More than the sum of its parts — pattern mining, neural networks, and how they complement each other

A dissertation submitted towards the degree
Doctor of Natural Sciences (Dr. rer. nat.)
of the Faculty of Mathematics and Computer Science
of Saarland University

by Jonas Fischer

Saarbrücken, 2022

Dean	Jürgen Steimle
Date of the defence	28th of July, 2022

Examination board

Chair	Prof. Dr. Sven Rahmann
Reviewer	Prof. Dr. Jilles Vreeken
Reviewer	Prof. Dr. Gerhard Weikum
Reviewer	Prof. Dr. Srinivasan Parthasarathy
Academic staff	Dr. Paramita Mirza

Abstract

In this thesis we explore pattern mining and deep learning. Often seen as orthogonal, we show that these fields complement each other and propose to combine them to gain from each other's strengths. We, first, show how to efficiently discover succinct and non-redundant sets of patterns that provide insight into data beyond conjunctive statements. We leverage the interpretability of such patterns to unveil how and which information flows through neural networks, as well as what characterizes their decisions. Conversely, we show how to combine continuous optimization with pattern discovery, proposing a neural network that directly encodes discrete patterns, which allows us to apply pattern mining at a scale orders of magnitude larger than previously possible. Large neural networks are, however, exceedingly expensive to train for which 'lottery tickets' – small, well-trainable sub-networks in randomly initialized neural networks – offer a remedy. We identify theoretical limitations of strong tickets and overcome them by equipping these tickets with the property of universal approximation. To analyze whether limitations in ticket sparsity are algorithmic or fundamental, we propose a framework to plant and hide lottery tickets. With novel ticket benchmarks we then conclude that the limitation is likely algorithmic, encouraging further developments for which our framework offers means to measure progress.

Zusammenfassung

In dieser Arbeit befassen wir uns mit Mustersuche und Deep Learning. Oft als gegensätzlich betrachtet, verbinden wir diese Felder, um von den Stärken beider zu profitieren. Wir zeigen erst, wie man effizient prägnante Mengen von Mustern entdeckt, die Einsichten über konjunktive Aussagen hinaus geben. Wir nutzen dann die Interpretierbarkeit solcher Muster, um zu verstehen wie und welche Information durch neuronale Netze fließen und was ihre Entscheidungen charakterisiert. Umgekehrt verbinden wir kontinuierliche Optimierung mit Mustererkennung durch ein neuronales Netz welches diskrete Muster direkt abbildet, was Mustersuche in einigen Größenordnungen höher erlaubt als bisher möglich. Das Training großer neuronaler Netze ist jedoch extrem teuer, für das 'Lotterietickets' – kleine, gut trainierbare Subnetzwerke in zufällig initialisierten neuronalen Netzen – eine Lösung bieten. Wir zeigen theoretische Einschränkungen von starken Tickets und wie man diese überwindet, indem man die Tickets mit der Eigenschaft der universalen Approximierung ausstattet. Um zu beantworten, ob Einschränkungen in Ticketgröße algorithmischer oder fundamentaler Natur sind, entwickeln wir ein Rahmenwerk zum Einbetten und Verstecken von Tickets, die als Modellfälle dienen. Basierend auf unseren Ergebnissen schließen wir, dass die Einschränkungen algorithmische Ursachen haben, was weitere Entwicklungen begünstigt, für die unser Rahmenwerk Fortschrittsevaluierungen ermöglicht.

This thesis is dedicated to my parents and my sister, who supported me during all—first small and then steadily growing—steps of my life.

Acknowledgments

My gratitude goes out to all individuals that took part in this amazing journey, my colleagues and friends that I met on the way and that provided a great environment to work in, and Valentina Galata in particular, for drawing the amazing thesis cover. Thanks go out also to my collaborators and mentors on the way for their great contributions which served for outstanding joint projects bridging different fields. I further want to thank my parents, my sister, and my girlfriend, for their constant moral support and genuine interest in what I am doing. Finally, special thanks go out to my supervisor Jilles. It was a great journey and, as we say in german, "Der Weg ist das Ziel", it was indeed the journey itself that provided for all the fun. Yet, I am also more than happy with where it ended: this thesis. All of this would not have been possible without you, the freedom you gave me to follow my own ideas, and your trust in me. Thank you!

TABLE OF CONTENTS

INTRODUCTION

1

MOTIVATION

Asking the right question and forming a matching hypothesis lays at the heart of scientific research. These hypotheses then need to be validated by thorough testing, which requires gathering data from carefully designed experiments. Gathering large amounts of such experimental data routinely, supported by recent technological leaps, massively propelled research. These advances are particularly strong in sciences such as molecular biology and medicine. With these large amounts of data produced on a daily basis, the question extended from whether the collected data supports or refutes a given hypothesis, but also what else we can learn and infer by exploiting the richness of the data. For example, gene expression measured in lung tissue could contain information about gene interactions, which could give insight in a disease, or even its cause and effects, and could lead to the discovery of marker genes or potential drug targets. That is, rather than explicitly formulated by the scientist, a hypothesis – e.g., a particular gene is a marker of the disease – is generated by exploring the data, rather than consulting the data with a hypothesis.

This is precisely the question that *data mining* asks: What can we learn from data? As part of this field, in *pattern mining*, the goal is to automatically discover human-interpretable *patterns* that capture the main regularities observed across the features of a given data set. These patterns thus generate insights into the processes underlying the data and therewith allow domain experts to postulate new hypotheses. Considering the gene expression example from above, pattern mining can reveal interesting interaction patterns of genes, across whole popu-

lations or differential between diseased and healthy individuals. Those patterns can then be leveraged for further investigation by a domain expert as potential biomarkers, drug targets, or to improve the foundational understanding of gene regulation as a whole.

In *machine learning* the data is instead exploited with a different goal, that is, to learn predictive models. In the last decade artificial neural networks stood at the forefront of machine learning, solving complex tasks in e.g. image recognition and natural language processing often at human-like performance or even surpassing it (He et al., 2015; Hu et al., 2018; Amodei et al., 2016; Devlin et al., 2019). These models, furthermore, solved long-open computational challenges in science such as predicting protein folding (Jumper et al., 2021) or controlling nuclear fusion reactors (Degrave et al., 2022). While these models steadily produce new sensational results and excell in predictive performance, they are, however, opaque and do not lend themselves for easy interpretation. At the heart of these successes lays that these models also rely on patterns in the data, which begs the question what regularities these networks exploit to achieve their performance, and, what we can learn from them. This task is again exploratory in nature, but now is concerned with explaining a model, e.g. a trained neural network, in human-interpretable terms, rather than the data. When training a neural network to predict lung diseases, we are likely to find that the model uses the presence of a pattern of genes to predict a disease state. Such information can on the one hand be used to get insights about the data, and on the other hand can provide clues about model robustness, reliability, and generalization, which are critical factors considering for instance an AI system in medical care.

The performance of neural networks also comes at a different cost apart from opaqueness, they are prohibitively expensive to train and deploy. Learning smaller networks would not only be a remedy, but these would also be inherently more interpretable. There is, hence, a huge incentive to find ways to reduce their size, opposing the trend of ever-growing networks built to solve more and more complex tasks. Although it should theoretically be possible and despite tremendous research efforts, small neural networks, so far, could not reach the performance of their larger counterparts when trained from scratch.

In this thesis we study each of these problems in turn.

CHALLENGES

Arguably the central challenge in data mining is that of discovering interpretable patterns that capture the regularities in the data at hand. The types of data considered for discovery include graphs, sequences, continuous, integral, and nominal or binary data. We are here interested in binary data, both fundamen-

tal building block, as well as naturally arising in many application domains including biology and medicine. There, experts are for example interested to find the patterns of genetic and epigenetic control that govern cellular regulation in (healthy) individuals, and the aberrant patterns linked to diseases. These patterns could be of genes switched on or off, epigenetic marks such as DNA methylation or histone modifications being set or not, or the presence of a mutation in the genome, all of which are naturally modeled by binary variables.

More generally, we can define this instance of pattern mining as

Problem 1 (The pattern set mining problem)**.** *Given binary data $D \in \{0,1\}^{n,m}$ of n samples over m binary variables, find a set of patterns $\mathcal{P}$ over a language $\mathcal{L}$ that captures the non-spurious and non-redundant regularities in the data.*

As we will see, how we define the pattern language $\mathcal{L}$ is essential to the descriptive power of patterns $P \in \mathcal{P}$ and consequently what types of regularities we can capture in data. Despite a classical problem of data mining, methods mostly consider the language of logical conjunctions, or association rules over those conjunctive statements. One of the first fields of application for pattern mining was shopping transaction data (Agrawal et al., 1993), e.g. the buying history of individuals at a retailer, yet, as we will see the state-of-the-art for pattern mining does not scale well beyond a few thousand of variables, rendering it impractical for this application. More importantly, these approaches are also not suited to take on the emerging biological applications, which come with similarly large and high-dimensional data. Besides, many approaches find millions of spurious or redundant results, as we also find in our experiments, which are hardly interpretable by a human expert. There is, hence, ample room for improvement.

Strongly related is the problem of finding patterns that describe regularities in data partitions. For example, what are epigenetic patterns that are distinct to healthy individuals, and what are those distinct to patients? This problem is referred to as subgroup discovery, emerging pattern mining, or variants of significant pattern mining, depending on considered data type, label type, and methodology (Atzmueller, 2015; García-Vico et al., 2018; Llinares-López et al., 2015). The overarching problem of discovering regularities describing a given partitioning we will, here, call differential pattern mining, emphasizing the differential descriptions that these patterns yield, detaching it from specific data types or methodologies. Given data D along with a binary label L that partitions the data into two parts D_1, D_2, we define this problem as

Problem 2 (The differential pattern mining problem)**.** *Given two partitions of binary data $D_1 \in \{0,1\}^{n_1,m}$, $D_2 \in \{0,1\}^{n_2,m}$, find patterns $\mathcal{P}_1, \mathcal{P}_2$ that capture the regularities distinct to each partition.*

Several methods have been proposed that find interesting differential patterns, yet, as we will see, often have issues partially shared with the traditional

pattern mining methods. For example, some find mostly redundant or spurious results, suffer from multiple hypothesis testing, or have difficulties with highly imbalanced data partitions. What all of them have in common is that they barely scale to a few thousand features.

Both the presented pattern mining problems lay at the heart of data mining and are formulated general enough to be fitted to a wide range of applications. In this thesis, we also consider the problems of explaining – in human-interpretable terms – the decision process of machine learning models, and what data makes them err. This is especially important for opaque neural network models acting in a safety-critical environment, but also might generate new insights into the regularities underlying the data. Finding patterns in a neural network that explain its decision process is an instance of Problem 1, where the data D consists of the neuron activations observed for each sample. To describe associations of neuron activations across, for example, network layers, however, requires a rich pattern language $\mathcal{L}$. Similarly, to understand what patterns in the data make a network err, we can define data partitions D_1, D_2 to be that input data which lead to correct respectively incorrect prediction for the given network, which makes the problem of understanding misclassifications an instance of Problem 2.

Lastly, we turn to the problem of finding small but well performing neural networks. Neural networks are widely celebrated for their successful application to challenging tasks, but are immensely over-parameterized and, hence, expensive to train and hard to interpret. Instead of training a small network from scratch, which commonly fails to reach the desired performance, recent proposals suggested to identify a small sub-network θ' of an initialized large network θ, where θ' has orders of magnitude fewer parameters but can be trained with equal success, referred to as *lottery ticket*.

Problem 3 (Identifying lottery tickets)**.** *Given an initialized neural network with parameters θ, identify a small sub-network $\theta' \subset \theta$ with $|\theta'| \ll |\theta|$, that trains to equal performance as the full network.*

Recent theoretical and empirical results confirmed that these sub-networks or lottery tickets indeed exist, and most successful approaches to identify lottery tickets rely on pruning of a large network. Such pruning approaches, if efficient enough, bear the promise to overcome the issues of overparametrization, being inherently more interpretable and saving a significant amount of resources. We next discuss the contributions we make to each of the three problems.

CONTRIBUTIONS

The main focus of this work in the context of pattern mining is two-fold. First, we want to be able to find richer structures that better capture the regularities

of the given data. In the previous section we intentionally left the definition of a pattern language $\mathcal{L}$ vague and will briefly discuss the importance of it and its impact on the results here. Second, we want approaches that scale well with increasing number of samples n as well as features – or items – m in the data. This allows to handle the interesting emerging applications in biology and medicine, as well as to cope with the ever-growing data collections for the classical exploratory problems, such as transactional retail data.

Contribution 1. The first contribution that we make is that of discovering sets of robust rules $X \to Y$ which express dependencies of a set of co-occurring features Y on a set of co-occurring features X. We show that, in contrast to the state-of-the-art, our rule sets are robust to noise in the data, and can express conjunctive statements of arbitrary length in both heads X and tails Y. The discovery of such rules is an instance of *Problem 1*, where the pattern language $\mathcal{L}$ is that of rules of the form described above. We approach this problem with a pattern set mining approach and identify a good rule set in terms of the Minimum Description Length (MDL) principle. We further propose an efficient bottom-up heuristic that allows to find succinct, non-redundant, and non-spurious sets of rules that capture important regularities in the data, which we show on both synthetic as well as real world data. In our experiments we further confirm that our approach scales to thousands of rows and columns in the data, which makes it well suited for the emerging biological applications such as gene expression data.

Contribution 2. Rules capture important interpretable dependencies in data. Biological and medical data, however, also contain relationships of mutual exclusivity, such as genes acting as antagonists in pathways, or replacable subcomponents in protein complexes. The existing works are usually not able to capture those relationships to their full extent, despite the importance of such patterns for drug discovery, discovery of biomarkers, and the foundational understanding of gene regulation. In our second contribution, we reconsider *Problem 1* by proposing an expressive pattern language of co-occurrences and mutual exclusivity, and mixtures of those. Formulated as a pattern set mining problem, we identify descriptive pattern sets in terms of MDL, and propose an approximation that allows to scale to tens of thousands of features, showing its exceptional power in a novel application to single-cell gene expression data.

Contribution 3. Existing work on pattern mining, including our previous contributions, usually explore the discrete space of all patterns or pattern sets by combinatorial optimization techniques. While we are able to consider an order of magnitude more features by more efficient pruning and approximations as well as optimized code, we slowly reach a limit of the considerable

data with respect to size. This limitation is largely due to the enumeration of the search space. To be able to scale to hundred thousands or even millions of features, typically encountered for example at large online retailers or in genetic variation studies, we, hence, need to rethink how we model pattern mining problems. In our third contribution, we revisit *Problem 1*, and present a novel type of binarized autoencoder, which is able to discover human-interpretable sets of conjunctive patterns through gradient based optimization. It thus provides a link between learning of discrete patterns and continuous optimization, overcoming the issues of traditional combinatorial search in pattern mining. Putting these models into practice, we consider the large 1000 Genomes dataset, which captures genomic variations of a human population, and show that our models are able to retrieve insightful biological patterns across several hundred thousands of mutations present in this data.

Contribution 4. With the exceptional performance of neural networks (NNs) there is a strong research drive to understand what these models learn to succeed in their task. In our fourth contribution, we turn to the problem of understanding these black box models through the lense of pattern mining. Building on Contribution 1, we next turn to a variation of *Problem 1*, where we consider binarized activations of NNs as data D and are interested in finding rules that give insights into the inner workings of NNs. Contemporary work on explaining NNs mostly focuses on local explanations, on input-output relationships, or aims to find interpretable models that mimic the NNs' decision. Also classical rule mining fails as it is neither expressive enough nor finds crisp descriptions of the activation patterns, which are necessary for human interpretation. We here fill this gap by extending the language of rules further to express disjunctive statements in both heads and tails in a set mining framework. We are thus able to capture regularities across e.g. class labels, which are inherently disjunct, and the noisy co-activations observed in real-world NNs. In applications to convolutional neural networks (CNNs) trained on real-world data, we get a glimpse into how networks learn to generalize across classes, distinguish between them, and how information flows through the network.

Contribution 5. Analyzing how NNs arrive at a decision is crucial to understand how they 'reason', but also whether their reasoning is robust and reliable. What is even more important is to understand where they fail, both to understand their shortcomings as well as to improve them. Proposing a solution to *Problem 2*, we then turn towards analyzing misclassification of NNs on complex language tasks using our proposal, comparing it to a wide range of approaches from rule mining, subgroup discovery, statistical pattern mining, and classical machine learning on both synthetic and real world data. While existing work shows to struggle with the amounts of noise and the high imbal-

ance of data of misclassified and classified samples, our approach discovers patterns of mutually excluvise and co-occuring words that differentiate correctly from incorrectly predicted instances. We show that these patterns not only provide interesting insights in what a network deems challenging, but are also actionable.

The success of neural networks not only comes at the cost of black box models, but is usually also overshadowed by the often prohibitively expensive training process. The research efforts to overcome these issues through smaller, yet equally well performing neural networks, captured by *Problem 3*, recently gained traction by the conjecture and formal proof of the lottery ticket hypothesis (LTH). We consider two types of tickets, *weak lottery tickets*, which are sub-networks that can be trained to the same performance as their dense counterpart, and *strong lottery tickets*, which are sub-networks that, *at initialization*, perform as good as their dense counterpart. Identifying the latter can be seen as the holy grail in the quest for resource-efficient networks, as these tickets are efficient at inference and save training entirely, but are inherently difficult to find. While not as resource-saving, substantial progress regarding *Problem 3* has been made by discovering weak tickets, which found wide-spread application in computer vision (Frankle et al., 2020), natural language processing (Yu et al., 2020), and adversarial robustness (Fu et al., 2021). Our contributions discuss and answer several fundamental questions pertaining to lottery tickets.

Contribution 6. Current initialization schemes for NNs initialize bias terms to zero. Considering NNs with ReLU activations, which are standard in modern architectures, we formally show that zero-initialized biases prevent strong lottery tickets from obtaining the universal approximation property. Based on this insight, we extend multiple initialization schemes to non-zero biases, show that these initializations are well trainable, and prove the existence of strong lottery tickets in this setting. On benchmark data, we show the practical benefits of these initializations, which equip the tickets with the ability for universal approximation, leading to tickets of greater sparsity. We additionally use a theoretical insight on parameter rescaling during the pruning process to further improve the sparsity of discovered strong tickets.

Contribution 7. The state-of-the-art lottery ticket pruners successfully retrieve well performing sub-networks for a variety of tasks. However, their sparsity and, hence, their ability to save resources, is limited. Furthermore, due to missing benchmarks these methods were only evaluated against their dense counterpart and existing pruning methods. This issue brings up the following essential question: Is the lack of sparse solutions fundamental, i.e. sparser

tickets do not exist, or of algorithmic nature, i.e. the current algorithms can not identify an existing sparser solution?

To discuss this problem, we first provide a lower bound on the probability of existence for strong lottery tickets, and derive a framework to plant and hide lottery tickets in target networks. Generating hand-crafted ground truth tickets for three challenging problems, which we hide in large randomly initialized networks, we then set on to test the state-of-the-art's ability to retrieve them, providing for the first time a ticket pruning benchmark. We identify several shortcomings of existing work and highlight their potential for improvement and can answer that the lack of sparse solutions is due to an algorithmic limitation of current methods rather than a fundamental limitation. Furthermore, the observed trends are consistent with results on real world image data, which shows that our benchmarks, while artificial in nature, reflect realistic conditions, serving as a measure of progress in the quest for ticket pruning.

The list of publications corresponding to all contributions can be found in Tab. 1.1. In each of the publications the author of this thesis was involved as first author, contributing to the main ideas, theoretical work, implementation, experiments, and writing of the manuscript. For contribution 5 and 6 there was a shared first authorship, where both authors contributed equally to the work, taking part in each of the previously mentioned steps.

ORGANIZATION

This thesis is organized into nine chapters, with Chapters 2-8 each dedicated to one contribution (see Tab. 1.1) and the last chapter containing a conclusion of the work. For a better read, the chapters are ordered thematically rather than by problem statements. That is, we first discuss how to discover sets of robust rules in Chapter 2 and how to leverage rule set mining to explain neural networks in Chapter 3. We then extend the classical conjunctive pattern language to combinations of conjunctive and mutual exclusivity statements in Chapter 4, and propose to identify label-descriptive patterns over such a richer language in Chapter 5, where we apply it to better understand the systematic errors made by neural networks in natural language processing. In Chapter 6 we then discuss how to overcome scalability issues of current pattern mining approaches by introducing a novel type of binarized autoencoder that allows to explore the discrete search space of patterns through efficient continuous optimization. In Chapter 7 we extend the strong lottery tickets to neural network initialization schemes with nonzero biases to equip these tickets with the universal approximation property. We then propose a framework to plant and hide lottery tickets in randomly initialized neural networks in Chapter 8 and generate extremely

Authors	Title	Venue
Fischer, J, and Vreeken, J	*Sets of robust rules, and how to find them.*	ECMLPKDD 2019
Fischer, J, and Vreeken, J	*Discovering succinct pattern sets expressing co-occurrence and mutual exclusivity.*	KDD 2020
Fischer, J, and Vreeken, J	*Differentiable pattern set mining.*	KDD 2021
Fischer, J, Oláh, A, Vreeken, J	*What's in the box? Exploring the inner life of neural networks with robust rules.*	ICML 2021
Hedderich*, M, Fischer*, J, Klakow, D, Vreeken, J	*Label-descriptive patterns and their application to characterizing classification errors.*	ICML 2022
Fischer*, J, Gadhikar*, A, Burkholz, R	*Towards strong pruning for lottery tickets with nonzero bias.*	*preprint*
Fischer, J, and Burkholz, R	*Plant 'n' Seek: Can you find the winning ticket?*	ICLR 2022

Table 1.1: *List of publications.* Given are the publications ordered by contribution number. Equal contribution of authors is indicated by *.

sparse lottery ticket benchmarks to challenge the state-of-the-art pruners, for the first time allowing to quantify the (in)efficiency of these algorithms with respect to the sparsity of the discovered tickets.

Each of these chapters is organized into a general introduction to the specific topic, discussion of the related work and necessary notation, followed by theoretical, methodological, and experimental contributions, and ended with a discussion and conclusion section. We lay out the main related work of pattern mining in the first chapter, and discuss the additional work relevant to each contribution in the corresponding chapters. At the end of this thesis, in Chapter 9, we will conclude the work and discuss its relevance within data mining and machine learning.

SETS OF ROBUST RULES, AND HOW TO FIND THEM

2

2.1 INTRODUCTION

Association rules are among the most important primitives in data mining. Rules of the form $X \rightarrow Y$ are not only simple to understand, but they are also simple to act upon, and, most importantly, can express important *local* structure in the data. The problem is, however, that there are so many of them, and that telling the interesting from the uninteresting rules has so far proven extremely difficult. Both traditional algorithms based on support and confidence (Agrawal and Srikant, 1994), as well as modern approaches based on statistical tests (Hämäläinen, 2012), typically discover orders of magnitude more rules than the data has rows – even when the data consists of nothing but noise. In this paper we show how to discover a small, non-redundant set of noise-resistant rules that together describe the data well.

To succinctly express subtly different structures in data, we allow *multiple* items in the consequent of a rule. To illustrate, while rule sets $R_1 = \{A \rightarrow B, A \rightarrow C\}$ and $R_3 = \{A \rightarrow BC\}$ both express that B and C appear frequently in the context of A, the former states they do so independently, while the latter expresses a dependency between B and C. We additionally allow for *patterns*, which are simply rules like $R_4 = \{\emptyset \rightarrow ABC\}$ and express unconditional dependencies. Real data is often noisy, and hence we can allow rules to hold *approximately*. That is, for a transaction $t = ABC$, our models may infer that

This chapter is based on Fischer and Vreeken (2019).

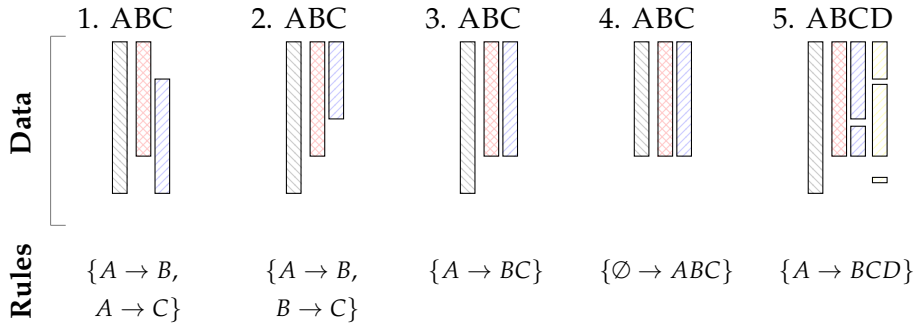


Figure 2.1: *Five toy databases with corresponding rule sets.* 1) B and C occur in the context of A but independently of each other, 2) C occurs in the context of B, which in turn occurs in the context of A, 3) B and C show strong joint dependence in the context of A, 4) A, B, C show strong unconditional dependence, and 5) a rule with noise, BCD occuring jointly in the context of A.

rule $R_5 = \{A \to BCD\}$ holds, even though item D is not present in t. We call these noise-resistant, or *robust* rules. To determine the quality of a rule set for given data, we rely on information theory.

In particular, we define the rule set mining problem in terms of the Minimum Description Length (MDL) principle (Grünwald, 2007). Loosely speaking, this means we identify the best rule set as that one that compresses the data best. This set is naturally non-redundant, and neither under- nor over-fitting, as we have to pay for every additional rule we use, as well as for every error we make. We formally show that the resulting problem is neither submodular, nor monotone, and as the search space is enormous, we propose Grab, an efficient any-time algorithm to heuristically discover good rule sets directly from data. Starting from a singleton-only model, we iteratively refine our model by considering combinations of rules in the current model. Using efficiently computable tight estimates we minimize the number of candidate evaluations, and as the experiments show, Grab is both fast in practice, and yields high quality rule sets. On synthetic data, Grab recovers the ground truth, and on real-world data it recovers succinct models of meaningful rules. In comparison, state of the art methods discover up to several millions of rules for the same data, and are hence hardly useful.

2.2 RELATED WORK

Pattern mining is arguably one of the most important and well-studied topics in data mining. We aim to give a succinct overview of the work most relevant

to ours. The first, and perhaps most relevant proposal is that of association rule mining (Agrawal and Srikant, 1994), where in an unsupervised manner the goal is to find all rules of the form $X \rightarrow Y$ from the data that have high frequency and high confidence. As it turns out to be straightforward to distill the high-confidence rules from a given frequent itemset, research attention shifted to discovering frequent itemsets more efficiently (Mannila et al., 1994; Zaki et al., 1997; Han et al., 2000) and in settings of distributed (Otey et al., 2003) or evolving databases (Veloso et al., 2002). While all these approaches indeed efficiently mine all rules, they suffer from the pattern explosion problem – where millions of itemsets or rules are found even for small data sets. To tackle this issue, Webb and Zhang (2005) suggested a top-k approach that efficiently mines the k most interesting rules using efficient branch and bound search (Webb, 1995). In a related line of work, research focused instead on eliminating redundancy in the discovered rule sets based on the concepts of maximal frequent itemsets and closedness properties (Bayardo, 1998; Pasquier et al., 1999; Calders and Goethals, 2007; Moerchen et al., 2011). Wang and Parthasarathy (2006) instead propose to summarize itemsets through markov random fields, modeling items as random variables.

Frequency alone, however, turned out to be a bad measure of interestingness, as it leads to spurious patterns (Webb, 2007; Vreeken and Tatti, 2014). To alleviate this, statistically sound measures were proposed to mine individual patterns based on margins (Pellegrina and Vandin, 2018; Pellegrina et al., 2019; Papaxanthos et al., 2016) or based on richer background knowledge (Jaroszewicz and Simovici, 2004; Tatti, 2008; De Bie, 2011), or to use well-founded information theoretic approaches to mine sets of patterns as a whole, based on the Minimum Description Length Principle (Vreeken et al., 2011; Smets and Vreeken, 2012) or maximum entropy modeling (Mampaey et al., 2012; Dalleiger and Vreeken, 2020). Perhaps because it is already difficult enough to determine the interestingness of a pattern, let alone a rule, most proposals restrict themselves to patterns. Key exceptions are Kingfisher (Hämäläinen, 2012), which proposes an upper bound for Fisher's exact test that allows to efficiently mine significant dependency rules, yet suffers from the multiple test correction problem, and an extension of Magnum Opus, a frequency based pattern and rule mining framework refined with statistically sound testing of discovered patterns (Webb, 2007, 2011). Notably, however, both these proposals can only discover exact rules with a single item consequent.

Less directly related to our problem setting, but still relevant, are supervised descriptive rule discovery approaches (Wang and Rudin, 2015; Papaxanthos et al., 2016), emerging pattern mining (Dong and Li, 1999), contrast sets (Bay and Pazzani, 2001), and subgroup discovery (Wrobel, 1997). For a general overview we refer to Zimmermann and Nijssen (2014) respectively Novak et al. (2009). We will further discuss this line of research in Chapter 5. For now,

however, we are not interested in rules that explain a certain target, but rather aim for a set of rules that together explains *all* of the data well.

Our approach is a clear example of pattern set mining (Vreeken and Tatti, 2014). That is, rather than measuring the quality of individual patterns – or rules – we measure quality over a set of patterns (Vreeken et al., 2011; Fowkes and Sutton, 2016). Information theoretic approaches, such as MDL and the Maximum Entropy principle, have proven particularly successful for measuring the quality of sets of patterns (Vreeken et al., 2011; Mampaey et al., 2012). Most pattern set approaches do not account for noise in the data, with Asso (Miettinen and Vreeken, 2014), Hyper+ (Xiang et al., 2008), and Panda (Lucchese et al., 2010) as notable exceptions. However, extending any of the above from patterns to rules turns out to be far from trivial, because rules have different semantics than patterns. Pack (Tatti and Vreeken, 2008) uses MDL to mine a small decision tree per item in the data, and while not technically a rule-mining method, we can interpret the paths of these trees as rules. In our experiments we will compare to Kingfisher as the state-of-the-art significance based rule miner, Hyper+ as a representative of noise resilient, frequency based pattern miner, and Pack as an MDL based pattern miner, which output can be translated into rules. While Magnum Opus suits our goal, it is only commercially available and hence excluded from our experiments.

2.3 PRELIMINARIES

In this section we discuss preliminaries and introduce notation.

2.3.1 NOTATION

We consider binary transaction data D of size n-by-m, with $n = |D|$ transactions over an alphabet $\mathcal{I}$ of $m = |\mathcal{I}|$ items. In general, we denote sets of items as $X \subseteq \mathcal{I}$. A transaction t is an itemset, e.g. the products bought by a customer. We write $\pi_X(D) := \{t \cap X \mid t \in D\}$ for the projection of D on itemset X. The transaction set, or selection, T of itemset X is the set of all transactions $t \in D$ that contain X, i.e. $T_X = \{t \in D \mid X \subseteq t\}$. We write $n_X = |T_X|$ to denote the cardinality of a transaction set. The *support* of an itemset X is then simply the number of transactions in D that contain X, i.e. $\mathit{support}(X) = |T_X|$.

An association rule $X \to Y$ consists of two non-intersecting itemsets, the antecedent or head X, and consequent or tail Y. A rule makes a statement about the conditional occurrence of Y in the data where X holds. If $X = \emptyset$, we can interpret a rule as a pattern, as it makes a statement on where in the whole data the consequent holds. Throughout this manuscript, we will use A, B, C to refer to sets of single items and X, Y, Z for itemsets of larger cardinality.

2.3.2 MINIMUM DESCRIPTION LENGTH

The Minimum Description Length (MDL) principle (Rissanen, 1978; Grünwald and Roos, 2019) is a computable and statistically well-founded approximation of Kolmogorov Complexity (Li and Vitányi, 1993). For given data D, MDL identifies the best model M^* in a given model class $\mathcal{M}$ as that model that yields the best lossless compression. In one-part, or, *refined* MDL we consider the length in bits of describing data D using the entire model class, $L(D \mid \mathcal{M})$, which gives strong optimality guarantees (Grünwald, 2007) but is only attainable for certain model classes. In practice we hence often use two-part, or, *crude* MDL, which is defined as $L(M) + L(D \mid M)$. Here the length of the description of the model $L(M)$, and the length in bits of the description of the data $L(D \mid M)$ using M are computed separately. We will use two-part codes where we have to, and one-part codes where we can. Note that in MDL we are only concerned with code *lengths*, not materialized codes. Also, as we are interested in measuring lengths in bits, all logarithms are to base 2, and we follow the convention $0 \log 0 = 0$.

2.4 THEORY

To use MDL in practice, we first need to define our model class $\mathcal{M}$, how to describe a model M in bits, and how to describe data D using a model M. Before we do so formally, we first give the intuitions.

2.4.1 THE PROBLEM, INFORMALLY

Our goal is to find a set of rules that together succinctly describe the given data. Our models M hence correspond to sets R of rules $X \to Y$. A pattern ABC is simply a rule with an empty head, i.e. $\emptyset \to ABC$. A rule *applies* to a transaction $t \in D$ if the transaction supports its head, i.e. $X \subseteq t$. For each transaction to which the rule applies, the model specifies whether the rule *holds*, i.e. whether Y is present according to the model. We can either be strict, and require that rules only hold when $Y \subseteq t$, or, be more robust to noise and allow the rule to hold even when not all items of Y are part of t, i.e. $Y \setminus t \neq \emptyset$. In this setting, the model may state that rule $A \to BCD$ holds for transaction $t = ABC$, even though $D \notin t$ (see Fig. 2.1.5). A model M hence needs to specify for every rule $X \to Y \in R$ a set of transactions $T^M_{Y|X}$ where it asserts that Y holds in the context of X, and, implicitly also $T^M_{\not Y|X}$, the set of transactions where it asserts Y does not hold. Last, for both these we have to transmit which items of Y are actually in the data; the fewer errors we make here, the cheaper it will be to transmit. To ensure that we encode any data D over $\mathcal{I}$, we require that a model M contains

at least singleton rules, i.e. $\emptyset \to A$ for all $A \in \mathcal{I}$. Cyclic dependencies would prevent us from decoding the data without loss. Any valid model M can hence be represented as a directed acyclic graph (DAG), in which the vertices of the graph correspond to rules in R, where vertex $r = X \to Y$ has incoming edges from all vertices $r' = X' \to Y'$ for which $X \cap Y'$ is non-empty.

We explicitly allow for rules with non-singleton tails, as this allows us to succinctly describe subtly different types of structure. When B happens independently of C in T_A (Fig. 2.1.1), rule set $R_1 = \{A \to B, A \to C\}$ is a good description of this phenomenon. In turn, when C occurs often – but not always – in T_B, which in turn happens often in T_A (Fig. 2.1.2) rule set $R_2 = \{A \to B, B \to C\}$ is a good description. To succinctly describe that B and C are statistically dependent in T_A (Fig. 2.1.3) we need rules with multiple items in its tail, i.e. $R_3 = \{A \to BC\}$. Finally, if A, B, and C frequently occur jointly, but conditionally independent of any other variable, we need patterns to express this, which are just consequents in the context of the whole database $R_4 = \emptyset \to ABC$.

2.4.2 MDL FOR RULE SETS

Next, we formalize an MDL score for the above intuition. We start by defining the cost of the data given a model, and then define the cost of a model.

COST OF THE DATA We start with the cost of the data described by an individual rule $X \to Y$. For now, assume we know $\pi_X(D)$ and T_X. We transmit the data over Y in the context of X, i.e. $D_{Y|X} = \pi_Y(T_X)$, in three parts. First, we transmit the transaction ids where model M specifies that both X and Y hold, $T^M_{Y|X}$, which implicitly gives $T^M_{\not Y|X} = T_X \setminus T^M_{Y|X}$. We now, in turn transmit that part of $D_{Y|X}$ corresponding to the transactions in $T^M_{Y|X}$, resp. that part corresponding to $T^M_{\not Y|X}$. We do so using optimal data-to-model codes, i.e. indices over canonically ordered enumerations,

$$L(D_{Y|X} \mid M) = \log \binom{|T_X|}{|T^M_{Y|X}|} + \log \binom{|T^M_{Y|X}| \times |Y|}{\mathbb{1}(T^M_{Y|X})} + \log \binom{|T^M_{\not Y|X}| \times |Y|}{\mathbb{1}(T^M_{\not Y|X})} ,$$

where we write $\mathbb{1}(T^M_{Y|X})$ for the number of 1s in $T^M_{Y|X}$, i.e.

$$\mathbb{1}(T^M_{Y|X}) = \sum_{t \in T^M_{Y|X}} |t \cap Y| \leq |T^M_{Y|X}| \times |Y| .$$

We define $\mathbb{1}(T_{\not Y|X})$ analogue.

When the model makes exact assertions on Y holding when X is present, i.e. when $T^M_{Y|X} = T_{Y|X}$, the second term vanishes, and analogously for the third term when $T^M_{\not Y|X} = T_{\not Y|X}$. Both terms vanish simultaneously only when $D_{Y|X} \in \{\emptyset, Y\}^{|D_X|}$. This is trivially the case when Y is a singleton.

The overall cost of the data given the model simply is the sum of the data costs per rule,

$$L(D \mid M) = \sum_{X \to Y \in M} L(D_{Y|X} \mid M)$$

To decode the data, the recipient will of course need to know each rule $X \to Y$. These are part of the model cost.

Cost of the Model To encode a rule, we first encode the cardinalities of X and Y using $L_{\mathbb{N}}$, the MDL-optimal code for integers $z \geq 1$, which is defined as $L_{\mathbb{N}}(z) = \log^* z + \log c_0$, where $\log^* z = \log z + \log\log z + ...$, and c_0 is a normalization constant such that $L_{\mathbb{N}}$ satisfies the Krafft-inequality (Rissanen, 1983). We can now encode the items of X, resp. Y, one by one using optimal prefix codes, $L(X) = -\sum_{x \in X} \log \frac{s_x}{\sum_{i \in \mathcal{I}} s_i}$. Last, but not least we have to encode its parameters, $|T^M_{Y|X}|$, $\mathbb{1}(T^M_{Y|X})$, and $\mathbb{1}(T_{\not Y|X})$. These we encode using a refined, mini-max optimal MDL code. In particular, we use the regret of the Normalized Maximum Likelihood code length (Kontkanen and Myllymäki, 2007) for the class of binomials,

$$L_{pc}(n) = \log \left(\sum_{k=0}^{n} \frac{n!}{(n-k)!k!} \left(\frac{k}{n}\right)^k \left(\frac{n-k}{n}\right)^{n-k} \right) ,$$

which is also known as the parametric complexity of a model class. Kontkanen and Myllymäki (2007) showed that this term can be computed in time $O(n)$ in a recursive manner. We obtain the model cost $L(X \to Y)$ for a rule $X \to Y$ by

$$\begin{aligned} L(X \to Y) = {} & L_{\mathbb{N}}(|X|) + L(X) + L_{\mathbb{N}}(|Y|) + L(Y) + \\ & L_{pc}(|T_X|) + L_{pc}(|T^M_{Y|X}| \times |Y|) + L_{pc}(|T^M_{\not Y|X}| \times |Y|) \, . \end{aligned}$$

From how we encode the data we can simply ignore the last two terms for rules with $|Y| = 1$. The overall cost of a model M then amounts to

$$L(M) = L_{\mathbb{N}}(|R|) + \sum_{X \to Y \in R} L(X \to Y) \, ,$$

where we first send the size of rule set R, and then each of the rules in order defined by the spanning tree of the dependency graph. This concludes the

definition of our two-part MDL score given as

$$L(D, M) = L(M) + L(D \mid M).$$

For the interested reader, example computations are provided in Appendix A.1.1, which give practical intuition about the basic and noise encoding.

2.4.3 THE PROBLEM, FORMALLY

We can now formally define the problem in terms of MDL.

Problem 4 (Minimal Rule Set Problem)**.** *Given data D over items $\mathcal{I}$, find that rule set R and that set of $\mathcal{T}$ of tid-lists $T^M_{Y|X}$ for all $X \to Y \in R$, such that for model $M = (R, T)$ the total description length,*

$$L(D, M) = L(M) + L(D \mid M)$$

is minimal.

Solving this problem involves enumerating all possible models $M \in \mathcal{M}$. There exist $\sum_{i=0}^{|\mathcal{I}|} \left(\binom{|\mathcal{I}|}{i} \times 2^i \right) = 3^{|\mathcal{I}|}$ possible rules – where the second term in the sum describes all possible partitions of i items into head and tail, and the equality is given by the binomial theorem. Assuming that the optimal $T^M_{Y|X}$ are given, there are generally $2^{3^{|\mathcal{I}|}}$ possible models. The search space does not exhibit any evident structure that can be leveraged to guide the search, which is captured by the following two theorems. We postpone the proofs to Appendix A.1.

Theorem 2.1 (Submodularity)**.** *The search space of all possible sets of association rules 2^Ω, where Ω are rules over $\mathcal{I}$, when fixing a dataset and using the description length $L(D, M)$ as set function, is not submodular. That is, there exists a data set D s.t. $\exists X \subset Y \subseteq \Omega, z \in \Omega.\ L(D, X \cup \{z\}) - L(D, X) \leq L(D, Y \cup \{z\}) - L(D, Y)$.*

Theorem 2.2 (Monotonicity)**.** *The description length $L(D, M)$ on the space of all possible sets of association rules 2^Ω is not monotonously decreasing. That is, there exists a data set D s.t. $\exists X \subset Y \subseteq \Omega.\ f(X) \leq f(Y)$.*

Hence, we resort to heuristics.

2.5 GRAB

In this section we introduce GRAB, an efficient heuristic for discovering good solutions to the Minimal Rule Set Problem. GRAB consists of two steps, candidate generation and evaluation, that are executed iteratively until convergence of $L(D, M)$, starting with the singleton-only rule set $R_0 = \{\emptyset \to A \mid A \in \mathcal{I}\}$.

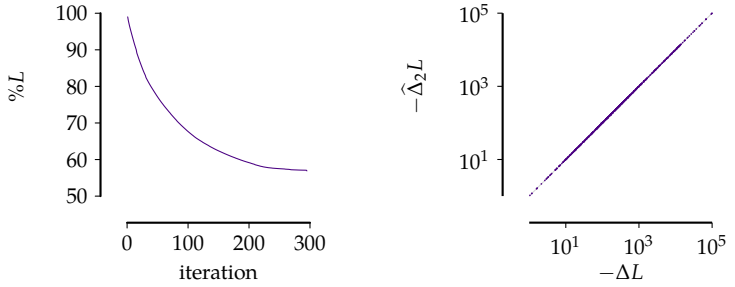
CANDIDATE GENERATION From the current rule set R we iteratively discover that refined rule set R' that minimizes the gain $\Delta L = L(D, M') - L(D, M)$. As refinements we consider the combination of two existing rules into a new rule for which we give pseudocode in Alg. 2.1.

In particular, we generate candidate refinements by considering all pairs $r_1 = X \to Y, r_2 = X \to Z \in R$, assuming w.l.o.g. $n_{XY} \geq n_{XZ}$, and merging the tails of r_1 and r_2 to obtain candidate rule $r_1' = X \to YZ$ (line 4), and merging the tail of r_1 with the head to obtain candidate rule $r_2' = XY \to Z$ (line 9). We now construct refined rule sets R_1' and R_2' by adding rule r_1' resp. r_2'. To reduce redundancy, we remove r_2 from both R_1' and R_2', and r_1 from R_1', taking care not to remove singleton rules. We only evaluate those refined rule sets R' for which the corresponding dependency graph is acyclic (line 7, 12), and select the one with minimal gain $\Delta L < 0$. Although we consider only pairs of rules for merging, GRAB converges fastly (see Fig. 2.2a).

Algorithm 2.1: GENERATECANDIDATES (D, M)

input : Database D, model $M = (R, \mathcal{T})$
output: Ordered candidate refinement set C

1 $G \leftarrow$ dependency graph of rule set R;
2 $C \leftarrow \emptyset$ // Candidate list
3 **for** $r_1 = X \to Y, r_2 = X \to Z \in R, n_Y \geq n_Z$ **do** // Rule pairs with same head
4 $r_1' \leftarrow X \to YZ$; $R_1' \leftarrow R + r_1' - r_1 - r_2$ // Merge the tails
5 **if** $\widehat{\Delta}_1(r_1') < 0$ **then** // Check gain estimates
6 **if** $\widehat{\Delta}_2(r_1') < 0$ **then**
7 **if** $G + r_1' - r_1 - r_2$ is acyclic **then**
8 $C \leftarrow C \cup (\widehat{\Delta}_2(r_1'), R_1')$
9 $r_2' \leftarrow XY \to Z$; $R_2' \leftarrow R + r_2' - r_2$ // Merge the head
10 **if** $\widehat{\Delta}_1(r_2') < 0$ **then** // Check gain estimates
11 **if** $\widehat{\Delta}_2(r_2') < 0$ **then**
12 **if** $G + r_2' - r_2$ is acyclic **then**
13 $C \leftarrow C \cup (\widehat{\Delta}_2(r_2'), R_2')$
14 **return** C



(a) Convergence of L.

(b) Estimate vs. actual gain for L.

Figure 2.2: GRAB *searches efficiently and estimates accurately.* For *DNA* we show (left) the convergence of the relative compression of model M at iteration i against the singleton model M_s, $\%L = \frac{L(D,M)\times 100}{L(D,M_s)}$, and (right) the correlation between estimated and actual gain of all evaluated candidates in real data.

GAIN ESTIMATION To avoid naively evaluating the gain ΔL of every candidate, we rely on accurate gain estimations. In particular, we consider two different estimates; the first estimate is very inexpensive to compute, but overly optimistic as it assumes a perfect overlap between the two rules. The second estimate is computationally more costly, as it requires us to compute the intersection of the tid lists of the two original rules. In practice, however, it is exact (see Fig. 2.2b).

Depending on how we combine two rules r_1 and r_2, we need different estimate definitions. In the interest of readability, we here consider one case in detail: that of combining singleton rules $r_1 = \emptyset \to A$ and $r_2 = \emptyset \to B$ into $r = A \to B$. For the remaining definitions we refer to Appendix A.1.4.

Following the general scheme described above, for the first estimate $\widehat{\Delta}_1$ we assume that $T_B \subseteq T_A$. With singleton tails we do not transmit any errors. Thus, we only subtract the old costs for r_2 and add the estimated cost of sending where the new rule r holds, as well as the estimated regret for the new matrices,

$$\widehat{\Delta}_1(r) = -\log \binom{n}{n_B} + \log \binom{n_A}{n_B} + L_{pc}(n_A) + L_{pc}(n_B) + L_{pc}(n_A - n_B) - L_{pc}(n) \ .$$

For the tighter, second estimate $\widehat{\Delta}_2$ we instead need to retrieve the exact number of usages of the rule by intersecting the tid lists of merged rules. The change in model costs $L(M)$ by introducing r appearing in $L(M)$ is trivially computable and thus abbreviated by $\hat{L}(M)$. For formerly covered transactions that are not covered by the new rule, we need to send singleton rules with adapted costs, which is estimated through simple set operations on the tid lists. Additionally, we need to subtract the model costs for r_2, in case B is completely covered by r,

ensured by the indicator variable I. We hence have

$$\widehat{\Delta}_2(r) = -\log\binom{n}{n_B} + \log\binom{n_A}{|T_A \cap T_B|} + \log\binom{n}{|T_B \setminus T_A|} + \hat{L}(M) + L_{pc}(n_A) + L_{pc}(|T_A \cap T_B|) + L_{pc}(n_A - |T_A \cap T_B|) - I(T_B \subseteq T_A) \times L_{pc}(n)\ .$$

Grab first computes the first order estimate $\widehat{\Delta}_1$ per candidate, and only if this shows potential improvement, it computes the second order estimate $\widehat{\Delta}_2$. Out of those, it evaluates all candidates that have the potential to improve over the best refinement found so far. In the next paragraph we describe how to efficiently compute the overall score $L(D, M)$.

Efficiently computing $L(D, M)$ To get the codelength of a rule set with a new candidate, two steps are carried out, which we summarize in Alg. 2.2. First, the data is covered with the new rule to determine where the rule holds and what error matrices to send. Covering the data is straightforward, but to find the error matrices we have—unless we rely on a user-defined threshold—to optimize for the best point to split between additive and destructive noise. We observe that each rule encoding is independent of every other rule (except singletons), that is, changing the error matrices for one rule does not change the matrices for any other rule as we always encode all transactions where the antecedent is fulfilled.

With this in mind, it is clear that we can optimize the split point for each rule $X \to Y$ separately. Thus, we find a partitioning of T_X into $T^M_{Y|X}$ and $T^M_{\not Y|X}$ that minimizes the contribution of this rule to the overall costs:

$$\Delta_{T_X, T^M_{Y|X}, T^M_{\not Y|X}, \mathbb{1}(T^M_{Y|X}), \mathbb{1}(T^M_{\not Y|X})} = L_{pc}(|T_X|) + L_{pc}(|T^M_{Y|X}| \times |Y|) + L_{pc}(|T^M_{Y|X}| \times |Y|) + \log\binom{|T^M_{Y|X}| \times |Y|}{\mathbb{1}(T^M_{Y|X})} + \log\binom{|T^M_{\not Y|X}| \times |Y|}{\mathbb{1}(T^M_{\not Y|X})}\ .$$

We can also view the problem from a different angle, namely, for each transaction $t \in T_X$ we count how many items of Y are present, which yields a vector of counts B, $B[i] = |\{t \in T_X \mid |t \cap Y| = i\}|$. For fixed split point k, we get the

Algorithm 2.2: COVER

input : Database D, model $M = (R, \mathcal{T})$, refined rule set R'
output: Model $M' = (R', \mathcal{T}')$

$T_{Y|X}^{M'} \leftarrow$ (Equation (2.1)) // Initialize where new rule holds
for $I \in \mathcal{I}$ **do** // For each singleton
  $T_I^{M'} \leftarrow T_I$ // Reset usage to baseline model
  for $\{X \to Y \in R' \mid I \in Y\}$ **do** // For rule tail containing I
    $T_I^{M'} \leftarrow T_I^{M'} \setminus T_X$ // Remove these transactions from list
return
$(R', \{T_I^{M'} \mid I \in \mathcal{I}\} \cup \{T_{U|V}^{M} \in \mathcal{T} \mid U \to V \in R \cap R'\} \cup \{T_{Y|X}^{M'}\})$

additive and destructive matrix sizes $\mathbb{1}(\cdot)^k$ and transaction set sizes $|\cdot|^k$:

$$|T_{Y|X}^{M}|^k := \sum_{i=k}^{|B|} B[i] \qquad |T_{Y|X}^{M}|^k := \sum_{i=0}^{k-1} B[i]$$
$$\mathbb{1}(T_{Y|X}^{M})^k := \sum_{i=k}^{|B|} B[i] \times i \qquad \mathbb{1}(T_{Y|X}^{M})^k := \sum_{i=0}^{k-1} B[i] \times i \,.$$

To find the best split k^* we optimize along k using the two equation sets above, which is computable in linear time with respect to the size of the consequent,

$$k^* = \operatorname*{argmin}_{k=1\ldots|B|} \left(\Delta_{T_X, T_{Y|X}^{M}, T_{Y|X}^{M}, \mathbb{1}(T_{Y|X}^{M}), \mathbb{1}(T_{Y|X}^{M})} \right) . \tag{2.1}$$

This yields the best splitpoint k^* for how many items of the consequent are required for a rule to *hold* in terms of our MDL score and thus implicitly gives the error matrices.
Putting everything together, we have GRAB, given in pseudo-code as Alg. 2.3.

COMPLEXITY In the worst case we generate all pairs of combinations of rules, and hence at each step GRAB evaluates a number of candidates quadratic in the size of the rule table. Each evaluation of the $O(3^{2^m})$ candidates requires a database cover which costs time $O(n \times m)$, and singleton transaction set update, thus giving an overall time in $O(3^{2^m} \times m \times n)$. However, MDL ensures that the number of rules is small, and hence a more useful statement about runtime is thus given in the following theorems that are based on the size of the output or

Algorithm 2.3: GRAB

input : Dataset D
output: Heuristic approximation to M

$M \leftarrow \{\emptyset \rightarrow \{A\} \mid A \in \mathcal{I}\}$ // Initialize model with singletons
do
  $C \leftarrow$ generateCandidates(D, M)
  $M^* \leftarrow M$; $\Delta^* \leftarrow 0$
  while C contains a refinement R with $\widehat{\Delta}_2 < \Delta^*$ **do**
    $R' \leftarrow$ refinement $R \in C$ with best $\widehat{\Delta}_2$
    $M' \leftarrow$ cover(D, M, R') // Construct model M'
    $\Delta' \leftarrow L(D, M') - L(D, M)$ // Compute exact gain
    if $\Delta' < \Delta^*$ **then**
      $M^* \leftarrow M'$; $\Delta^* \leftarrow \Delta'$;
  if $M^* \neq M$ **then** // Update best model
    $M \leftarrow M^*$
while $L(D, M) < L(D, M^*)$
return M

in other words the number of mined rules.

Theorem 2.3 (GRAB candidate evaluations)**.** *Given that we mine k rules for a given dataset D, GRAB evaluates $O((m+k)^3)$ candidates.*

Proof. In each step of the algorithm we add at most one rule to the rule table. We start initially with all p singletons in the table. At step i we thus generate $(p+i)^2$ many new candidates. If we mine k rules overall, there are $\sum_{i=0}^{k}(p+i)^2$ candidates that need to be evaluated in the worst case. We first do an index shift to obtain $\sum_{j=p}^{p+k} j^2$. We now proof a closed form solution to this sum by rearranging the sum of cubes

$$\sum_{i=0}^{k} i^3 = \sum_{i=0}^{k} \left((i+1)^3\right) - (k+1)^3 \,.$$

Next we expand the right hand side

$$\sum_{i=0}^{k} i^3 = \sum_{i=0}^{k} \left(i^3\right) + 3\sum_{i=0}^{k} \left(i^2\right) + 3\sum_{i=0}^{k} \left(i\right) + \sum_{i=0}^{k} \left(1\right) - (k+1)^3$$

and rearrange

$$3\sum_{i=0}^{k} i^2 = -3\sum_{i=0}^{k}(i) - \sum_{i=0}^{k}(1) + (k+1)^3 \,.$$

Expanding the remaining cubic term and applying Gauss' formula to the one sum we get

$$3\sum_{i=0}^{k} i^2 = -3\frac{k(k+1)}{2} - (k+1) + k^3 + 3k^2 + 3k + 1 \,.$$

Simplifying, we obtain

$$\sum_{i=0}^{k} i^2 = k^3 + \frac{3}{2}k^2 + \frac{k}{2} \,.$$

We thus evaluate $\sum_{j=p}^{p+k} j^2 = (p+k)^3 + \frac{3}{2}(p+k)^2 + \frac{p+k}{2} - ((p-1)^3 + \frac{3}{2}(p-1)^2 + \frac{p-1}{2}) = O((p+k)^3)$ many candidates. □

This theorem gives us insight in how many times GRAB calls COVER. For the runtime analysis, we know that in step i our rule table has size at most $m+i$ and GRAB has to compute the cover of the newest rule in time $O(n \times m)$ and update the singleton costs in time $O((m+i) \times m \times n)$.

Theorem 2.4 (GRAB runtime)**.** *Given that we mine k rules for a given dataset D, the overall runtime of GRAB is* $O((m+k)^4 \times m \times n)$.

Proof. We know that $O((p+i) \times p \times n)$ is the dominating factor in the evaluation. In each round i, we evaluate $(p+i)^2$ candidates in the worst case, for each of which we have time $O((p+i) \times p \times n)$. We thus get a runtime in $O((p+i)^3 \times p \times n)$ at round i, hence, given that we mine k rules, an overall runtime of $\sum_{i=0}^{k} O((p+i)^3 \times p \times n)$.

We proof now the closed form solution for the sum $\sum_{i=0}^{k} i^3 = \left(\frac{k(k+1)}{2}\right)^2$ by induction.

Base case: trivial.

Induction hypothesis: Assume closed form holds for k=j.

Induction step $j \to j+1$*:*

$$\begin{aligned}\sum_{i=0}^{j+1} i^3 &= \sum_{i=0}^{j} (i^3) + (j+1)^3 \\ &= \left(\frac{j(j+1)}{2}\right)^2 + (j+1)^3 \\ &= \frac{1}{4}\left(j^2(j+1)^2 + 4(j+1)(j+1)^2\right) \\ &= \frac{1}{4}\left((j+1)^2(j^2+4j+4)\right) \\ &= \frac{1}{4}\left((j+1)^2(j+2)^2\right) \\ &= \left(\frac{(j+1)(j+2)}{2}\right)^2.\end{aligned}$$

Plugging in this closed form into $\sum_{i=0}^{k} O((p+i)^3 \times p \times n)$ similar as before, we thus get an overall runtime of $O((p+k)^4 \times p \times n)$. □

In practice, however, the runtime is much lower both due to our gain estimates and because we only allow to merge rules with the same head.

2.6 EXPERIMENTS

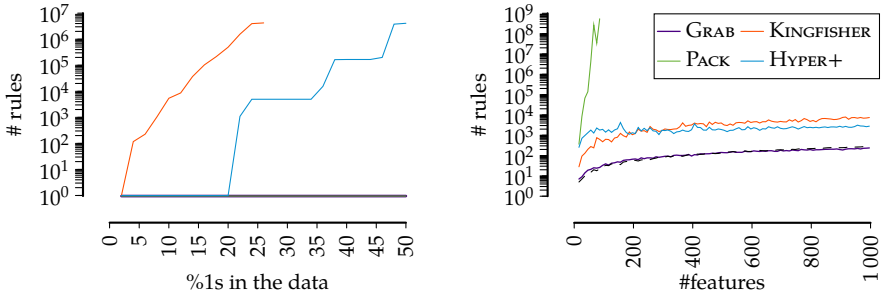
In this section we empirically evaluate GRAB quantitatively and qualitatively on both synthetic and real-world data. We implemented GRAB in C++. We make all code and data available for research purposes.[1] All experiments were executed single-threaded on Intel Xeon E5-2643 v3 machines with 256 GB memory running Linux. We report the wall-clock running times.

We compare to state of the art methods for mining statistically significant patterns and association rules.[2]

In particular, we compare to HYPER+ (Xiang et al., 2008), which mines noise-resistant patterns, KINGFISHER (Hämäläinen, 2012), which is a state of the art algorithm for the discovery of statistically significant rules under Fisher's exact test (Fisher, 1922), and PACK (Tatti and Vreeken, 2008), an MDL-based method that yields a binary tree per item $A \in \mathcal{I}$ of which we can interpret the paths to leafs as rules $X \to A$.

[1] http://eda.mmci.uni-saarland.de/prj/grab/

[2] We leave classical support-confidence methods out for the simple fact that even for trivial data APRIORI results in millions of rules (see Appendix A.1.3).



(a) Rules found in pure noise data of different sparsity. No rule should be found.

(b) Data with planted rules, ground truth is given by dashed, black line.

Figure 2.3: *Results on synthetic data.* Visualized are results on random data (left) and data with planted rules for different data dimensionalities (right).

Synthetic data First, we consider data with known ground truth. As a sanity check, we start our experiments on data without any structure. We draw datasets of 10000-by-100 of d% 1s, and report for each method the average results over 10 independent runs in Fig. 2.3a. We find that both Kingfisher and Hyper+ quickly discover up to millions of rules. This is easily explained, as the former relies on statistical significance only, and lacks a notion of support, whereas the latter does have a notion of support, but lacks a notion of significance. Pack and Grab, however, retrieve the ground truth in all cases.

Next, we consider synthetic data with planted rules. We generate datasets of $n = 20000$ transactions, and vary m from 10 to 1000 items. We generate rules that together cover all features. We sample the cardinality of the heads and tails from a Poisson with $\lambda = 1.5$. To avoid convoluting the ground truth via overlap, or by rules forming chains, we ensure that every item A is used in at most one rule. Per rule, we choose confidence c uniformly at random between 50 and 100%. We then randomly partition the n transactions into as many parts as we have rules, and per part, set the items of the corresponding rule head X to 1, and set Y to 1 for c% of transactions within the part. Finally, we add noise by flipping 1% of the items in the data – we use this low noise level to allow for a fair comparison to the competitors that do not explicitly model noise.

We provide the results in Fig. 2.3b. We observe that unlike in the previous experiment, here Pack strongly overestimates the number of rules – it runs out of memory for data of more than 92 features. Kingfisher and Hyper+ both discover over an order of magnitude more rules than the ground truth. Grab, on the other hand, is the only one that reliably retrieves the ground truth.

Real-World Data Second, we verify whether Grab also yields meaningful results on real data. To this end we consider 8 data sets over a variety of domains, for which we provide basic statistics in Table 2.1. In particular, we consider

- **Abstracts** (Tatti and Vreeken, 2008) A collection of $m = 3\,933$ stemmed words (features) for $n = 859$ abstracts of ICDM 2001-2007 (samples).
- **Accidents** (Geurts et al., 2003) A dataset of roughly $n = 340\,000$ records of traffic accidents in Belgium with binarized meta information about the accident's location and circumstances summing up to $m = 468$ features.
- **Adult** (UCI repository[3]) A collection of Census income data with $m = 97$ binary variables providing information about $n = 10\,830$ persons and if their income exceeds 50 000.
- **Covtype** (UCI repository) Data about forest cover types for $n = 581\,012$ small areas along with meta information about the area, such as soil type and slope, building a data set with $m = 105$ binary features.
- **DNA** (Myllykangas et al., 2006) Microarray measurements of DNA amplification comprising data about copy number amplification for $n = 4\,590$ cases and $m = 391$ sites (binarized).
- **Mammals** (Mitchell-Jones, 1999) A record of $n = 2\,183$ geo locations across europe along with longitude an latitude stating for $m = 121$ different mammals if it has been sighted at this location.
- **Mushrooms** (UCI repository) A collection of hypothetical samples of $n = 8\,124$ Mushrooms generated from The Audubon Society Field Guide to North American Mushrooms, with $m = 119$ binary attributes such as cap shape or spore color.
- **Plants** (UCI repository) For $m = 70$ states in the United States and Canada, for $n = 22\,632$ plants information has been gathered, whether the plant appears in the state or not.

We run each of the methods on each data set, and report the number of discovered non-singleton rules for all methods and the average number of items in head and tail for Grab in Table 2.1. We observe that Grab retrieves much more succinct sets of rules than its competitors, typically in the order of tens, rather than in the order of thousands to millions. The rules that Grab discovers are also more informative, as it is not constrained to singleton-tail rules. This is also reflected by the number of items in the consequent, where the average tail size is much larger than 1 for e.g. *Mammals* and *Plants*, where we find multiple rules with more than 10 items in the consequent.

To qualitatively evaluate the rules that Grab discovers, we investigate the results on *Abstracts* and *Mammals* in closer detail. For *Abstracts* we find patterns such as $\emptyset \to \{nearest, neighbor\}$, $\emptyset \to \{pattern, frequency\}$, $\emptyset \to \{naive, bayes\}$, and, notably, $\emptyset \to \{association, rule\}$. Further, we find meaningful rules, including $\{high\} \to \{dimension\}$, $\{knowledge\} \to \{discovery\}$, $\{ensembl\} \to$

[3] Dua and Graff (2017)

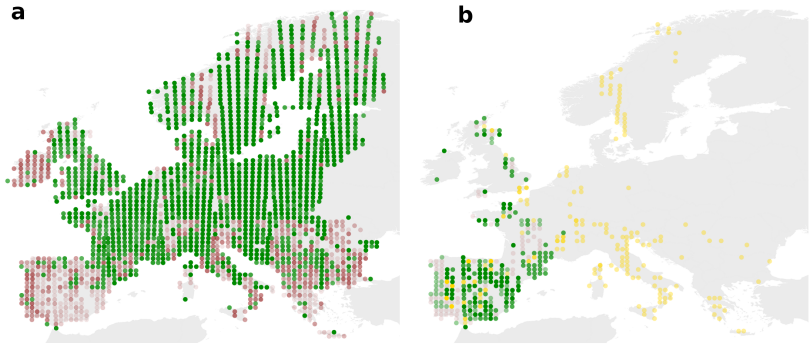


Figure 2.4: *Example rules for Mammals.* Shown are the inferred presence (green) and absence (red) of a pattern $\emptyset \rightarrow \{$*common squirrel, deer, ermine, marten, mice**$\}$ and b rule $\{$*Southwest European cat*$\} \rightarrow \{$*Mediterranean mice*, Iberian rabbit*$\}$. The intensity of the colour indicates how many items of the tail hold – the ideal result is hence dark green and light red. Yellow dots indicate presence of animals which are part of the tail of the rule where animals of the corresponding rule head were not sighted.

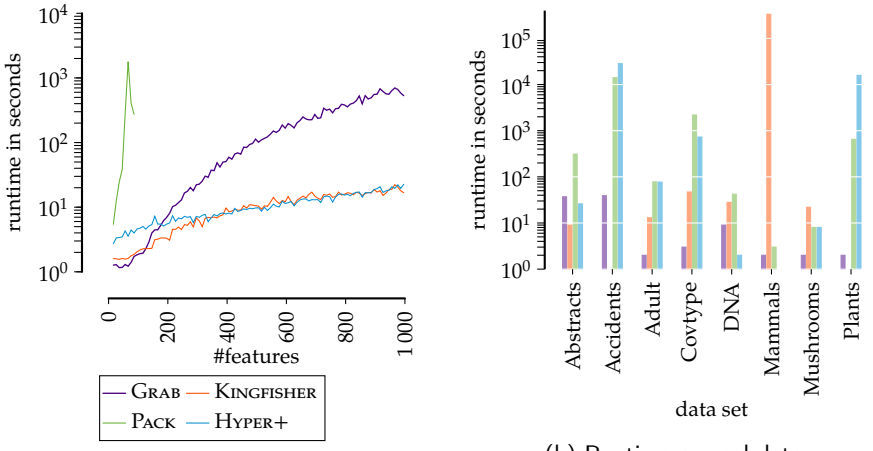
$\{$*bagging, boosting*$\}$, and $\{$*support*$\} \rightarrow \{$*vector, machin, SVM*$\}$. All patterns and rules correspond to well-known concepts in the data mining community.

On *Mammals*, Grab finds large patterns such as $\emptyset \rightarrow \{$*red deer, European mole, European fitch, wild boar, marten, mice**$\}$, and $\emptyset \rightarrow \{$*common squirrel, deer, ermine, marten, mice**$\}$, that correspond to animals that commonly occur across Europe, with multiple mouse species (items) indicated by mice*. In addition, it also discovers specific patterns, e.g. $\emptyset \rightarrow \{$*snow rabbit, elk, lynx, brown bear*$\}$, which are mammals that appear almost exclusively in northeastern Europe. We visualized the second rule in Figure 2.4a to show that the consequent should hold in most of the cases, but not necessarily need to be always present. Moreover, Grab is able to find meaningful rules in the presence of noise, e.g. $\{$*Southwest European cat*$\} \rightarrow \{$*Mediterranean mice*, Iberian rabbit*$\}$, where the rule should only hold in southwest europe. For the rule that Grab discovers this is indeed the case, although the data contains (likely spurious) sightings of Iberian rabbits or Mediterranean mice in Norway (see Fig. 2.4b) and some sightings of mice alone, along the Mediterranean sea.

Runtime and Scalability Last, but not least, we investigate the runtime of Grab. We first consider scalability with regard to number of features. For this, in Fig. 2.5a we give the runtimes for the synthetic datasets we used above. From the figure we see that while Grab is not as fast as Kingfisher and Hyper+, it scales favourably with regard to the number of features. Although it considers

			Grab			Hyper+	Kingfisher	Pack
Dataset	n	m	$\overline{\lvert X\rvert}$	$\overline{\lvert Y\rvert}$	$\lvert R\rvert$	$\lvert R\rvert$	$\lvert R\rvert$	$\lvert R\rvert$
Abstracts	859	3933	0.9	1.2	29	1508	42K	334
Accidents	339898	468	1	1.1	138	65M	*mem*	69M
Adult	10830	97	1	1.1	27	26K	9K	68M
Covtype	581012	105	1.3	1.1	41	13K	43K	286M
DNA	1316	392	1	1.7	147	49	140K	451
Mammals	2183	121	1.5	2	38	*mem*	$\geq$ 10M	2K
Mushroom	8124	119	1.6	1.5	65	13K	81K	7K
Plants	34781	69	1.2	3.2	20	6M	*mem*	910

Table 2.1: For Grab, the size of the rule set and average size of head $\overline{\lvert X\rvert}$ and tail $\overline{\lvert Y\rvert}$ are given. For the other methods, number of found rules are given, *mem* indicates an aborted run due to memory usage $>$ 256GB.



(a) Runtime on synthetic data.

(b) Runtime on real data.

Figure 2.5: *Scalability.* On the left side, runtimes are visualized on a logarithmic y-axis for synthetic data of varying number of features (x-axis). On the right, runtimes (logarithmic y-axis) are depicted for 8 real world data sets (x-axis). KINGFISHER did not finish on *Accidents* and *Plants*, HYPER+ did not finish on *Mammals*.

a much larger search space, GRAB only needs seconds to minutes. On real data GRAB is the fastest method for five of the data sets, and only requires seconds for the other datasets, whereas the other methods take up to hours for particular instances (compare Figure 2.5b).

2.7 DISCUSSION AND CONLUSION

We considered the problem of non-parametrically discovering sets of rules for a given dataset, and proposed to mine succinct and non-redundant sets of rules on solid information theoretic grounds. We showed that the problem does not lend itself for efficient optimization, and offered GRAB as a remedy, a highly efficient heuristic that greedily approximates the MDL optimal result. The experiments showed that GRAB has the among the state-of-the-art unique ability to mine small, non-redundant sets of highly informative noise-resistant rules and patterns non-parametrically. On synthetic data it recovered the ground truth, without picking up noise. Similarly, on real world data, it retrieved concise, easily interpretable, and highly meaningful rule sets, as opposed to the state of the art that discovered thousands, up to millions of rules even for very small data.

For instance, the results on the *Mammals* data provided evidence that Grab recovers known population structures, even in the presence of noise. The results on the ICDM *Abstracts* data were equally promising, with rule $\{support\} \rightarrow \{vector, machin, svm\}$ as a notable example. In contrast to machine learning, in data mining "support" is ambiguous. In the ICDM abstracts it means the support of a pattern, as well as support vector machines, and the rule expresses this. To verify this, we additionally ran Grab on abstracts from the Journal of Machine Learning Research (JMLR), where it indeed instead recovered the pattern $\emptyset \rightarrow \{support, vector, machin, svm\}$.

Our proposed algorithm is unique in that it can discover both patterns and rules, is noise-resistant and allows rules and patterns to hold approximately, and, can discover rules with non-singleton consequents. Due to careful implementation and accurate gain estimates, it scales well in the number of transactions, as well as in the number of features. In practice, Grab can consider up to several thousand features in reasonable time, thus offers insights into relevant datasets with its expressive statement of rules and patterns.

For applications in for instance bioinformatics we need algorithms that scale up to hundreds of thousands or even millions of features, and pattern languages that consider statements such as mutual exclusivity to capture the fine details of biological interactions. We will explore new pattern languages and scalable approaches in the subsequent chapters, that enable generating new insights for such large and complex biological datasets. Before that, we will make use of Grab to shed light onto the inner workings of neural networks in the next chapter, thereby extending the pattern language of Grab to be able to express disjunctive statements.

EXPLORING NEURAL NETWORKS THROUGH ROBUST RULES

3

3.1 INTRODUCTION

Neural networks achieve state of the art performance in many settings. However, how they perform their tasks, how they perceive the world, and especially, how the neurons within the network operate in concert, remains largely elusive. While there exists a plethora of methods for explaining neural networks, most of these focus either on the mapping between input and output (e.g. model distillation) or only characterize a given set of neurons, but can not identify which set to look at in the first place (e.g. prototyping). Here, we introduce a new approach to explain how the neurons in a neural network interact. In particular, we consider the activations of neurons in the network over a given dataset, and propose to characterize these in terms of rules $X \rightarrow Y$, where X and Y are sets of neurons in different layers of the network. A rule hence represents that neurons Y are typically active when neurons X are. For robustness we explicitly allow for noise, and to ensure that we discover a succinct yet descriptive set of rules that captures the regularities in the data, we formalize the problem in terms of the Minimum Description Length principle.

To discover good rule sets in practice, we propose the unsupervised ExplaiNN algorithm, which builds on the foundations of Grab, extending its

This chapter is based on Fischer, Oláh, and Vreeken (2021b).

pattern language to disjunctions to reflect the relevant information in a network, such as rules spanning multiple classes which are naturally disjunct. We confirm that ExplaiNN is able to retrieve those disjunctive rules in a brief study on synthetic data, and show that the rules we discover on real world networks give clear insight in how networks performs their tasks. As we will see, these identify what the network deems similar and different between classes, how information flows within the network, and which convolutional filters it expects to be active where. An ablation study further confirms that the rule sets are of high quality and capture class-relevant information succinctly. We find that our rules are easily interpretable, give insight in the differences between datasets, show the effects of fine-tuning, as well as super-charge prototyping as they tell which neurons to consider in unison. ExplaiNN thus enables us to peek into the black box of neural networks to get a deeper understanding how networks perform their tasks.

3.2 Related Work

Explaining neural networks is of widespread interest, and especially important with the emergence of applications in healthcare and autonomous driving. We here introduce the work most relevant to ours, while we refer to surveys for more information (Adadi and Berrada, 2018; Ras et al., 2018; Xie et al., 2020; Gilpin et al., 2018). There exist several proposals for investigating how networks arrive at a decision for a given sample, with saliency mapping techniques for CNNs among the most prominent (Bach et al., 2015; Zhou et al., 2016; Sundararajan et al., 2017; Shrikumar et al., 2017). Although these provide insight on what parts of the image are used, they are inherently limited to single samples, and do not reveal structure across multiple samples, let alone classes. For explaining the inner working of a CNN, research mostly focuses on feature visualization techniques (Olah et al., 2017) that produce visual representations of the information captured by neurons (Mordvintsev et al., 2015; Gatys et al., 2015). Although these visualizations provide insight on how CNNs perceive the world (Øygard, 2016; Olah et al., 2018) it has been shown that concepts are often encoded over multiple neurons, and that inspecting individual neurons does not provide meaningful information about their role (Szegedy et al., 2014; Bau et al., 2017). How to find such groups of neurons, and how the information is routed between layers in the networks remains unsolved.

An orthogonal approach is model distillation, where we train easy-to-interpret white box models to mimic the decisions of a neural network (Ribeiro et al., 2016; Frosst and Hinton, 2017; Bastani et al., 2017; Tan et al., 2018). Rules of the form (*if–then*) are easily interpretable, and hence a popular technique for model distillation (Taha and Ghosh, 1999; Lakkaraju et al., 2017). Existing

techniques (Robnik-Šikonja and Kononenko, 2008; Özbakır et al., 2010; Barakat and Diederich, 2005) aim for rules that directly map input to output, rather than providing insight into how information flows through the network. A notable exception is the work of Tran and d'Avila Garcez (2018) which propose to mine all sufficiently strong association rules – suffering from the well-known pattern explosion – and restrict themselves to Deep Belief Networks only. In contrast, Chu et al. (2018) propose to explain NNs by deriving decision boundaries of a network using polytope theory. While this approach permits strong guarantees, it is limited to very small ($<$ 20 hidden neurons) piecewise linear NNs. In sum, existing methods either do not give insight in what happens inside a neural network, and/or, are not applicable to the type or size of state-of-the-art convolutional neural networks. While Zhang et al. (2018) show how we can gain insight into convolutional layers of CNNs by building an explanatory graph over sets of neurons, they – in contrast to what we propose – do not elucidate the relation between such filters and subsequent dense layers, nor to the network output.

Instead, we propose to mine sets of rules to discover groups of neurons that act together across different layers in feed forward networks, and so reveal how information is composed and routed through the network to arrive at the output. To discover rules over neuron activations, we need an unsupervised approach. As discussed in the previous chapter, rule mining methods based on frequency (Agrawal and Srikant, 1994; Bayardo, 1998; Moerchen et al., 2011) or statistical testing (Hämäläinen, 2012; Webb, 2010) typically return millions of rules even for small datasets, thus thwarting the goal of interpretability. We therefore take a pattern set mining approach similar to Grab, where we are after that set of rules that maximizes a global criterion, rather than treating each rule independently. Although providing succinct and accurate sets of rules, Grab is limited to conjunctive expressions. This is too restrictive for our setting, as we are also after rules that explain shared patterns between classes, and are robust to the inherently noisy activation data, which both require a more expressive pattern language of conjunctions, approximate conjunctions, and disjunctions for both heads and tails of rules. We hence present ExplaiNN, a non-parametric and unsupervised method that learns sets of such rules efficiently.

3.3 THEORY

We first informally discuss how to discover association rules between neurons. We then formally introduce the concept of robust rules, and how to find them for arbitrary binary datasets, last, we show how to combine these ideas to reveal how neurons are orchestrated within feedforward networks.

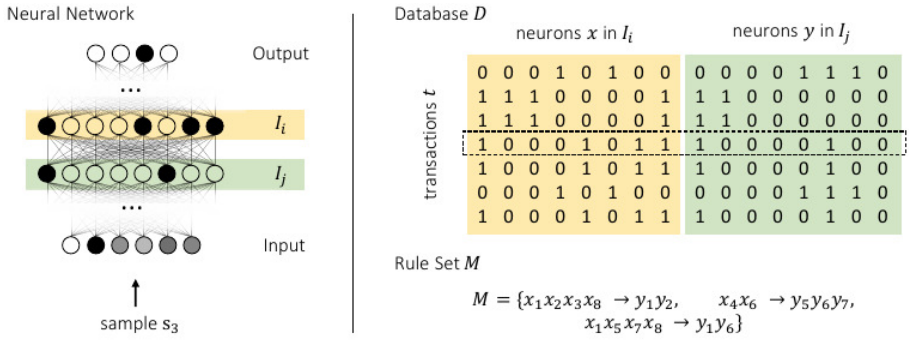


Figure 3.1: *Overview.* For a given network (left), binarized activations are gathered for the layers I_i, I_j for each sample, and summarized in the binary database D (right). Rules are discovered over D, where a good rule set M is given with at the bottom right, with rules $X \rightarrow Y,\ X \in I_i,\ Y \in I_j$.

3.3.1 Patterns of neuron co-activation

Similar to neurons in the brain, when they are active, artificial neurons send information along their outgoing edges. To understand flow of information through the network, it is hence essential to understand the activation patterns of neurons between layers. Our key idea is to use recent advances in pattern mining to discover a succinct and non-redundant set of rules that together describe the activation patterns found for a given dataset. For two layers I_i, I_j, these rules $X \rightarrow Y, X \subset I_i, Y \subset I_j$ express that the set of neurons Y are usually co-activated when neurons X are co-activated. That is, such a rule provides us *local* information about co-activations within, as well as the dependence of neurons between layers. Starting from the output layer, we discover rules between consecutive layers I_j, I_{j-1}. Discovering overlapping rules between layers $X \rightarrow Y$ and $Y \rightarrow Z, X \subset I_j, Y \subset I_{j-1}, Z \subset I_{j-2}$, allows us to trace how information flows through the entire network.

Before we can mine rules between two sets of neurons I_i and I_j of a network, for example two layers, we have to obtain its binarized activations for a given data set $\mathcal{D} = \{(s_k, o_k)\}$ consisting of samples s_k and output labels o_k. In particular, for each sample s_k and neuron set I_i, we take the tensor of activations ϕ_i and binarize it to ϕ_i^b. For networks with *ReLU* activations, which binarize naturally at threshold 0, we might lose some information about activation strength that is eventually used by subsequent layers. This binarization however allows us to derive crisp symbolic, and directly interpretable statements on how neurons interact. Furthermore, binarization reflects the natural on/off state of biological neurons, also captured by smooth step functions such as sigmoid or tanh used in artifical neural networks. We gather the binarized activations into a dataset

D where each row t_k corresponds to the concatenation of binarized activations ϕ_i^b and ϕ_j^b of neuron sets I_i and I_j for input sample s_k, i.e., $t_k \in D$ is a binary vector of length $|I_i| + |I_j|$. See Fig. 3.1 for a toy example.

Next, given binary activation data D, our goal is to find that set of rules that together succinctly describe the observed activations. The Minimum Description Length (MDL) principle lends itself as an objective to find such sets. Similar to the previous chapter, our model class $\mathcal{M}$ is the superset of all possible rules over D, and by MDL we identify the best model M^* as the one that compresses the data best. In contrast to Grab, which models the traditional rule language restricted to conjunctions over items which is not sufficient for our application, we consider an extended language that allows for partial disjunctions of items (neurons, labels) and introduce a codelength function $L(D, M)$ to instantiate MDL for our model class of rule sets. This is necessary to properly model rules over networks, where neuron activations tend to be noisy, and labels are inherently mutually exclusive.

3.3.2 MDL FOR ROBUST RULES

Our goal is to find a set of rules M that, in terms of description length $L(D, M)$, best describes a binary database $D = \{t \mid t \subset \mathcal{I}\}$ that consists of transactions t that are subsets of items $\mathcal{I}$. Each rule is of the form $X \to Y,\ X, Y \subset \mathcal{I}$, and indicates that Y is strongly associated with X, i.e. occurs mostly in transactions where X is present. To recapitulate, we say a rule $X \to Y$ *applies* to a transaction t iff $X \subset t$ and say a rule *holds* for t if additionally $Y \subset t$. We indicate these transactions sets as $T_X = \{i \mid t_i \in D,\ X \subset t_i\}$, respectively $T_{Y|X} = \{i \mid t_i \in T_X,\ Y \subset t_i\}$. Based on these definitions of rule transaction sets, we can now formally introduce our codelength function $L(D, M)$.

Baseline model Our base model $M_{ind} = \{\emptyset \to I \mid \forall I \in \mathcal{I}\}$ consists of singleton rules only, i.e. it models that all items $\mathcal{I}$ are generated independently. To send the n transactions of D using M_{ind}, we simply send for each item I in which out of all transactions in the database it appears. We can do so optimally using a log binomial code as introduced in the previous chapter, which is given by $\log \binom{|T_\emptyset|}{|T_{I|\emptyset}|} = \log \binom{n}{|T_I|}$. To unambiguously decode, the recipient needs to know each $|T_I|$, which we can optimally encode via the parametric complexities of the binomials (see Eq. 2.4.2).We thus have $L(D, M_{ind}) = \sum_{I \in \mathcal{I}} \left(\log \binom{n}{|T_I|} + L_{pc}(n) \right)$. M_{ind} serves as our baseline model, and its singleton rules are a required part of any more complex model as they ensure we can always send any data over $\mathcal{I}$.

NON-TRIVIAL MODELS A non-trivial model M contains rules of the form $X \to Y$, $X, Y \subset \mathcal{I}$ that are not part of M_{ind}. Similar to GRAB, the idea is that we first transmit the data for where these non-trivial rules hold, and then send the remaining data using M_{ind}. To determine where such a rule applies, the receiver needs to know where X holds, and hence the data over X needs to be transmitted first. To ensure that we can decode the data, as before, we only consider models M for which the rules form an acyclic graph. We thus get a codelength

$$\begin{aligned} L(D \mid M \cup M_{ind}) = &\left(\sum_{X \to Y \in M} \log \binom{|T_X|}{|T_{Y|X}|} \right) \\ &+ \left(\sum_{\emptyset \to I \in M_{ind}} \log \binom{n}{|T'_I|} \right) , \end{aligned}$$

where $T'_I = \{t \in D \mid (I \in t) \wedge (\forall X \to Y \in M.\ I \in Y \Rightarrow t \notin T_{Y|X})\}$ is the modified transaction set containing transactions with item I not covered by any non-trivial rule.

In addition to the parametric complexities of the binomial codes, the model cost of a non-trivial model also includes the cost of transmitting the non-trivial rules. To transmit a rule $X \to Y$, we follow the approach from GRAB and first send the cardinalities of X resp. Y using the universal code for integers $L_{\mathbb{N}}$. Knowing the cardinalities, we can then send the items of X resp. Y one by one using an optimal prefix code given by $L(X) = -\sum_{x \in X} \log \frac{|T_x|}{\sum_{I \in \mathcal{I}} |T_I|}$. For a particular rule $X \to Y \in M$, the model costs for a rule, respectively the full model thus amount to

$$\begin{aligned} L(X \to Y) = &L_{\mathbb{N}}(|X|) + L_{\mathbb{N}}(|Y|) \\ &+ L(X) + L(Y) + L_{pc}(|T_X|) , \\ L(M \cup M_{ind}) = &|\mathcal{I}| \times L_{pc}(n) + L_{\mathbb{N}}(|M|) \\ &+ \sum_{X \to Y \in M} L(X \to Y) . \end{aligned}$$

We provide an example calculation in App. A.1.1. With these definitions, we have an MDL score that identifies the best rule set M^* for data D as

$$M^* = \operatorname*{argmin}_{M \in \mathcal{M}} \left(L(M \cup M_{ind}) + L(D \mid M \cup M_{ind}) \right) ,$$

where $\mathcal{M}$ contains all possible rule sets over the items in D.

ROBUST RULES In real world applications, we need a score that is robust against noise. The key problem with noisy data is that a single missing item in a transaction can cause a whole rule not to hold or apply. To discover rules that generalize well, we need to explicitly account for noise. The idea is to let rules apply, and hold, also when some items of head respectively tail are missing. Specifying how many items l, and k, out of all items in the rule head, respectively tail, need to be part of a transaction, we relax the original rule definition to account for missing items, or in other words, noise.

Furthermore, as output neurons – the classes – are only active mutually exclusively, rules need to be able to model disjunctions. Setting $l = 1$ and $k = 1$ means that only one of the items of head respectively tail need to be present, thus coincidentally corresponding to a disjunction of items in the head and tail of the rule $X \to Y$. This disjunctive form allows to model the activations of output neurons, corresponding to class predictions, correctly, whereas $l = |X|$ and $k = |Y|$ correspond to the original stringent rule definition of conjunctions. Varying between the two extremes accounts for varying levels of noise. The optimal l and k are those that minimize the MDL score.

To ensure a lossless encoding, we need to make sure that the receiver can reconstruct the original data. Thus, for the previously introduced relaxed definition of when rules hold and apply, we send for each rule the corresponding number of items l that need to be present for it to apply using $L_{\mathbb{N}}(l)$ bits. Knowing each l, the receiver can reconstruct where each rule applies. Sending where a rule holds now leaves the receiver with an approximation of the data. To be able to reconstruct the actual data, we leverage the error matrices introduced in the previous chapter that when XORed with the approximation yield the original data. These two matrices $\mathcal{X}^{+}_{X \to Y}$, and $\mathcal{X}^{-}_{X \to Y}$ correct for the errors made in the part where the rule applies and holds, respectively applies but does not hold. As discussed, these error matrices are part of the model M and have to be transmitted with an adapted $L(D, M)$.

COMPLEXITY OF THE SEARCH To discover rules over the activations of layers I_i, I_j, we have to explore all rules formed by subsets of neurons in I_i for the head, combined with any subset of neurons of I_j for the tail. There exist $2^{|I_i|} \times 2^{|I_j|}$ such rules, and hence $2^{2^{|I_i|+|I_j|}}$ distinct models would need to be explored. In the previous chapter, we showed that the rule set search space does not lend itself to efficient search as it is neither monotone nor submodular, the counterexamples also holding for this model definition. In fact, for robust rules, we additionally have to consider where rules should apply respectively hold – optimizing k and l – which results in approximately $2^{|I_i| \times |I_j| \times 2^{|I_i|+|I_j|}}$ models. This can be seen

from combinatorics, as for two layers I_i, I_j, we enumerate all possible rules by

$$\begin{aligned}
&\underbrace{\Big(\sum_{k=0}^{|I_i|} k \times \binom{|I_i|}{k}\Big)}_{\text{Possibilities for head}} \times \underbrace{\Big(\sum_{l=0}^{|I_j|} l \times \binom{|I_j|}{l}\Big)}_{\text{Possibilities for tail}} \\
&\leq |I_i|\Big(\sum_{k=0}^{|I_i|} \binom{|I_i|}{k}\Big) \times |I_j|\Big(\sum_{l=0}^{|I_j|} \binom{|I_j|}{l}\Big) \\
&= |I_i|\, 2^{|I_i|} \times |I_j|\, 2^{|I_j|} = |I_j|\, |I_i|\, 2^{|I_i|+|I_j|},
\end{aligned}$$

where the first sum enumerates all heads of size k, the binomial coefficient describes the ways of drawing heads of such size, and the term k is the number of models given by the robust head encoding. Similarly, the second sum enumerates all tails of size l, the binomial coefficient describes the drawing of such tails, and the term l is the number of ways to place the error correcting matrices for the robust tail encoding. As in theory we can have any subset of these rules as a model, we thus get approximately $2^{(|I_j|\times|I_i|\times 2^{|I_i|+|I_j|})}$ many different models. Exhaustive search is therewith infeasible, which is why we present ExplaiNN, a heuristic algorithm to efficiently discover good sets of rules.

3.4 EXPLAINN

ExplaiNN is based on the idea of iteratively refining the current model by merging and refining already selected rules, similar to the previously proposed Grab. In contrast to Grab, to permit scaling up to the size of a typical neural network under our more expressive pattern language, we introduce a two-step procedure, which first iteratively introduces new single-consequent rules to the model, and then merges rules in the model, if it decreases the MDL score. In particular, as search steps we consider first introducing new rules to M, by taking a good set of items $X \subset I_i$ for the head and a single item $A \in I_j$ for the tail and refine the model to $M' = M \oplus \{X \to A\}$, seeing if it decreases the overall MDL costs according to Eq. 3.3.2. Second, we merge two existing rules $r_1 = X \to Y_1 \in M$ and $r_2 = X \to Y_2 \in M$, to form a new rule $r' = X \to Y_1 \cup Y_2$, and refine the model to $M' = M \oplus \{r'\}$. For a rule r', the refinement operator $\oplus$ is adding the rule $r' = X \to Y$ to M, and removes the merged rules that led to r', if any. Moreover, it updates the singleton transaction lists T_A for all items $A \in Y$, removing all transactions where r' holds.

We next discuss how to efficiently search for candidate rules with heads that can express anything from conjunctions to disjunctions. Immediately after,

Algorithm 3.1: GenCandNew

input : Dataset D over sets of neurons I_i, I_j, Model M, tail item A, threshold θ
output: Best refinement M'

$H_A \leftarrow \emptyset$ // head items, in decreasing order of confidence
for $x \in I_i$ **do** // For each neuron in I_i
  $\sigma_{x,A} \leftarrow \frac{|T_x \cap T_A|}{|T_x|}$ // Compute conditional frequency
  if $\sigma_{x,A} > \theta$ **then**
    insert $(x, \sigma_{x,A})$ **into** H_A // Add neuron x to list
$M' \leftarrow \emptyset$
$\Delta_{min} \leftarrow 0$ // gain estimate in bits
for $t = 1 ... |H_A|$ **do**
  $M' = M \oplus \{H_A[:t] \rightarrow A\}$ // Refinement: use first t neurons
  $\Delta_t \leftarrow L(D, M) - L(D, M')$
  if $\Delta_t < \Delta_{min}$ **then**
    $\Delta_{min} \leftarrow \Delta_t$
    $M' \leftarrow M \oplus \{H_A[:t] \rightarrow A\}$ // Update best rule set
return M'

we present the full algorithm ExplaiNN for mining high quality rule sets for two arbitrary sets of neurons (e.g. layers) of a neural network.

3.4.1 SEARCHING FOR CANDIDATES

A key component of ExplaiNN is the candidate generation process, which implements the two possible steps of generating new and merging existing rules. Given two neuron sets I_i, I_j, to efficiently discover rules that are both robust to noise, and may include disjunctively active neurons in the head, we can not enumerate all possible rule heads for each individual tail neuron, as this would result in $|I_j| \times 2^{|I_i|}$ many rules. Instead, we keep a list H_y for each item $y \in I_j$, storing all head neurons $x \in I_i$ for which y is frequently active when x is active, that is $\sigma_{x,y} = \frac{|T_x \cap T_y|}{|T_x|} > \theta$, where θ is a confidence threshold. We consider a rule $X \rightarrow Y$ to be good, if when neurons X are active, the neurons Y are also likely to be active, which is directly represented by the confidence θ. The lists are sorted decreasing on σ. We search in each H_y for the rule with highest gain over all unions of first $t = 1 \dots |H_y|$ neurons in the list. We add

that rule $X \to y$ with highest gain to the candidate list. To compute the gain, we consider all possible values $k = 1 \dots |X|$ to determine for which transactions $T_X^k = \{t \in D \mid |X \cap t| \geq k\}$ the rule should robustly apply, where $k = 1$ corresponds to disjunction and $k = |X|$ to conjunction of neurons. We provide this idea in pseudocode as Alg. 3.1.

Algorithm 3.2: GenCandMerges

input : Dataset D, Model M, overlap threshold μ
output: Candidates C sorted by gain Δ

$C \leftarrow \emptyset$ // Candidate rule merges
for $r_1 = X_1 \to Y_1, r_2 = X_2 \to Y_2 \in M$ **do** // For each rule pair
  if $|X_1 \ominus X_2| \leq \mu$ **then**
    /* Gain of adding conjunction of heads */
    $\Delta_\cap \leftarrow L(D, M \oplus \{X_1 \cap X_2 \to Y_1 \cup Y_2\}) - L(D, M)$
    if $\Delta_\cap < 0$ **then**
      /* Add to candidates */
      insert $(X_1 \cap X_2 \to Y_1 \cup Y_2,\ \Delta_\cap)$ **into** C
    /* Gain of adding disjunction of heads */
    $\Delta_\cup \leftarrow L(D, M \oplus \{X_1 \cup X_2 \to Y_1 \cup Y_2\}) - L(D, M)$
    if $\Delta_\cup < 0$ **then**
      /* Add to candidates */
      insert $(X_1 \cup X_2 \to Y_1 \cup Y_2,\ \Delta_\cup)$ **into** C
return C

For an individual neuron y, such a rule would be optimal, but, our goal is to discover groups of neurons that act in concert. To this end we hence iteratively merge rules with *similar* heads – similar, rather than same, as this gives robustness both against noise in the data, as well as earlier merging decisions of the algorithm. We give pseudocode for this procedure as Alg. 3.2. For two rules $X_1 \to Y_1,\ X_2 \to Y_2$ with symmetric difference $X_1 \ominus X_2 = (X_1 \setminus X_2) \cup (X_2 \setminus X_1)$, we consider possible candidate rules $X_1 \cup X_2 \to Y_1 \cup Y_2$ and $X_1 \cap X_2 \to Y_1 \cup Y_2$, iff $|X_1 \ominus X_2| \leq \mu$ for some threshold $\mu \in \mathbb{N}$. For example, $\mu = 1$ corresponds to the case that one head has one label more than the other, all other labels are the same.

Both parameters θ and μ are simple, yet effective runtime optimizations. The best results with respect to MDL will always be obtained with the largest search space, i.e. with θ and μ set to 0, respectively $|X_1| + |X_2|$. Besides impacting run-time, many of those rules may be uninteresting from a user-perspective, μ

and θ allow to directly instruct ExplaiNN to ignore such rules.

3.4.2 EXPLAINN

Assembling the above, we have ExplaiNN, which, given two sets of neurons I_i, I_j and a database of activations of these neurons, yields a heuristic approximation to the MDL optimal model M^*. By first introducing all relevant single neuron rules, it then proceeds by iteratively merging existing rules using the approach described above, until it can achieve no more gain. For efficiency, we separate the generation of the new rules from the merging of existing rules. In practice, this does not harm performance, as we allow merging of similar heads and can thus revert too greedy decisions introduced earlier. Furthermore, by observing that independent rules $X_1 \to Y_1$, $X_2 \to Y_2$, with $Y_1 \cup Y_2 = \emptyset$ do not influence each others impact on codelength, we can add all independent rules with the highest respective gain at once. We provide pseudocode as Alg. 3.3.

3.4.3 COMPLEXITY OF EXPLAINN

The main algorithm of ExplaiNN is separated into the consecutive steps of generating new rules, and merging of existing rules. For the generation of new rules, we consider all possible single items A from neuron set I_j as tails and subsets of items from neuron set I_i as heads. We, here, only consider subsets formed by the first $t = 1...|I_i|$ items from the list of neurons $x \in I_i$ sorted by conditional frequency (see Alg. 3.1). Due to the sorting we, hence, require time in $O(n \times |I_j| \times |I_i| \log |I_i|)$ for generation of new rules, where the factor n comes from intersecting transaction lists T to compute the gain. We can have at most $|I_j|$ generated rules before considering rule merges – i.e. each single neuron appears in a rule – and in every merging step we combine two rules, effectively reducing the rule set size by 1. In each such step, we consider any pair-wise combination of rules out of which there are $|I_j|^2$ many. For each rule combination, the length of the new rule tail is of size at most $|I_j|$, which is the number of levels we need to check where a rule applies for our noise encoding. We thus need time in $O(n \times |I_j|^4)$, where the factor n is again coming from the transaction list intersections in the gain estimation. In summary, ExplaiNN thus needs time in $O(n \times (|I_j| \times |I_i| \log |I_i| + |I_j|^4))$. As MDL ensures we consider models that tend to be succinct and hence capture only relevant structure in the data, ExplaiNN is in practice much faster and easily scales to several thousands of neurons.

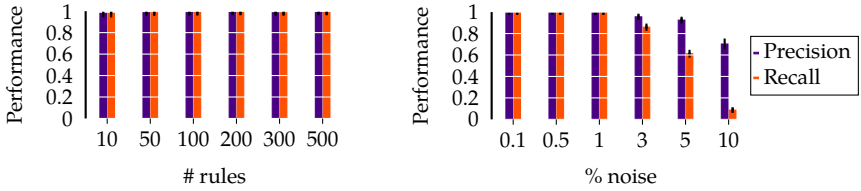
Algorithm 3.3: ExplaiNN

input : Dataset D over sets of neurons I_i, I_j, frequency threshold θ, overlap threshold μ
output: Best model M^*

$M \leftarrow \{\emptyset \rightarrow A \mid A \in I_j\}$ // Initialize model with baseline rules
for $A \in I_j$ **do**
  $R' \leftarrow \text{GenCandNew}(D, M, A, \theta)$ // Alg. 3.1
  $M' \leftarrow M \oplus R'$
  if $L(D, M') < L(D, M)$ **then**
    $M \leftarrow M'$
do
  $\hat{M} \leftarrow M$
  $C \leftarrow \text{GenCandMerges}(D, M, \mu)$ // Alg. 3.2
  $\mathcal{Y} \leftarrow \emptyset$ // Keep track of independence of merged rules
  for $X \rightarrow Y \in C. Y \notin \mathcal{Y}$ **do**
    $M' \leftarrow M \oplus \{X \rightarrow Y\}$ // Refine model, test gain
    if $L(D, M') < L(D, M)$ **then**
      $\hat{M} \leftarrow M'$
      $\mathcal{Y} \leftarrow Y$
while $M \neq \hat{M}$
return C

3.5 EXPERIMENTS

In this section we empirically evaluate ExplaiNN on synthetic data with known ground truth and real world data to explore how CNNs perceive the world. Other approaches to discover patterns based on e.g. frequency measures or statistical testing have already been shown to yield millions or billions of rules or patterns, most spurious and redundant, and many more than anyone would be willing to investigate as we have seen in the previous chapter. We hence focus on evaluating our method for the task of finding activation patterns. Here, we look at CNNs as they count towards the most widespread use of feedforward networks and naturally lend themselves for visualization, which helps us to interpret the discovered rules. We compare to traditional prototyping and activation map approaches on *MNIST* (LeCun and Cortes, 2010), and examine which information is used how to arrive at classification for *ImageNet* (Rus-



(a) Mutual exclusive head items. (b) Varying levels of noise.

Figure 3.2: *Evaluation of rule quality.* Performance of ExplaiNN as precision and recall on data with varying number of planted rules with mutual exclusive head items (left) and co-occurring head items with varying noise (right). 10% noise corresponds to more noise than signal in the data. We provide the average and standard error across 10 repetitions.

sakovsky et al., 2015). Finally, we investigate the effect of fine-tuning in transfer learning on the Oxford *Flower* data (Nilsback and Zisserman, 2008). The implementation of ExplaiNN is publicly available.[1] For the below experiments, running on commodity hardware, ExplaiNN took minutes for *MNIST* and *Flower*, and up to 6 hours for *ImageNet*– yielding from a few hundred up to 3000 rules, for the smaller, respectively larger networks, and earlier, respectively later layers.

3.5.1 Recovering Ground Truth

We discussed and evaluated conjunctive rules in the previous chapter. To evaluate how well ExplaiNN can recover the ground truth from data in settings of high noise and mutually exclusive features, we first generate synthetic binary data sets of 10000 samples and introduce $\{10, 50, 100, 200, 300, 500\}$ rules with up to 5 items in head and tail, respectively. For each rule, the frequency is drawn from a uniform distribution $U(.02, .08)$, the confidence is drawn from $U(.5, 1)$. We introduce noise by flipping 0.1% of the entries chosen uniformly at random, and add 5 noise features with frequency equal to those of rules. In the first set of experiments, we set head items mutual exclusively, in line of finding rules over the NN output labels. ExplaiNN achieves high recall and precision (see Figure 3.2a) in terms of retrieving exact ground truth rules, and does not retrieve any redundant rules. Next, we investigate the impact of noise on the performance, generating data of 10000 samples and 100 rules similar to above, with head items now set co-occuring, varying the level of noise in $\{0.1\%, 0.5\%, 1\%, 3\%, 5\%, 10\%\}$ bitflips in the matrix, where 10% noise means

[1] http://eda.mmci.uni-saarland.de/prj/explainn/

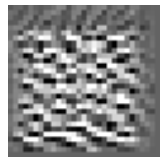
more noise than actual signal. ExplaiNN is robust to noise, even when facing almost the same amount of noise and signal (see Fig. 3.2b).

3.5.2 How neural networks perceive the world

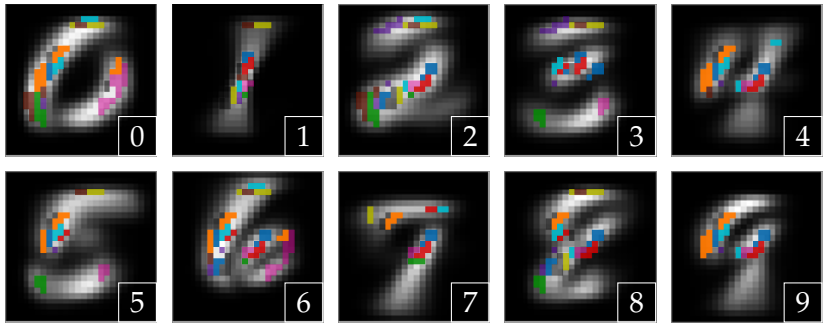
How information is filtered As a first real world example, we consider the *MNIST* data of handwritten digits. We train a simple CNN of 2 convolutional and one fully connected layer using Keras, achieving 99% classification accuracy on test data (see Supp. A.2.2 for details). We are interested in what the individual filters learn about the digits, and how ExplaiNN reveals shared features across several classes. We compare the found patterns against average activation maps and single neuron prototypes. We see that whereas the average activation maps per class do not reveal the purpose of a filter, the rules learned by ExplaiNN clearly identify which pixels *together* trigger a filter. For example, in filter 2 in layer 1 the prototype does not reveal any insight from its maze-like visualization (see Fig. 3.3a), and average activation maps just show the number given by a (single) class, whereas the discovered rules identify shared structure, such as curvatures across digits (see Fig. 3.3c). Finally, we provide the discovered rules for filter 12 in convolutional layer 1 in Fig. 3.3d. We observe that this filter acts as a negative, essentially an imprint of the digit. These examples provide information on how CNNs exploits different *local* structures to classify images based on the immediate information that rules provide on images and filters. Next, we will investigate how networks cope with more complex data, leveraging rules to super-charge prototyping.

How information flows To understand the inner life of neural networks in a more complex setting, we examine the activations for the *ImageNet* data set of pretrained VGG-S and GoogLeNet architectures (Chatfield et al., 2014; Szegedy et al., 2015). We focus on analyzing the VGG-S results for which an optimized and highly interpretable prototyping method to visualize multiple neurons exists (Øygard, 2016), and provide results for GoogLeNet in App. A.2.2. Mining for rules from the output to the last layer, ExplaiNN yields rules with individual heads spanning multiple labels, and tails spanning multiple neurons, which together encode the information shared between labels. In Fig. 3.4, 3.5, we provide a general and diverse overview of the results based on prototyping. We focus on rules with multiple neurons in the tail, as such class and multiclass prototypes can hardly be found by hand. Overall, we observe that the larger the number of neurons in the tail, the sharper and more interesting the resulting protoype. Furthermore, we found that for many prototypes spanning multiple classes, we discover multiple rules the same classes (e.g. `Black Grouse`) and the protoypes indicate that only a fraction of information, such as patterns, a

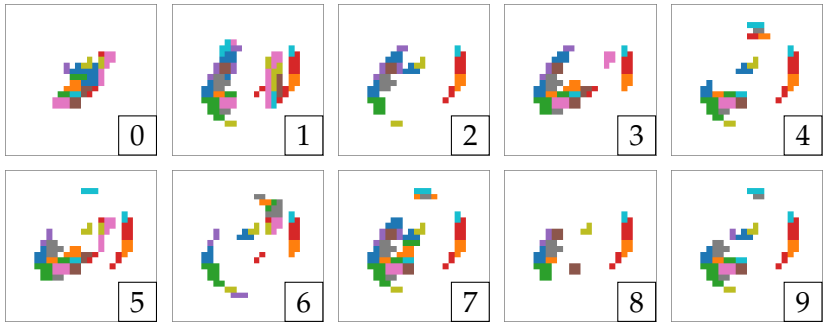
(a) Prototype for filter 2 in CONV1.



(b) Prototype for filter 36 in CONV2.



(c) Average activation of neurons for digit classes for filter 2 in the first convolutional layer. Overlayed are ExplaiNN rules, where pixel groups of the same colour (e.g. purple pixels top left for classes 2, 3) belong to a single rule spanning multiple classes.



(d) Visualizations for filter 12 in the first convolutional layer. This filter seems to capture the 'negatives' or imprints of the handwritten digits.

Figure 3.3: *MNIST*. Visualized are prototypes (a,b), average activations (c), and rules (c,d) from different convolutional layers from a network trained on MNIST.

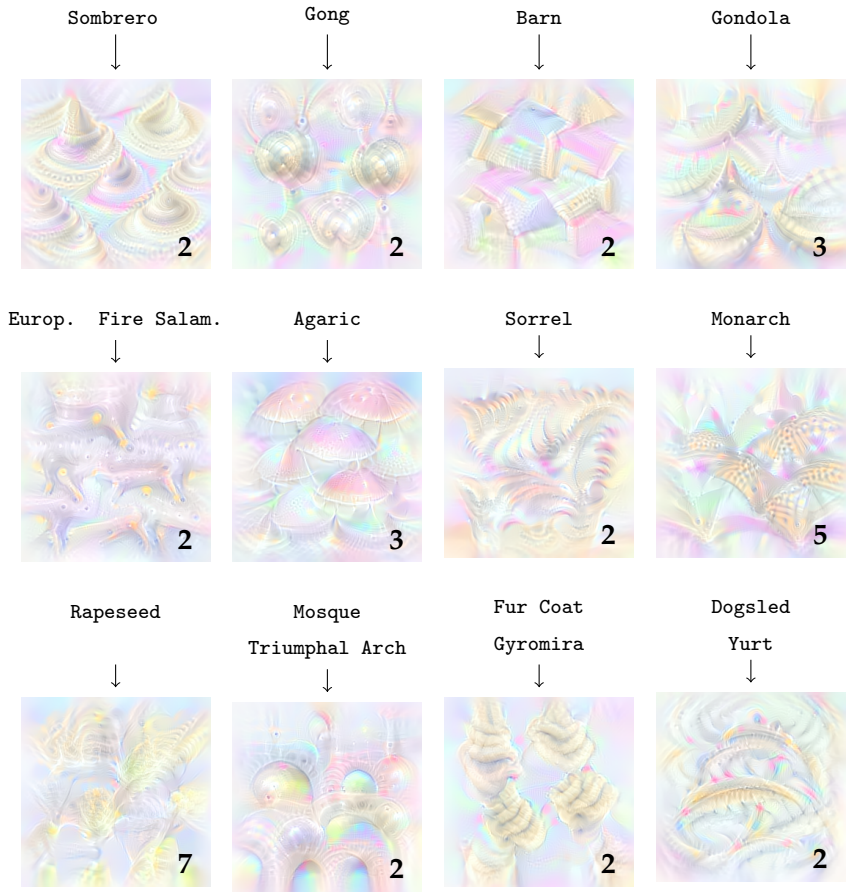


Figure 3.4: *Diverse prototypes*. Visualized are prototypes for rules found in the VGG-S network for *ImageNet* data between the output and last hidden layer. The class labels corresponding to the output are given above each image, the size of the group of neurons that this picture was generated from is given in the bottom right.

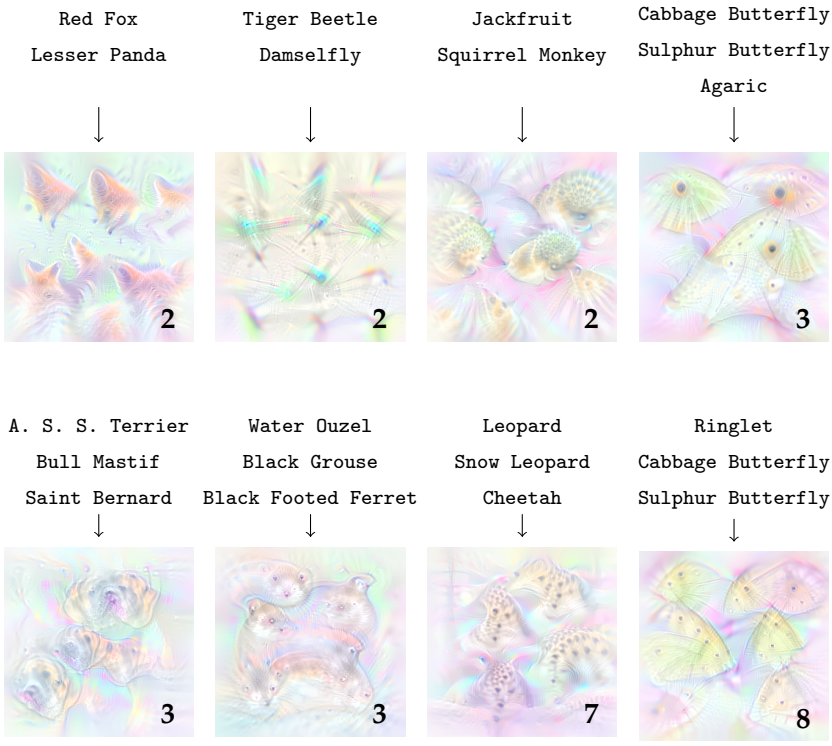
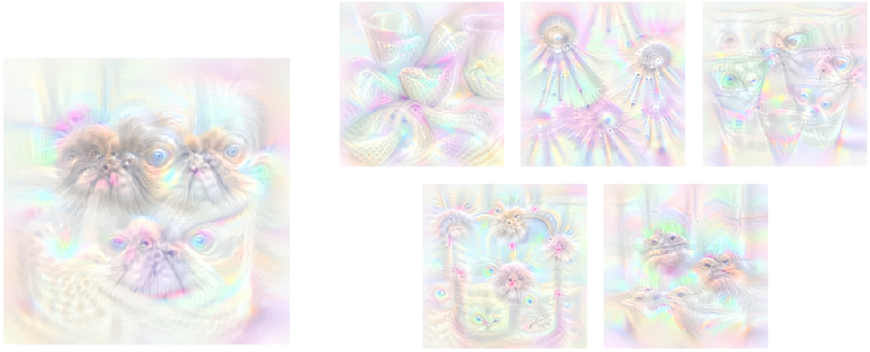


Figure 3.5: *More prototypes*. Visualized are prototypes for rules found in the VGG-S network for *ImageNet* data between the output and last hidden layer. The class labels corresponding to the output are given above each image, the size of the group of neurons that this picture was generated from is given in the bottom right.



(a) Visualization for whole tail (b) Visualization for the units in the tail individually

Figure 3.6: *Characteristic faces.* From the data for all dog breed categories, ExplaiNN discovered the rule between the labels {`Japanese spaniel, Pekinese, Shih-Tzu, Lhasa, Affenpinscher, Pug, Brabancon griffon`}, and 5 units from the *FC7* layer, for which a prototype is given in the top image. The units together capture the characteristic face of these breeds, wherease individual units (bottom) give only little insight about the encapsulated information.

colored leg or beak, or a color patch, is used per rule, such that together the sets of neurons compose the relevant information to arrive at the class prediction.

In Fig. 3.4, the first row of the panel are examples of neuron groups that learn typical shapes of objects, such as `Sombrero` or `Gondola`. The second row contains groups of neurons capturing typical patterns and colors for individual classes, such as yellow patches on black skin of the `Fire Salamander`, red caps with white dots of the `Agaric` mushroom, the typical leaf with red veins of `Sorrel` or the wings of a `Monarch` butterfly. The third row contains common features between two classes that are together captured by the same group of neurons, like the arch-like structures and round rooftops found for certain `Triumphal Archs` and `Mosques`, the layered and intertwined worm-like shapes of many `Fur Coats` and the `Gyromira` mushroom, or the characteristic traditional covering of yurts and the front part of dogsleds.

In Fig. 3.5, groups of neurons that are shared between multiple classes are visualized both revealing surprising similarities, as well as confirming that the network learns similarities that we also use as a human. In the first row, the neurons described by the first two images capture the typical shape and red color of the ears shared between the `Red Fox` and the `Lesser Panda`, respectively the insect legs and shiny turquoise color of the body of `Tiger Beetles` and `Damselflies`. Intriguingly, the network also learns a roundish shape and distinct pattern between the `Jackfruit` and the `Squirrel Monkey`.

At this point, we would like to invite the reader to look up how the top of the head of such a monkey looks like, it resembles surprisingly well the size, color, shape, and texture of a Jackfruit. For the last picture in the first row of this panel, we see dotted wings that clearly are related with the associated labels `Cabbage Butterfly`, and `Sulphur Butterfly`. But opposed to visualizations related to other Butterflies (given in both panels), the wings are all oriented in a distinct way, which resemble the cap of a dotted mushroom, which might explain the association with the `Agaric` mushroom. In the second row, we observe that the network captures common features shared between similar classes – in this case closely related animals – with the same set of neurons, which matches human intuition. To get an idea whether it is indeed the combination of neurons that captures this information, we look at the examples of faces of certain dog breeds, where we observe that if we visualize these neurons individually (Fig. 3.6), it is hard to extract anything meaningful from the images: the information is really encoded in the set of neurons that act together.

We also observe cases where rules describe how the network discriminates between similar classes. We give an example in Fig. 3.7 for the neurons ExplaiNN discovers to be associated with just huskies, just malamutes, and both of these classes together. These dog breeds are visually similar, sharing a black–white fur pattern, as well as head and ear shapes. These traits are reflected by the neurons corresponding to the rule for both breeds. Looking closer, we can see that distinct traits, the more pointy ears of the husky, respectively the fluffy fur of the malamute, are picked up by the neurons discovered for the individual classes. Beside discovering what shared and distinct traits the network has learned for classes, we also find out when it learns differences *across* samples of the *same* class. As one example, for the dog breed Great Danes, we discover three rules that upon visualization each correspond to visually very different sub-breeds, whereas a simple class prototype does not reveal any such information (Fig. 3.8).

Next we investigate the information flow within the network, by iteratively finding rules between adjacent layers, starting with rules $X \rightarrow Y$ from output layer to last fully connected layer FC7. Based on this set of rules, we then apply ExplaiNN to discover rules $Y \rightarrow Z$ between FC7 and FC6, where heads Y are groups of neurons found as tails in the previous iteration. We recursively apply this process until we arrive at a convolutional layer. This gives us traces of neuronal activity by chaining rules $X \rightarrow Y \rightarrow Z \rightarrow \cdots$ discovered in the iterative runs. We visualize two such traces in Fig. 3.10, which give insight in how the network perceives different classes, passing on information from layer to layer.

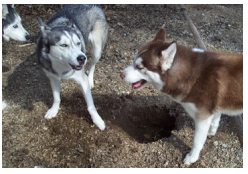
One example of a discovered trace is for the class `totem pole` (Fig. 3.10a). We observe that the set of neurons discovered for FC7 and FC6 each yield prototypes that resemble the animalistic ornaments of such totem poles, which

Siberian husky AND NOT Malamute

Siberian husky AND Malamute

Malamute AND NOT Siberian husky

(a) Rule prototypes for layer FC7.



(b) Examples from Imagenet. Left: Malamute, Right: Siberian Husky.

Figure 3.7: *Neurons discriminating Huskies and Malamutes.* (a) Huskies and Malamutes are very similar looking dog breeds. (b) Prototypes for rules $X \rightarrow Y$ discovered for classes X, `Siberian husky` (red frame), class `Malamute` (yellow frame), resp. both (orange frame) and neurons Y in FC7. The neurons associated with both classes represent typical features shared between the two classes, those associated only with `Siberian huskies` show their slightly sharper, more defined head, while those associated only with `Malamutes` capture their more fluffy fur.

can also be found in the training data. Interestingly, we see that the neuron sets found for different filters of the last convolutional layer CONV5 together detect parts of the object, including the vertical pole, the rooftop-like structures, and eyes that are often found in the animalistic ornaments, both found at the top of a totem. These filters act in a highly specific manner, detecting only specific parts of the image, such as thinner or wider vertical structures in the center, or objects at the top center of the image.

We also find signs of overfitting, e.g. when considering the information trace for a set of dog breeds (Fig. 3.10b). Note that for readability, we here only show a subset of the discovered rules with other rules showing similar prototypes. We observe that the prototypes for FC7 and FC6 both show side-views of animals. The networks seems to learn features that are specific to side photos of dogs, which are prevalent in the training data, also indicated by the filter prototypes. For the filters, we see that the network acts on very specific parts of the image, detecting structures at the bottom that resemble

Class prototype

Top: Imagenet example images
Bottom: Rule based prototypes

Figure 3.8: `Great Danes`. The left image shows the visualization for the whole class `Great Danes`. This visualization could not highlight many characteristic features, as there is a large diversity within the class. On the right side 3 images from the dataset are shown, along with 3 rules that ExplaiNN finds in connection with the class label. We are able to pick up trends, that are not characteristic to the whole class, but only a subset.

paws and pairs of front and hind legs, and at the top of the image, which resemble dog faces and clouds. We also find more abstract features with groups of filters detecting horizontal edges, which reminds of the back of the dog in side-view. While there is room for interpretation of prototypes, the discovered traces provide evidence on how the network perceives the world, as information from prototypes can be interpreted across layers, and in combination with the spatial location of activations in the filters.

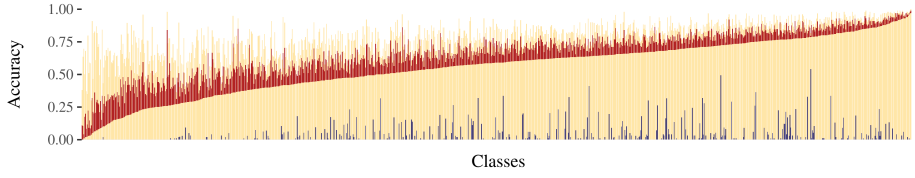


Figure 3.9: *Ablation study.* Accuracy per class of VGG-S before (yellow) and after (blue) intervention on weights connecting neurons to class given by a rule, and 90% quantile of accuracies obtained for randomized intervention (red). Intervention was done by setting the corresponding weight to zero.

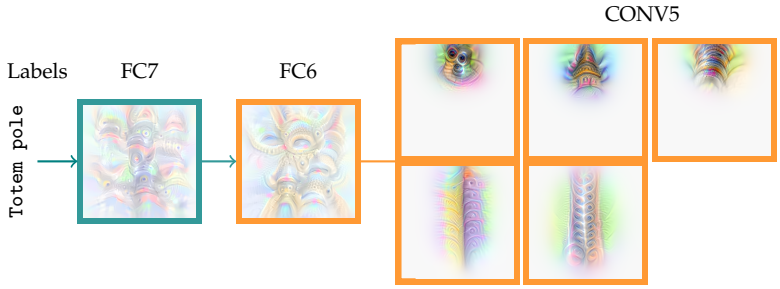
Rules carry class information To quantitatively assess the rules that ExplaiNN discovers, we here consider the VGG-S network for *ImageNet* and intervene on those neurons in the last fully connected layer that ExplaiNN finds to be class-associated. For each class c, we set incoming weights from neurons y to 0, for which we have discovered a rule $X \rightarrow Y, c \in X, y \in Y$, comparing classification rate before and after intervention. As baseline, we additionally intervene on an equally sized random subset of all weights leading to class c, again measuring classification rate after intervention. We see that for all classes, performance drops much more strongly for the actual interventions than for the random ones, in most cases even to 0 (see Fig. 3.9). This gives evidence that the discovered rules capture information necessary for classification. We further observe that under intervention the model often predicts closely related classes, e.g. `Fire Salamander` to `Spotted Salamander`, `Barbell` to `Dumbbell`, or `Palace` to `Monastery`, which gives insight towards similarity of classes, robustness of predictions, and therewith sensitivity to adversarial attacks.

The effect of fine tuning Finally, we show that ExplaiNN provides insight into the effect of fine-tuning in transfer learning. For this we consider Oxford *Flower* data (Nilsback and Zisserman, 2008), which consists of 8k images of 102 flower types. For investigation, we consider both the vanilla VGG-S network trained on *ImageNet* from above, and a fine-tuned version from the Caffee model zoo.[2] We run ExplaiNN to obtain rules between the output and the final layer of both networks. We provide examples in Fig. 3.11. The visualizations show, as expected, a strong emphasis on colour and shape of the corresponding flower. Interestingly, the visualizations of the same neurons for the original VGG-S show almost identical shapes and pattern, but with less intense colour, and in both observe prototypes with animal-like features such as eyes or beaks. This might be an indication that information about these specific flowers is already in the vanilla network hidden in some specific combination of neurons obtainable by rewiring, although the network never had to classify those, nor has it probably seen these flowers in the original *ImageNet* data.

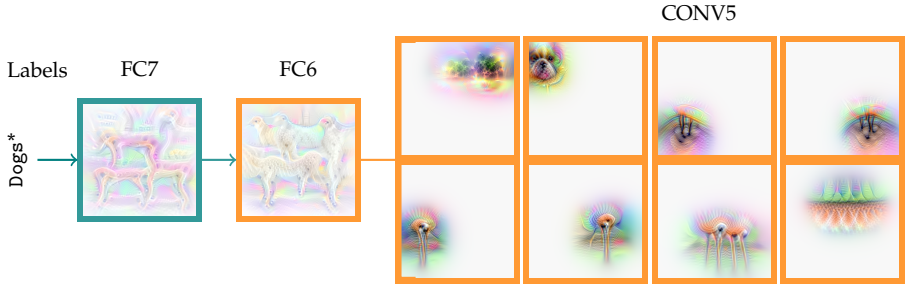
3.6 Discussion and Conclusion

The experiments show that ExplaiNN is able to discover distinct groups of neurons that *together* capture traits shared and distinct between classes, within-class heterogeneity, and how filters are used to detect shared features, segment background, or detect edges locally. Neither of these are revealed by activation maps, which miss the local information that patterns provide, nor by saliency

[2] https://github.com/jimgoo/caffe-oxford102



(a) Information flow for class `Totem Pole`.



(b) Part of an information flow for *`{Black-and-tan coonhound, english foxhound, borzoi, ibizan hound, saluki, scottish deerhound, curly-coated retriever, entle bucher, mexican hairless}`.

Figure 3.10: *Information flow.* Example rule cascades discovered for *ImageNet*. For each rule $X \rightarrow Y$, the group of neurons of tail Y are used to generate a prototype (images in colored frames). To discover these rule cascades, we first mine rules between output and FC7. We use the tails of these rules (neurons of FC7) as heads to mine rules to the next layer (FC6). Finally, we use the tails of those rules to mine rules between FC6 and CONV5.

maps, which investigate network attention for an individual image alone. Prototyping is a great tool for visualizing neuron information content, but, by itself is limited by the massive number of possible combinations of neurons, requiring thousands of hours to painstakingly handpick and connect the information of just individual neurons (Olah et al., 2020). Combining ExplaiNN with prototyping permits exploring networks beyond single neurons, by automatically discovering which neurons act in concert, which information they encode, and how information flows through the network.

In particular, we discover distinct groups of neurons in fully connected layers that capture shared respectively distinct traits across classes, which helps in understanding how the network learns generality but still can discriminate

Figure 3.11: *Flower visualizations.* For rules found between output and last fully connected layer, we visualize the neurons in the tail of the rule for the fine-tuned VGG-S network (first), the original VGG-S network (second), and example images for the flower classes (right).

between classes. Due to the local information that our rules provide, we can also detect differences in the perception across samples of a single class, where for example different groups of neurons describe visually different sub-breeds of a class of dogs. By connecting rules that we find across several layers, we trace how information is gathered and combined to arrive at a classification, from filters that detect typical class specific features in the image, through fully connected layers where multiple neurons together encode the combined information, up to the final classification output. Applying ExplaiNN to investigate the impact of fine-tuning in transfer learning, we found that for groups of neurons in the given fine-tuned CNN, surprisingly, the contained information is almost identical to the original CNN, but capturing the traits of the new classes almost perfectly. For the given task, fine-tuning thus mostly resulted in routing information differently, rather than learning to detect new features.

Overall, ExplaiNN performs well and finds surprising results that help to understand how CNNs perceive the world. While many important tasks are solved by such networks, attention based architectures play an important role in e.g. language processing. Although rules can likely also help to understand what these models learn, these networks encode an entirely different type of information that is inherently hard to understand and visualize, and hence an

exciting challenge for future work. Here, our main interest was characterizing information flow through neural networks, and hence, we focused on subsequent layers. ExplaiNN, however, operates on arbitrary sets of neurons, thus naturally allows investigating e.g. residual networks, where the previous *two* layers contribute information. Currently scaling to thousands of neurons, it will make for engaging future work to scale to entire networks at once.

Within the scope of this thesis, we will now turn back to the foundations of pattern mining, looking at the goal of finding more expressive patterns. In particular, in the next chapter we discuss the importance of patterns of mutual exclusivity, and how to find those in combination with co-occurences of features.

PATTERNS OF CO-OCCURRENCE AND MUTUAL EXCLUSIVITY

4

4.1 INTRODUCTION

Classical pattern and rule mining, which we discussed in the last chapters, is concerned with finding local associations or co-occurences, essentially modeling logical *conjunctive* statements over the attributes of the data. An essential primitive for describing relationships in many interesting applications such as biology, however, is *mutual exclusivity*. To capture relevant and interesting structure in such data, discovering patterns of mutual exclusivity can in general reveal valuable insight that goes well beyond what simple associations or co-occurrences are able to express.

We are particularly interested in discovering a small, non-redundant and easily interpretable set of patterns that together summarize the data and clearly express the significant co-occurrences and mutual exclusivity within. In supermarket basket analysis, patterns of mutual exclusivity allow to express typical buying preferences, such as products of *either* the one *or* the other brand. By combining information of mutual exclusivity with co-occurrences, we can discover the ingredients of a fancy dinner with meat and its vegetarian replacement.

While transaction data is the classic application for pattern mining, the key motivation for this work comes from biology, in particular from single cell

This chapter is based on Fischer and Vreeken (2020).

sequencing analysis. We consider binary data where for each cell (transactions) we are given which genes are active or which epigenetic features are present (items), and want to gain insight in how these interact. The traditional task is to discover groups of significantly co-activated genes, which are of interest because such genes may be part of genetic pathways or encode part of protein complexes. Patterns of co-occurrence only tell part of the story, however. Genes with *mutually exclusive* activation allow us additionally to discover e.g. *antagonistic* relationships within pathways, such as gene co-activations that are lethal, and *exchangable* sub-components in protein complexes, changing the function of the protein complex – analogous to exchanging the bit of a screwdriver.

Things become even more interesting when we consider a pattern language that additionally allows *combinations* of mutual exclusivity and co-occurrence, as this enables us to discover and succinctly describe a much larger class of possible interactions. For example, we can then express that the co-activation of two genes A and B is mutually exclusive with the co-activation of genes C, D and E, or that we often see either gene A or B activated, but whichever one it is, always together with one out of C, D, and E. Neither of these interactions would be possible to capture with statements on co-occurrence or mutual exclusivity alone, we truly need a pattern language that includes both.

In this chapter we define exactly such a pattern language, with the goal to discover the set of patterns over this language that together summarize the data best. We formalize this problem in terms of the Minimum Description Length principle, which permits a score such that it is robust against noise in the occurrences of these patterns, and, can avoid spurious results through an efficient statistical test for K-ary mutual exclusivity. As the combinatorial problem of mining the best set of patterns does not lend itself to efficient exact search, we propose Mexican, a highly efficient bottom-up heuristic to discover good pattern sets from data.

We evaluate Mexican on both synthetic and real-world data. On synthetic data we confirm that, unlike the state of that art, Mexican is robust to noise and reconstructs the ground truth, and on a wide range of real world datasets, we find that it discovers small sets of patterns that we confirm to provide meaningful information. For example, from single cell sequencing data we discover previously unknown patterns of mutual exclusivity that reflect driving factors of local processes, which can be confirmed with results from the literature.

4.2 Related Work

We discussed related work from classical (conjunctive) pattern mining in the second chapter. To the best of our knowledge, the task of mining (non-redundant, significant) patterns of mutual exclusiveness has not yet been explored. There

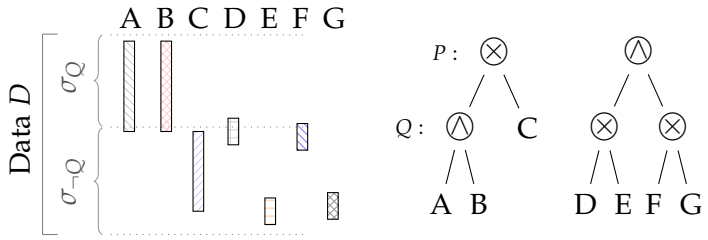


Figure 4.1: *Toy database with example pattern forest.* Left: Database D of items $A, ..., G$. Item presence is indicated by bars. Transactions where pattern-subtree Q holds are indicated by σ_Q, the remainder of data where the tree does not apply by $\sigma_{\neg Q}$. Right: Pattern trees of database D.

do however exist proposals to generalize pattern mining towards richer boolean expressions, including disjunctions (Zaki et al., 2010; Shima et al., 2004; Nanavati et al., 2001), but these are again limited to frequency as a measure of interestingness. Closer to our goal are approaches to mine association rules with negative dependencies (Hämäläinen, 2012; Antonie and Zaïane, 2004), as whenever we discover both $A \to \neg B$ and $B \to \neg A$, we can conclude mutual exclusiveness between items A and B, i.e. infer $A \otimes B$. This gives us the first baseline approach we will consider in the experiments, i.e. we mine significant association rules using Kingfisher (Hämäläinen, 2012), and post-process its results to identify patterns of mutual exclusivity.

Another related approach is that of mining low entropy sets (Heikinheimo et al., 2007), which are itemsets for which the contingency table exhibits low entropy. Low entropy sets hence generalize frequency, and can detect any type of dependency, including mutual exclusivity. We consider this as the second baseline, where we simply mine itemsets with an entropy lower than τ, and post-process the result to identify those that exhibit mutual exclusivity.

4.3 NOTATION

We use the same notation of a database D over items $\mathcal{I}$, transactions $t \in \mathcal{P}(\mathcal{I})$, and itemsets $X \subseteq \mathcal{I}$ as before. To express the richer pattern language of co-occurrence, mutual exclusivity, as well as combinations thereof, we enrich the notation of traditional pattern mining. First, we need the projection $\pi_X(D)$ of a database D onto an itemset X, which yields the intersection of each transaction $t \in D$ with X, i.e. $\pi_X(D) = \{t \cap X \mid t \in D\}$. Second, we need the selection $\sigma_c(D)$ of a logical condition c over database D, which yields all transactions $t \in D$ that satisfy c, i.e. $\sigma_c(D) = \{t \in D \mid c(t) \equiv \top\}$. We call the number of transactions where this condition *holds*, its support $supp_D(c) = |\sigma_c(D)|$.

We denote the logical k-ary AND by $\textcircled{\wedge}_{c_1,\ldots,c_k}$. For a given transaction t it resolves to $\top$ iff all the given conditions hold, i.e. $\left(\textcircled{\wedge}_{c_1,\ldots,c_k}(t) \equiv \top\right) \leftrightarrow \left(\forall_{i=0}^{k} c_i(t) \equiv \top\right)$. Analogue, we denote the logical k-ary XOR by $\otimes_{c_1,\ldots,c_k}$, which resolves to $\top$ iff exactly one of the provided conditions holds, i.e. $\left(\otimes_{c_1,\ldots,c_k}(t) \equiv \top\right) \leftrightarrow \left(\exists_j\left((c_j(t) \equiv \top) \wedge (\forall_{i\neq j} c_i(t) \equiv \bot)\right)\right)$. We connect conditions back to items $I \in \mathcal{I}$ by introducing the base case $\textcircled{\wedge}_I(t) \equiv \top \leftrightarrow I \in t$. Vice versa, $it(c)$ gives us the itemset X of all items involved in a condition c. We then have the projection of D onto condition c as $\pi_{it(c)}(D)$.

To ease notation, we will write $\odot(c_1, \ldots, c_k)$ wherever t is clear from context. In addition, we will directly use single items $I \in \mathcal{I}$ as conditions, instead of writing $\textcircled{\wedge}(I)$. As an example, we will write $\textcircled{\wedge}(A, B, C)$ and $\otimes(A, B, C)$ to denote the pattern of co-occurence pattern resp. mutual exclusivity over ABC.

We can express more complex patterns by using hierarchies of conditions, e.g. we can express by $\otimes(\textcircled{\wedge}(A, B), \textcircled{\wedge}(C, D))$ that AND-pattern AB holds mutually exclusively with AND-pattern CD. In other words, a condition c forms a *pattern tree*, with conditions $c \in \{\otimes, \textcircled{\wedge}\}$ as inner nodes, and items $I \in \mathcal{I}$ as leafs. From now on, we will use formulas, patterns, and pattern trees interchangably.

4.4 THEORY

To solve our problem using MDL, we need to formally define a model class $\mathcal{M}$. As we use two-part codes, we will further need to define a code length function that gives the number of bits needed to describe a model, and a code length function that yields the number of bits needed to describe the data at hand given a model. Before we formally introduce our approach, we provide the intuitions of the problem and how MDL naturally lends itself to solve it.

4.4.1 THE PROBLEM, INFORMALLY

Given a database, our goal is to find a set of patterns that together succinctly describe the data. Here, we are interested in patterns that capture co-occurence as well as mutually exclusive relationships in the data. These can be simple relationships, such as the co-occurence of items A, B, C, captured by $\textcircled{\wedge}(A, B, C)$, or the mutual exclusive occurence of them, captured by $\otimes(A, B, C)$. We are also interested in more complex, nested relationships, e.g. two item sets $X = \{X_1, ..., X_i\}$ and $Y = \{Y_1, ..., Y_j\}$, with items in X co-occuring, and items in Y co-occuring, but X and Y occuring mutually exclusive, which results in $\otimes(\textcircled{\wedge}(X_1, ..., X_i), \textcircled{\wedge}(Y_1, ..., Y_j))$. With $X = \{A, B\}$, $Y = \{C\}$ we show this pattern tree in Fig. 4.1.

We hence define a model $M \in \mathcal{M}$ as a set of pattern trees $\mathcal{P}$, which we refer to as a pattern forest. We require that every M always contains the singleton tree $\textcircled{\wedge}(I)$ for each item $I \in \mathcal{I}$. This gives us the baseline encoding that ensures we can always model any data D over $\mathcal{I}$. Wherever a non-singleton pattern tree in M holds, however, we will transmit the corresponding data accordingly. As an example, considering Fig. 4.1 again, we can succinctly transmit where A, B, C hold by sending all transactions where pattern tree P holds in one go, rather than sending this for each item individually. This allows us to detect patterns even of low support, as if the corresponding items co-occur sufficiently strongly, encoding them together will reduce the code length.

Overall, we aim to find that model $M^* \in \mathcal{M}$ such that the overall cost for the model and data is minimal.

4.4.2 MDL FOR PATTERN FORESTS

We will now formalize an MDL score based on the intuition of pattern forests above. First, we describe how to compute the model costs and then define the cost of transmitting data given a model .

Encoding a Model

To transmit a model, we first send how many pattern trees are there in our model M. We then send each pattern tree along with the items used in each tree. Our model costs are thus defined as

$$L(M) = L_{\mathbb{N}}(|M|) + \sum_{P \in M} \Bigg(\underbrace{\log \binom{|\mathcal{I}|}{|it(P)|} + L_{pc}(|\mathcal{I}|, 2)}_{\text{NML code}} + L^D(P) \Bigg) + \sum_{I \in \mathcal{I}} L_{st}(I) \ ,$$

where the last term is the cost for the singleton stumps. We transmit the number of trees in the forest using the MDL-optimal code $L_{\mathbb{N}}(n)$ for integers. To send an individual pattern tree $P \in M$, we first transmit which items $it(P) \subseteq \mathcal{I}$ are used in the tree, which we do using a refined MDL code, in particular the Normalized Maximum Likelihood (NML) code for multinomials. Once we know the relevant items, we proceed to transmit the actual tree using $L^D(P)$, which we define next.

We encode trees recursively, starting from the root. For every node we have to use 1 bit to encode whether it is an internal node or a leaf. If P is a leaf, we are done, and have $L^D(P) = 1$. If P is an internal node, we have to additionally

encode the type of the operator ($\textcircled{\wedge}, \otimes$), the number of children $|ch(P)|$, and the items $it(Q)$ each child $Q \in ch(P)$ contains, after which we can recurse.

For an internal $\textcircled{\wedge}$ node P with children $ch(P) = \{Q_1, \ldots, Q_k\}$, and $D' = \sigma_P(D)$ that part of the data where P holds,

$$L^D(P) = 2 + L_{\mathbb{N}}(|ch(P)|) + L_{pc}(|D|, 2) + \sum_{Q_i \in ch(P)} L^{D'}(Q_i) + L_{pc}(|it(P)|, |ch(P)|) + \log \binom{|it(P)|}{|it(Q_1)|, \ldots, |it(Q_k)|},$$

where the terms on the second line together encode the relevant items per child. Analogue, whenever P is an $\otimes$ node, we have

$$L^D(P) = 2 + L_{\mathbb{N}}(|ch(P)|) + \sum_{i=1}^{k} \left(L_{pc}(|D'_{\geq i}|, 2) \right) + \sum_{Q \in ch(P)} L^{D'_{\geq i}}(q) + L_{pc}(|it(P)|, |ch(P)|) + \log \binom{|it(P)|}{|it(Q_1)|, \ldots, |it(Q_k)|},$$

where $D'_{\geq i} = D \setminus (\cup_{j=1}^{i-1} \sigma_{Q_j})$ that data excluding transactions covered by children before Q_i. Finally, for singleton trees P_I we get

$$L_{st}(P_I) = 1 + L_{pc}(|D|, 2) \ ,$$

where we use 1 bit to indicate the root is a leaf node, and the parametric complexity L_{pc} for the log binomial over all rows.

Encoding the Data

To encode data D using a model M, we make use of the information that the pattern trees $P \in M$ provide about the dependencies in the data. In particular, we use a pattern P to encode that part of the data where P holds, while we encode the remaining data using the singleton trees P_I for each singleton $I \in \mathcal{I}$. We will start with encoding the data for which a singleton tree P_I holds, i.e., where I is present. To do so we use optimal data-to-model codes,

$$L^D\left(\pi_{P_I}(D) \mid P_I\right) = \log \binom{|D|}{|\sigma_{P_I}|} .$$

For a non-singleton pattern tree, things are slightly more involved. First, we have to encode where the root node, i.e. the logical formula for this tree, holds, after which we can recurse. We have two cases, that of an $\textcircled{\wedge}$ node, and that of an $\otimes$ node. We start with the former, where we consider data D and a tree

$P_{\textcircled{\wedge}}$ with root node $\textcircled{\wedge}$ with children $ch(P) = \{Q_1, \ldots, Q_k\}$. We first encode for which of the $|D|$ transactions P holds, after which we can recurse for each child Q only for that part of the data $D' = \sigma_{P_{\textcircled{\wedge}}}(D)$ where $P_{\textcircled{\wedge}}$ holds,

$$L^D(\pi_{P_{\textcircled{\wedge}}}(D) \mid P_{\textcircled{\wedge}}) = \log \binom{|D|}{|\sigma_{P_{\textcircled{\wedge}}}|} + \sum_{i=1}^{k} L^{D'}(\pi_{Q_i}(D') \mid Q_i) \ ,$$

where $\pi_P(D)$ is the projection of data D on pattern P. Analogue, for a pattern tree $P_{\otimes}$ with an $\otimes$ as root node, we iteratively encode where each child holds, while actively using information about already transmitted children. We have

$$L^D(\pi_{P_{\otimes}}(D) \mid P_{\otimes}) = \sum_{i=1}^{k} \left(\binom{D'_{\geq i}}{|\sigma_{Q_i}|} + L^{D'_{\geq i}}(\pi_{Q_i}(\sigma_{P_{\otimes}}) \mid Q_k) \right) .$$

where $D'_{\geq i} = D \setminus (\cup_{j=1}^{i-1} \sigma_{Q_j})$ that data excluding data covered by children before Q_k. Importantly, this encoding is independent of the order in which we iterate over the children. We will provide a proof for the case of 3 children, the case for an arbitrary number of l children follows the same reasoning.

Theorem 4.1. *Given a node $P = \otimes(i, j, k)$ with corresponding margins n_i, n_j, n_k of the children, it does not matter in which order we send where the children hold using $L^{D'}(\pi_{P_{\otimes}}(D') \mid P)$.*

Proof. We essentially need to show that we can flip the children order without changing the cost, for that assume a new order $P = \otimes(k, i, j)$, then we show that

$$\begin{aligned} &\log \binom{n}{n_i} + \log \binom{n - n_i}{n_j} + \log \binom{n - n_i - n_j}{n_k} \\ \stackrel{!}{=} &\log \binom{n}{n_k} + \log \binom{n - n_k}{n_i} + \log \binom{n - n_i - n_k}{n_j} \ . \end{aligned}$$

We use the definition of the binomial with factorials, use the standard rules for logarithmic arithmetic to pull the binomials apart, and then add new terms

that add up to 0 to derive the equation above.

$$
\begin{aligned}
&\log \frac{n!}{(n-n_i)!n_i!} + \log \frac{(n-n_i)!}{(n-n_i-n_j)!n_j!} + \log \frac{(n-n_i-n_j)!}{(n-n_i-n_j-n_k)!n_k!} \\
&= \log(n!) - \cancel{\log((n-n_i)!)} - \log(n_i!) + \cancel{\log((n-n_i)!)} \\
&\quad - \cancel{\log((n-n_i-n_j)!)} - \log(n_j!) + \cancel{\log((n-n_i-n_j)!)} \\
&\quad - \log((n-n_i-n_j-n_k)!) - \log(n_k!) \\
&\quad \underbrace{+\log((n-n_k)!) - \log((n-n_k)!)}_{=0} \\
&\quad \underbrace{+\log((n-n_i-n_k)!) - \log((n-n_i-n_k)!)}_{=0} \\
&= \log \frac{n!}{(n-n_k)!n_k!} + \log \frac{(n-n_k)!}{(n-n_i-n_k)!n_i!} + \log \frac{(n-n_i-n_k)!}{(n-n_i-n_j-n_k)!n_j!}
\end{aligned}
$$

The other permutations as well as the case for more than 3 children follow the same reasoning. □

As an example of a pattern tree $P_{\otimes}$, consider Fig. 4.1, where we would like to encode data for pattern P. Following to the equation above, for an $\otimes$ pattern we first encode for each children where they hold, and then recurse, which yields $\log \binom{|D|}{|\sigma_Q|}$ bits for identifying transactions of Q and $\log \binom{|\sigma_{\neg Q}|}{|\sigma_C|}$ bits for identifying transactions of the leaf C. The recursion on the $\bigwedge$ child Q yields 0 bits, as we already identified the corresponding transactions with the code for the root node. The same is the case when we recurse on the leaf nodes.

Putting all together, we now send for each pattern tree where it holds and transmit the remaining data with singleton trees,

$$
L(D \mid M) = \sum_{P \in M} \left(L^D(\pi_P(D) \mid P) \right) + \sum_{I \in \mathcal{I}} \left(L^D(\pi_I(D) \mid I) \right),
$$

where the last term corresponding to the singleton trees only transmits transactions that are not covered by a pattern, i.e. we consider a modified transaction multiset $\sigma'_I(D) = \{t \in D \mid I \in t \wedge (\forall P \in M.\ I \notin \pi_P(t))\}$. With the above, we have a lossless encoding for a dataset D given a model M.

4.4.3 THE PROBLEM, FORMALLY

With the scores above, we can now formally state the problem.

Problem 5 (Minimal Pattern Forest Problem)**.** *Given a database D over items $\mathcal{I}$, find the smallest set of pattern trees M that minimizes the total description length*

$$L(D, M) = L(M) + L(D \mid M) \ .$$

Although our defined model encoding allows for arbitrary hierarchies over $\otimes$ and $\oplus$, we can apriori reject certain combinations because we know they will not be insightful. For example, if a node and its children are all $\oplus$, we can obtain a much simpler model without loss by merging these nodes into a single $\oplus$ node. We can hence exclude directly nested $\oplus$ nodes from our search.

We also exclude pattern trees with directly connected $\otimes$ nodes – not for reasons of inefficiency, but rather because we are not interested in what these represent. Consider, for example a nested $\otimes$ pattern such as $\otimes(A, \otimes(B, C))$. Rather than expressing that *one* of either A, B, or C holds, which is what we are explicitly interested in, this pattern expresses that an odd number of its items is true. Although arguably an interesting statement to discover in data, it is not mutual exclusivity and hence not what we are after.

Overall, we therefore enforce alternating layers of operators, that is a children of $\otimes$ can only be a $\oplus$ or a leaf, and a children of $\oplus$ can only be $\otimes$ or a leaf. To keep all discovered patterns interpretable, we restrict ourselves in practice to pattern trees of depth at most 2.

Discovering the exact solution to the *Minimal Pattern Forest Problem* requires the enumeration of all possible sets of pattern trees, for the simple reason that our score does not exhibit any trivially exploitable structure such as convexity, monotonicity, or sub-modularity. However, the model space is of size

$$\begin{aligned} |\mathcal{M}| = \sum_{k=1}^{|\mathcal{I}|/2} \sum_{i=2k}^{|\mathcal{I}|} \binom{|\mathcal{I}|}{i} \sum_{\substack{l_1+..+l_k=i \\ l_1,..,l_k \geq 2}} \binom{i}{l_1,..,l_k} \\ \cdot \prod_{j=1}^{k} 2 \sum_{m=0}^{l_j/2} \sum_{\substack{h_1,..,h_m \geq 2 \\ 2m \leq h_1+..+h_m \leq l_j}} \binom{h_1+..+h_m}{h_1,..,h_m} (l_j - (h_1+..+h_m))! \ . \end{aligned}$$

In order of terms in the first line, the first sum indicates the number of trees in a model, the second sum the number of items covered by these trees, the first binomial the number of subset of this size, the third sum together with the first multinomial indicates the possible partitionings of this item subset over the trees. In the second line we describe the number of trees possible, given by the first product, and then multiply by 2, which is the number of ways we can fix the operators. We then have to go over all different tree shapes. For that, we first sum over all the possible number of inner nodes in the tree. The second sum in the second line together with the first multinomial gives the partition of

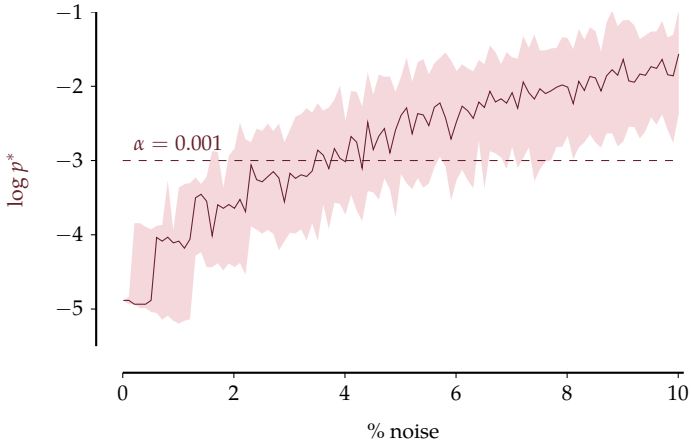


Figure 4.2: *Noise sensitivity of Fisher's exact test.* Exact p-value p^* for experiments of 2-ary $\otimes$ with varying levels of noise in the data. Significance threshold α is given as dashed line, 25% and 75% quantiles are indicated by the red band.

items over the leafs of these inner nodes. The last term gives the combinations of distributing the remaining items over the leafs that are children of the root. It is thus practically infeasible to enumerate this search space. Hence, we resort to heuristics.

4.4.4 THE CHANCE OF BEING EXCLUSIVE

Before we present our algorithm we first discuss the issue of spurious discoveries. That is, especially in sparse data, we are very likely to find that two or more items are perfectly mutually exclusive but just so by chance. To rule out that we include such spurious patterns in a model, we introduce a statistical test that supplements our MDL score, that is able to rule out patterns in a model that are likely to arise by chance. Formally, we want a statistical test that yields the likelihood of seeing an MDL gain similar or better than observed for a given $\otimes$ pattern over itemset X, assuming independence between the items $I \in X$. It turns out that the MDL gain for pattern with two items $\otimes(A, B)$ is monotone in the joint count.

Theorem 4.2 (Monotonicity of gain)**.** *For items A, B with marginals n_A, n_B and joint n_{AB}, the MDL gain of adding $\otimes(A, B)$ to the model M is smaller than for any $n'_{AB} < n_{AB}$.*

Proof. We will show that the MDL costs for a joint count $n_{AB} + 1$ is the cost for joint count n_{AB} plus some $\log \epsilon > 0$. We will start off with the MDL costs,

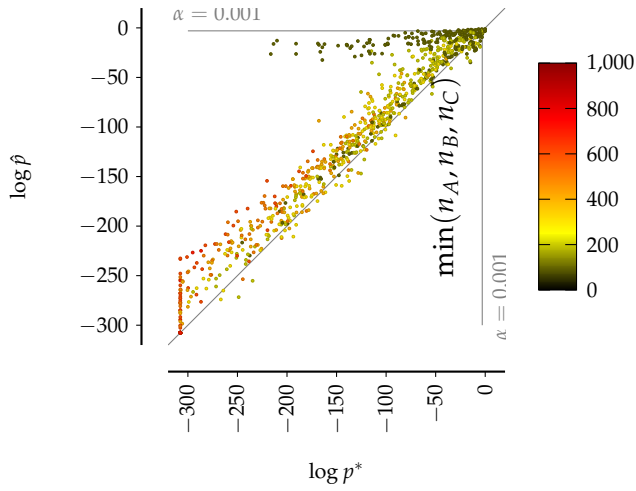


Figure 4.3: *Approximating Fisher's exact test.* Approximate p-values $\hat{p}$ against exact p-values p^* for experiments of 3-ary $\otimes$ with varying margins. Minimum margin for an experiment is indicated by color.

given by the costs of transmitting transactions covered by the XOR pattern and transmitting the transactions where A, B overlap using the singleton code. The model costs can be ignored as they are the same for different joint counts of the same pattern. Hence, we get

$$
\begin{aligned}
&\log\binom{n}{n_{\mathrm{A}}-1} + \log\binom{n-n_{\mathrm{A}}+1}{n_{\mathrm{B}}-1} + 2\cdot\log\binom{n}{n_{\mathrm{AB}}+1} \\
&\overset{(1)}{=} \log\left(\frac{n_{\mathrm{A}}}{n-n_{\mathrm{A}}+1}\binom{n}{n_{\mathrm{A}}}\right) + 2\cdot\log\left(\frac{n-n_{\mathrm{AB}}}{n_{\mathrm{AB}}+1}\binom{n}{n_{\mathrm{AB}}}\right) \\
&\quad + \log\left(\frac{n-n_{\mathrm{A}}+1}{n_{\mathrm{B}}-1}\binom{n-n_{\mathrm{A}}}{n_{\mathrm{B}}-2}\right) \\
&\overset{(2)}{=} \log\left(\frac{n_{\mathrm{A}}}{n-n_{\mathrm{A}}+1}\binom{n}{n_{\mathrm{A}}}\right) + 2\cdot\log\left(\frac{n-n_{\mathrm{AB}}}{n_{\mathrm{AB}}+1}\binom{n}{n_{\mathrm{AB}}}\right) \\
&\quad + \log\left(\frac{n-n_{\mathrm{A}}+1}{n_{\mathrm{B}}-1}\,\frac{n_{\mathrm{B}}}{n-n_{\mathrm{A}}-n_{\mathrm{B}}+1}\,\frac{n_{\mathrm{B}}-1}{n-n_{\mathrm{A}}-n_{\mathrm{B}}+2}\binom{n-n_{\mathrm{A}}}{n_{\mathrm{B}}}\right)
\end{aligned}
$$

$$\overset{(3)}{=} \log \binom{n}{n_\mathrm{A}} + \log \binom{n-n_\mathrm{A}}{n_\mathrm{B}} + 2 \cdot \log \binom{n}{n_\mathrm{AB}} + \log \Bigg(\underbrace{\frac{n_\mathrm{A} n_\mathrm{B} (n-n_\mathrm{AB})^2}{(n-n_\mathrm{A}-n_\mathrm{B}+1)(n-n_\mathrm{A}-n_\mathrm{B}+2)(n_\mathrm{AB}+1)^2}}_{=\epsilon} \Bigg) .$$

For equality (1), we use the well known equations $\binom{n}{k-1} = \frac{k}{n-k+1}\binom{n}{k}$ and $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$ to change the n, respectively k of the binomial coefficient by 1. Equality (2) is a recursive application of the first binomial coefficient equation. Equality (3) is essentially reordering and cancelling out terms and pulling the 2 into the logarithm. Starting with the costs of transmitting the data using the pattern with joint count $n_{AB} - 1$ we thus arrived at the costs of transmission costs for the pattern with joint n_{AB} plus some $\log \epsilon$. It remains to show that $\epsilon \geq 1$.

We will now bound each of the terms in the denominator of ϵ by one of the numerator terms from above. Assume that *a)* $n_A, n_B \geq 1$, *b)* $n_A > n_{AB}$, and *c)* $n_B > n_{AB}$. Then we get

$$n_A \overset{b)}{\geq} n_{AB} + 1, \quad n_B \overset{c)}{\geq} n_{AB} + 1 ,$$
$$n - n_{AB} \overset{b)}{\geq} n - n_A + 1 \overset{a)}{\geq} n - n_A - n_B + 2 > n - n_A - n_B + 1 .$$

It follows that $\epsilon \geq 1$, which complestes the proof. □

By this theorem we hence know that any datasets for A, B with similar or better gain are exactly those datasets with smaller or equal joint count.

For the simple case of two variables, we can obtain an exact p-value through Fisher's exact test (Fisher, 1922). Fisher's exact test leverages the fact that an observed joint count with fixed marginal counts follows a hypergeometric distribution, hence we can compute the exact p-value by

$$p_2^* = \sum_{i=0}^{n_{AB}} \frac{\binom{n_A}{i}\binom{n-n_A}{n_B-i}}{\binom{n}{n_B}} ,$$

where n_X is the number of rows that contain X. For given significance thresholds α, this yields an exact test to decide for the simple case of 2-ary $\otimes$ pattern if it is significant or not – and which allows evaluating the influence of noise on mutually exclusivity over A, B.

To do so, we generate data of $n = 1000$ transactions, where we plant a perfect $\otimes$ pattern with margins $n_A = n_B = 100, n_{AB} = 0$, and vary the level of noise. In particular, we add noise by flipping $\{0.01, 0.11, ..., 10\}$ % entries

uniformly at random. For each noise level we generate 100 folds, measure the p-value, for which we report the median and 25% to 75% quantile range in Figure 4.2. We see that already at very modest amounts of noise (2%) the observed gains could just as well be by chance, and clearly illustrates the need for a statistical test.

However, while with Fisher's exact test we have a test for the case of 2 variables, there is no hypergeometric distribution we can use for three variables – we have to resort to plain combinatorics, and enumerate all possibilities of observing data with the given marginals, given by

$$p_3^* = \sum_{i=0}^{n_{AB}} \sum_{j=0}^{n_{AC}} \sum_{k=0}^{n_{BC}} \sum_{l=0}^{n_{ABC}} \binom{n}{n_{\mathrm{A}}} \binom{n_A}{n_{\mathrm{AB}}} \binom{n_{\mathrm{AB}}}{n_{\mathrm{ABC}}} \binom{n - n_{\mathrm{A}}}{n_{\mathrm{B}} - n_{\mathrm{AB}}} \binom{n_{\mathrm{B}} - n_{\mathrm{AB}}}{n_{\mathrm{BC}} - n_{\mathrm{ABC}}}$$
$$\binom{n - n_{\mathrm{A}} - n_{\mathrm{B}} + n_{\mathrm{AB}}}{n_{\mathrm{C}} - n_{\mathrm{BC}} - n_{\mathrm{AC}} + n_{\mathrm{ABC}}} \binom{n_{\mathrm{A}} - n_{\mathrm{AB}}}{n_{\mathrm{AC}} - n_{\mathrm{ABC}}} \Big/ \left(\binom{n}{n_{\mathrm{A}}} \binom{n}{n_{\mathrm{B}}} \binom{n}{n_{\mathrm{C}}} \right) .$$

While somewhat doable for patterns of three items and small joint counts, this quickly becomes infeasible otherwise: if we generalize this formula to K-ary XOR, the number of sums grows to $2^K - K - 1$, while the number of terms in each summation grows to $2^K - 1$. That is, the computational complexity of just computing the p-value for K-ary XOR grows exponentially in K. As we are explicitly interested in arbitrary sized sets of mutually exclusive items, we hence need an alternative solution. To this end, we present a good approximation[1] for the p-value of K-ary XOR, for which the runtime is independent of K.

Suppose we know that for a 3-ary XOR ABC that $\otimes(A, B)$ is significant for a given significance threshold α under p_2^*. Then, we can use an adaptation of Fisher's exact test to approximate p_3^*, by treating AB as a new item that we call $\otimes_{AB}$. Furthermore, we denote by n_{AB} the number of rows that contain both A and B and are thus not mutually exclusive. We, hence, approximate by

$$\hat{p}_3 = \sum_{i=0}^{n_{ABC}} \frac{\binom{n_{\otimes_{AB}} + n_{AB}}{i} \binom{n - n_{\otimes_{AB}} - n_{AB}}{n_C - i}}{\binom{n}{n_C}} .$$

More generally, for two sets of variables X, Y for which we know $\otimes(X)$ and $\otimes(Y)$ is significant, the p-value of $\otimes(X \cup Y)$ is approximated by

$$\hat{p} = \sum_{i=0}^{n_{\tilde{X}\tilde{Y}}} \frac{\binom{n_{\tilde{X}}}{i} \binom{n - n_{\tilde{X}}}{n_{\tilde{Y}} - i}}{\binom{n}{n_{\tilde{Y}}}} ,$$

[1] We refer to it as Fischer's inexact test.

where $n_{\widetilde{X}} = |\{t \in D \mid (t \cap X) \neq \emptyset\}|$ is the number of rows where at least one of the items of X is present. Note that we thus essentially condensed X and Y each in a new item $\widetilde{X}$ and $\widetilde{Y}$. Since we are now working on the case of two variables again, Theorem 4.2 applies and we can leverage the fact that the gain is monotone in the joint count. Furthermore, we can use the following known recursive definition of terms q_i – the i-th term in the summation – that drastically reduces numerical instabilities for large n,

$$\hat{p}^0 = \frac{\binom{n-n_{\widetilde{X}}}{n_{\widetilde{Y}}}}{\binom{n}{n_{\widetilde{Y}}}} ,$$

$$\hat{p}^i = \hat{p}^{i-1} \frac{(n_{\widetilde{X}} - i)(n_{\widetilde{Y}} - i)}{(i+1)(n - n_{\widetilde{X}} - n_{\widetilde{Y}} + i + 1)} .$$

We provide a proof of this recurrence in Appendix A.3.1.

To explore how well we approximate the true p-value, we generate data with $n = 500, m = 3$, vary the marginal densities for each of the three variables in $\{5, 10, ..., 100\}$, the joints between each of the two variables in $\{0, 5, 10, 15\}$ and the full joint in steps of 2 up to the minimum pairwise joint. This leaves us with a total of 144000 experiments covering the space of very sparse to dense dataset margins and combinations of those. In our tests, $\hat{p}_3$ shows to be a good, slightly more conservative approximation of p_3^* that deviates from the exact value only if an item I has low support n_I in the order of ten rows (see Fig. 4.3), while computation is on average 200 times faster. Knowing that with larger K-ary XOR the running time increases exponentially for the exact test, while our approximation remains constant regarding K, we get a good, computable approximation that allows us to prune for insignificant mutually exclusive patterns. While due to multiple hypothesis testing issues that plague every significant pattern mining approach, we cannot claim significance of the $\otimes$ patterns, we can leverage this test as a powerful filtering technique to prevent many spurious patterns to be even considered in the model.

4.5 MEXICAN

To discover high quality pattern forests in practice, we propose the MEXICAN algorithm,[2] which leverages heuristics to efficiently explore the search space. To discover patterns representing mutual exclusivity and co-occurrences that are easily interpretable, we restrict the depth of pattern trees to 2 and alternating operators between layers. The MEXICAN algorithm can however be trivially adapted to discover patterns of arbitrary depth d.

[2] MEXICAN is short for *M*utual *EX*clus*I*ve and Conjunctive p*A*tter*N*s.

4.5.1 MERGING TREES

The main idea of MEXICAN is that, instead of enumerating all possible models, we iteratively refine the current model by combining pattern trees in the current model. We do so as follows.

MEXICAN starts with a model M that only consists of singleton trees. We can combine two singleton trees A and B by introducing a new root, yielding $\otimes(A, B)$ and $\oplus(A, B)$ as candidate pattern trees. A pattern tree $\otimes(A, B)$ and a singleton C we can combine in the following two ways. We either merge C into the XOR, and have $\otimes(A, B, C)$, or we create a new *conjunctive* node with both patterns as children, i.e. we have $\oplus(\otimes(A, B), C)$. Analogue, a pattern tree $\oplus(A, B)$ and singleton C we can again either merge, and have $\oplus(A, B, C)$, or combine under a new mutual exclusivity root node, and have $\otimes(\oplus(A, B), C)$.

If we have two pattern trees with root nodes of the same type, e.g. $\oplus(A, B)$ and $\oplus(C, D)$, resp. $\otimes(A, B)$ and $\otimes(C, D)$, we can either merge them and obtain $\oplus(A, B, C, D)$ resp. $\otimes(A, B, C, D)$, or combine them by introducing a new root node of the alternate kind, and have $\otimes(\oplus(A, B), \oplus(C, D))$ resp. $\oplus(\otimes(A, B), \otimes(C, D))$. Overall, we require that the depth of the new tree is ≤ 2, and that operators alternate along a path from root to leaf.

Summarizing the above, we create candidate patterns from pairs of pattern trees P and R as follows:

- Make R and P children of a new root $r \in \{\otimes, \oplus\}$
- Make R a new child of P
- Merge R with existing child Q of P
  - if $root(R) = root(Q)$, add children of R to children of Q
  - if Q is singleton, make Q child of R, and R child of P
- Merge R and P that have same root operator

We thus obtain an algorithm `mergeTrees(`M`)` that given a current model M yields a set of candidate patterns to refine M. Although heuristic, this scheme does explore large parts of the relevant search space: intuitively, a conjunction specifies that items usually co-occur, and hence that all subsets of these items usually co-occur, and similarly so for mutual exclusive patterns. This is captured by the idea of our bottom-up search.

4.5.2 ALGORITHM

Putting together the candidate generation strategy and the MDL code length definitions, we arrive at MEXICAN, for which we give the pseudocode in Algorithm 4.1. Starting with the set of all singleton trees as initial model (line 1), we iteratively refine our model by generating a set $\mathcal{C}$ of candidate pattern trees using `mergeTrees` (line 3), from which we find the pattern P that gives the best gain $\Delta = L(D, M \oplus P) - L(D, M)$ in terms of our previously defined MDL score (line 7). In case the candidate is an $\otimes$ pattern, we also compute

Algorithm 4.1: MEXICAN

input : Dataset D, Significance threshold α
output: Heuristic approximation to the optimal model M^*

1 $M \leftarrow \mathcal{I}$ // Initialize model with singletons
2 **do**
3 $\quad \mathcal{C} \leftarrow \texttt{mergeTrees}(M)$ // Generate candidates
4 $\quad M' \leftarrow M$
5 $\quad \Delta' \leftarrow 0$
6 $\quad$ **for** $P \in \mathcal{C}$ **do**
7 $\quad\quad \Delta \leftarrow L(D, M \oplus P) - L(D, M)$ // Compute gain
8 $\quad\quad p \leftarrow 0$
9 $\quad\quad$ **if** $root(P) = \otimes$ **then** // If candidate is XOR
10 $\quad\quad\quad p \leftarrow \hat{p}$ of P // Compute p-Value
11 $\quad\quad$ **if** $\Delta < \Delta'$ and $p < \alpha$ **then** // Update current best
12 $\quad\quad\quad M' \leftarrow M \oplus P$
13 $\quad\quad\quad \Delta' \leftarrow \Delta$
14 $\quad M \leftarrow M'$ // Update best model
15 **while** $L(D, M)$ decreases
16 **return** M

the p-Value estimate $\hat{p}$ (line 9, 10) and filter it out if it is above the signifance threshold α (line 11). We add the pattern P with the highest gain to the model, $M \oplus P$, while removing all non-singleton patterns that were used to construct P from M (line 12). If there is no candidate that would decrease the current codelength, we terminate and return the current best model (line 15, 16).

4.5.3 COMPLEXITY

As in the previous chapters, we will analyze the complexity in terms of the size of the discovered model rather than the input. The following theorems capture complexity properties of MEXICAN in terms of the number of found patterns k.

Theorem 4.3 (#Candidates of MEXICAN)**.** *Given that we mine k pattern trees, with at most l leafs each, for a given dataset D, MEXICAN evaluates at most $O((k \cdot l + m)^2 l^2)$ candidates per iteration.*

Proof. In a given iteration, we can have at most m singletons and $k' = \lfloor \frac{kl}{2} \rfloor$ pattern trees that together form a model. The term k' is the upper bound that

we get when we divide the trees in the final model into trees of size 2, which is the smallest possible tree as otherwise we would have a singleton. We can merge each tree with any other tree resulting in $(k' + m)(k' + m - 1)/2$ tree pairs to look at. Without making further assumptions on the shape or content of the tree, we have to assume that we can merge one tree in any leaf or inner node of the other tree. There are at most l leafs and at most $\lfloor l/2 \rfloor + 1$ inner nodes, using the constraint that a tree has at most depth 2. For each merge, including merging under a new root or as a new child, we can use two different operators. This gives at most $(k' + m)(k' + m - 1) \cdot 2 \cdot (2 \cdot (l + (\lfloor l/2 \rfloor + 1) + (\lfloor l/2 \rfloor + 1)) + 1)$ candidates. □

It now remains to include the number of iterations, which results in the following time complexity of MEXICAN.

Theorem 4.4 (Runtime of MEXICAN)**.** *Given that we mine k pattern trees with at most l leafs each for a given dataset D, the runtime of MEXICAN is in $O(k \cdot l \cdot (k \cdot l + m)^2 l^2)$.*

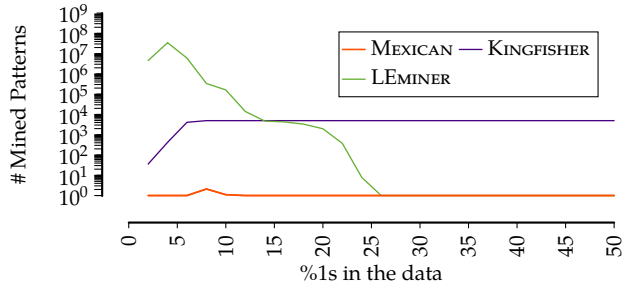
Proof. From Th. 4.3 and the observation that in each iteration one of the tree grows by at least 1 in the number of leafs because of the pairwise merge, we know that there are at most $k \cdot l$ merges possible. Otherwise we would get a tree that is larger than any tree in the final model. □

4.6 EXPERIMENTS

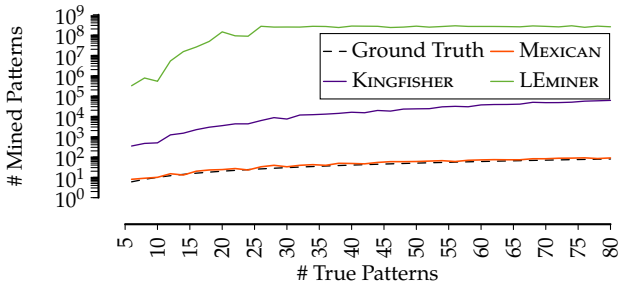
We empirically evaluate MEXICAN on both synthetic data with known ground truth as well as on real world data. For that, we implemented MEXICAN in C++. There is no direct competitor that is able to mine mutually exclusive patterns, we instead compare to the two closest cousins. The first method that we compare to is an implementation for mining low entropy patterns, LEMINER (Heikinheimo et al., 2007). To compare to LEMINER, we postprocessed the results to extract patterns where at least some subset of features has few overlapping transactions with the rest of the features, which can be seen as mutual exclusive patterns over itemsets. The second method is an adaptation of KINGFISHER that derives mutual exclusive patterns from statistically significant association rules (Hämäläinen, 2012). We reimplemented KINGFISHER in C++, such that it scales to larger datasets and processed the results to merge rules $A \to \neg B$ and $B \to \neg A$ to $\otimes(A, B)$. Note that KINGFISHER is only able to find rules with single items in the consequent, hence we can only mine for mutual exclusive feature pairs. We want to emphasize at this point that both methods LEMINER and KINGFISHER were originally not designed to discover patterns of mutual exclusivity and the comparison is thus slightly unfair. However, these methods come closest to what we do. All experiments were carried out single-threaded on Intel Xeon

E5-2643 v3 machines with 256GB RAM running Linux. MEXICAN finishes within seconds to minutes on all synthetic and real data, with the exception of the single cell and Instacart data, for which it needs hours, due to the large width respectively height of the data sets. The code and all data is available online[3].

4.6.1 SYNTHETIC DATA



(a) Patterns found in pure noise data of different sparsity.



(b) Patterns found in synthetic data with planted patterns. Numbers of LEMINER are a lowerbound as each run was terminated after 1 hour.

Figure 4.4: *Synthetic data*. Top: #Patterns found in pure noise data. Bottom: #Patterns found in data with planted patterns.

As a sanity check, we first consider data without any structure. We generate data sets of size 1000×100 with d% 1s that are set uniformly at random, and report the average number of found patterns across 10 folds for each method. We show the results in Fig. 4.4a, and see that while MEXICAN recovers the ground

[3] http://eda.mmci.uni-saarland.de/prj/mexican/

truth in all cases but one, its competitors KINGFISHER and LEMINER, discover up to millions of spurious patterns.[4]

SIMPLE PATTERNS

To evaluate whether these methods can recover ground truth beyond pure noise, we generate data where we plant 10 pure $\otimes$, respectively pure $\oplus$ patterns over at most 5 items in data with $n = 10000$ rows. We add noise to this data by flipping 0.1% of the entries – an average of 10 entries per column – uniformly at random, ensuring that $\otimes$ patterns are not spurious (see also Fig. 4.2).

We generate 10 datasets for each of the two setups, and record the number of patterns each method discovers. We find that on average KINGFISHER reports hundreds, and LEMINER more than a million patterns for each of the two experiments. In contrast, MEXICAN recovers the 10 patterns in the $\oplus$ data exactly, and on average 7.6 patterns for the $\otimes$ data. As on data without noise it does recover all patterns, this is likely due to the (strong) effect of noise on the significance of $\otimes$ patterns of higher arity.

NESTED PATTERNS

Next, we investigate how well MEXICAN performs on data that contains more complex patterns, in particular including nested logical formulas, we plant $k \in \{5, 7, ..., 80\}$ pattern trees into $n = 10000$ rows. We draw uniformly at random $c \in \{2, 3, 4\}$ children for the root. For each of those children we draw up to 2 nodes as their children. We then draw an operator $o \in \{\oplus, \otimes\}$ for each inner node. For each pattern tree we draw rows from $\mathcal{N}(500x, 10x)$, where x is the maximum number of children of any $\otimes$ subtree, or 1 if none exists, to avoid overly sparse features. We then partition the rows for each $\otimes$ subtree according to a multinomial over the children. Finally, we apply noise as in the previous experiment. We run each of the methods, and report the number of discovered patterns in Figure 4.4b. We see that, KINGFISHER and LEMINER discover thousands up to billions of patterns, whereas MEXICAN recovers models that are – both in numbers, as well as upon close inspection – very close to the ground truth.

4.6.2 REAL-WORLD DATA

To evaluate MEXICAN on real data, we look at six different binarized datasets over different domains. We consider data on Belgian traffic *Accidents*, lemmatized words from abstracts of Deep Learning and Quantum Computing papers on

[4]Note that we observed a similar behaviour of state-of-the-art methods based on frequency or statistical testing on pure noise also in the Experiments of previous chapters

			KF	LE	MEXICAN		
Dataset	n	m	$\#\otimes$	$\#\odot^k$	$\#\wedge$	$\#\otimes$	$\#\odot^k$
Accidents	$340K$	468	$38K$	$216M^*$	1	34	6
DLQ	$10K$	$4.2K$	$830K$	$34M^*$	10	130	4
DNA	$1.3K$	392	$20K$	$1B^*$	114	0	0
Mammals	$2.2K$	121	$5K$	$198K$	4	15	12
SC	65	$20.2K$	$63M$	0^*	131	500	41
Instacart	$2.6M$	1236	-	-	1	95	3

Table 4.1: Reported are number of rows n and columns m in data set and number of found AND and XOR patterns $\#\wedge$, $\#\otimes$, and nested patterns $\#\odot^k$ for KINGFISHER (KF), LEMINER (LE), and MEXICAN. By * we indicate the forced termination of LEMINER after either 1 Billion patterns were mined, or >500GB of disk space was consumed. Resulting numbers are patterns returned on termination, hence a lower bound.

ArXiv (*DLQ*) (Dalleiger and Vreeken, 2020), *DNA* amplification data (Myllykangas et al., 2006), European *Mammals* (Mitchell-Jones, 1999), and Single Cell RNA-sequencing (*SC*) data (Ardakani et al., 2018). Data dimensions and results are reported in Tab. 4.1.

The results show that MEXICAN is able to retrieve succinct, hence interpretable, sets of patterns, where the other methods discover orders of magnitude more patterns than there are rows and columns in the data. Furthermore, these results show that our method scales up to many thousands of features and millions of rows. Both KINGFISHER and LEMINER are not able to process *Instacart*. Examining the results of MEXICAN by operator type, we observe that MEXICAN for some data sets discovers many $\otimes$ patterns, especially sparse data such as *Instacart*, whereas other data that has highly correlated margins, such as *DNA*, yields more $\wedge$ patterns. When appropriate, MEXICAN discovers also more complex, nested patterns that describe the data well, which we examine further below.

The experiments further show that MEXICAN is fast despite the theoretically challenging problem, taking only seconds for *DNA* and *Mammals*, up to hours on data of very high dimensionality or sample size, such as *Instacart* and *SC*. So far, we focused on the pattern set size, which is a good indicator of accessibility of the sets to a human expert, however does not provide any qualitative statement about the results. In the following we examine the quality of pattern sets for *DLQ*, and *SC*.

DLQ data

For the *DLQ* dataset we find informative patterns that capture the discrepancies between papers of the two communities. By observing that discovered patterns reflect classical word co-occurences, such as $\textcircled{\wedge}$(monte, carlo) or $\textcircled{\wedge}$(nearest, neighbour), and the XOR pattern $\otimes(\textcircled{\wedge}(\text{deep}, \text{learning}), \text{quantum})$ that summarizes how the data was generated, it is clear that Mexican is able to infer patterns that capture main properties of the data. Close inspection of the patterns further reveals that we are able to retrieve more subtle distinctions between the Deep Learning and Quantum Computing fields, such as $\otimes$(adversarial, free), or $\otimes$(stochastic, superposition). Furthermore, Mexican discovers larger patterns that might indicate subdomains within Deep Learning or Quantum Computing, such as $\otimes$(operation, radiation, autoencoder), or $\otimes$(layer, atom, cryptography, hamiltonians, reality, remote).

Single Cell Data

Finally, we look at the case of the single cell sequencing data set. Single cell sequencing is a recent breakthrough that enables to measure genetic and epigenetic features for separate, single cells instead of cell batches that were measured previously. Hence, instead of analysing only global averages over all cell states, we can now obtain the state of each cell individually, and thus capture the patterns underlying cellular dynamics. The major challenge of such data is the large number of features as we are interested in analyzing all (several tens of thousands of) genes. Furthermore, there is only a limited understanding of the whole gene regulatory system. This is both a limitation in terms of how we can validate but also an opportunity to suggest new relationships derived from our discoveries.

For the *SC* data, many patterns discovered by Mexican reflect distinct local mechanisms within the cellular life[5]. One of the discovered patterns is $\textcircled{\wedge}$(POP5, ZCCHC17), which can be seen as a pattern of protein production, one of the most crucial functions of a cell. Both of these proteins are essential for protein synthesis, POP5 as part of the machinery detecting which building block to add to the amino acid chain (van Eenennaam et al., 2001), and ZCCHC17 in the complex actually generating the chain (Chang et al., 2003). A pattern of another important cellular process is $\textcircled{\wedge}$(IMPDH2, UMPS). Both of these genes and associated proteins are essential for the synthesis of building blocks of the DNA and thus are key for cell proliferation. In particular, UMPS is responsible for the final two steps of pyrimidine synthesis, a buildung block for certain nucleotids (the N in DNA)(Krungkrai et al., 2001). IMPDH2 catalyzes a crucial

[5]Please note that due to the history of discovery, many genes have been assigned more than one name. The aliases can be looked up at e.g. https://www.genecards.org/.

step in the synthesis of the nucleotide Guanine (Carr et al., 1993). Due to its direct effect on the supply of Guanine and thus the rate of cell proliferation, IMPDH2 is also used as drug target for cancer treatment (Naffouje et al., 2019).

An example of a mutual exclusivity pattern is $\otimes$(ZNF692, BRF1) for which we can hypothesize an antagonistic role of the two genes influencing gene regulation. In particular, ZNF692 encodes a protein that acts as a transcriptional repressor (Inoue and Yamauchi, 2006), while BRF1 is a component of RNAPolIII, responsible for the transcription of genes (Hsieh et al., 1999).

Finally, with the pattern $\otimes$(RP11.446E9.1, SETD1B, MIOX), we can suggest novel relationships between genes. MIOX is related to the Polyol metabolism, and is tightly regulated by DNA methylation in its promoter (Sharma et al., 2017). SETD1B, on the other hand, is part of a complex modifying Histone proteins (Schultz et al., 2002). These modifications are known epigenetic markers for gene regulation. Thus, the Lysine modification introduced by SETDB1 might play a repressive role in the regulation of MIOX, an interesting subject of further study.

These findings show that Mexican discovers a succinct set of patterns that characterise cellular mechanisms, and thus can be leveraged to guide future research by suggesting potential relationships as subject for further investigations.

4.7 Discussion and Conclusion

In contrast to robust rules, this chapter focused on the problem of discovering patterns of co-occurence and mutual exclusivity. In particular, we proposed a pattern language over logical conjunctions and mutual exclusivity, and defined the goal of discovering succinct and non-redundant sets of patterns over this language that together generalize the data. As such, the pattern language in combination with a scalable algorithm lends itself for exploring high-dimensional biological datasets. For example, it can be leveraged to discover new gene relationships from expression data, where mutual exclusive patterns model interesting biological concepts which would otherwise be missed.

We defined the problem in terms of the Minimum Description Length principle, and as the resulting score does not lend itself for efficient exact search, we proposed an effective heuristic approach called Mexican to efficiently approximate the optimal solution. To filter out spurious results, we suggest a statistical test for K-ary mutual exclusivity and propose a computationally efficient approximation to it.

With the among the state of the art unique ability of discovering both co-occurrences as well as mutual exclusive relationships, we show that Mexican gives an extended view on the distribution of the data by conducting experi-

ments with synthetic and real world data. On synthetic data, we showed that MEXICAN is able to recover the ground truth without picking up on noise, where the state-of-the-art methods discovered millions of patterns even when there are none. Through experiments on real data we confirmed that MEXICAN consistently returns succinct pattern sets interpretable by human experts, scaling up to millions of rows and thousands of features. Close inspection revealed that patterns MEXICAN discovered are indeed meaningful and correspond to domain knowledge.

One major application of a pattern language equipped with mutual exclusivity is biological data on the gene regulatory system, where such patterns could indicate replacable sub-components or antagonistic players in a pathway. To showcase the efficacy of MEXICAN in this domain, we considered a case study on single cell RNA sequencing data. The discovered patterns reflect local cellular mechanisms that we validated with the literature, but also suggest new relationships of genes about only little is known so far, which could be subject to further experiments.

Although MEXICAN meets the goals we set initially for this work and yields highly encouraging results, we are interested in even more scalable approaches, which could potentially scale to hundreds of thousands if not millions of features. Examples of such challenging datasets include transactional data of large retailers, or data of human variation, which are important to understand for clinical settings as well as drug development. Scaling to these orders of magnitude larger number of features, however, requires a paradigm shift from how traditionally pattern mining problems are modeled. Before we discuss a solution to this problem, in the next chapter we investigate how we can mine subgroups at scale, considering patterns of conjunctions and mutual exclusivity, and how we can use this method to understand which input makes neural networks misclassify for complex natural language and vision tasks.

DISCOVERING LABEL-DESCRIPTIVE PATTERNS

5

5.1 INTRODUCTION

State-of-the-art deep learning methods achieve human-like performance on challenging tasks, such as in computer vision or linguistics. As much as 'to err is human', these models make errors, too.[1] Some of these errors are due to noise that is inherent to the process we want to model, and therewith relatively benign. Systematic errors, on the other hand, e.g. those due to bias or misspecification, are much more serious, as these lead to models that are inherently unreliable. If we know under what conditions a model performs poorly, we can actively intervene, e.g. by augmenting the training data, and so improve overall reliability and performance. Before we can do so, we first need to know whether a model makes systematic errors, and if so, how to characterize them in easily understandable terms.

We propose a method that allows us to do so for arbitrary classification models, by partitioning the input data according to correctness of the model's predictions, and then mining those partition-specific patterns that together describe the partitions most succinctly. To get a good understanding of what

This chapter is based on Hedderich, Fischer, Klakow, and Vreeken (2022).

[1] The full proverb goes 'to err is human, to forgive is divine'. Let us hope we never have to rely on the forgiveness of neural networks.

causes systematic errors, we consider a rich pattern language that allows us to express conjunctions, mutual exclusivity, and nested combinations thereof. This task is an instance of the more general problem of label description, where for given labeled data we are interested in a non-redundant and easily interpretable description of the associations between the data and the given labels. We formulate this problem in terms of the Minimum Description Length principle, by which we identify the best set of patterns as the one that best compresses the data without loss. As the search space is twice exponential, and does not exhibit any easy-to-exploit structure, we propose the efficient and hyper-parameter-free Premise algorithm to heuristically discover the *premises* under which we see the given labels.

The label description problem is obviously related to classification. Here, we however are not so much interested in prediction, but rather description and therewith value interpretability of the results over accuracy. This notion we share with subgroup discovery (Novak et al., 2009; van Leeuwen and Knobbe, 2012; Sutton et al., 2020), emerging pattern mining (Dong and Li, 1999), and significant pattern mining (Pellegrina et al., 2019), which aim to discover those conditions under which a target attribute has an exceptional distribution and hence are closely related to finding descriptive rules for the target attribute (Agrawal et al., 1993; Hämäläinen, 2012). The key difference is that we are not interested in discovering *all* patterns that are strongly associated, but rather want a small and non-redundant set of patterns. As such, our approach is an instance of pattern set mining (Vreeken and Tatti, 2014; Fischer and Vreeken, 2019, 2020; Proença and van Leeuwen, 2020), but distinct from existing work in the sense that it has to scale to large input domains, discovers noise-robust patterns, considers a richer pattern language, and partitions of the data, all at the same time.

We evaluate Premise both on synthetic and real-world data. We show that, unlike the state of the art, Premise is robust to noise, scales to large numbers of items, deals well with imbalance, and association to labels. Through two case studies we show that Premise discovers patterns that provide clear insight into the systematic errors of NLP classifiers, and, perhaps most importantly, are indeed actionable. In particular, we show these patterns elucidate the biases of recent Visual Question Answering (VQA) classifiers, and that we can improve the performance of a neural Named Entity Recognition (NER) model by acting on the patterns Premise discovers.

5.2 Related Work

We discussed pattern mining in the related work of the previous chapters. Approaches to mine patterns or pattern sets are unsupervised in nature, i.e. they

do not take label information into account. Rule mining, however, naturally lends itself to describe associations between labels and items. As discussed before, most existing methods evaluate patterns individually, thereby discovering millions of rules even if the data is pure noise. GRAB, which we discussed in the first chapter, instead mines small sets of rules that together summarize the data well, and CLASSY (Proença and van Leeuwen, 2020) discovers rule lists that characterize a given label. We compare to both GRAB and CLASSY in the experiments. We can already reveal that it does not work well in our setting: the approaches to mine rules and rule lists do not scale well and are sensitive to label imbalancing. Furthermore, they are inherently not able to reflect that the consequence is likely to co-occur with the antecedent *and* unlikely to occur otherwise. As a result, they fail to pick up subtle patterns, e.g. one that occurs in 70% of the misclassified instances, and 40% of the correctly classified instances.

More close to our work is supervised pattern mining, out of which subgroup discovery (Wrobel, 1997; Novak et al., 2009; García-Vico et al., 2018) and emerging pattern mining (Dong and Li, 1999) are among the most prominent representatives (Novak et al., 2009). Emerging pattern mining returns all patterns that meet a user-specified 'growth' threshold, and hence suffers from the same problems as frequent pattern mining. Subgroup discovery instead returns the top-k patterns that correlate most strongly. This keeps the result sets of manageable size, but does not solve the problem of redundancy (van Leeuwen and Knobbe, 2012). Statistical pattern mining aims to discover patterns that correlate *significantly* to a class label (Llinares-López et al., 2015; Papaxanthos et al., 2016; Pellegrina and Vandin, 2018). While it is (relatively) easy to test one pattern for significance, in practice we have to evaluate many millions of candidates, and hence multiple hypothesis testing becomes a serious problem: these methods tend to discover millions of 'significant' patterns even from small data, making the results hard to use.

Specifically to explain classifiers, several approaches aim to capture dependencies of features or attributes that a classifier uses to make a prediction, e.g. in terms of patterns or rules (Henelius et al., 2014; Barakat and Diederich, 2005), by model distillation (Frosst and Hinton, 2017; Lakkaraju et al., 2017), or to discover patterns of neurons within neural networks that drive a decision (Fischer et al., 2021b). These, however, focus on the dependencies the classifier exploits for successful prediction as opposed to understanding where – or why – something goes wrong. Here, Duivesteijn and Thaele (2014) use the CORTANA tool (Meeng and Knobbe, 2011) to explain where a classifier performs particularly poorly in terms of feature subspaces. SliceFinder (Chung et al., 2020) follows a similar idea. However, both models were only evaluated on data with less than 50 features. Our experiments show that these methods do not scale well to the feature spaces common in NLP data.

For specific applications, such as characterizing classification errors in NLP

models, there exists manual approaches based on challenging test sets (Gardner et al., 2020; Ribeiro et al., 2020) or testing a hypothetical cause for misclassification (Rondeau and Hazen, 2018; Wu et al., 2019; Lee et al., 2019a). Such manual approaches, however, require existing knowledge about the difficulties of the models. In contrast, LIME (Ribeiro et al., 2016) and Anchors (Ribeiro et al., 2018) analyze and describe the decision boundary of each instance, thus providing only local explanation for individual samples. Furthermore, none of the methods scales well to the data that we consider.

5.3 Notation

As in the previous chapters, we consider binary transaction data D over a set of items $\mathcal{I}$, and use the same definitions of transactions, itemsets, and projections. Additionally, each transaction $t \in D$ is assigned a binary label $\ell(t) \in \{l_-, l_+\}$. For ease of notation, we define the partition of the database according to this binary label $D^- = \{t \in D \mid \ell(t) = l_-\}$ and $D^+ = \{t \in D \mid \ell(t) = l_+\}$. Moreover, we use the language of k-ary conjunctions and XOR from Mexican. We are here specifically interested in patterns of AND operator over XOR operations, i.e. $\bigwedge(\bigotimes_{c_1,\ldots,c_k}, \ldots, \bigotimes_{c'_1,\ldots,c'_k})(t)$. An XOR operation is called clause, $\gamma(c)$ lists all clauses in conjunctive condition c. To simplify notation, we drop t where it is clear from context and write I for conditions on a single item $c(I)$. In this chapter, we use condition and pattern interchangeably.

5.4 Theory

To discover those patterns best describing the given labels, we here introduce the class of models $\mathcal{M}$ and corresponding codelength functions. Before we define these formally, we give the intuition.

5.4.1 The problem, informally

Given a dataset of binary transaction data and corresponding binary labels, we aim to find a set of patterns that together identify the partitioning of the data according to the labels. As an application, consider the input words of an NLP task as transactions, along with labels that express whether an instance is misclassified by a given model. We are now interested in patterns of words that describe these labels. In essence, we want to find word combinations such as $\bigwedge(\textit{how, many})$, or mutual exclusive patterns, e.g. $\bigotimes(\textit{color, colour})$, that capture synonyms or different writing styles, all occuring predominantly when a misclassifcation happens. The pattern language we use is a combination of

the two, namely conjunctions of mutual exclusive clauses, e.g. $\owedge(\mathit{what}, \otimes(\mathit{color}, \mathit{colour}))$. We provide an example in Figure 5.1.

We thus define a model $M \in \mathcal{M}$ as the set of patterns $\mathcal{P}$ that help to describe given labels. Similar to the previous chapters, to ensure that we can always encode any data, M contains all singleton words $I \in \mathcal{I}$, describing the entire data D label unspecific. Here, the model containing all singletons also acts as a baseline implementing the assumption that there are no associations that describe the label. Whenever there is a structure in the labels that can be explained by a pattern, we transmit data corresponding to a label (D^+, D^-) separately. This allows us to more succinctly transmit where patterns hold.

Let us consider the example in Figure 5.1, where we would first send $\owedge(A, \otimes(B, C))$ occurrences in D^+, and then its occurrences in D^-. Thus, we identify where A, C, and D hold at once, and we leverage the fact that $\owedge(A, \otimes(B, C))$ occurs predominantly in D^+, resulting in more efficient transmission. Intuitively, a bias of a pattern to occur in one label more than in the other corresponds to a large deviation between the conditional probability – the pattern occurrence conditioned on the label – and the unconditional probability – the pattern occurence in the whole database. In this case, the codes are hence more efficient when sending pattern separately for D^+ and D^-. Coming back to the example, F however occurs similarly often in both labels – there is almost no deviation between conditional and unconditional probability – hence it is unlikely that it identifies a structural error. Here, the baseline encoding transmitting F as singleton in all of D will be most efficient. This approach allows us to identify patterns that occur predominantly for one of the labels as the patterns that yield better compression when conditioned on the labels, and thus characterise labels in easily understandable terms.

We are hence after the model $M^* \in \mathcal{M}$ that minimizes the cost of transmitting the data and model. In the following sections, we will formalize this intuition using an MDL score to identify that pattern set that best describes the data given the labels. We will first detail how to compute the encoding cost for the data given the model and then the cost for the model itself.

5.4.2 COST OF DATA GIVEN MODEL

Let us start by explaining how to encode a database D with singleton items I in the absence of any labels, which will later serve as the baseline encoding corresponding to independence between items and labels. To encode in which transaction an item I holds, we use optimal data-to-model codes as introduced before. Hence, the codelength for transmitting the data is

$$L\left(\pi_I\left(D\right) \mid I\right) = \log \binom{|D|}{|\sigma_I\left(D\right)|} .$$

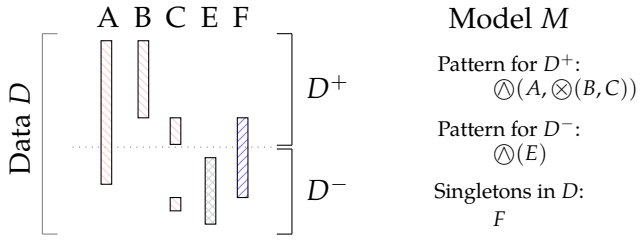


Figure 5.1: *Example database and model.* Left: a toy database D over a set of items, separated by labels into D^+ and D^-. Right: the corresponding model M containing patterns describing data partitions D^- and D^+ induced by labels l_- and l_+

Taking into account the partitioning of D along the label, yielding D^+ and D^-, we encode I separately:

$$L\left(\pi_I\left(D\right) \mid I\right) = \log\left(\begin{pmatrix} |D^-| \\ |\sigma_I\left(D^-\right)| \end{pmatrix}\right) + \log\left(\begin{pmatrix} |D^+| \\ |\sigma_I\left(D^+\right)| \end{pmatrix}\right).$$

As such, we explicitly reward patterns (here, singletons) that have a different distribution between the unconditional probability – i.e. frequency in D of I – and the conditional probability of I conditioned on the label – i.e. frequency in D^- respectively D^+. It models the property that we are interested in; a pattern that characterizes a certain label. It is straightforward to extend to patterns of co-occurring items $P = \bigwedge(X_1, \ldots, X_k)$ by selecting on transactions where the pattern holds

$$L\left(\pi_P\left(D\right) \mid P\right) = \log\left(\begin{pmatrix} |D^-| \\ |\sigma_P\left(D^-\right)| \end{pmatrix}\right) + \log\left(\begin{pmatrix} |D^+| \\ |\sigma_P\left(D^+\right)| \end{pmatrix}\right).$$

There might be transactions where individual items of P are present, but not all of P holds. To ensure a lossless encoding, the singleton code $L(\pi_I\left(D\right) \mid I)$ is modified to cover all item occurrences left unexplained after transmitting $\mathcal{P}$. Hence, we get

$$L_s(\pi_I\left(D\right) \mid P) = \log\left(\begin{pmatrix} |D| \\ |\sigma_I\left(D\right) \setminus \left(\bigcup_{P\in\mathcal{P}, I\in P}\ \sigma_P\left(D\right)\right)| \end{pmatrix}\right).$$

For patterns expressing conjunctions over mutual exclusive items, such as $\bigwedge(\bigotimes(A, B), \bigotimes(C, D))$, we first send for both D^- and D^+ for which transactions the pattern holds, after which we specify which of the items is active where. We do that one by one, as we know that when the pattern holds and A is present, B cannot be present too. With each transmitted item of the clause, there are thus

fewer transactions where the remaining items could occur, hence the codelength is reduced. More formally, the codelength for a pattern P of conjunctions of clauses is given as

$$L(\pi_P(D) \mid P) = \sum_{l \in \{-,+\}} \log \binom{|D^l|}{|\sigma_P(D^l)|} + \sum_{\substack{cl \in \\ \gamma(P)}} \sum_{\substack{I \in \\ cl}} \log \binom{|\sigma_P\left(D^l\right)| - \sum_{I' \in cl, I' \leq I} |\sigma_{I'}\left(\sigma_P\left(D^l\right)\right)|}{|\sigma_I\left(\sigma_P\left(D^l\right)\right)|},$$

assuming a canonical order on $\mathcal{I}$. With clauses of only length 1 we arrive at a simple conjunctive pattern, and the function resolves to the codelength function for conjunctive patterns discussed above. Note here that the codelength is the same regardless of the order assumed on the $\mathcal{I}$, which we discussed in the previous chapter.

This concludes the definition of codelength functions for transmitting the data. The overall cost of transmitting the data D given a model M is hence

$$L(D \mid M) = \Big(\sum_{P \in \mathcal{P}} L(\pi_P(D) \mid P)\Big) + \Big(\sum_{I \in \mathcal{I}} L_s(\pi_I(D) \mid P)\Big).$$

5.4.3 COST OF THE MODEL

Let us now discuss how to transmit the model M for pattern set $\mathcal{P}$. First, we transmit the number of patterns $|\mathcal{P}|$ using the MDL-optimal code for integers $L_{\mathbb{N}}(|\mathcal{P}|)$. Then, for each pattern P, we transmit the number of clauses via $L_{\mathbb{N}}(|\gamma(P)|)$. For each such clause, we transmit the items it contains using a log binomial, requiring $\log \binom{|\mathcal{I}|}{|cl|}$ bits plus the parametric complexity term $L_{pc}(|\mathcal{I}|)$. Lastly, we transmit the parametric complexities of all binomials used in the data encoding.

Combining the above, the overall model cost is

$$L(M) = L_{\mathbb{N}}(|\mathcal{P}|) + \sum_{P \in \mathcal{P}} (L_{\mathbb{N}}(|\gamma(P)|) + L_{pc}(|D^+|) + L_{pc}(|D^-|)) + \sum_{cl \in P} \left(\log \binom{|\mathcal{I}|}{|cl|} + L_{pc}(|\mathcal{I}|)\right) + \sum_{I \in \mathcal{I}} L_{pc}(|D|),$$

by which we have a lossless MDL score.

5.4.4 THE PROBLEM, FORMALLY

Based on the above, we can now formally state the problem.

Problem 6 (Minimal Label Description Problem)**.** *Given data D over $\mathcal{I}$ and partitions D^- and D^+, find model $M \in \mathcal{M}$ that minimizes the codelength $L(D) + L(D \mid M)$.*

Solving this problem through enumeration of all models is computationally infeasible, as the size of the model space is

$$|\mathcal{M}| = 2^{\sum_{i=1}^{|\mathcal{I}|} \binom{|\mathcal{I}|}{i} \times \sum_{j=1}^{i} \{^{i}_{j}\}} ,$$

where the first term in the summation specifies the number of possible item combinations in a pattern of length i, the second term counts the number of possible ways to separate them into j different clauses via the Stirling number of the second kind and the exponent is introduced as a model M consists of arbitrary combinations of patterns. The MDL score for such complex model classes does not lend itself for easy-to-exploit structure such as monotonicity. Hence, we resort to an efficient bottom-up search heuristic for discovering good models which we introduce in the next section.

5.5 PREMISE

To find good pattern sets in practice, we present PREMISE, which discovers *P*atterns *RE*constructing *MIS*classification *E*rrors, by efficiently exploring the search space in a bottom-up heuristic fashion.

5.5.1 CREATING AND MERGING PATTERNS

PREMISE starts with a model M that contains only singletons. It then iteratively improves the model by adding, extending, and merging patterns until it can not achieve more gain in the MDL score. To ease the explanation, we first introduce the setting with conjunctive patterns only. We start with an empty set of patterns M, the dataset is initially encoded only using singletons. We then search for candidate patterns that improve $L(M, D)$. These can be created in the following ways:

- *single items*: $I \in \mathcal{I}$ that improves the MDL score when transmitted separately for D^- and D^+,
- *pairs of items*: a new conjunctive pattern $\bigwedge(I_1, I_2) \in \mathcal{I} \times \mathcal{I}$,
- *patterns and items*: a new conjunctive pattern $\bigwedge(P, I)$ by merging an existing pattern $P \in M$ with an $I \in \mathcal{I}$,

- *pairs of patterns*: a new conjunctive pattern $\textcircled{\wedge}(P_1, P_2)$ obtained by merging two existing patterns $P_1, P_2 \in M$.

We can speed up the search by pruning infrequent and therewith uninteresting patterns. Pairs of items for which the transaction sets barely overlap are unlikely to compress well as conjunctive patterns. Hence, we introduce a minimum overlap threshold of 0.05 in all experiments. This straightforwardly leads to algorithm `createCandidates` given as pseudocode in Alg. 5.1, that, based on a current model M, outputs a set of possible candidate patterns that we will consider as additions to the model.

5.5.2 FILTERING NOISE

Additionally to the MDL score, we use Fisher's exact test, which we discussed in the previous chapter, as a filter for spurious candidate patterns. Fisher's exact test allows to assess statistically whether two items co-occur independently based on contingency tables. We assume the hypothesis of homogeneity; in our case that there is no difference in the pattern's probability between D^- and D^+. We can then compute the p-value for the one-sided test directly via

$$p = \sum_{i=0}^{\min(a,d)} \frac{\binom{a+b}{a-i}\binom{c+d}{c+i}}{\binom{n}{a+c}}.$$

with $c = |\sigma_P(D)|$, $a = |D| - c$, $d = |\sigma_P(D^+)|$, $b = |D^+| - d$ and $n = |D|$ for a pattern P labeled with l_+. For patterns labeled with l_-, the other tail of the distribution is tested (with a and b as well as c and d switching places). A general problem for statistical pattern mining is the lack of an appropriate multiple test correction. We here however only use the test to *filter* candidates, false positive patterns passing the test are still evaluated in terms of MDL.

5.5.3 THE PREMISE ALGORITHM

Combining the candidate generation with our MDL score, we obtain PREMISE. We give the pseudo-code in Algorithm 5.2. Starting with the empty model, we generate candidates as discussed in the previous section, and for each of those, we compute the gain in terms of MDL (line 7) as well as the pattern's p-value (line 8). We select the candidate below a significance threshold α that reaches the highest gain (line 9-11) and add it to the model. If we created the pattern through a merge, we remove its parent patterns from M. We repeat the process until no candidate provides further gain in codelength.

Algorithm 5.1: createCandidates

input : D, patterns $\mathcal{P}$ in current M, max neighbour distance K
output: Set of candidate patterns $\mathcal{P}$

/* Define $nb(I, 0) = I$ for simplicity */
$C \leftarrow \{\}$
/* Single item and its neighbours */
for $I \in \mathcal{I}$ **do**
  $A \leftarrow \{\}$
  for $k \in \{0, \ldots, K\}$ **do**
    $A \leftarrow A \cup \{nb(I, k)\}$
    $C \leftarrow C \cup \{\bigotimes(A)\}$
/* Pairs of items and their neighbours */
for $(I_1, I_2) \in \mathcal{I} \times \mathcal{I}$ **do**
  $A_1 \leftarrow \{\}$
  for $k_1 \in \{0, \ldots, K\}$ **do**
    $A_1 \leftarrow A_1 \cup \{nb(I_1, k_1)\}$
    $A_2 \leftarrow \{\}$
    for $k_2 \in \{0, \ldots, K\}$ **do**
      $A_2 \leftarrow A_2 \cup \{nb(I_2, k_2)\}$
      $C \leftarrow C \cup \{\bigwedge(\bigotimes(A_1), \bigotimes(A_2))\}$
/* Pattern + item and its neighbours */
for P in $\mathcal{P}$ **do**
  for $I \in \mathcal{I}$ **do**
    $A \leftarrow \{\}$
    for $k \in \{0, \ldots, K\}$ **do**
      $A \leftarrow A \cup \{nb(I, k)\}$
      $C \leftarrow C \cup \{\bigwedge(\gamma(P) \cup \{A\})\}$
/* Pattern + Pattern */
for $(P_1, P_2) \in \mathcal{P} \times \mathcal{P}$ **do**
  $C \leftarrow C \cup \{\bigwedge(\gamma(P_1) \cup \gamma(P_2)\}$
/* filter criteria: minimum overlap of .05, fisher's exact test */
$C \leftarrow \texttt{Filter}(C)$
return C

Algorithm 5.2: PREMISE

input : D, significance threshold α
output: approximation M of M^*

repeat
  $\Delta' \leftarrow 0$
  $M' \leftarrow M$
  $C \leftarrow$ `createCandidates`(M)
  for $P \in C$ **do**
    $\Delta \leftarrow L(D, M \oplus P) - L(D, M)$ // gain
    $p \leftarrow$ `FisherExactTest`(P) // p-value
    if $p < \alpha$ and $\Delta < \Delta'$ **then**
      $\Delta' \leftarrow \Delta$
      $M' \leftarrow M \oplus P$
  $M \leftarrow M'$
until $\Delta' = 0$
return M

5.5.4 MUTUAL EXCLUSIVITY

In our practical applications from NLP, we are interested in finding clauses expressing words that are synonyms, that reflect similar concepts, or language variations, such as $\bigwedge(\textit{which}, \bigotimes(\textit{color}, \textit{colour}))$ or $\bigotimes(\textit{could}, \textit{can})$. Such statements, however, require a richer pattern language than given by the purely conjunctive patterns discovered by the state-of-the-art. We discussed above how to identify the best model over such a richer pattern language of clauses in terms of MDL. Instead of enumerating all possible clauses exhaustively or searching for an XOR structure similar to the approach of MEXICAN, for NLP applications, we follow a more informed approach, taking into account information from pre-trained, classifier-independent word embeddings.

For the clauses of mutually exclusive items, we are mostly interested in finding words that are synonyms or that reflect similar concepts, such as $\bigotimes(\textit{color}, \textit{colour})$ or $\bigotimes(\textit{could}, \textit{can})$. Research in NLP has proposed various techniques for identifying such pairs including manually created ontologies such as WordNet (Miller, 1995) or word embeddings that are learned through co-occurrences in text and map words to vector representations. This information about related words can be used to guide the search for mutually exclusive patterns. Using such pretrained embeddings rather than deriving them from the given input data has the advantage that we are independent of the size of the input data

set, and receive reliable embeddings, which were trained on very large, domain independent text corpora.

While our approach is independent of the specific method, we have chosen FastText word embeddings trained on CommonCrawl and Wikipedia (Grave et al., 2018). In contrast to word ontologies, word embeddings have a broader vocabulary coverage. They also do not impose strict restrictions such as a particular definition of synonyms and instead reflect relatedness concepts learned from the text. FastText embeddings have the additional benefit that they use subword information, removing the issue of out-of-vocabulary words. The word embeddings are independent of the machine learning classifier we study. As measure of relatedness m between two items I_1, I_2, we use cosine similarity, i.e. $m = cos(emb(I_1), emb(I_2))$ where emb is the mapping between an item/word and its vector representation. We define $nb(I, k)$ as the k-closest neighbours $I' \in \mathcal{I}$ of I, i.e. those items for which $m(I, I')$ is the k-highest. Examples for words and their neighbours in FastText embeddings are given in Table 5.1.

Based on the information of the embedding, we derive $\bigotimes$-clauses. For each item I, we explore mutual exclusivity in its $1 \dots K$ closest neighbours, i.e. from $\bigotimes(I, nb(I, 1))$ until $\bigotimes(I, nb(I, 1), \dots, nb(I, K))$ where K is the maximum neighbourhood size. For that, we adapt the `createCandidates` algorithm so that whenever we consider merging with an item I, we also consider merging with the $\bigotimes$-clauses containing additionally the $1, 2, \dots K$ closest neighbours (see Alg. 5.1).

Since not all words have K neighbours that represent similar words, we additionally filter neighbourhoods such that $\frac{\bigcap_I \sigma_I(D)}{\bigcup_I \sigma_I(D)} < a$ and $m(I, nb(I, k)) > b_k$ for all items I in the clause, i.e. we require that their transactions barely overlap (mutual exclusivity), and that their embeddings are reasonably close. In all experiments we set $K = 5$, $a = 0.05$ and b_k to the 3rd quartile of $\{m(I, nb(I, k)) \mid I \in \mathcal{I}\}$.

In the general case for arbitrary labeled data, we could follow the MEXICAN approach to search for potential XOR structure, which however would lead to a much increased search space and hence computational costs, without any benefits for the specific applications.

5.5.5 COMPLEXITY

As before, we analyze the complexity of PREMISE in terms of the size of the model it discovers. Consider PREMISE finds k conjunctive patterns of maximum length l for a dataset with m items. Since in every round either a new singleton or pair is generated that belongs to one of the k final patterns, or two existing patterns are merged, the algorithm runs $O(k\,l)$ rounds. In each round, the dominating factor is the candidate generation, out of which there are $O(m)$ potential singletons, $O(m^2)$ pairs, and at maximum $O(k\,l)$ pattern merges, corresponding to the case

Word	**5-nearest neighbours**
photo	photograph, photos, picture, pic, pictures
color	colour, colors, purple, colored, gray
can	could, will, may, might, able
say	know, think, tell, mean, want

Table 5.1: Words and their nearest neighbours on *Visual7W*.

that all parts of the final patterns exist as singleton patterns in the current round. Hence, we get a worst case time complexity of $O(k\,l\,(k\,l + m^2))$ for Premise with conjunctive statements.

For clauses containing mutual exclusivity, for all practical applications we consider XOR statements of the c closest words in a given embedding, where c is a small constant. We hence consider $O(m\,c)$ single XOR clauses, $O((m\,c)^2)$ pairs, and at maximum $O(k\,l)$ pattern merges, where again this corresponds to the case that all parts of the final patterns exist as singleton patterns in the current round. Hence we get a worst case time complexity of $O(k\,l\,(k\,l + (m\,c)^2))$. For the general case, when searching for arbitrary AND and XOR combinations, we refer to the previous chapter.

5.6 EXPERIMENTS

We evaluate our approach on synthetic data with known ground truth, as well as on real world NLP tasks to characterise misclassifications. We compare against significant pattern mining (SPuMaNTe, Pellegrina et al., 2019), rule set mining (Grab, Fischer and Vreeken, 2019), rule lists (Classy, Proença and van Leeuwen, 2020), top-k subgroup discovery (Subgroup-Discovery, Lemmerich and Becker, 2018) and the subgroup discovery tool Cortana (Meeng and Knobbe, 2011; Duivesteijn and Thaele, 2014). As representatives of interpretable, global machine learning models we consider the rule-learner Ripper (Cohen, 1995) and patterns derived from classification trees (Tree). Due to runtime issues, we compare to the local explainability method (Anchors, Ribeiro et al., 2018) only in the NER experiment. For similar reason, we exclude SliceFinder (Chung et al., 2020), and disjunctive emerging patterns (Vimieiro, 2012); neither completed a single run within 12 hours.

Experiments were performed on an Intel i7-7700 machine with 31GB RAM running Linux. For the single-threaded C++ implementation of Premise, all synthetic data experiments finished within minutes for the moderately sized data sets, and within hours for the larger datasets with 5k and 10k items. On

the VQA datasets PREMISE finished within 20 minutes and on the NER data within 4 hours.

For the decision tree, patterns are exracted from a tree trained on the misclassification data. Each of the tree's inner nodes is a binary decision regarding the presence of an item and a pattern is the conjunctive path from the tree's root to one of its leafs. The model is trained with Gini impurity as decision criterion in the implementation from scikit-learn.

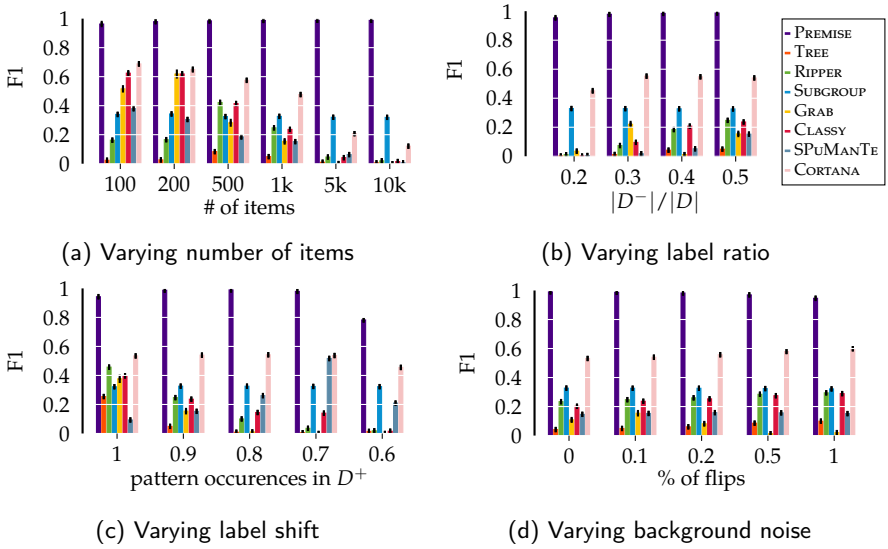
For subgroup discovery, the implementation by Lemmerich and Becker (2018) is used with depth-first search and weighted relative accuracy as quality function. The size of the result set and the maximum depth are set to the ground truth for the synthetic data and to 100. On the synthetic data, it hence has an advantage over all other approaches which would not hold in a real-world scenario. Maximum depth is set to 5 for the VQA datasets. SPuManTe is used with the authors' suggested parameters, setting its sample size to the dataset size. The subgroup discovery tool CORTANA (Meeng and Knobbe, 2011) as used by Duivesteijn and Thaele (2014) expects two numeric labels, one for ground truth and one for prediction probability. We, therefore, split the misclassification label into two labels that disagree if an instance is misclassified. As quality measure, we use negative r and we follow the authors approach of only considering subgroups that cover 1% of the data to prevent overfitting. The maximum depth is set to ground truth for the synthetic data and to 5 for VQA. The beam width is kept to 100 and the quality threshold to 0.2 following the default settings.

We modified GRAB for the task at hand by restricting the possible rule-heads to the labels only, but allowing tails over all other items. For CLASSY we used the publicly available implementation by the authors as used in the original publication. Minimum support is set to 1 and maximum rule length to the ground truth for the synthetic data and 5 for the VQA datasets.

For Visual7W and LXMERT, we use the published, pretrained models by the corresponding authors. For LXMERT, the minival version of the development set is used. For the LSTM+CNN+CRF classifier for NER, we follow the specific set-up of Hedderich et al. (2020) with English FastText embeddings. OntoNotes was split and preprocessed using the script from https://github.com/yuchenlin/OntoNotes-5.0-NER-BIO. The fine-tuning data consists of 240 instances/sentences as two patterns did not match any training data. Fine-tuning on the additional data is performed for 30 epochs. As labels, the intersection between CoNLL03 and OntoNotes is used (PER, LOC, ORG) in the BIO2 format.

Code and datasets are publicly available.[2]

[2] https://github.com/uds-lsv/premise



(a) Varying number of items

(b) Varying label ratio

(c) Varying label shift

(d) Varying background noise

Figure 5.2: *Synthetic data results.* We report mean and standard error over 10 repetitions for all synthetic data experiments. As competitors only recover fragments of patterns, the results are in terms of a soft F1 score, which also rewards the discovery of fragments.

.

5.6.1 SYNTHETIC DATA

Unless specified differently, for each of the experiments we generate a data matrix with 10 000 samples, half of which get label l_-. The set of items $\mathcal{I}$ has size 1000. We draw patterns of length $2-5$ from $\mathcal{I}$ with replacement until 50% of items are covered. For each pattern we then draw $k \sim \mathcal{N}(150, 20)$ and set the items of the pattern in $.9k$ random transactions from D^+, and $.1k$ transaction from D^- to 1. This corresponds to a typical sparsity level for pattern mining problems. Additionally, for each item that is part in a pattern, we let it occur in $k \sim \mathcal{N}(50, 20)$ random transactions from D. For all items not part of a pattern, we let them occur in $k \sim \mathcal{N}(150, 20)$ transactions from D. Lastly, we introduce background noise by flipping .1% of the entries in the data matrix at random.

A standard metric to evaluate success of a model is the F1 score – the harmonic mean between precision and recall – which for discovered pattern set P_d and ground truth pattern set P_g is defined as

$$\mathrm{F1}(P_d, P_g) = |P_d \cap P_g| / \left(|P_d \cap P_g| + \frac{1}{2} |P_d \ominus P_g| \right),$$

where $\ominus$ is the symmetric difference between two sets. As competitors only

recover fragments of patterns and hence obtain very low F1 scores, we instead report a soft F1 score that rewards also fragments. We define it as harmonic mean between a soft precision and a soft recall:

$$\text{SoftPrec}(P_d, P_g) = \sum_{p_d \in P_d} \operatorname*{argmax}_{p_g \in P_g} \frac{|p_d \cap p_g|}{|p_g|} ,$$

$$\text{SoftRec}(P_d, P_g) = \sum_{p_g \in P_g} \operatorname*{argmax}_{p_d \in P_d} \frac{|p_d \cap p_g|}{|p_g|} ,$$

$$\text{F1}(P_d, P_g) = \frac{2 * \text{SoftPrec} * \text{SoftRec}}{\text{SoftPrec} + \text{SoftRec}} .$$

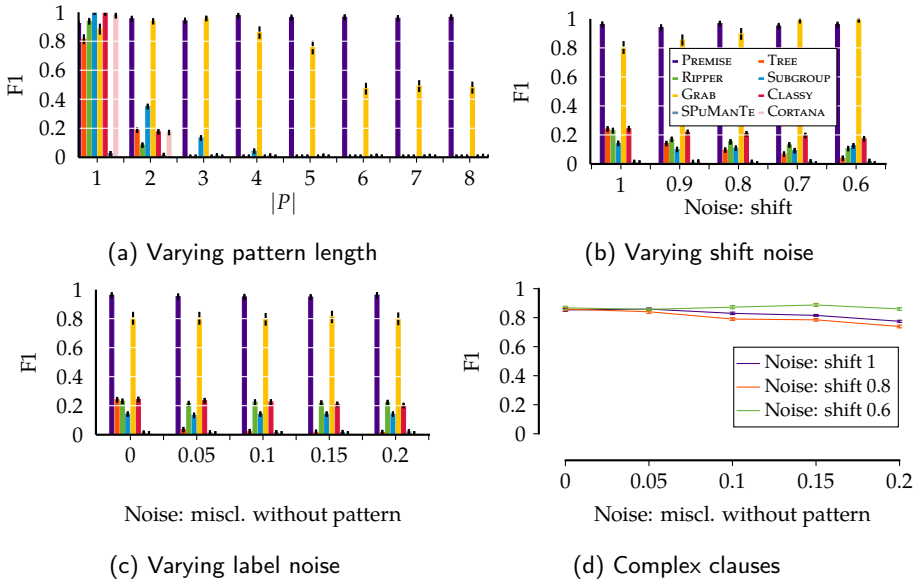
Results for experiments measured as vanilla (hard) F1 score are given in App. Fig. 9.

Scalability. First, we investigate how the different methods scale to larger item sets $\mathcal{I}$ in Fig. 5.2a. We observe that the performance of most existing methods deteriorates already for data with several hundred items, only Premise can robustly scale to data with more items. The subgroup discovery approaches of standard Subgroup-Discovery and Cortana scale also to these larger datasets, the performance, however, is in a range of around .3 respectively .2 in terms of soft F1 score. Note that we let subgroup discovery retrieve the top k patterns, where k equals the number of ground truth patterns, it hence has an advantage over all other approaches which would not hold in a real-world scenario. Subgroup discovery, however, still only yields (soft) F1 scores of around .35, whereas Premise recovers ground truth patterns with close to optimal F1 scores.

Label imbalancing. To investigate the effect of label imbalancing, we vary the proportion of transactions having label l_-, i.e. $|D^-|/|D|$, visualized in Fig. 5.2b. An imbalancing of labels is commonly encountered in real world datasets, where e.g. the number of misclassified samples makes up only a small fraction of overall samples. We again find that only Premise is consistently robust across the varying levels. Again, as opposed to all other competitors, vanilla Subgroup-Discovery and Cortana show decent performance of .35 respectively .5 soft F1 across all ratios. Premise again achieves soft F1 scores close to 1.

Varying label shift. Next, we look at label shift, the effect of patterns occurring not exclusively in one of the labels in Fig. 5.2c. This is again a likely event in real world data. We adapt the occurrence of patterns between 1, meaning the pattern exclusive occurs in one partition of the database, to .6, meaning that 60% of the transactions where a pattern occurs have one label, the others have the other label. Similar to before, we observe subgroup discovery approaches and Premise are robust to this change. For SPuMaNTe, we find the best performance for a label shift of .7 while for Ripper, Classy and Grab performance drops even for slight label shifts.

Robustness to background noise. Finally, we look at how the methods cope with background noise, by flipping a fraction of entries of the data matrix. We find that Ripper, Classy, Premise, SPuManTe, and Subgroup Discovery are robust even to large amount of noise – to the extent of how they perform without noise (see Fig. 5.2d). Grab on the other hand performs much less well when there is more noise than actual signal.



(a) Varying pattern length

(b) Varying shift noise

(c) Varying label noise

(d) Complex clauses

Figure 5.3: *Synthetic text data results.* On synthetic text data, varying the number of items per pattern (a), the amount of *shift noise* (b), and the amount of *label noise* (c), we visualize the results in terms of F1 score with respect to the ground truth for existing methods and Premise. We additionally provide the results of Premise on data containing patterns of mutual exclusive clauses for varying amounts of *shift noise* (d).

Synthetic text data. For an evaluation with known ground truth more similar to the NLP application domain, we evaluate how well all methods cope with item – or token – distributions similar to real text. To obtain such a synthetic data set with similar item/token distributions as natural language text, we derive transactions/instances from the around 3.4k sentences in the development set of the PennTreebank Corpus (Marcus et al., 1993). In particular, we draw 12 distinct patterns, for each pattern choosing items from the vocabulary tokens at random. To ensure that we introduce only new patterns into the data, we verify that none of the items in the patterns co-occur in the original data. We then insert each pattern into a random subset of the PennTreebank instances, where

the number of instances to be covered is drawn from a normal $\mathcal{N}(150, 20)$. The data contains 6k unique items.

Then, we generate four different sets of experiments. In the first set, we introduce conjunctive patterns varying pattern length of the introduced patterns between 1 and 8 without noise. In the second set of experiments we vary the amount of *shift noise*, introducing shifts of $\{0.6, 0.7, 0.8, 0.9, 1\}$, and choosing pattern length uniformly in 1 to 5. In the third set we instead change the amount *label noise*, varying in $\{0, 0.05, 0.1, 0.15, 0.2\}$. In the fourth set of experiments, we introduce patterns consisting of conjunctions of mutual exclusive itemsets. The number of clauses per pattern and the number of items for each clause is chosen uniformly at random between 1 and 5. A pattern is only added to an instance if this would not break the mutual exclusivity assumptions of all patterns. For the word neighborhoods, items in the same clause obtain embeddings located around a randomly chosen centroid. All other items obtain random embeddings. We repeat all experiments 10 times and report the F1 score as average across repetitions in Fig. 5.3.

For patterns of length 1, i.e. single items, all methods except for SPuMaNTe perform very good, with vanilla Subgroup-Discovery, Cortana, and Classy reaching perfect soft F1 scores, and Premise close to optimal F1 scores. For most considered competitors, the performance, however, deteriorates quickly for slightly longer ground truth patterns. A notable exception is Grab, which is able to retrieve longer patterns and is resistant to shift and noise in the form of non-systematic label errors, yet fails to discover long patterns with more than 5 items accurately. Premise outperforms all competitors, achieving consistently high F1 scores beyond .9. For complex patterns consisting of conjunctive clauses of disjunctions, which none of the competitors can express, we verify that Premise is able to retrieve them even in the presence of noise.

Overall, these results show that existing methods are challenged by datasets of larger scale. Furthermore, we find that the state-of-the-art breaks down when we have label imbalancing. In contrast, Premise solves both these challenges, which is important for characterising misclassifications for real world machine learning models.

5.6.2 Real data: visual question answering

Visual Question Answering (VQA) is the popular and challenging task of answering textual questions about a given image. We analyze the misclassification of Visual7W (Zhu et al., 2016) and the state-of-the-art LXMERT (Tan and Bansal, 2019), both specific architectures for different VQA tasks. Visual7W reaches 54% accuracy in 4-option multiple choice, LXMERT a validation score of 70%. Both classifiers perform far from optimal and thus serve as interesting applica-

			Premise			Tree		Ripper		Subgrp.		SpuMan.		Classy		Grab		Corta.	
Dataset	$\lvert\mathcal{I}\rvert$	$\lvert D\rvert$	k^-	k^+	$\overline{\lvert p\rvert}$	k	$\overline{\lvert p\rvert}$	k	$\overline{\lvert p\rvert}$	k	$\overline{\lvert p\rvert}$	k	$\overline{\lvert p\rvert}$	k	$\overline{\lvert p\rvert}$	k	$\overline{\lvert p\rvert}$	k	$\overline{\lvert p\rvert}$
Visual7W	2429	28032	29	26	3.38	4309	3.55	0	0.00	100	2.32	575	2.92	19	1.26	1	1	15	2.26
LXMERT	5351	25994	41	34	2.69	3371	2.71	3	3.00	100	2.52	951	3.90	36	1.28	1	1	2	3

Table 5.2: *VQA data statistics*. For the two VQA classifiers, we provide general statistics about data dimensions, and for each method the number of discovered patterns ($k = |P|$) or if applicable number of patterns explaining misclassification ($k^- = |P^-|$), respectively correct classification ($k^+ = |P^+|$) and the average pattern length $\overline{|p|}$.

tions for describing (misclassification) labels. We derive misclassification data sets from applying the classifiers to the development sets.

In Tab. 5.2 we give statistics about the data and retrieved patterns. Both the tree based method and SPuMaNTe retrieve many hundred or thousand patterns making it difficult to interpret the results. Furthermore, we know from the previous experiments that these methods find thousands of patterns even when there exist only few ground truth patterns. Subgroup-Discovery requires the user to specify the number of patterns a-priori, which is not known. The discovered patterns are highly redundant with often ten or more patterns expressing the same cause for misclassification. It is thus hard to get a full description of what goes wrong, it lacks the power of set mining approaches that evaluate patterns *together*. Cortana filters more strongly, the discovered patterns are, however, still redundant. Most patterns found by Classy consist of only one token, Grab and Ripper fail to retrieve meaningful results.In App. Tab. 5.2, we provide further statistics about the data and retrieved patterns.

In Tab. 5.3 we provide an excerpt of the patterns found by Premise. We can clearly see the advantage of the richer pattern language, allowing to find patterns with related concepts such as $\owedge(\textit{what}, \otimes(\textit{color, colors, colour}))$. Generally, the patterns found by Premise highlight different types of wrongly answered questions, including counting questions, identification of objects and their colors, spatial reasoning, and higher reasoning tasks like reading signs. Furthermore, Premise retrieves both frequent patterns, such as $\owedge(\textit{how, many})$ and rare patterns such as $\owedge(\textit{on, wall, hanging})$.

Premise also discovers patterns that are biased towards correct classification. These can indicate issues with the dataset. For instance, $\owedge(\textit{who, took}, \otimes(\textit{photo, picture, pic, photos, photograph}))$, although a difficult question, is nearly always answered by "photographer" and thus easy to learn. Another problematic question is indicated by the pattern $\owedge(\textit{clock, time})$, where usually the answer is "UNK", the actual time being replaced with the unknown word token by the limited vocabulary of Visual7W. The pattern hence indicates a setting where the VQA classifier undeservedly gets a good score.

By adding additional information as items to each instance, it is possible to gain further insights. Appending the correct output to each instance, we observe for the question when the picture was taken two different trends. On the one hand, the discovered pattern $\owedge(\textit{when}, \otimes(\textit{daytime, nighttime}))$ is associated with correct classification, the pattern $\owedge(\textit{when}, \otimes(\textit{evening, morning, afternoon, lunchtime}))$, on the other hand, points towards misclassification. This is intuitively consistent as the answers "daytime" and "nighttime" are easier to choose based on a picture.

We observe in the discovered patterns that the Visual7W and LXMERT classifiers share certain issues, like the counting questions. However, no patterns regarding color or spatial position are retrieved. This might indicate that the

Pattern	Example
UNK	how are the UNK covered
⋀(*how, many*)	how many elephants are there
⋀(*what,* ⊗(*color,colors, colour*)	what color is the bench
⋀(*on, top, of*)	what is on the top of the cake
⋀(*left, to*)	what can be seen to the left
⋀(*on, wall, hanging*)	what is hanging on the wall
⋀(*how, does, look*)	how does the woman look
⋀(*what, does,* ⊗(*say, like think, know, want*))	what does the sign say

(a) Visual7W

Pattern	Example
⋀(*How, many*)	How many kites are flying?
⋀(*hanging, from*)	What is hanging from a hook?
⋀(⊗(*kind, sort*), of)	What kind of birds are these?
⋀(⊗(*would, could, might, can*), *you*)	How would you describe the decor?
⋀(*name, of*)	What is the name of this restaurant?
number	What is the pitchers number?
⊗(*letter, letters*)	What letter appears on the box?
⋀(*How, much,*⊗(*cost,costs*))	How much does the fruit cost?

(b) LXMERT

Table 5.3: *VQA example patterns.* Our method discovers meaningful and easily interpretable patterns. For Visual7W (left) and LXMERT (right), we show a subset of the patterns highlighting different reasons for misclassification along with examples from the corresponding datasets.

more recent LXMERT classifier can handle these question types better.

5.6.3 REAL DATA: NAMED ENTITY RECOGNITION

A machine learning classifier might perform well during development, its performance when deployed "in the wild" however is often much worse. Understanding the difference is important for being able to improve the classifier. To evaluate on a different task, we chose Named Entity Recognition (NER). This is a sequence labeling task that identifies entities like persons, locations and organizations in text and it is the basis for many more advanced tasks such as text search or virtual assistants. Here, we investigate the popular

LSTM+CNN+CRF architecture (Ma and Hovy, 2016) for NER. The classifier is trained on the standard NER dataset CoNLL03, where it achieves a good performance (F1-score of 0.93). On OntoNotes, a dataset covering a wider range of topics, the performance drops to 0.61 F1 on the development set. We take each sentence as one instance and label it as misclassification if at least one token in the sentence is predicted incorrectly. This results in a misclassification dataset with about 16k instances and 23k unique items.

Anchor (Ribeiro et al., 2018) allows to obtain conjunctive patterns to explain NLP instances locally. It took, however, several days to analyze all misclassifications on modern GPU hardware due to the necessary, repeated queries to the NER classifier. Anchor finds 4.1k patterns with many redundant and overly long and specific patterns. As expected from a local method, the patterns are highly specific and thus identify problems of the model for particular instances rather than identifying the general (global) issues that the model has. Premise retrieves a concise set of 190 patterns. An example is $\bigwedge$(*-LRB-*, *-RRB-*) that indicates different preprocessing of the text, where *-LRB-* is an alternative form for the opening bracket, which is specific to the OntoNotes data, and thus should be properly handled by the NER classifier. Patterns also indicate problems with differing labeling conventions. We find the single item pattern *'s* because this token is included as part of the entity in OntoNotes (e.g. "Samuel Alito 's") but excluded in CoNLL03. Similar differences can be seen for other patterns like $\bigwedge$(*Wall, Street*) which have differing labels in the two datasets. We also find issues with the OntoNotes dataset itself. It contains bible excerpts which are not labeled at all. We discover this via several found patterns that describe this domain containing *God*, *Jesus*, and *Samuel*.

As there is no ground truth available and to empirically validate that the found patterns affect the classifier's performance, we select the top 50 patterns according to gain in MDL and for each pattern sample 5 sentences containing it uniformly at random from the OntoNotes training data. The CoNLL03 classifier is then fine-tuned on this data. Sampling and fine-tuning is repeated 20 times with different seeds. Using the pattern-guided data, the performance is improved to 0.67 mean F1 score (SE 0.003) compared to sampling fully at random where only a small improvement to 0.62 (SE 0.005) is achieved. This shows that the patterns discovered by Premise provide actionable insights into how a classifier can be improved.

5.7 Discussion and Conclusion

We considered the problem of finding interpretable and succinct descriptions of a given label, and proposed to discover succinct pattern sets to describe the labels based on the Minimum Description Length Principle. To solve this

formulation in practice, we proposed an efficient bottom-up heuristic Premise.

Experiments showed that Premise provides concise and interpretable descriptions of labeled data. On synthetic data we found that the state-of-the-art methods across different fields related to supervised pattern mining, including subgroup discovery, emerging pattern mining, statistical pattern mining, rule mining, discovery of rule lists, and tree based classification, all have severe difficulties finding the ground truth pattern set, while Premise accurately retrieves it. It thus renders itself as the most suitable method for challenging real world applications revolving around characterising misclassifications of Natural Language Processing models. For such tasks, the labels are inherently imbalanced and the sets of items – in this case tokens – is large. Besides, to capture the structures of word associations, we need to consider a richer pattern language capturing mutual exclusiveness, which out of existing methods only Premise is able to express.

On two models for Visual Question Answering, we set for characterising their misclassifications. While some of the competing methods retrieved reasonable explanations, these were highly redundant and barely interpretable for human experts. Moreover, important concepts, such as patterns that are similar across related words or synonyms, were completely missed. Premise, on the other hand, discovered succinct sets of patterns that provide interesting characterizations, revealing that models struggle with counting, spatial orientation, reading, and identifies shortcomings in training data. For a popular Named Entity Recognition classifier, we consider a model applied to text of a different source and characterize the resulting classification errors. Compared to the state-of-the-art local explanation method Anchors, Premise retrieves a more succinct set of patterns in less time and we also show that the obtained insights are actionable.

In summary, our method showed to be the only approach that scales well to data typical in real world problem settings, while at the same time being robust to noise, and label imbalance. With these abilities, combined with a more expressive pattern language compared to the state-of-the-art, Premise enables to get a better understanding about the general issues of classification models rather than instance specific (local) issues. It hence fills the gap of a robust approach to describe labels in terms of human-interpretable patterns, suited to take on problems such as characterizing misclassifications of large-scale machine learning models.

While our approach scales already to tens of thousands of features, it makes for engaging future work to go beyond that, towards hundreds of thousands of features or to extend the work on characterizing misclassifications incorporating elements of the classifier itself, such as neuron activations, leveraging methodological insights gained from ExplaiNN. It, however, turns out that finding even classical conjunctive patterns sets on such a scale is extremely challeng-

ing, with contemporary work limited to (at most) a few thousands of features, the methods discussed in the previous chapters to around twenty-thousand features. These limitations are largely due to the combinatorial search that is commonly employed to explore the discrete search space of patterns. To scale to hundreds of thousands of features thus requires a fundamentally different approach. In the next chapter we provide a solution by linking the learning of discrete variables to continuous optimization in the context of pattern mining.

DIFFERENTIABLE PATTERN SET MINING

6

6.1 INTRODUCTION

Modern approaches to pattern mining are designed as a model selection problem to overcome the drawbacks of the traditional approaches that swamp the analyst with extremely many results, most of which are redundant or spurious. This model selection explicitly asks for a small *set* of patterns that together generalize the data well. This solves one problem, but creates another: the search space for pattern sets is even larger than that of patterns alone – doubly exponential in the number of features – and does not exhibit any structure that permits efficient search. As a result, existing methods rely on heuristics, and are applicable only to modestly sized data of at most a few thousand features, by which modern biological applications, such as genome-wide association studies or single-cell sequencing data with hundreds of thousands of features, are far out of reach.

In this paper we propose a radically different strategy to pattern set mining that scales extremely well in both the number of samples n and number of features m, naturally handles noise, and copes equally well with sparse and dense data. We achieve this by taking a *differentiable* rather than a combinatorial approach. Our key idea is to learn a special kind of interpretable neural autoencoder for binary data, which we refer to as BINAPS, short for Binary Pattern Networks. These networks consist of two linear layers – the encoding hidden

This chapter is based on Fischer and Vreeken (2021).

and the decoding output layer – with continuous weights, sharing weights between encoder and decoder. By using a novel type of binary activation function and binarizing weights during each forward pass, we can interpret the neurons in the hidden layer as 'classic' conjunctive patterns. Here, we propose a reconstruction loss that properly accounts for dense respectively sparse data, and as such the networks are equally well applicable to dense biological datasets as well as classical sparse transaction data.

One key benefit of our formulation is that it naturally allows for discovering noisy patterns: the network is rewarded for reconstructing the data, and hence automatically learns how much and which parts need to occur for a pattern to be considered as 'present'. Similar to other pattern set mining approaches, such as tiling (Geerts et al., 2004) or boolean matrix factorization (Miettinen, 2010; Miettinen and Vreeken, 2011), deciding the optimal size of the pattern set – the size of the hidden layer – is NP-hard. We however also show that in practice this is not a problem at all: initialized with a sufficiently large capacity, our networks drive the weights of edges towards 'surplus' neurons to zero and can so find an (almost) optimal number of patterns, even when initialized with as many hidden neurons as there are features. The overall most important benefit of BinaPs in contrast to existing methods, is however its massive scalability: the differentiable formulation is not only much easier to optimize, but also allows us to leverage the power of modern GPUs.

Evaluating BinaPs against the state-of-the-art on synthetic data, we show that it accurately retrieves the ground truth, and is robust to noise. BinaPs is the only method applicable on datasets with truly large n and m, on which we confirm that it discovers meaningful patterns that provide non-trivial insight. It is also applicable to moderately sized data, on which we show that it performs at least as well as the state-of-the-art. As a case study, we consider a biological dataset of over two hundred thousand features on which BinaPs are able to mine patterns in mere minutes, recovering structure which can be confirmed by the literature, and generating new insights that could be leveraged by experts, such as potential roles of genes for which the functions are so far unknown.

6.2 Related Work

We reviewed pattern mining literature in the first chapter, emphasizing the development of modern approaches as a model selection problem, in which patterns are evaluated *together* using a global score, e.g. based on MDL or maximum entropy (Vreeken et al., 2011; Smets and Vreeken, 2012; Mampaey et al., 2012; Dalleiger and Vreeken, 2020). While these methods retrieve succinct and interpretable summarizations of the data as opposed to traditional approaches based on frequency, this comes at the cost of a twice exponential search space

that does not expose any easy to exploit structure. Hence, heuristic algorithms are employed that add patterns one by one in a bottom-up fashion, and thus can get stuck in local minima. Furthermore, despite efficient heuristics these approaches still do not scale to modern data such as retail databases or many biological problem settings.

With the advent of Deep Learning, efficient GPU-based implementations to train neural networks experienced wide-spread adoption. In unsupervised settings, autoencoders can be trained to yield succinct representations of complex datasets, also providing local information in analogy to patterns (Kramer, 1991; Masci et al., 2011; Kingma and Welling, 2014). Such networks are, however, inherently hard to interpret – especially with respect to how the structures that the network actually learned link to the given input – and lack the clear interpretability of patterns barring their use for exploration. For low resource and embedded devices, network architectures with binarized $\{-1, 1\}$ weights and activations have been proposed, thus making model storage and application much more memory efficient (Saad and Marom, 1990; Courbariaux et al., 2015; Rastegari et al., 2016; Hubara et al., 2016; Li and Liu, 2016; Deng et al., 2018), for more information we refer to the review of Simons and Lee (2019). While these works provide discrete-valued connection between neurons and input, these networks have no incentive to learn easy-to-interpret structures reflecting associations between input features. Preliminary experiments confirm this: these networks do not learn any such relation between weights, neurons, and actual ground truth patterns. Inspired by the link between continuous optimization of an objective and learning discrete variables, we here propose to learn interpretable patterns using a novel constrained network architecture.

6.3 THEORY

In this section, we propose BINAPS, a novel type of binarized neural autoencoder capable of learning pattern sets. We start by briefly introducing notation and then provide an informal overview of how BINAPS work. After introducing them formally, we discuss practical considerations, and provide theoretical analysis.

6.3.1 NOTATION

To be able to easily connect classical notation of machine learning and neural networks with pattern mining, we will redefine databases D in terms of matrix notation. We now consider binary input data $D \in \{0, 1\}^{n \times m}$ of n samples and m features. A sample $s_i \in D$ denotes the ith row in the data matrix $D[i]$. We denote the value of feature $j \in \{1, \ldots, m\}$ for the ith sample by $D[i, j]$. We are

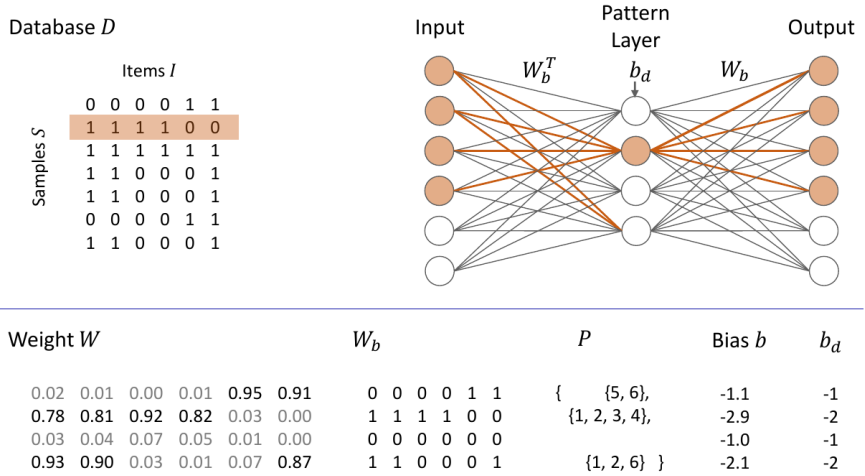


Figure 6.1: *Example database and* BINAPS. A binary database D is given in the top left, a binary pattern network with pattern layer size $k = 4$ on the right. Example continuous weights W and bias b used during backpropagation, and their *binarized*, respectively *discretized* counterparts W_b and b_d used during the forward pass, are given at the bottom, alongside the pattern set P that we can directly derive from these weights. Neuron activations (orange) for the orange sample in D are given according to the example binarized weights and bias.

interested to find a set of patterns P, where each pattern $p \in P$ is a set of feature indices $p \subset \{1, 2, \ldots, m\}$ representing feature co-occurrences. To learn good sets of patterns P, we propose a binarized neural network architecture, where we generally denote W for a weight matrix, W_b for its binarized version, b for a bias, and b_d for its discretized version. Activation functions are denoted as λ, functions applying a part of the network to an input, e.g. the encoding layer, as f. To ease notation, whenever we apply a univariate function to a vector, we apply it to every entry in the vector. For instance, an activation function $\lambda : \mathbb{R} \to \{0, 1\}$ applied to $x \in \mathbb{R}^3$ yields $\lambda(x) \in \{0, 1\}^3$. Partial derivatives of a function f with respect to parameter x are written as $\frac{\partial f}{\partial x}$, matrices M are generally denoted with capital letters, and vectors v with lower case letters.

6.3.2 THE IDEA IN A NUTSHELL

For a given binary database, we aim to find that set of patterns P that together succinctly describe the data. That is, a good set of patterns should cover the database non-redundantly while minimizing the reconstruction loss if we re-

construct the database from P alone. For continuous and unstructured data, autoencoder proved to be a successful tool to capture the main structure in the data by minimizing reconstruction loss. An autoencoder is a neural network consisting of task-specific encoding layers that end in an embedding layer, and a decoder – which is most often symmetric to the encoder – to reconstruct the input from the embedding layer. The embedding layer is usually small compared to the input layer, thus imposing an information bottleneck and forcing the network to learn relevant and shared structure between inputs. Neural networks are, however, inherently hard to understand for a human, as connections between input and neurons are non-symbolic and often non-linear.

Here, we propose a novel kind of neural autoencoder, where weights and activations are taking values in $\{0, 1\}$ during the forward pass. To learn in small noisy steps during backpropagation, for training we use continuous versions of the weights, optimizing reconstruction loss with respect to these continuous weights. The autoencoder consists of one linear hidden layer – the pattern layer – and one linear output layer. For each neuron in the hidden layer, incoming binary weights indicate whether an input item is part of the encoded pattern, e.g. binarized weight $W_b[i, j]$ means that input item j is part of the pattern given by hidden neuron i. As such, each neuron in the hidden layer corresponds to a pattern p, while all neurons together correspond to the pattern set P.

We construct binarized versions of the weights that we use in the forward pass through the autoencoder, i.e. to reconstruct the input. To ensure that the hidden neurons correspond to actual patterns, and that such patterns are interpretable, two additional constraints are introduced. First, we introduce a discrete negative bias to the pattern layer to make sure that a minimum number of items is required to be present in the input for the neuron to fire – respectively the pattern to hold. Second, we "mirror" the weight matrix, such that the weight of the decoding layer is the weight of the encoding layer transposed. This ensures that the input/output relation is fixed, thus ensures that if a neuron fires – meaning a pattern over a set of input neurons holds – it activates exactly those output neurons that are part of the pattern. We give an example network and toy database in Fig. 6.1.

6.3.3 BINAPS – BINARY PATTERN NETWORKS

We define Binary Pattern Networks (BinaPs) as autoencoder with two linear layers. Each layer has binary weights and binary activations, and a continuous version of weights and bias for backpropagation. We first formally define the functions that this stack of layers encodes given binary weights – corresponding to the forward pass of the network – and then provide the derivatives of a loss function with respect to the continuous version of the weights – corresponding

to the backward pass – and show how we can binarize the continuous weights for the next iteration.

The forward pass

A binary linear layer is defined as

$$f_{W_b}(x) = xW_b^\intercal \ ,$$

with input x and binarized weights W_b. For sample $s \in \{0,1\}^m$, in the forward pass, the function of the encoding layer takes the form

$$f_E(s) = \lambda_E(f_{W_b}(s)) \ ,$$

where $W_b \in \{0,1\}^{k \times m}$ is the binary weight matrix for the hidden layer of size k, and $\lambda_E : \mathbb{R} \to \{0,1\}$ is a binary activation function.

The function of the decoding layer, taking as input the result of the encoding layer $y \in \{0,1\}^k$, is defined as

$$f_D(y) = \lambda_D(f_{W_b^\intercal}(y)) \ ,$$

where W_b is the same weight matrix as in the encoding layer.

The activation function consists of two parts, a learnable bias term b that allows to train how many items of this pattern have to be present in the input for the neuron to fire, and a binarization function to produce the output signal. First, we define the clamping function, which clamps value x to boundaries a, b

$$\text{clamp}(x, a, b) = \begin{cases} a & \text{if } x < a \ , \\ x & \text{if } a \leq x \leq b \ , \\ b & \text{if } b < x \ . \end{cases}$$

The activation of the encoding layer is then given as

$$\lambda_E(x) = \text{round}(\text{clamp}(x + b_d, 0, 1)) \ ,$$

where $x \in \mathbb{N}^k$, and $b_d \in \{-\infty, \ldots, -2, -1\}^k$ is the discretized bias parameter. The activation function of the decoding layer is $\lambda_D(x) = \text{round}(\text{clamp}(x, 0, 1))$, which does not involve a bias term as we want to reconstruct an item if it appears in any pattern. This concludes the definitions of all components of BinaPs in a forward pass, which is thus computed by $f_D(f_E(s))$ for given input s.

The backward pass

To learn a good model for a given dataset D, our aim is to minimize the reconstruction loss across all samples $s \in D$

$$L(D; W, b) = \sum_{s \in D} ||f_D(f_E(s)) - s|| \ .$$

Given the loss term for a given batch of samples, we can then compute derivatives with respect to the continuous weights W and biases b, and backpropagate through the network. Networks are trained using gradient descent, for which the derivatives with respect to the parameters can be obtained using the chain rule. For the linear layer, the derivative with respect to the weights W, input x, and incoming gradient flow g_o is, similar to vanilla neural networks, given as

$$\frac{\partial f_{W_b}}{\partial W} = g_o^\intercal x \ , \qquad \frac{\partial f_{W_b}}{\partial x} = g_o W \ .$$

The activation functions λ in each layer resemble a step function, hence there does not exist an analytical solution to their derivative. For the decoder activation λ_D we use a version of the straight-through-estimator $\frac{\partial \lambda_D}{\partial x} = \mathbb{1} g_o$ as suggested by Bengio et al. (2013).

For the hidden layer, the straight-through-estimator fails, as wrongly predicted items backpropagate a negative gradient to weights through the activation, even though the neuron was not active. In other words, patterns get penalized for wrong reconstruction, regardless of whether they were actually taking part in the reconstruction. Thus, we propose the *gated* straight-through-estimator, which distinguishes the two cases of whether λ_E was activated on input x or not. For given input x and incoming gradient flow g_o, we define gradients with respect to bias b and input x as

$$\frac{\partial \lambda_E}{\partial x} = \begin{cases} g_o & \text{if } \lambda_E(x) = 1 \\ 0 & \text{if } \lambda_E(x) = 0 \end{cases} , \qquad \frac{\partial \lambda_E}{\partial b} = \begin{cases} g_o & \text{if } \lambda_E(x) = 1 \\ \max(0, g_o) & \text{if } \lambda_E(x) = 0 \end{cases} .$$

With the partial derivatives defined, the chain rule allows us to propagate gradients through the network and update the continuous parameters W and b accordingly. After backpropagation, we clamp the weights to be in range $[0, 1]$, which allow us to use a stochastic binarization. We clamp the bias to be at maximum -1, which means that the bias acts as a learnable threshold for a minimum number of items to be present for a pattern to hold.

From continuous to discrete.

In the next forward pass, we then binarize respectively discretize the weight and bias parameter. As the weights are in range $[0, 1]$, we can use a stochastic interpretation and treat them as a Bernoulli variable, the binarization thus becomes a draw from a Bernoulli $\mathcal{B}$,

$$W_b[i, j] = \mathcal{B}(W[i, j]) \ .$$

Hence, the binary versions of the weights change probabilistically in every iteration. For the discretization of the bias we ceil the values $b_d[i] = \lceil b[i] \rceil$.

The algorithm.

With all information in place, we are able to train a binary pattern network, for which we provide pseudocode as Alg. 6.1. In practice, we use batch learning, where line 3 is replaced by an appropriate batch slicing of the database, and all operations become the corresponding tensor operations. The pattern set P encoded by the network is given by a binarization of the final weight matrix $W_P = \text{round}(W)$, where row i encodes a pattern that contains each item j for which $W_P[i, j] = 1$.

Reconstruction loss for transaction data.

Binary matrices, and in particular sparse transaction databases, impose an additional challenge when optimizing the loss. In analogy to a classification problem with imbalanced class labels, where there is an intrinsic bias towards correctly classifying the overrepresented class, sparsity introduces a huge imbalance when it comes to reconstruction as 1s are highly underrepresented yet most important for mining patterns. In extreme cases, such as very sparse data usually given by supermarket transaction data, a model achieves very low reconstruction loss by predicting 0s everywhere – similar to always predicting the overrepresented class in the classification analogy. We thus propose a sparsity dependent reconstruction loss, which for given sample $D[i]$, and reconstruction z_i, is

$$L_\alpha(D[i]; W, b) = \sum_{j \in [1,m]} \big((1 - D[i, j])\alpha + D[i, j](1 - \alpha)\big)|z_{ij} - D[i, j]| \ ,$$

where $\alpha = \frac{\#1s}{\#1s + \#0s}$ is the sparsity of the data, and z_{ij} is the reconstructed feature j for sample i. The loss function thus avoids aforementioned intrinsic biases by properly modeling sparsity in the reconstruction.

Algorithm 6.1: BINAPs training

input : dataset D, initial BINAPs (W^1, b^1), number of epochs $e_{\max}$, batch size l
output : pattern set P

$t \leftarrow 0$ // Step
for $e = 1 \dots e_{\max}$ **do** // For each epoch
  for $i = 1 \dots n$ **do** // For each sample
    // **Forward pass**
    $W_b^t \leftarrow \mathcal{B}(W^t)$ // Binarize weights
    $b_d^t \leftarrow \lceil b^t \rceil$ // Discretize bias
    $y_i \leftarrow \text{round}(\text{clamp}(D[i](W_b^t)^\intercal + b_d^t, 0, 1))$ // f_E
    $z_i \leftarrow \text{round}(\text{clamp}(y_i W_b^t, 0, 1))$ // f_D
    $L(D[i]; W, b)$ // Compute loss
    // **Backward pass**
    $g_W \leftarrow \frac{\partial L}{\partial W}$ // Weight gradient
    $g_b \leftarrow \frac{\partial L}{\partial b}$ // Bias gradient
    $W^{t+1} \leftarrow \text{update}(W^t, g_W)$ // Optimizer step
    $b^{t+1} \leftarrow \text{update}(b^t, g_b)$
    $t \leftarrow t + 1$
$P = \{\{j \mid \text{round}(W^t[i, j]) = 1\} \mid i \in 1 \dots k\}$ // Pattern set
return P

ESCAPING POOR LOCAL MINIMA.

The probabilistic nature of our binarization allows us to escape poor local minima due to the stochasticity introduced by drawing the binarized weights each round. By bounding the continuous version of the weights by a very small but positive value from below, we further ensure that with low probability, every item could potentially still be learned for a neuron. In particular, we adapt the clamping of weights after updating the weights during backpropagation in iteration t such that

$$W^{t+1} = \text{clamp}(W^{t+1}, 1/m, 1) \,,$$

where the minimum $1/m$ is chosen such that on expectation 1 out of m items is randomly assigned to the pattern of a neuron[1], independent of data dimensions. This helps to escape poor local minima by e.g. preventing neurons from dying – meaning that the gradient of corresponding weights and biases would always be zero.

Initialization.

To initialize the model, we can again take advantage of the stochasticity of the weights. By setting the initial weights $W_{ij}^0 = 1/m$ for data of size m, we have uniform chances of setting any particular weight to 1, and have an initialization that is insensitive to the number of features m, thus allowing to learn even small patterns properly early on in the optimization. To enforce learning of proper patterns at the beginning of the optimization, we set all bias terms in the encoding layer to -1, which means that at least two items have to be present for a neuron to fire.

Size of hidden layer.

So far, we assumed the architecture, and in particular the size of the hidden layer c, to be fixed. This hyperparameter, which we call *capacity*, corresponds to the maximum size a retrieved pattern set can attain. Existing work, often assumes the size of the optimal pattern set k^* to be given, as deciding k^* for BMF (Miettinen et al., 2008) and related problems such as minimum tiling of databases is NP-hard (Geerts et al., 2004), where the optimal pattern set is the smallest set of patterns that together reconstruct the database. Similarly, to decide if the hidden layer capacity c corresponds to the size of an optimal pattern set, is NP-hard.

Theorem 1 (Estimating hidden layer size c^* is NP-hard)**.** *For a given database D consisting of patterns P, estimating the smallest hidden layer size $c^* = |P|$ that would result in perfect reconstruction, is NP-hard.*

Proof. We prove the theorem by introducing the concept of bipartite dimensions of a graph, which is known to be NP-hard, on which we reduce our problem.

Definition 1 (Bipartite Dimension)**.** *For a bipartite graph $G = (V_1 \cup V_2, E)$, the bipartite dimension is the minimum number of bicliques between V_1 and V_2 required to cover all edges E.*

Lemma 2. *Deciding the bipartite dimension for a graph G is NP-hard* (*Garey and Johnson, 2000, GT18*).

[1] After initialization, each incoming edge is is a Bernoulli random variable with same probability, hence the number of items in the pattern follows a Binomial.

For a given database D of items $\mathcal{I}$ and sample identifier $\mathcal{S} = \{1, 2, \ldots, n\}$, the binary database matrix corresponds to an adjacency matrix of the (undirected) bipartite graph spanned by the two node sets of all items and all sample identifiers, with an item and an identifier having an edge iff the item occurs in the corresponding sample. More formally, we define $G_D = (\mathcal{I} \cup \mathcal{S}, \{(i, j) \in \mathcal{I} \times \mathcal{S} \mid D[i, j] = 1\})$. The perfect reconstruction of the database D with minimal number of patterns thus corresponds to the minimal number of bicliques of G_D, with each biclique corresponding to a pattern. Hence, the minimal number of patterns k^* and thereby the hidden layer size c^* required to achieve perfect reconstruction corresponds to the bipartite dimension of the underlying graph, which is NP hard to decide (Lemma 2). □

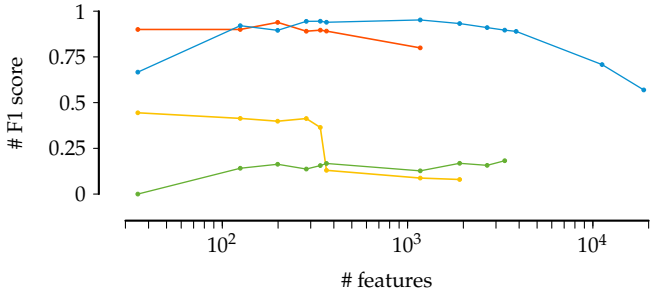
The good news is, however, that binary pattern networks – as opposed to other approaches such as BMF – are robust to optimistic estimates of k^*, i.e. the capacity c of the network only serves as an upper bound for the number of patterns it will retrieve. In practice, we can set e.g. $c = m$ and the network simply sets incoming weights to surplus neurons to 0, thus shrinking the pattern set to an almost optimal size. It is important to note that by setting c to the number of items, the network does not memorize the input, as it is forced to learn proper patterns due to the bias term $b \leq -1$.

Complexity.

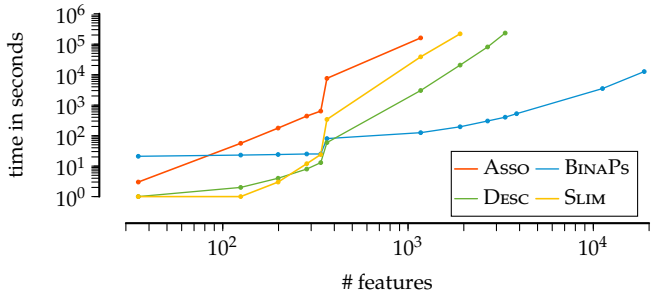
The complexity of learning binary pattern networks is the same as for vanilla neural networks, which is $O(ntm^2c)$ for t iterations and data of n samples and m items and a BinaPs of capacity c. This term is dominated by the matrix multiplication, for simplicity we assumed naive matrix multiplication. As the search space for pattern sets is twice exponential in the number of features and usually does not lend itself for easy to exploit structure, a low-degree polynomial complexity is great news. State-of-the-art approaches based on e.g. information theory do not provide differentiable objectives, the major issue being the discreteness of the search space, hence they resort to heuristic algorithms. Here, by having a link between the discrete search space and the continuous loss, we circumvent this problem, thus resulting in a differentiable objective optimizable in polynomial time. In practice, GPUs drastically speed up the optimization even further, allowing us to scale up to and beyond hundreds of thousands of features.

6.4 EXPERIMENTS

Here, we empirically evaluate BinaPs on both synthetic as well as real world data. For that, we implemented our approach in `PyTorch` and compare to state



(a) F1 score on scale data (higher is better).



(b) Runtime on scale data (lower is better).

Figure 6.2: BinaPs *is scalable*. For synthetic data with known ground truth of varying number of features and planted patterns, we show the F1 score (a) and the runtime (b) for all methods. Experiments were aborted when exceeding 3 days of runtime.

of the art pattern mining methods that leverage different objectives and approaches. We use the publicly available implementations of all methods, and make our implementation available.[2] All experiments of existing methods were carried out on Intel Xeon E5-2643 v3 machines with 256GB RAM running Linux, BinaPs were trained on an NVIDIA A100-SXM4 GPU with 40GB memory. Existing methods supporting multithreading were executed in parallel on 16 cores. In particular, we compare to Asso, which is based on boolean matrix factorization, and use an automated rank selection approach (Miettinen and Vreeken, 2011), Slim, an information theoretic pattern set miner based on the Minimum Description Length principle (Smets and Vreeken, 2012), and Desc, which mines pattern sets using maximum entropy modeling (Dalleiger and Vreeken, 2020). Preliminary experiments showed that vanilla binary networks (Courbariaux et al., 2015), do not yield any evident easy-to-interpret relation between

[2] http://eda.mmci.uni-saarland.de/prj/binaps/

weights, neurons, and ground truth patterns. Individual experiments were stopped if taking more than 3 days. Asso is given the maximum k, and BinaPs capacity c, equal to the number of features m for all experiments. Despite being large overestimates of the actual pattern set sizes, both methods show to be robust to this choice, hence we did not optimise these parameters further. For all methods, we optimised hyperparamters for each dataset separately, for details we refer to App. A.5.1.

6.4.1 RECOVERING GROUND TRUTH

To evaluate all methods with respect to scalability and robustness to noise, we first generate synthetic data with known ground truth. We consider data with various number of planted patterns, where each pattern is assigned 2 to 10 features at random. Futhermore, we draw for each pattern a random subset of the samples of size drawn from $\mathcal{N}(.05n, .005n)$. Here, .05 corresponds to the mean density of patterns, similar to sparse real world data, and n is the number of samples. In a first set of experiments, we vary the number of samples n, to show that BinaPs can handle small data well. In a second set of experiments, we vary the number of planted patterns, comparing BinaPs with existing methods with respect to scalability. In a third set of experiments, we introduce varying amounts of noise, to evaluate how well the approaches can cope with noise. To evaluate how well the planted pattern sets are retrieved, we consider F1 – the harmonic mean between precision and recall – which is defined as

$$\mathrm{F1}(P_d, P_g) = \frac{|P_d \cap P_g|}{|P_d \cap P_g| + \frac{1}{2}|P_d \ominus P_g|},$$

measuring how well discovered pattern set P_d matches the ground truth P_g, with $\ominus$ the symmetric difference between two sets. An intersection of two pattern sets P, P' is defined as true matches between individual patterns, i.e. $P \cap P' = \{p \mid p \in P \wedge p \in P'\}$.

Small n. We start by confirming whether BinaPs can cope with *small* data. To this end we generate synthetic data density and distribution as above, planting 100 different patterns in $n = \{1000, 1100, \ldots, 10\,000\}$ samples, flipping 0.1% of data entries at random, corresponding to a signal to noise ratio of 20. Although 1000 samples may not sound very small, the probability for creating spurious but statistically significant co-occurrences for just one pattern when planting 100 patterns is already 23% (see App. A.5.2 for the derivation). In other words, considering fewer than 1000 samples does not make sense.

We report the results in terms of the F1 score in Fig. 6.3. We see that BinaPs robustly retrieves nearly all patterns even when the data consists of just a few thousand samples, already ably recovers 75% of the patterns completely

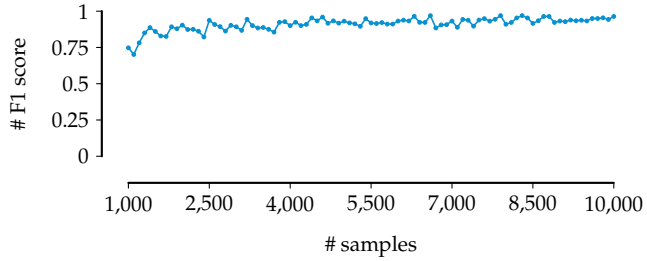


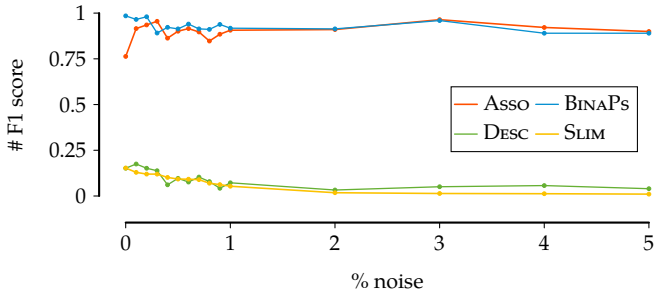
Figure 6.3: BINAPS *is applicable on data with few samples.* F1 score for pattern sets discovered by BINAPS on synthetic data with varying number of samples.

from just 1000 samples. When investigating the retrieved pattern sets for the data of 1000 samples, we see that as expected by the above analysis, some of the retrieved patterns correspond to frequent co-occurences of ground truth patterns, which are counted as errors in the F1 score.

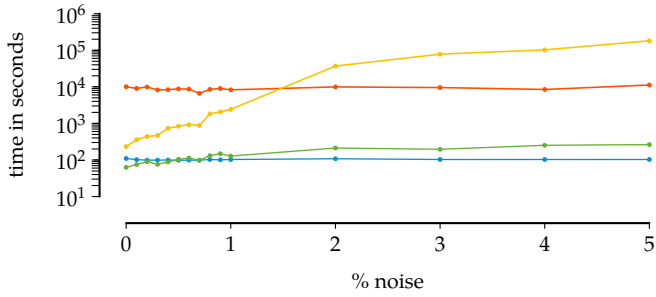
Recovering ground truth. To evaluate all methods with respect to scalability, we generate data with pattern sizes and densities as before, distributing patterns over $n = 10\,000$ samples uniformly at random, varying the number of different planted patterns in $\{10, 30, \ldots, 90\}$. As above, we introduce .1% of noise. Scaling further, we generate data of $n = 100\,000$ samples and plant $\{100, 300, \ldots, 900, 1000, 3000, 5000\}$ patterns, all of same frequency and level of noise. Here, we generate large data sets in terms of number of samples to avoid introducing spurious pattern sets, as two ground truth patterns are likely to co-occur significantly in small datasets.

The results show that state-of-the-art methods SLIM and DESC have trouble retrieving the ground truth (see Fig. 6.2a). They overfit respectively underfit. ASSO as well as BINAPS do retrieve ground truth pattern sets accurately. We observe that ASSO, SLIM, and DESC all scale unfavourably, with ASSO performing overall weakest in terms of runtime (Fig. 6.2b), taking days for relatively small data sets, similar to SLIM, and not being able to process even medium sized data sets. DESC shows to scale slightly better, being able to process moderately sized data sets, the retrieved patterns however not matching ground truth. In contrast, BINAPS scales to tens of thousands of patterns while retrieving pattern sets accurately, being able to process datasets with 100k samples and close to 20k features in slightly more than an hour.

Noise robustness. Next, we investigate the impact of noise on how well methods are able to retrieve the correct pattern set. We generate data of $n = 100\,000$ samples and 100 planted patterns with distributions as above, but now varying the percentage of noise in $\{0, 0.001, 0.002, \ldots, 0.009, 0.01, 0.02, \ldots, 0.05\}$, where 0.05 corresponds to a signal to noise ratio of 1. We visualize the results in



(a) F1 score on noise data (higher is better).



(b) Runtime on noise data (lower is better).

Figure 6.4: BinaPs *is robust to noise.* For synthetic data with known ground truth and varying the level of noise, we show the F1 score (a) and the runtime (b) for all methods. With 5% noise, we have a 1:1 ratio of signal-to-noise.

Fig. 6.4a-6.4b. Both Asso and BinaPs are robust to high amounts of noise, even for a signal to noise ratio of 1. Slim and Desc, on the other hand, are very sensitive to noise, both retrieving more spurious patterns the more noise is present. On these datasets BinaPs run for minutes, whereas the competitors take up to several hours.

6.4.2 Quantitative results on real data

To evaluate BinaPs on real data, we consider 5 datasets of different dimensions. In particular, we consider click-stream data of the hungarian on-line news portal *Kosarak*;[3] data on Belgian traffic *Accidents* (Geurts et al., 2003), *DNA*-amplification data (Myllykangas et al., 2006), online grocery shopping data from *Instacart*, for which we give more information in App. A.5.3, and single nucleotide polymorphisms in genomes of human individuals from the 1000 *Genomes* project (The 1000 Genomes Project Consortium, 2015). We provide details on how we processed *Genomes* in App. A.5.4. We provide statistics about the data and results in Table 6.1.

Whereas all methods are able to handle the small *DNA* and *Accidents* data, there are already orders of magnitude differences in runtime, with BinaPs finishing in minutes and Asso and Slim taking hours up to a day. With increasing number of columns, existing approaches fail to scale, Asso not being able to handle *Instacart*. On larger data such as *Kosarak*, all existing approaches did not terminate within 3 days, or run out of memory. BinaPs being the only approach to reliably handle large – both in n as well as m – databases, it retrieves patterns for *Kosarak*, and the challenging *Genomes* data.

Looking at the results, BinaPs retrieves succinct and non-redundant pattern sets. Furthermore, these easy-to-interpret pattern sets are much smaller than the initial capacity c given to the network, despite not explicitly penalizing model complexity. While retrieving equally succinct pattern sets, Asso fails to scale to moderately sized data. Slim, an MDL based approach, finds thousands of partially redundant patterns already for medium sized data, which make it hard to analyze as a whole. Desc underfits for *Accidents* respectively *Instacart* only returning patterns of length 2.

6.4.3 Qualitative results on real data

Here, we will first compare the results on two small datasets, and then move on to analyze the insights gained for the new *Genomes* data set based on patterns found by BinaPs.

[3] http://fimi.uantwerpen.be/

			# Patterns				Runtime			
Dataset	# rows	# cols	Asso	**BinaPs**	Desc	Slim	Asso	**BinaPs**	Desc	Slim
DNA	2458	391	134	131	345	281	4m	26s	20s	2s
Accidents	340183	468	133	78	215	12261	12h	6m	14m	21h
Instacart	2704831	1235	*n/a*	328	712	8119	∞	44m	25m	8h
Kosarak	990002	41270	*n/a*	302	*n/a*	*n/a*	∞	5h	∞	∞
Genomes	2504	226623	*n/a*	42	*n/a*	*n/a*	∞	9m	∞	∞

Table 6.1: *Results on Real World Data.* For five real world datasets, we give the number of rows, number of columns, and for Asso, BinaPs, Desc, and Slim we report the number of discovered patterns and runtime rounded to the most significant order of magnitude. Individual experiments were aborted after 3 days, resp. when exceeding the available 256GB of RAM, corresponding entries are marked with *n/a* and ∞.

DNA On this dense data, BinaPs and Asso discover compact patterns that exhibit a block structure, with consecutive items merged into a feature. Such block-like structures correspond to the larger chromosomal areas where DNA copy number amplification happens, and hence represent biologically meaningful patterns. For this particular data set, we can see the disadvantages of iterative heuristics such as from Slim, which learns parts of this block structure early on, but then overfits to overly large patterns that only appear in few samples and without evident block structure. These larger patterns are however likely due to random co-occurrences. Desc here only finds short patterns, likely being caught in a local minimum.

Instacart BinaPs discover patterns describing dense, noisy blocks, for example a pattern spanning dozens of fruits, which are bought together in arbitrary combinations. Slim breaks such dense blocks in many (thousands) of individual patterns, each containing a small subset of items. Desc again underfits, finding patterns of length 2 only. BinaPs also discover small patterns that reveal the general buying behaviour of customers such as a pattern of different prepared dishes {Pizza, Ceasar salad, Hummus wrap, Filled wrap}, or certain food styles, such as {Chicken Wrap, Chile con queso}.

Genomes Last, we consider a dataset that motivated this work. The data contains information about variants of a population of human individuals for single nucleotide positions within genes, and, with over 200 thousand features, *Genomes* is far beyond what existing methods can consider. BinaPs, however, is able to discover a succinct pattern set in mere minutes. Here, we analyze the discovered patterns in more detail.

Biologically, we expect sequence stretches to be conserved and hence variants to be inherited "together". A first positive observation is that most patterns that BinaPs discover indeed show such blocks of variants that are close-by on the genome.

Within such blocks, the rare variants often lie on the same allele – indicated by consecutive "0|1" variants – meaning that often one parent has the "common" reference variants for consecutive sites, whereas the other parent has the "rare" variants. Encouragingly, individual BinaPs patterns show this, and even more interesting, BinaPs often discover a second pattern for the same sites that show the opposite, for example most sites to be "1|0". Whether this is due to phasing or has biological meaning, we leave to the experts.

Taking a closer look at individual patterns, we find interesting connections between variants and the genes they occur in. For example, we find a pattern spanning multiple variants in the genes NUCB2, NCR3LG1, ABCC8, and ROMO1. Variants at NUCB2 and ABCC8 have together been associated with type 2 diabetes and high blood pressure in Japanese population (Sakamoto et al., 2007). For NCR3LG1 we know that it is close-by to ABCC8, but similar to ROMO1 there has not been any connection made between the variants of these

genes, and hence could provide interesting insights for experts.

Another highly interesting pattern is over SF3A1, RRP7A, and Z82190, where SF3A1 and RRP7A encode proteins that are part of the ribosomal complex, the factory that produces proteins in a cell. Z82190 is a so far uncharacterized gene, and this pattern hence suggests what its role could be, which can guide future studies.

BINAPs discovers both large patterns of many variants within few genes, but also discover associations over many genes at once. As last example, we consider the pattern of variants in FUBP1, SELENOI, ZKSCAN5, TMEM225B, ASPN, SOX6, PLEKHA7, and ALX3. ASPN and ALX3 are taking part in chondrogenesis, an essential developmental process producing elastic tissue covering ends of bones and joints. FUBP1 and SOX6 are genes linked to other developmental processes. For ZKSCAN5 and TMEM225B only little is known so far. These results encourage for further in-depth analysis by domain experts.

Overall, BINAPs retrieve informative patterns over variant sites, which can give interesting insights to relations between variants, genes, and their function.

6.5 DISCUSSION AND CONCLUSION

We considered the problem of pattern set mining and propose BINAPs, a novel kind of binarized neural networks that are capable of discovering a compact set of interpretable patterns that together describe the data well. In experiments on both synthetic as well as real data, BINAPs reliably recovered the ground truth and showed promising results on several large scale real world datasets, discovering succinct descriptions of the data while scaling to hundreds of thousands of samples and features. The results further showed that BINAPs discover patterns of varying length and frequency, that describe true structure of the given data, on sparse and dense data as well as small to large n, as well as m.

Naturally related to BINAPs, boolean matrix factorization showed good results on small data sets, but fails to scale beyond small data. Furthermore, it generally requires the rank of the factorization – the size of the pattern sets – to be given, wheras BINAPs only need a large enough capacity of the pattern layer, and shrinks the number of patterns – or factors – if necessary. State-of-the-art information theoretic methods, based on the Minimum Description Length principle or Maximum Entropy distributions, tended to over- or underfit, which is likely due to their heuristic combinatorial search rather than their objective. These methods did often recover (many) small fragments of ground truth patterns, which may be puzzled together by domain experts. In contrast, BINAPs were able to discover entire patterns, or larger chunks.

With their ability to learn patterns using continuous, gradient-based optimization, BinaPs pave the way towards exploratory studies of challenging data sets comprising hundreds of thousands of features that state-of-the-art algorithms for pattern set mining fail to process, while providing patterns in the interpretable language of conjunctions over input symbols.

While BinaPs can reveal interesting structure in data and provide insights for human experts, it would make for interesting future work to extend BinaPs to richer pattern languages. For example, by developing an architecture that reflects statements similar to the previously introduced Mexican, to mine relationships of mutual exclusivity, which capture interesting structures in real world datasets. Similarly, investigating deeper networks would make for engaging future work as more involved architectures could for example allow to extract pattern hierarchies from the data. Simply making the network larger however comes at the cost of reduced interpretability of the patterns and, hence, needs to be carefully balanced.

STRONG PRUNING FOR LOTTERY TICKETS WITH NONZERO BIAS

7

7.1 INTRODUCTION

Challenging tasks across different domains, from protein structure prediction for drug development to detection in complex scenes for self driving cars, have recently been solved through deep neural networks (NNs). This success, however, is due to heavy overparametrization and comes at the expense of large amounts of computational resources that these models require to be trained and to be deployed. While training small NNs from scratch commonly fails, the lottery ticket hypothesis (LTH) conjectured by Frankle and Carbin (2019) bears a potential solution. The LTH states that within a large, randomly initialized NN there exists a well trainable, much smaller subnetwork, or 'ticket', which can be identified by pruning the large NN. Thus, both training and deployment becomes computationally much cheaper at the expense of the pruning algorithm. Even more promising is the conjecture of the existence of 'strong lottery tickets' (SLTs) by Ramanujan et al. (2020), which are subnetworks of randomly initialized NNs that do *not* require any further training, for which expensive training might become obsolete. This strong lottery ticket hypothesis (SLTH) was later proven for networks without biases (Malach et al., 2020; Pensia et al., 2020; Orseau et al., 2020).

This chapter is based on Fischer, Gadhikar, and Burkholz (2021a).

Whereas standard intialization schemes – and hence SLTs – use zero biases, most successfully trained NN architectures have nonzero biases. Such nonzero bias terms are important to equip NNs with the universal approximation property (Scarselli and Tsoi, 1998). Serving as our main motivation, we show that NNs without biases, which include SLTs, fail to achieve this in general. To enable successful training by pruning alone, we, hence, generalize common initialization schemes, including the 'looks linear' orthogonal initialization, to nonzero biases. We show that a signal that computes the output of a function or gradients can propagate through such nonzero bias initialized networks. This means they are trainable and gradient-based scoring functions, which are typically used by lottery ticket pruners, are computable. Moreover, we formally prove the SLTH in this setting and provide empirical evidence that our theory derives realistic conditions, in which lottery ticket pruning is feasible.

For the discovery of SLTs in practice, Ramanujan et al. (2020) proposed Edge-Popup, to the best of our knowledge the only algorithm capable of doing so. Their proposal is, however, not suited to recover bias parameters and finds only relatively dense tickets. We here extend their approach in multiple ways to recover strong tickets that include bias parameters and that are sparser than the ones obtained by vanilla Edge-Popup. In particular, we extend the popup scores to bias terms, and slowly anneal the sparsity of the network to the desired target sparsity. In addition, our LTH proof conjectures that a rescaling of the parameters is necessary to find SLTs of high sparsity. We propose an efficient optimization procedure to find such a good scaling factor in each epoch.

In a synthetic data study, we show that SLTs recovered by Edge-Popup from NNs with nonzero bias initialization outperform tickets found in networks initialized with zero bias. Furthermore, we show that on this data, Edge-Popup with rescaling finds much sparser tickets of higher quality. While these results are encouraging, we discuss that on image data, nonzero biases might play a less prominent role for the current generation of SLT pruning algorithms, which identify subnetworks of suboptimal sparsity with few bias parameters in the recovered ticket (Fischer and Burkholz, 2022). We anticipate that our generalization of strong tickets and initialization schemes could pave the way for the next generation of algorithms for the discovery of parameter efficient subnetworks at initialization.

7.2 Related work

The lottery ticket hypothesis (Frankle and Carbin, 2019) has spurred the development of neural network pruning algorithms that either prune before (Wang et al., 2020; Lee et al., 2019b; Tanaka et al., 2020; Verdenius et al., 2020), during (Frankle and Carbin, 2019; Srinivas and Babu, 2016; You et al., 2020; Lee et al.,

2020; Liu et al., 2021a,a; Weigend et al., 1991; Savarese et al., 2020; Chen et al., 2021; Su et al., 2020; Ma et al., 2021), or after training (Savarese et al., 2020; LeCun et al., 1990; Hassibi and Stork, 1992; Dong et al., 2017; Li et al., 2017; Molchanov et al., 2017; Zhang et al., 2021). In particular for pruning before and during training, their main objective is to identify a subnetwork of a randomly initialized neural network – a lottery ticket – that can be trained to achieve a similar performance as the fully trained large network. Usually, these methods try to find lottery tickets in a 'weak' (but powerful) sense, that is to identify a sparse neural network architecture that is well trainable starting from its initial parameters. These methods score edges in terms of network flow, which can be quantified by gradients at different stages of pruning, or based on edge weights, and prune all edges with the lowest scores until the desired sparsity is achieved (Frankle et al., 2021).

Zhou et al. (2019) and Ramanujan et al. (2020) postulated an even stronger hypothesis, as they realized that the randomly initialized neural network contains so called 'strong' lottery tickets that do not require further training after pruning. Malach et al. (2020); Pensia et al. (2020); Orseau et al. (2020) proved their existence by deriving realistic lower bounds on the width of the original randomly initialized neural network that contains a lottery ticket with a given probability.

However, the proposed pruning algorithm for SLTs, i.e., edge-popup (Ramanujan et al., 2020), as well as the existence proofs only handle neural network architectures with zero biases. A reason might be that very large neural networks can compensate for missing biases in the studied application, i.e. image classification. Yet, this compensation usually requires a higher number of weight parameters but state-of-the-art algorithms and proofs for SLTs do not cover highly sparse tickets as we also show in the next chapter (Fischer and Burkholz, 2022). Another reason might be that most neural network initialization schemes propose zero biases. Exceptions include a data dependent choice of biases (Yang et al., 2019) and a random scheme that does not try to prevent exploding or vanishing gradients in deep neural networks (Hanin and Rolnick, 2019). The recent trend in the search for weak lottery tickets towards rewinding parameters to values obtained early during training (Frankle et al., 2020) also results in lottery ticket initialization with nonzero biases.

None of these nonzero bias initialization schemes are designed in support of the existence of SLTs. We fill this gap with this work.

7.3 NOTATION

Let $f(x)$ denote a bounded function, without loss of generality $f : [-1,1]^{n_0} \to [-1,1]^{n_L}$, that is parameterized as a deep neural network with architecture

$\bar{n} = [n_0, n_1, ..., n_L]$, i.e., depth L and widths n_l for layers $l = 0, ..., L$ with ReLU activation function $\phi(x) := \max(x, 0)$. It maps an input vector $\boldsymbol{x}^{(0)}$ to neurons $x_i^{(l)}$ as:

$$\boldsymbol{x}^{(l)} = \phi\left(\boldsymbol{h}^{(l)}\right), \boldsymbol{h}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{x}^{(l-1)} + \boldsymbol{b}^{(l)},$$
$$\boldsymbol{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}, \boldsymbol{b}^{(l)} \in \mathbb{R}^{n_l}$$

where $\boldsymbol{h}^{(l)}$ is the pre-activation, $\boldsymbol{W}^{(l)}$ is the weight matrix, and $\boldsymbol{b}^{(l)}$ is the bias vector of layer l. For convenience, the parameters of the network are subsumed in a vector $\boldsymbol{\theta} := \left(\left(\boldsymbol{W}^{(l)}, \boldsymbol{b}^{(l)}\right)\right)_{l=1}^{L}$. We also write $f(x \mid \boldsymbol{\theta})$ to emphasize the dependence of f on its parameters $\boldsymbol{\theta}$. The supremum norm of any function g is defined with respect to the same domain $||g||_\infty := \sup_{x \in [-1,1]^{n_0}} ||g||_2$.

Assume furthermore that a ticket f_ϵ can be obtained by pruning a large mother network f_0, which we indicate by writing $f_\epsilon \subset f_0$. The sparsity level ρ of f_ϵ is then defined as the fraction of nonzero weights that remain after pruning, i.e., $\rho = \left(\sum_l ||\boldsymbol{W}_\epsilon^{(l)}||_0\right) / \left(\sum_l ||\boldsymbol{W}_0^{(l)}||_0\right)$, where $||\cdot||_0$ denotes the l_0-norm, which counts the number of nonzero elements in a vector or matrix. Another important quantity that influences the existence probability of lottery tickets is the in-degree of a node i in layer l of the target f, which we define as the number of nonzero connections of a neuron to the previous layer plus 1 if the bias is nonzero, i.e., $k_i^{(l)} := ||\boldsymbol{W}_{i,:}^{(l)}||_0 + ||b_i^{(l)}||_0$, where $\boldsymbol{W}_{i,:}^{(l)}$ is the i-th row of $\boldsymbol{W}^{(l)}$. The maximum degree of all neurons in layer l is denoted as $k_{l,\max}$. In the formulation of theorems, we make use of the universal constant C that can attain different values.

7.4 TYPES OF LOTTERY TICKETS

As introduced before, there exist different types of lottery tickets depending on the amount of pruning and training involved in their discovery. To clarify the differences between them, we provide an overview in Fig. 7.1 and formally define these tickets in terms of the network parameters θ. Frankle and Carbin (2019) first conjectured the LTH, suggesting that within large, randomly initialized NNs there exist small subnetworks that – after training – perform as well as the large network after training. These were later referenced as 'weak tickets', which can be informally defined as follows.

Definition 7.1 (Weak lottery tickets)**.** *Given a randomly initialized neural network* $f_0(x \mid \boldsymbol{\theta_0})$*, a small subnetwork* $f_\epsilon(x \mid \boldsymbol{\theta_0'}), \boldsymbol{\theta_0'} \subset \boldsymbol{\theta_0}$ *of sparsity* $\frac{|\boldsymbol{\theta_0'}|}{|\boldsymbol{\theta_0}|}$ *is called a* weak

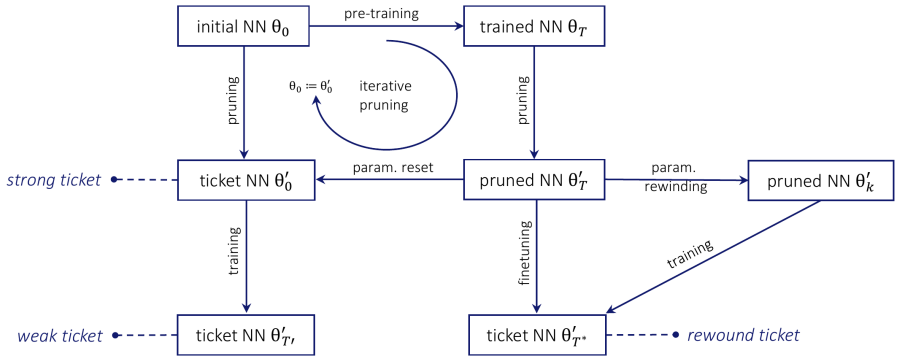


Figure 7.1: *Types of lottery tickets.* From a large neural network given by parameters θ, the goal is to identify a small subnetwork $\theta' \subset \theta$, $\frac{|\theta'|}{|\theta|} \ll 1$ that performs as well as the original network. If this subnetwork at initialization θ'_0 performs as good as the large network after training, θ_T, it is called a 'strong ticket'. If it requires further training from the initialization, $\theta'_{T'}$, it is called 'weak ticket'. If it is trained from an intermediate training point from the pruning process, it is called 'rewound ticket'.

ticket, *if it can be trained to match the performance of the fully trained large network, i.e.* $\mathcal{L}\left(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}'_{T'}\right) \approx \mathcal{L}\left(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}_T\right)$ *for loss* $\mathcal{L}$.

They key task of weak lottery ticket pruning is to discover such a small subnetwork. First proposals aimed to identify weak tickets in one shot, i.e. prune a network once to the desired target sparsity. Later, iterative – or multishot – approaches have been shown to be more successful, where throughout the iterations the sparsity of the network is iteratively reduced to the desired target sparsity. Essentially, in each iteration we use weak lottery ticket pruning to identify a slightly smaller network than before, reset the parameters to initialization, and then repeat on the smaller network. All of these approaches usually rely on pre-training, i.e. they train a network for few epochs and then identify the most important parameters based on a score, e.g. based on their magnitude or taking into account gradient flows (Frankle and Carbin, 2019; Wang et al., 2020; Lee et al., 2019b; Tanaka et al., 2020).

For iterative magnitude pruning (IMP), Frankle et al. (2020) showed that it is necessary to rewind the parameters to values earlier in training rather than to initialization, to be applied successfully to complex image tasks. These tickets can be defined as follows.

Definition 7.2 (Rewound lottery tickets)**.** *Given a randomly initialized neural network* $f_0(x \mid \boldsymbol{\theta_0})$*, a small subnetwork* $f_\epsilon(x \mid \boldsymbol{\theta'_0}), \boldsymbol{\theta}'_0 \subset \boldsymbol{\theta}_0$ *identified by an iterative pruning approach is called a* rewound ticket*, if it can be trained from an intermediate*

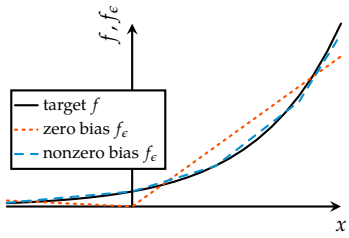


Figure 7.2: *Function approximations.* Depicted are approximations of an exponential function (solid black) by sparse NNs with nonzero bias (dashed blue), which are piecewise linear functions, and zero bias (dotted orange), which are two linear pieces through the origin, which exemplify Lem. 7.1.

training point from the pruning process θ'_k to match the performance of the fully trained large network, i.e. $\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}'_{T^*}) \approx \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}_T)$ for loss $\mathcal{L}$.

One special case is to skip reset entirely, which is often referred to as ticket finetuning in the literature (Liu et al., 2021a).

Both weak and rewound tickets can give substantial gain in terms of resources, especially during deployment. Yet, we have to invest into training for several epochs, where in iterative pruning approaches we start with rather large networks, and require full training of the small ticket network. Strong lottery tickets provide a remedy, as they do not require any further training, first conjectured by Ramanujan et al. (2020) and later formally proven to exist (Malach et al., 2020; Pensia et al., 2020; Orseau et al., 2020).

Definition 7.3 (Strong lottery tickets)**.** *Given a randomly initialized neural network $f_0(x \mid \boldsymbol{\theta}_0)$, a small subnetwork $f_\epsilon(x \mid \boldsymbol{\theta}'_0), \boldsymbol{\theta}'_0 \subset \boldsymbol{\theta}_0$ is called a* strong ticket*, if it, at initialization, matches the performance of the fully trained large network, i.e. $\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}'_0) \approx \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}_T)$ for loss $\mathcal{L}$.*

These tickets can be seen as the holy grail of NN pruning, as they bear the promise of extreme resource efficiency by skipping training entirely. So far, only one algorithm exists, which retrieves relatively dense strong tickets. In this chapter, we will focus on the foundations of strong lottery tickets. We will see why (good) strong lottery tickets currently are hard to find and how to overcome some of these issues, thereby improving the capabilities of the state-of-the-art strong tricket pruner.

7.5 Motivation: Universal approximation

Why do we need to initialize nonzero biases? With nonzero biases, neural networks have the universal approximation property (Scarselli and Tsoi, 1998).

Thus, for a given $\epsilon > 0$ and arbitrary continuous function g, a large enough neural network can approximate g up to error ϵ. Standard neural networks and also weak lottery tickets are able to learn nonzero biases even from zero initialization. However, this is not the case for strong lottery tickets. Strong lottery tickets rely on pruning alone to obtain a model with high performance. Nonzero biases need to be available from the initialization for them to be universal approximators with ReLU activations. The following Lemma captures the problem with zero bias networks, which is a factorization of univariate ReLU networks of *arbitrary depth and width*.

Lemma 7.1. *Univariate neural networks with ReLU activations $f : \mathbb{R} \to \mathbb{R}^{n_L}$ of arbitrary depth L and layer widths $n_1, \ldots, n_L$, and without biases, represent a function $f(x) = \boldsymbol{W}_+\phi(x) + \boldsymbol{W}_-\phi(-x), \boldsymbol{W}_+, \boldsymbol{W}_- \in \mathbb{R}^{n_L \times 1}$.*

Proof. We prove by induction over the number of hidden layers L of the network. First, assume that $L = 1$, i.e. $f^{(1)}(x) = \phi(\boldsymbol{W}^{(1)}x)$, with $\boldsymbol{W}^{(1)} \in \mathbb{R}^{n_1 \times 1}$. For any output neuron $f_j^{(1)}(x),\ j = 1, \ldots, n_1$, we have

$$\begin{aligned} f_j^{(1)}(x) &= \sum_{i=1}^{n_0} \phi(\boldsymbol{W}_i^{(1)}x) \\ &= \begin{cases} \sum_{i=1}^{n_0} \boldsymbol{W}_i^{(1)} I(\boldsymbol{W}_i^{(1)} > 0)\phi(x), & \text{if } x \geq 0 \\ \sum_{i=1}^{n_0} -\boldsymbol{W}_i^{(1)} I(\boldsymbol{W}_i^{(1)} < 0)\phi(-x), & \text{otherwise,} \end{cases} \end{aligned}$$

where $I(.)$ is the indicator function. For networks with a single layer, we thus get $f^{(1)}(x) = \boldsymbol{W}_+^{(1)}\phi(x) + \boldsymbol{W}_-^{(1)}\phi(-x)$ with

$$\begin{aligned} \boldsymbol{W}_+^{(1)} &= \boldsymbol{W}_i^{(1)} I(\boldsymbol{W}_i^{(1)} > 0)\,, \\ \boldsymbol{W}_-^{(1)} &= -\boldsymbol{W}_i^{(1)} I(\boldsymbol{W}_i^{(1)} < 0)\,, \end{aligned}$$

which proves our claim for $L = 1$.

Next, our induction hypothesis is that $f_j^{(l)} = \boldsymbol{W}_+^{(l)}\phi(x) + \boldsymbol{W}_-^{(l)}\phi(-x)$. For networks with $l + 1$ layer, $f^{(l+1)}(x) = \boldsymbol{W}^{(l+1)}\phi\left(f^{(l)}(x)\right), \boldsymbol{W}^{(l+1)} \in \mathbb{R}^{n_{l+1} \times n_l}$,

for each output neuron $f_j^{(l+1)}(x),\ j = 1, \dots, n_{l+1}$ we obtain

$$
\begin{aligned}
f_j^{(l+1)}(x) &= \sum_{i=1}^{n_{l+1}} \phi\left(f^{(l)}(x)\right)_i W_{ji}^{(l+1)} \\
&= \sum_{i=1}^{n_{l+1}} \phi\left(W_+^{(l)}\phi(x) + W_-^{(l)}\phi(-x)\right)_i W_{ji}^{(l+1)} \quad \text{(induction hypothesis)} \\
&= \begin{cases} \sum_{i=1}^{n_{l+1}} \phi\left(W_+^{(l)} x\right)_i W_{ji}^{(l+1)}, & \text{if } x \geq 0 \\ \sum_{i=1}^{n_{l+1}} \phi\left(-W_-^{(l)} x\right)_i W_{ji}^{(l+1)}, & \text{otherwise} \end{cases} \\
&= \begin{cases} \sum_{i=1}^{n_{l+1}} W_{+,i}^{(l)} W_{ji}^{(l+1)} x, & \text{if } x \geq 0 \wedge W_{+,i} > 0 \\ \sum_{i=1}^{n_{l+1}} -W_{-,i}^{(l)} W_{ji}^{(l+1)} x, & \text{if } x < 0 \wedge W_{-,i} > 0 \\ 0, & \text{otherwise.} \end{cases}
\end{aligned}
$$

Hence, we get

$$
\begin{aligned}
W_+^{(l+1)} &= W^{(l+1)} W_+^{(l)} \\
W_-^{(l+1)} &= -W^{(l+1)} W_-^{(l)},
\end{aligned}
$$

which concludes the induction step, and the proof. □

This factorization shows that ReLU networks without biases are greatly limited in terms of the functions they can express, which we visualize for an example in Fig. 7.2. From this Lemma we derive the following insight about neural networks without biases – and hence strong lottery tickets – which forms the motivation of our work.

Corollary 3. *Neural networks with ReLU activations and without biases are not universal function approximators.*

Proof. We prove by contradiction. Assume that such a network is a universal function approximator, i.e. for any function g and any ϵ there exists a network that can approximate g up to error ϵ.

Let us try to approximate the constant function $g(x) = 0.5$ on the domain $[-1, 1]$ with a neural network without biases. From Lemma 7.1, we know that an univariate ReLU network without biases represents a function $f(x) = w_+\phi(x) + w_-\phi(-x)$ with two parameters $w_+, w_- \in \mathbb{R}$. The minimum mean squared error with respect to $g(x)$ that f can achieve is $\int_{-1}^{1}(g(x) - f(x))^2 dx = 1/8$ for $w_+ = 3/4$ and $w_- = 3/4$ (see Appendix A.6.1 for derivation). Thus for any $\epsilon < 1/8$, $f(x)$ fails to approximate $g(x)$ up to error ϵ, which is a contradiction.

Note that a neural ReLU network with nonzero biases can represent the function $g(x) = 0.5$ perfectly. For instance, a network of depth $L = 1$ with one neuron in the intermediary layer is sufficient, as $0.5 = \phi(0.5)$.

Although $g(x) = 0.5$ is an obviously simple function, it already serves as a counterexample and hence suffices to prove the theorem. However, for networks without bias this is a hard task, as it explicitly requires to model an argument independent offset – or bias. Yet, it is easy to see from Lemma 7.1 that these networks usually fail to guarantee universal approximation for univariate non-linear functions even without explicit biases. We provide $g(x) = e^x$ as an example in Appendix A.6.1. □

This theorem provides the theoretical motivation for why we need nonzero bias initializations for well performing strong lottery tickets, and why it is necessary to include bias terms when pruning for good strong lottery tickets.

7.6 INITIALIZING NONZERO BIASES

Common initialization schemes (e.g. He et al. (2015); Glorot and Bengio (2010); Burkholz and Dubatovka (2019)) set all biases to zero, while network weights are drawn randomly to obtain parameter diversity. Proofs of the strong lottery ticket hypothesis have focused on this setting, thereby foregoing the universal approximation property of deep neural networks (Scarselli and Tsoi, 1998), since pruning alone can only recover the zero-initialized biases.

Here, we propose the initialization of nonzero biases, which then become subject to pruning in addition to the network weights. How should these biases be initialized? A good approach has to fulfill two essential criteria. a) The randomly initialized neural network needs to be trainable by SGD. This property is also critical for most pruning algorithms, as they are inspired by SGD and define pruning scores based on gradients. b) The randomly initialized neural network should contain lottery tickets with high probability.

Before we can answer how to initialize biases, we first have to face a different issue pertaining strong lottery tickets. Standard initialization approaches, like He (He et al., 2015) or Glorot (Glorot and Bengio, 2010) initialization, achieve trainability by ensuring that the output of a deep neural network is contained within a reasonable range, thus rendering the computations of gradients numerically feasible. Network weights are commonly initialized according to a distribution with variance σ^2 that is inverse proportional to the number of neurons n in a layer, $\sigma^2 \propto 1/n$. In consequence, after pruning a high percentage of these weights, the network output is heavily down-scaled, which needs to be compensated by up-scaling the output, as also discovered experimentally by Ramanujan et al. (2020) and mentioned by Malach et al. (2020).

7.6.1 OUTPUT SCALING

For ReLU networks with zero biases, the appropriate output scaling after or during pruning is straight forward to compute, as networks of depth L are L-homogeneous in the network parameters. That is, multiplying each parameter with the same scalar σ leads to a scaling factor of σ^L: $f(x \mid \sigma\theta) = \sigma^L f(x \mid \theta)$. This holds no longer true for nonzero biases.
The following observation helps us to develop a notion that is similar to homogeneity for networks with nonzero biases.

Lemma 7.2. *Let $h(\boldsymbol{\theta_0}, \boldsymbol{\sigma})$ denote a transformation of the parameters $\boldsymbol{\theta_0}$ of the deep neural network f_0, where each weight is multiplied by a scalar σ_l, i.e., $h_{ij}^{(l)}(w_{0,ij}^{(l)}) = \sigma_l w_{0,ij}^{(l)}$, and each bias is transformed to $h_i^{(l)}(b_{0,i}^{(l)}) = \left(\prod_{m=1}^{l} \sigma_m\right) b_{0,i}^{(l)}$. Then, we have $f(x \mid h(\boldsymbol{\theta_0}, \boldsymbol{\sigma})) = \prod_{l=1}^{L} \sigma_l f(x \mid \boldsymbol{\theta_0})$.*

Proof. Let the activation function ϕ of a neuron either be a ReLU $\phi(x) = \max(x, 0)$ or the identity $\phi(x) = x$. A neuron $x_i^{(l)}$ in the original network becomes $g\left(x_i^{(l)}\right)$ after parameter transformation. We prove the statement by induction over the depth L of a deep neural network.

First, assume that $L = 1$ so that we have $x_i^{(1)} = \phi\left(\sum_j w_{ij}^{(1)} x_j + b_i^{(1)}\right)$. After transformation by h, i.e. $w_{ij}^{(1)} \mapsto \sigma_1 w_{ij}^{(1)}$ and $b_i^{(1)} \mapsto \sigma_1 b_i^{(1)}$, we receive $g\left(x_i^{(1)}\right) = \phi\left(\sum_j w_{ij}^{(1)} \sigma_1 x_j + \sigma_1 b_i^{(1)}\right) = \sigma_1 x_i^{(1)}$ because of the homogeneity of $\phi(\cdot)$. This proves our claim for $L = 1$.

Next, our induction hypothesis is that $g\left(x_i^{(L-1)}\right) = \prod_{m=1}^{L-1} \sigma_m x_i^{(L-1)}$. Scaling each parameter by applying h, we obtain

$$g\left(x_i^{(L)}\right) = \phi\left(\sum_j h_{ij}^{(L)}(w_{ij}^{(L)}) g\left(x_j^{(L-1)}\right) + h_i^{(L)}(b_i^{(L)})\right) \quad \text{(network definition)}$$

$$= \phi\left(\sum_j w_{ij}^{(L)} \sigma_L g\left(x_j^{(L-1)}\right) + b_i^{(L)} \prod_{m=1}^{L} \sigma_m\right) \quad \text{(def. of transformation)}$$

$$= \phi\left(\sum_j w_{ij}^{(L)} \sigma_L \prod_{m=1}^{L-1} \sigma_m x_j^{(L-1)} + b_i^{(L)} \prod_{m=1}^{L} \sigma_m\right) \quad \text{(induction hypothesis)}$$

$$= \prod_{m=1}^{L} \sigma_m x_i^{(L)} \quad \text{(homogeneity of } \phi\text{)},$$

which was to be shown. □

Lemma 7.2 suggests that if we scale each weight by a factor $\sigma_{w,l}$, scaling the corresponding biases by a factor $\sigma_{b,l} = \prod_{m=1}^{l} \sigma_{w,m}$ would result in the same network f without scaling of parameters. We only have to correct the output by dividing it with a factor $\prod_{l=1}^{L} \sigma_{w,l}$. From this observation, we directly derive our initialization proposal, as it suggests an equivalence (irrespective of scaling) between initialising parameters in $\theta_i \in U[0,1]$ and our more realistic setting. Concretely, we propose to replace $b_i^{(l)} = 0$ by $b_i^{(l)} \sim U([-\prod_{k=1}^{l} \sigma_{w,k}, \prod_{k=1}^{l} \sigma_{w,k}])$ or $b_i^{(l)} \sim \mathcal{N}\left(0, \prod_{m=1}^{l} \sigma_{w,m}\right)$, respectively, when the weights are $w_{ij}^{(l)} \sim U\left([-\sigma_{w,l}, \sigma_{w,l}]\right)$ or $w_{ij}^{(l)} \sim \mathcal{N}\left(0, \sigma_{w,l}\right)$.

As a general remark, note that the scaling factor of the output $\prod_{m=1}^{L} \sigma_{w,m}$ quickly approaches zero for increasing depth, which could render one-shot pruning numerically infeasible. For that reason, we propose a computationally cheap rescaling procedure that allows us to find significantly sparser strong lottery tickets. This is also helpful for maintaining the trainability of the pruned network (Hayou et al., 2021), which we discuss next in the context of initializations with nonzero biases.

7.6.2 SIGNAL PRESERVATION

The question remains whether input and gradient signals can still propagate through the large original network with such an initialization. A common criterion to prevent initial vanishing or exploding gradients, in particular in mean field analyses (Schoenholz et al., 2017), is to ensure that the squared signal norm of the input can propagate through the initial network. To bound the second moment of the squared output, we generalize Cor. 3 for normal distributions in Burkholz and Dubatovka (2019) to symmetric weight and bias distributions.

Lemma 7.3. *Assume that the weights and biases of a fully-connected deep neural network f are drawn independently from distributions that are symmetric around the*

origin 0 *with variances* $\sigma_{w,l}^2$ *or* $\sigma_{b,l}^2$*, respectively. Then, for every input* $\boldsymbol{x^{(0)}}$*, the second moment of the output is*

$$\mathbb{E}\left(\left\|f(\boldsymbol{x^{(0)}})\right\|_2^2\right) = \left\|\boldsymbol{x^{(0)}}\right\|_2^2 \Pi_{l=1}^{L} \frac{n_l \sigma_{w,l}^2}{2} + \sigma_{b,L}^2 \frac{n_L}{2} + \sum_{l=1}^{L-1} \sigma_{b,l}^2 \frac{n_l}{2} \Pi_{k=l+1}^{L} \frac{n_k \sigma_{w,k}^2}{2}.$$

Proof. First, let us focus on the distribution of a neuron $x_i^{(l)}$ given all neurons of the previous layer with $x_i^{(l)} = \phi\left(h_i^{(l)}\right)$. Since we assume that the weights and biases are distributed independently with zero mean, it follows that also the preactivation $h_i^{(l)} = \sum_j w_{ij}^{(l)} x_j^{(l-1)} + b_i^{(l)}$ has zero mean and variance $\mathbb{V}\left(h_i^{(l)} \mid \boldsymbol{x^{(l-1)}}\right) = \sum_j \left(x_j^{(l-1)}\right)^2 \mathbb{V}\left(w_{ij}^{(l)}\right) + \mathbb{V}\left(b_i^{(l)}\right) = \sigma_{w,l}^2 \left\|\boldsymbol{x^{(l-1)}}\right\|^2 + \sigma_{b,l}^2$, where $\mathbb{V}$ is the variance operator. It is furthermore symmetric around zero so that a neuron $x_i^{(l)} = \phi\left(h_i^{(l)}\right) \sim 0.5\delta_0 + 0.5 p_{h_{l,+}}$ is projected to zero with probability 0.5 and otherwise follows the distribution of the positive preactivation $p_{h_{l,+}}$, where δ_0 denotes the delta distribution at 0 and $h_{l,+}$ the random variable h_l conditional on $h_l > 0$. In consequence, the squared neuron value $(x_i^{(l)})^2 \sim 0.5\delta_0 + 0.5 p_{h_l^2}$ has expectation $\mathbb{E}\left(\left(x_i^{(l)}\right)^2\right) = 0.5\ \mathbb{E}\left(\left(h_i^{(l)}\right)^2\right) = 0.5\ \sigma_{w,l}^2 \left\|\boldsymbol{x^{(l-1)}}\right\|^2 + 0.5\ \sigma_{b,l}^2$.

Since all the neurons are independent and identically distributed given the neurons of the previous layer, we can easily deduce the expected signal norm

$$\begin{aligned}\mathbb{E}\left(\left\|\boldsymbol{x^{(l)}}\right\|_2^2 \;\middle|\; \left\|\boldsymbol{x^{(l-1)}}\right\|_2^2\right) &= \sum_{i=1}^{n_l} \mathbb{E}\left(\left(x_i^{(l)}\right)^2 \;\middle|\; \left\|\boldsymbol{x^{(l-1)}}\right\|_2^2\right) \\ &= \frac{n_l}{2}\left(\sigma_{w,l}^2 \left\|\boldsymbol{x^{(l-1)}}\right\|^2 + \sigma_{b,l}^2\right).\end{aligned}$$

This gives us the expected signal norm of an arbitrary layer conditioned on the previous layer. We can use this relationship to also compute the average squared signal norm of the output layer, which is

$$\mathbb{E}\left(\|f(\boldsymbol{x_0})\|_2^2\right) = \mathbb{E}\left(\left\|\boldsymbol{x^{(L)}}\right\|_2^2\right) = \mathbb{E}\left(\mathbb{E}\left(\left\|\boldsymbol{x^{(L)}}\right\|_2^2 \;\middle|\; \left\|\boldsymbol{x^{(1)}}\right\|_2^2\right)\right),$$

where the first equality is by definition of the network, and the second equality holds by law of total expectation. By recursively repeating this argument on

the inner expectation, we get

$$\begin{aligned}
&\mathbb{E}\left(\|f(x_0)\|_2^2\right) \\
&= \mathbb{E}\left(\mathbb{E}\left(\left\|x^{(L)}\right\|_2^2 \;\middle|\; \left\|x^{(1)}\right\|_2^2\right)\right) \\
&= \mathbb{E}\left(\mathbb{E}\left(\mathbb{E}\left(\left\|x^{(L)}\right\|_2^2 \;\middle|\; \left\|x^{(2)}\right\|_2^2\right) \;\middle|\; \left\|x^{(1)}\right\|_2^2\right)\right) \\
&= \mathbb{E}\left(\mathbb{E}\left(\ldots\mathbb{E}\left(\mathbb{E}\left(\left\|x^{(L)}\right\|_2^2 \;\middle|\; \left\|x^{(L-1)}\right\|_2^2\right) \;\middle|\; \left\|x^{(L-2)}\right\|_2^2\right)\ldots \;\middle|\; \left\|x^{(1)}\right\|_2^2\right)\right).
\end{aligned}$$

Using the derivation further above, which provides a solution to the expected signal norm for a layer conditioned on the previous layer, we can iteratively resolve the innermost expectation

$$\begin{aligned}
&\mathbb{E}\left(\|f(x_0)\|_2^2\right) \\
&= \mathbb{E}\left(\mathbb{E}\left(\ldots\mathbb{E}\left(\mathbb{E}\left(\left\|x^{(L)}\right\|_2^2 \;\middle|\; \left\|x^{(L-1)}\right\|_2^2\right) \;\middle|\; \left\|x^{(L-2)}\right\|_2^2\right)\ldots \;\middle|\; \left\|x^{(1)}\right\|_2^2\right)\right) \\
&= \mathbb{E}\left(\mathbb{E}\left(\ldots\mathbb{E}\left(\frac{n_L}{2}\left(\sigma_{w,L}^2\left\|x^{(L-1)}\right\|_2^2 + \sigma_{b,L}^2\right) \;\middle|\; \left\|x^{(L-2)}\right\|_2^2\right)\ldots \;\middle|\; \left\|x^{(1)}\right\|_2^2\right)\right) \\
&= \mathbb{E}\left(\mathbb{E}\left(\ldots\frac{n_L\sigma_{w,L}^2}{2}\mathbb{E}\left(\left\|x^{(L-1)}\right\|_2^2 \;\middle|\; \left\|x^{(L-2)}\right\|_2^2\right) + \frac{n_L\sigma_{b,L}^2}{2}\ldots \;\middle|\; \left\|x^{(1)}\right\|_2^2\right)\right).
\end{aligned}$$

Repeating this last argument provides the statement that was to be shown. □

For $\sigma_{w,l}^2 \approx 2/n_l$ (as usually realized by He initialization) and our choice $\sigma_{b,l} = \prod_{m=1}^{l}\sigma_{w,m}$, this implies that $\mathbb{E}\left(\|f(x_0)\|_2^2\right) \approx \left\|x^{(0)}\right\|_2^2 + 1$, which prevents initial signal and gradient explosions even for high depth L.

'Looks linear' and orthogonal initialization The above lemma assumes that the weights and biases are drawn independently at random. This does not hold for orthogonal weight initialization, whose benefits have been highlighted in numerous works in general (Pennington et al., 2017, 2018; Saxe et al., 2014) and in particular for lottery ticket pruning (Lee et al., 2020). The marginal distribution of each weight entry is still normally distributed as $w_{ij}^{(l)} \sim \mathcal{N}(0, 1/n_l)$ so

that our lottery ticket existence proof (in Section 7.7) still applies approximately to this setting. However, the main advantage of orthogonal weight initialization for trainability is usually induced by (approximate) dynamical isometry. For ReLU activation functions, this is not achievable simply by initializing the whole matrix $W^{(l)}$ as orthogonal (Pennington et al., 2017, 2018; Burkholz and Dubatovka, 2019). The solution is in fact based on the same insight that enables all current lottery ticket existence proofs for ReLUs, where the identity can be represented by $x = \phi(x) - \phi(-x)$.

As dynamical isometry can be achieved by a Jacobian that is similar to the identity, Burkholz and Dubatovka (2019); Balduzzi et al. (2017) could ensure perfect dynamical isometry for ReLUs by a 'looks linear' initialization of the weight matrix and zero biases so that the full signal is always preserved at initialization. Effectively, each neural network layer computes $\tilde{\boldsymbol{x}}^{(l)} = \phi\left(\boldsymbol{W}_0^{(l)}\tilde{\boldsymbol{x}}^{(l-1)}\right) - \phi\left(-\boldsymbol{W}_0^{(l)}\tilde{\boldsymbol{x}}^{(l-1)}\right)$, where the matrix $\boldsymbol{W}_0^{(l)} \in \mathbb{R}^{n_l/2} \times \mathbb{R}^{n_{l-1}/2}$ is orthogonal. Extending this idea by nonzero biases corresponds, effectively, to $\tilde{\boldsymbol{x}}^{(l)} = \phi\left(\boldsymbol{W}_0^{(l)}\tilde{\boldsymbol{x}}^{(l-1)} + \boldsymbol{b}_l\right) - \phi\left(-\boldsymbol{W}_0^{(l)}\tilde{\boldsymbol{x}}^{(l-1)} - \boldsymbol{b}_l\right)$, where the matrix $\boldsymbol{W}_0^{(l)} \in \mathbb{R}^{n_l/2} \times \mathbb{R}^{n_{l-1}/2}$. Concretely, we define

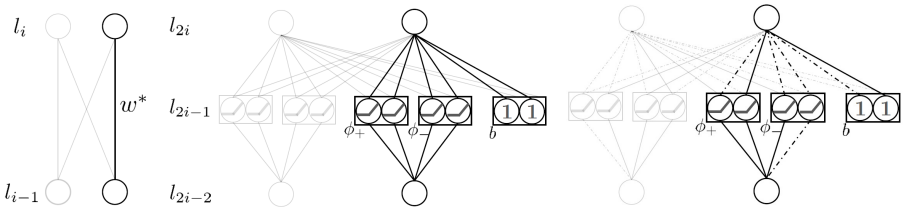
$$W^{(l)} = \begin{bmatrix} W_0^{(l)} & -W_0^{(l)} \\ -W_0^{(l)} & W_0^{(l)} \end{bmatrix},$$
$$b^{(l)} = \begin{bmatrix} b_0^{(l)} & -b_0^{(l)} \end{bmatrix}$$

for orthogonal, nonzero bias initialization with $\boldsymbol{b}_0^{(l)} \sim \mathcal{N}\left(0, \sigma_{b,l}^2 I\right)$ independently from the weights. Note that each entry of the weight matrix is again distributed as $w_{ij}^{(l)} \sim \mathcal{N}\left(0, 2/n_l\right)$ as in case of He initialization.

How should we choose the variance $\sigma_{b,l}^2$ of the biases? Similarly to Lemma 7.3, we can derive the variance of the output signal as

$$\mathbb{E}\left(\|\boldsymbol{f}(\boldsymbol{x_0})\|_2^2\right) = \left\|\mathbf{x}^{(0)}\right\|_2^2 + \sum_{l=1}^{L} \sigma_{b,l}^2 \frac{n_l}{2}.$$

Initially, the additional $\sum_{l=1}^{L} \sigma_{b,l}^2 \frac{n_l}{2}$ is easier to control than in Lemma 7.3, exactly because the weights do not scale the biases randomly. Note that we could improve this further and initialize the bias also dependent on the weights and make them cancel out to achieve again perfect dynamical isometry at initialization and $\mathbb{E}\left(||\boldsymbol{f}(\boldsymbol{x_0})||_2^2\right) = ||\mathbf{x}^{(0)}||_2^2$. However, this would depend on a carefully chosen dependence between weights and biases that gets destroyed during training and/or pruning. For that reason, we still assume a situation



(a) Target network f (b) Mother network f_0 (c) Strong lottery ticket f_ϵ

Figure 7.3: *Existence proof network construction.* (a) Two subsequent layers l_i, l_{i-1} of the target network f of depth L. (b) A mother network f_0 of depth $2L$, where intermediate layers with degree-one nodes are squeezed in between any pair of subsequent layers. The size of this layer is determined by the reduction on the subset sum problem. Bold neurons and connections are used to approximate w^* of the target f and the bias of the neurons in l_i of f. (c) Lottery ticket f_ϵ obtained from the mother network by pruning connections (dashed lines).

similar to Lemma 7.3 and initialize $\sigma_{b,l} = \prod_{m=1}^{l} \sigma_{w,m}$. This case is also supported (at least approximately) by our lottery ticket existence proof and respects the scaling of parameters as outlined in Lemma 7.2.

With the trainability of randomly initialized networks, we have fulfilled the first criterion of a good initialization proposal for nonzero biases. The second criterion, the existence of lottery tickets, is discussed in the next section.

7.7 EXISTENCE OF TICKETS WITH NONZERO BIASES

Proofs of the existence of lottery tickets have derived sufficient conditions under which pruning algorithms should have a good chance to find a winning ticket.

The first proof of the strong lottery ticket hypothesis (Malach et al., 2020) has shown that a weight-bounded deep neural target network of depth L and width n with ReLU activation functions is contained up to error ϵ with high probability in a larger deep neural network of double the depth of the target network, i.e., $2L$. Their strong requirement on the width of the large network to be of polynomial order $O(n^5 L^2/\epsilon^2)$ or, under additional sparsity assumptions, $O(n^2 L^2/\epsilon^2)$, was subsequently improved to a logarithmic dependency of the form $O(n^2 \log(nL)/\epsilon)$ for weights that follow an unusual hyperbolic distribution (Pensia et al., 2020) and $O(n \log(nL/\epsilon))$ for uniformly distributed weights (Orseau et al., 2020). The improvement is achieved by the insight that many different parametrizations exist that can compute almost the same function as the target network. All proofs have in common that two neural network layers

are needed to approximate a neuron $\phi(w^T x)$, which explains the $2L$ depth requirement.

7.7.1 SLTH WITH NONZERO BIAS INITIALIZATIONS

So far, all of these works assume that the target network has zero biases. This limits significantly the class of functions that we can hope to learn by pruning alone. To extend the existence proofs, the first question that we have to answer is: How does the error propagate through a network with nonzero biases? Similarly to Lemma 1 in (Pensia et al., 2020), we can deduce from the answer how close each parameter θ_ϵ needs to be to the target one in order to guarantee an ϵ approximation of the entire network.

Lemma 7.4 (Approximation propagation)**.** *Assume $\epsilon > 0$ and let the target network f and its approximation f_ϵ have the same architecture. If every parameter θ of f and corresponding θ_ϵ of f_ϵ in layer l fulfils $|\theta_\epsilon - \theta| \leq \epsilon_l$ for*

$$\epsilon_l := \epsilon \left(L \sqrt{n_l k_{l,max}} \left(1 + \sup_{x \in [-1,1]^{n_0}} \left\| x^{(l)} \right\|_1 \right) \prod_{k=l+1}^{L} \left(\left\| W^{(l)} \right\|_\infty + \epsilon / L \right) \right)^{-1},$$

then it follows that $\|f - f_\epsilon\|_\infty \leq \epsilon$.

Proof. Our objective is to bound $\|f - f_\epsilon\|_\infty \leq \epsilon$. We repeatedly use the triangle inequality and that $|\phi(x) - \phi(y)| \leq |x - y|$ is Lipschitz continuous with Lipschitz constant 1 to derive

$$\begin{aligned}
&\left\| x^{(l)} - x_\epsilon^{(l)} \right\|_2 \\
&\leq \left\| h^{(l)} - h_\epsilon^{(l)} \right\|_2 \\
&= \left\| W^{(l)} x^{(l-1)} + b^{(l)} - W_\epsilon^{(l)} x_\epsilon^{(l-1)} + b_\epsilon^{(l)} \right\|_2 \\
&= \left\| W^{(l)} x^{(l-1)} + b^{(l)} - W_\epsilon^{(l)} x_\epsilon^{(l-1)} + b_\epsilon^{(l)} + \underbrace{W_\epsilon^{(l)} x^{(l-1)} - W_\epsilon^{(l)} x^{(l-1)}}_{=0} \right\|_2 \\
&\leq \left\| \left(W^{(l)} - W_\epsilon^{(l)} \right) x^{(l-1)} \right\|_2 + \left\| b^{(l)} - b_\epsilon^{(l)} \right\|_2 + \left\| W_\epsilon^{(l)} \left(x^{(l-1)} - x_\epsilon^{(l-1)} \right) \right\|_2 \\
&\leq \epsilon_l \sqrt{m_l} \sup_{x \in [-1,1]^{n_0}} \left\| x^{(l-1)} \right\|_1 + \epsilon_l \sqrt{m_l} + \left(\left\| W^{(l)} \right\|_\infty + \epsilon_l \right) \left\| \left(x^{(l-1)} - x_\epsilon^{(l-1)} \right) \right\|_2
\end{aligned}$$

with $\epsilon_l \leq \epsilon / L$. m_l denotes the number of parameters in layer l that are smaller than ϵ_l and $\|W\|_\infty = \max_{i,j} |w_{i,j}|$. Note that $m_l \leq n_l k_{l,\max}$. The last inequality follows from the fact that all entries of the matrix $\left(W^{(l)} - W_\epsilon^{(l)} \right)$ and of the

vector $(\boldsymbol{b}^{(l)} - \boldsymbol{b}_\epsilon^{(l)})$ are bounded by ϵ_l and maximally m_l of these entries are nonzero. Furthermore, $\left\|\boldsymbol{W}_\epsilon^{(l)}\right\|_\infty \leq \left(\left\|\boldsymbol{W}^{(l)}\right\|_\infty + \epsilon_l\right)$ follows again from the fact that each entry of $\left(\boldsymbol{W}^{(l)} - \boldsymbol{W}_\epsilon^{(l)}\right)$ is bounded by ϵ_l.

Thus, at the last layer it holds for all $\boldsymbol{x} \in [-1,1]^{n_0}$ that

$$\begin{aligned}
&\|f(x) - f_\epsilon(x)\|_2 \\
&= \left\|\boldsymbol{x}^{(L)} - \boldsymbol{x}_\epsilon^{(L)}\right\|_2 \\
&\leq \sum_{l=1}^{L} \epsilon_l \sqrt{m_l} \left(1 + \sup_{x\in[-1,1]^{n_0}} \left\|\boldsymbol{x}^{(l-1)}\right\|_1\right) \prod_{k=l+1}^{L} \left(\left\|\boldsymbol{W}^{(l)}\right\|_\infty + \epsilon/L\right) \\
&\leq L\frac{\epsilon}{L} = \epsilon,
\end{aligned}$$

using the definition of ϵ_l as given in the theorem statement in the last step. □

Note that large weights in every layer could imply that ϵ_l is exponential in L. However, if we assume bounded weights so that $||\boldsymbol{W}^{(l)}||_\infty \leq 1$, we receive a moderate scaling of $\epsilon_l = C\epsilon/L$, where C depends on the maximum degree of the neurons $k_{l,\max} \leq n_{l-1} + 1$ and the size of the biases via $\sup_{x\in[-1,1]^{n_0}} ||\boldsymbol{x}^{(l)}||_1$. As we expect each output component of the target network f to be in $[0,1]$, reasonable choices of biases lead usually to $\sup_{x\in[-1,1]^{n_0}} ||\boldsymbol{x}^{(l)}||_1 \leq n_{l-1}$ and thus $\epsilon_l = \epsilon/(L(n_l+1)(n_{l-1}+1)e)$. Otherwise, we could rescale all parameters and thus the output to ensure desirable scaling. However, this would come at the expense of adapting the allowed error ϵ accordingly.

Next, we extend the proof of the existence of lottery tickets in Orseau et al. (2020) to nonzero biases. In addition, we generalize it to domains $[-1,1]^{n_0}$ (instead of balls with radius 1) and present sharper width estimates based on the in-degrees of neurons instead of the full target network width n_l. The big advantage of our initialization scheme is that we can directly transfer an approach that would assume uniformly distributed parameters in $\theta_i \sim U([-1,1])$.

Theorem 7.1 (Existence of lottery ticket)**.** *Assume that $\epsilon, \delta \in (0,1)$ and a target network f with depth L and architecture $\bar{n}$ are given. Each weight and bias of a larger deep neural network f_0 with depth $2L$ and architecture $\bar{n}_0$ is initialized independently, uniformly at random according to $w_{ij}^{(l)} \sim U([-\sigma_{w,l}, \sigma_{w,l}])$ and $b_i^{(l)} \sim U([-\prod_{k=1}^{l}\sigma_{w,k}, \prod_{k=1}^{l}\sigma_{w,k}])$. Then, with probability at least $1-\delta$, f_0 contains an approximation $f_\epsilon \subset f_0$ so that $\|f - \lambda f_\epsilon\|_\infty \leq \epsilon$ if for $l = 1, ..., L$*

$$n_{2l-1,0} = Cn_{l-1}\log\left(\frac{k_{l-1,max}n_l}{\min\{\epsilon_l, \delta/L\}}\right) \text{ and } n_{2l,0} = n_l,$$

where ϵ_l is given by Lemma 7.4 and the output is scaled by $\lambda = \prod_{l=1}^{2L} \sigma_{w,l}^{-1}$.

A similar statement holds also for normal distributions, i.e., $w_{ij}^{(l)} \sim \mathcal{N}\left(0, \sigma_{w,l}^2\right)$ and $b_i^{(l)} \sim \mathcal{N}\left(0, \prod_{k=1}^{l} \sigma_{w,k}^2\right)$. Note that, essentially, we receive the same scaling as in case of zero biases. Only the maximum degree $k_{l-1,\max}$ is modified by $+1$. The reason is that we can treat each bias as an additional weight in the construction of a ticket. We provide the full proof in the appendix and restrict ourselves in the following to the description of the main idea.

Proof Idea. Layer $2l-1$ and $2l$ of the large network serve the representation of the neurons in Layer l of the target network, e.g, neuron $\phi\left(\sum_j w_{ij} x_j^{(l-1)} + b_i\right)$. By using the identity $x = \phi(x) - \phi(-x)$ for ReLUs, we can express the preactivation also as $\sum_j \phi\left(w_{ij} x_j^{(l-1)}\right) - \phi\left(-w_{ij} x_j^{(l-1)}\right) + \operatorname{sign}(b_i)\phi(|b_i|)$. We provide a visualization of this construction in Fig. 7.3, where we highlight the construction for a single weight and neuron in the target, through one term $\phi\left(w_{ij} x_j^{(l-1)}\right)$ and bias $\phi(|b_i^{(l-1)}|)$, which are each approximated through a subset sum construction of randomly initialized weights and biases in f_0. The width of the intermediate layers needs to be only of order $\log(1/\epsilon)$ according to results by Lueker (1998) on the subset sum problem, which can be applied to finding lottery tickets (Pensia et al., 2020). Accordingly, with probability at least $1-\delta$, for each parameter θ (i.e., w_{ij}, $-w_{ij}$, or $|b_i|$) exists a subset S of n uniformly distributed parameters $X_i \sim U([-1,1])$ so that $|\theta - \sum_{i \in S} X_i| < \epsilon_l$ for $n \geq C \log 1/\min\{\epsilon_l, \delta\}$. Repeating this argument in combination with union bounds over k_i parameters per neuron, all neurons in a layer, and all layers, leads to the desired results. While this construction is purely theoretical in nature, it turns out that the mother network and approximation of the target have properties that are comparable to real world lottery ticket settings.

7.7.2 Proof construction – from theory to practice

We, here, conduct a study on the efficacy of the existence proof construction with respect to the networks f_0 and f_ϵ, the code is available online.[1] In particular, we show that the proof construction of mother networks f_0 of Thm. 7.1 admits for efficient approximations f_ϵ for a given target network f. As a case study, we consider a LeNet (LeCun et al., 1998) architecture of the form $784 \to 300 \to 100 \to 10$, where each fully connected layer is followed by ReLU activations. We train such a He initialized LeNet by Iterative Magnitude Pruning (Frankle and Carbin, 2019) on MNIST, which results in a small target network with around

[1] https://github.com/RelationalML/NonZeroBiases/releases/tag/nonzerobias

	Initialization	% Acc.	# Param.	Sparsity
Target f	He	97.96	18697	
Appr. f_ϵ	Orthogonal	97.98	106192	0.0027
Appr. f_ϵ	Uniform	97.94	112157	0.0029
Appr. f_ϵ	Normal	97.96	121153	0.0031

Table 7.1: *Constructed target networks.* Results of constructing lottery tickets to a pruned target network from LeNet on MNIST. Sparsity refers to the sparsity of approximations f_ϵ with respect to the mother network f_0.

20k parameters, and an accuracy of 97.96. We consider the resulting ticket as target network f.

As detailed in the Theorem 7.1, each layer in the target network can be approximated by two subsequent layers in a mother network f_0 of depth $2L$. Our approximation to the target lottery ticket is a subnetwork of this mother network, $f_\epsilon \subset f_0$. We construct each layer of f_ϵ by solving the subset sum problem neuron-wise, aiming to find the smallest subset of weights in f_0 that allow to approximate each weight in f, as implied by the proof. Our experiments verify that the constructed SLTs indeed do as well as the target network on MNIST data for our proposed nonzero initialization schemes, including a 'looks linear' orthogonal initialization (see Tab. 7.1). Furthermore, these tickets are of great sparsity with respect to the constructed mother network and only a factor of ~ 6 larger than the (real) target, providing evidence that extremely sparse solutions exist in large, overparametrized neural networks. Finally, although purely theoretical, the construction used within the proof leads to a mother network f_0 of architecture $784 \to 31400 \to 300 \to 12040 \to 100 \to 4040 \to 10$, which is well within the size of modern neural networks that are considered for pruning.

7.8 PARAMETER SCALING DURING PRUNING

According to Sec. 7.6.1 and our existence proof, we expect that the output of a pruned network usually does not match the right target range. The lottery ticket f_ϵ needs to be scaled by a factor $\lambda > 0$ (and usually $\lambda > 1$) . In the existence proof, $\lambda = \prod_{m=1}^{L} \sigma_{w,m}$ but this factor can be vanishing small for very deep networks. Furthermore, in many applications we do not know the exact size the of the network parameters and the output might also not be restricted to $[-1, 1]$. For these reasons, we propose to learn an appropriate output scaling factor $\lambda > 0$ that successively adapts the lottery ticket f_ϵ after each pruning

epoch. For regression minimizing the mean squared error with respect to N data samples with targets $y_{i,s}$, this scalar can be easily computed as

$$\lambda_{\text{mse}} = \left(\sum_{s=1}^{N} \sum_{i=1}^{n_L} y_{i,s} x_{i,s}^{(L)} \right) / \left(\sum_{s=1}^{N} \sum_{i=1}^{n_L} x_{i,s}^{(L)} x_{i,s}^{(L)} \right).$$

In case of a different loss $\mathcal{L}$, we only have to solve a one-dimensional optimization problem of the form

$$\min_{\lambda > 0} \mathcal{L}(y, \lambda x^{(L)})$$

which can, for instance, be achieved with SGD. To distribute the scaling factor on the different layers, we use again the parameter transformation in Lemma 7.2

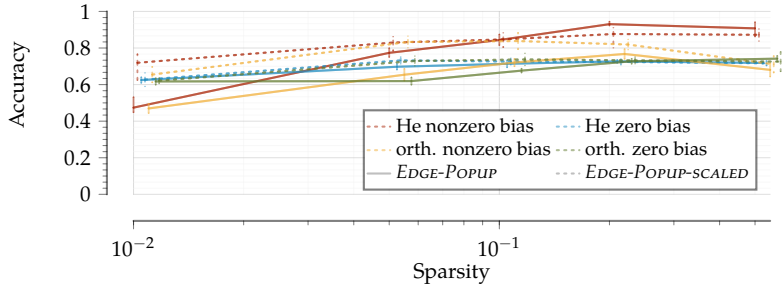
$$w_{ij}^{(l)} = w_{ij}^{(l)} \lambda^{1/L}, \quad b_i^{(l)} = b_i^{(l)} \lambda^{l/L},$$

which ensures that the overall output of the neural network is scaled by λ.

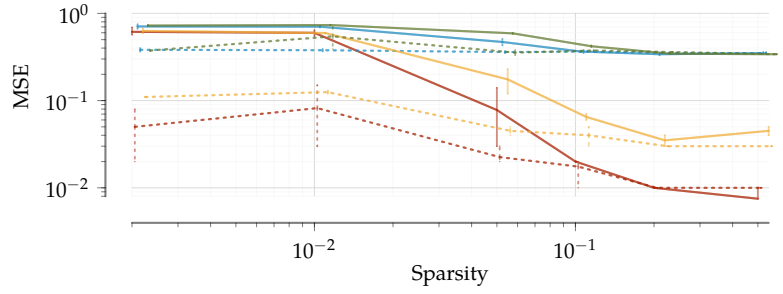
Combining this parameter rescaling for tickets found in each epoch with the SLT algorithm Edge-Popup, along with a slow annealing of the target sparsity throughout the course of pruning, we obtain Edge-Popup-scaled, for which we report pseudocode in Alg. A.1. As we show next, this parameter rescaling allows us to obtain much sparser lottery tickets than pure pruning.

7.8.1 A synthetic case study

We compare Edge-Popup with annealing of target sparsities against a version with scaling Edge-Popup-scaled for networks initialized with and without biases. On two synthetic benchmarks, which turn out to be challenging settings for SLT pruning, we evaluate these methods for both He as well as 'looks-linear' orthogonal initialization with and without nonzero bias extensions. All experiments are carried out on commodity hardware. `Edge-popup` training was conducted in the default way suggested by the original authors by SGD with momentum of 0.9 and weight decay 0.0005, combined with cosine annealing of the learning rate starting from 0.1. We train for $e = 10$ epochs and $e_a = 5$ annealing epochs in case of the shifted ReLU example and $e_a = 20$ annealing epochs in case of the ellipse for sparsity levels $0.01, 0.05$. In case of higher sparsity levels, $e_a = 10$ annealing epochs are sufficient. During annealing, we slowly reduce the sparsity over time by ρ^{i/e_a}, where ρ is the desired network sparsity, and i is the current epoch. We used a batch size of 32 in all experiments



(a) Ellipse data, higher is better.



(b) Shifted ReLU data, lower is better.

Figure 7.4: *Strong lottery ticket pruning with* Edge-Popup. Shown are results for discovered tickets of different target sparsities and initialization schemes obtained through Edge-Popup and Edge-Popup-scaled. We report the mean and obtained minimum and maximum performance on the validation data as error bars across 5 repetitions.

and report mean based on 5 repetitions. All code and data is publicly available[2]

A Shifted ReLU

First, we consider data of a regression problem that follows a shifted ReLU function $\phi_b(x) = \max(0, x + b)$ with $b = 0.5$. We thus draw $N = 10^4$ iid samples $x_i \sim U[-1, 1]$ with targets $y_i = \phi(x_i + 0.5) + n_i$ with independent Gaussian noise $n_i \sim \mathcal{N}(0, 0.01^2)$. For a network of depth 5, where each layer is of width 100, we retrieve strong lottery tickets at target sparsities $\{0.002, 0.01, 0.05, 0.1, 0.2\}$. We report the mean squared error (MSE) of the strong tickets on the test set in Fig. 7.4.

We observe that, consistently, the nonzero bias initialized networks enable the recovery of strong tickets with orders of magnitude smaller errors than in networks with zero-initialized bias. Furthermore, Edge-Popup-scaled with proper rescaling of parameters consistently outperforms the vanilla (unscaled) algorithm for extreme sparsities. At lower sparsity levels, rescaled Edge-Popup allows to retrieve well performing tickets that match the low error of their more dense counterparts, whereas vanilla Edge-Popup fails to find good tickets.

The Onion Slice

Next, we consider a classification problem, where points are arranged in elliptic rings, and each point is labeled by the ring it appears in. $N = 10^4$ inputs are again sampled iid from uniform distributions $x_1, x_2 \sim U[-1, 1]$ and one of four labels is assigned as target based on the value $y = 0.5(x_1 - 0.3)^2 + 1.2(x_2 + 0.5)^2$. Class boundaries are defined as $(0.2, 0.5, 0.7)$, while noise is introduced by flipping a label to a neighboring class with probability 0.01.

For networks of depth 5 and width 100, we retrieve strong lottery tickets at target sparsities $\{0.01, 0.05, 0.1, 0.2, 0.5\}$. We report the accuracy of tickets on the test set in Fig. 7.4. Tickets pruned from networks initialized with nonzero biases outperform their zero-bias counterparts. An exception to this rule is given by the unscaled Edge-Popup for sparsity 0.01, where both initialization approaches (with nonzero and zero biases) show unsatisfactory performance. In contrast, the rescaled Edge-Popup with nonzero bias He initialization is still able to retrieve extremely sparse tickets with more than 10 accuracy points margin to all other approaches.

[2] https://github.com/RelationalML/NonZeroBiases/releases/tag/nonzerobias

7.9 DISCUSSION & CONCLUSION

Strong lottery tickets, as currently defined in the literature, do not lend themselves as universal function approximators due to the limitation to the standard zero bias initialization schemes for neural networks. We hence transferred the strong lottery ticket hypothesis to neural networks with potentially nonzero initial biases and proved the existence of strong lottery tickets under realistic conditions with respect to the network width and initialization scheme. This generalization equips training by pruning for strong lottery tickets with the universal approximation property.

Along with the proof, we have extended standard initialization schemes to nonzero biases and formally shown that our proposal defines well trainable neural networks, while they support the existence of strong lottery tickets. These initialization schemes include the 'looks-linear' approach (Burkholz and Dubatovka, 2019; Balduzzi et al., 2017) that ensures initial dynamical isometry of ReLU networks, which often leads to favorable training properties.

Based on our theoretical insights, we have derived a parameter scaling strategy that enables pruning algorithms to find sparser strong lottery tickets. We have extended the Edge-Popup algorithm (Ramanujan et al., 2020) for strong lottery ticket pruning accordingly and demonstrated the utility of our innovations on two case studies. For imaging data, our nonzero bias initializations are well trainable, but the current generation of algorithms lacks the ability to draw an advantage over zero bias initialized networks (see App. A.6.4), likely due to their parameter-inefficacy (Fischer and Burkholz, 2022). With the development of pruning algorithms that can find highly sparse strong lottery tickets, we anticipate that nonzero bias initializations are important for lottery ticket pruning in theory as well as practice. In the next chapter, we leverage our nonzero bias initializations to construct and hide ground truth lottery tickets to answer fundamental question about current LTH algorithms.

PLANT 'N' SEEK: CAN YOU FIND THE WINNING TICKET?

8

8.1 INTRODUCTION

The lottery ticket hypothesis bears the promise of resource efficient training and deployment of highly performant neural networks. The existence of strong LTs, which do not need any further training at all, has been proven formally for networks without (Malach et al., 2020; Pensia et al., 2020; Orseau et al., 2020) and with potentially nonzero biases (Fischer et al., 2021a). While these types of proofs show existence in realistic settings, the sparsity of the constructed tickets is likely not optimal, as they represent a target parameter by multiple neurons of degree 1. The construction and proof of the generalized strong LT hypothesis raises two question – is the suboptimal sparsity merely an artifact of existence proofs or a general limitation of the pruning approach? And, if very sparse tickets exist, are current algorithms able to find them or are further improvements needed to achieve effective network compression? These questions cannot be answered by comparing LT pruning algorithms solely on standard benchmark datasets (Frankle et al., 2021), but demand the comparison with known ground truth LTs. The lack thereof was raised as an issue by Frankle et al. (2021). To fill this gap and generate baselines with known ground truth, we here propose an algorithm to plant and hide arbitrary winning tickets in

This chapter is based on Fischer and Burkholz (2022).

randomly initialized NNs and construct sparse tickets that reflect common challenges in machine learning. We use this experimental set-up to compare state-of-the-art pruning algorithms designed to search for lottery tickets.

Our results indicate that state-of-the-art methods achieve only sub-optimal sparsity levels. This suggests that previous challenges to identify highly sparse winning tickets as subnetworks of randomly initialized dense networks (Frankle et al., 2020; Ramanujan et al., 2020) can be explained by algorithmic limitations rather than fundamental problems with LT existence. In our experiments, the qualitative trends how methods compare to each other are consistent with previous results on image classification tasks (Tanaka et al., 2020; Frankle et al., 2021) indicating that our experimental set-up exposes pruning algorithms to realistic challenges. In addition, we identify an opportunity to improve state-of-the-art pruning algorithms in order to find strong LTs of better sparsity. Our proposed planting framework will enable the evaluation of future progress in this direction.

8.2 Related work

We gave an overview over the relevant LT pruning literature in the last chapter and, here, briefly discuss literature in the broader scope of finding small, well performing NNs. Apart from LT pruning, different approaches have been developed to reduce computational resources and perform structure learning, including dynamic sparse training (Evci et al., 2020; Liu et al., 2021b), adaptations (Frankle et al., 2020; Renda et al., 2020; Liu et al., 2021a) of Iterative Magnitude Pruning (IMP) (Han et al., 2015; Frankle and Carbin, 2019) and sparse regularization techniques (Weigend et al., 1991; Savarese et al., 2020). As these approaches do not identify LTs as subnetworks of randomly initialized NNs, they do not rely on the existence of planted tickets and are therefore beyond the scope of our experimental analysis. However, the ground truth tickets which we derived for planting could still provide an interesting baseline to explore whether sparse training of deep NNs can identify extremely sparse, hand designed NN architectures.

Dense NNs are known to find NN representations that are less sparse than hand crafted architectures (Denker et al., 1987), yet, the the explicit objective of sparse training is to address this issue. We provide the tools to evaluate progress in this direction by planting known ticket architectures. While the ultimate goal of deep learning is to solve problems with otherwise unknown solutions such as image classification (Frankle et al., 2021) or protein structure prediction (Tunyasuvunakool et al., 2021), the design of NN architectures for human solvable problems has already in the past provided important insights into NN properties, including universal approximation (Scarselli and Tsoi, 1998;

Yarotsky, 2018) or the importance of algorithmic alignment (Xu et al., 2020). NNs that compute polynomials (Scarselli and Tsoi, 1998; Yarotsky, 2018), xor gates (Rumelhart et al., 1986), discrete fast fourier transformation (Velik, 2008), symmetry groups (Sejnowski et al., 1986), general piecewise linear functions (Arora et al., 2018), or argmax (Xu et al., 2020) could also present interesting candidates for planting in future investigations.

8.3 EXISTENCE OF STRONG LOTTERY TICKETS

Pruning algorithms that search for strong lottery tickets achieve sparsity levels of around 0.5 but not substantially smaller if the resulting models should be able to compete with the accuracy of the entire, trained mother network (Ramanujan et al., 2020). Proofs of the existence of strong lottery tickets give no clear indication whether this is an algorithmic shortcoming, which could be overcome, or a fundamental limitation of pruning randomly initialized networks alone. The reason is that existing proofs (Malach et al., 2020; Pensia et al., 2020; Orseau et al., 2020; Fischer et al., 2021a) guarantee high existence probabilities of subnetworks that have double the depth and $2-30$ times the width of the target network and thus non-optimal sparsity. Based on their $2L$ construction, Malach et al. (2020) even went so far to conclude that training by pruning might be computationally at least as hard as training shallower NNs. However, it is well known that specific function classes can be approximated in significantly more parameter efficient ways by deeper NNs rather than shallower ones (Mhaskar et al., 2017; Yarotsky, 2018) and also be learned more efficiently (Schmidt-Hieber, 2020). Thus, by leveraging its full depth, the randomly initialized $2L$ deep NN might contain a much sparser strong LT than any of the ones whose existence has been proven.

As a first step towards making claims about the existence of very sparse representations, we therefore prove next a lower bound on the probability that a target NN of general architecture is contained in a larger, randomly initialized NN with the same depth as the target network. As many relevant targets have known representations of lower sparsity than what is covered by this bound, we will afterwards propose a planting algorithm to design experiments that can distinguish between algorithmic and fundamental limitations of pruning for strong LTs.

8.3.1 LOWER BOUND ON EXISTENCE PROBABILITY

Pruning a randomly initialized NN usually finds a strong LT that is close to a target network but does not recover the original parameters exactly. First, we need to understand how these errors in the parameters affect the final

network output and what error sizes are acceptable. For completeness, we restate Lemma 7.4 discussed in the previous chapter that guarantees an ϵ approximation of the entire network.

Lemma 8.1 (Error propagation). *Assume $\epsilon > 0$ and let the target network f and its approximation f_ϵ have the same architecture. If every parameter θ of f and corresponding θ_ϵ of f_ϵ in layer l fulfils $|\theta_\epsilon - \theta| \leq \epsilon_l$ for*

$$\epsilon_l := \epsilon \left(L\sqrt{n_l k_{l,max}} \left(1 + \sup_{x \in [-1,1]^{n_0}} \left\| \boldsymbol{x}^{(l)} \right\|_1 \right) \prod_{k=l+1}^{L} \left(\left\| \boldsymbol{W}^{(l)} \right\|_\infty + \epsilon / L \right) \right)^{-1},$$

then it follows that $\|f - f_\epsilon\|_\infty \leq \epsilon$.

Respecting the allowed errors ϵ_l, we can next establish a lower bound on the existence probability of a specific target network assuming standard initialization schemes with necessary nonzero bias initialization. The main argument is a union bound over matching each target neuron i (with k_i parameters) with neurons of the mother network in the corresponding layer.

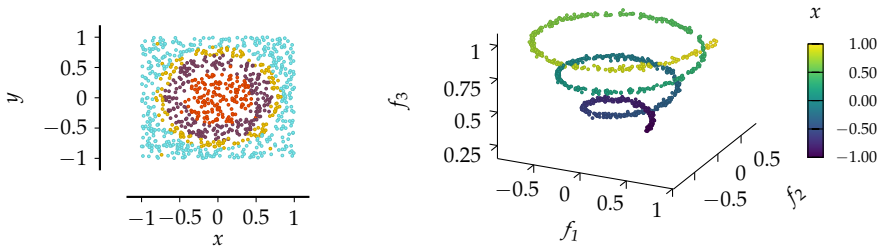
Theorem 8.1 (Lower bound on existence probability). *Assume that $\epsilon \in (0,1)$ and a target network f with depth L and architecture $\tilde{n}$ are given. Each parameter of the larger deep neural network f_0 with depth L and architecture $\tilde{n}_0$ is initialized independently, uniformly at random with $w_{ij}^{(l)} \sim U\left(\left[-\sigma_w^{(l)}, \sigma_w^{(l)}\right]\right)$ and $b_i^{(l)} \sim U\left(\left[-\prod_{k=1}^{l} \sigma_w^{(k)}, \prod_{k=1}^{l} \sigma_w^{(k)}\right]\right)$. Then, f_0 contains a rescaled approximation f_ϵ of f with probability at least*

$$\mathbb{P}\left(\exists f_\epsilon \subset f_0 : \ \|f - \lambda f_\epsilon\|_\infty \leq \epsilon\right) \geq \prod_{l=1}^{L} \left(1 - \sum_{i=1}^{n_l} (1 - \epsilon_l^{k_i})^{n_{l,0}} \right),$$

where ϵ_l is defined as in Eq. (7.4) and the scaling factor is given by $\lambda = \prod_{l=1}^{L} 1/\sigma_w^{(l)}$.

We could obtain similar results for initially normally distributed weights and biases, we would just have to substitute ϵ_l by $\epsilon_l/2$. A proof is provided in Appendix A.7.2.

Thm. 8.1 provides us with an intuition for what kind of targets we can expect to find. First of all, it tells us that a large number of nodes in a layer, and more importantly nodes with large in-degree k_i, render the existence of a specific network architecture as strong LT less likely. Each additional layer reduces the probability further. Moreover, we observe that the last layer is a bottleneck, as it usually has the same width as in the large initial network. A higher width of the mother network is clearly advantageous. Note that we could turn this theorem also into a lower bound on the width $n_{l,0}$ of the larger



(a) `Circle` of input (x, y) and output classes indicated by color.

(b) `Helix` with input x indicated by color, and output coordinates $(f_1(x), f_2(x), f_3(x))$ in euclidean space.

Figure 8.1: *Benchmark data.* Shown are samples from the `Circle` (left) and `Helix` (right) task.

mother network as it is common in existence proofs. Fig. 11 in Appendix A.7.2 supports this intuition with the visualization of an example. Assuming the same width $n_{l,0} = n_0$ and $n_l = n$ across layers, we would receive roughly $n_0 \geq C \log(Ln/\delta) \max_l \left(\epsilon_l^{-k_{\max}}\right)$. Even though it is polynomial in the relevant parameters, it only provides a practical existence proof for extremely sparse architectures. We therefore have to resort to planting to answer fundamental questions about abilities of pruning algorithms. In fact, the proof of the above theorem inspires the planting algorithm introduced next.

8.3.2 PLANTING STRONG LOTTERY TICKETS

As we have discussed, the LTs that exist with high probability rarely fulfill criteria of interest, such as low sparsity, favorable generalization properties, or adversarial robustness. We therefore propose to plant winning tickets with such desirable properties within randomly initialized neural networks. This approach offers the flexibility to design experiments of different degrees of difficulty and generate training and test data based on a ground truth.

A simple approach to planting a target f in a network f_0 would be to select a random subset of neurons in each layer and set them to their target values and otherwise randomly initialize the rest. This, however, would usually lead to a trivially detectable ticket because the target parameters are much larger than the initialized parameters of the larger mother network. The reason is that both networks produce output that lies in a similar range (ideally the one of the training labels). Yet, the target network has to achieve this by adding up a much smaller number of parameters. A different perspective on the same issue

is that a pruned lottery ticket needs to be scaled up to compensate for the lost parameters. Note the scaling factor λ in Theorem 8.1 for that purpose. Thus, at least, we would need to scale the target parameters appropriately during planting. To cover natural degrees of freedom in ReLU networks, we instead allow for different neuron-wise scaling factors $\lambda_j > 0$. Note that such a scaling can be compensated by rescaling of parameters in the next layer, as

$$x_k^{(l+1)} = \phi\left[\sum_i (w_{kj}^{(l+1)}/\lambda_j)\phi\left(\sum_i (w_{ji}^{(l)}\lambda_j)x_i^{(l-1)} + (b_j^{(l)}\lambda_j)\right) + b_k^{(l+1)}\right].$$

To apply only a small change to the random mother network during planting, we choose λ_j together with matching target neurons and mother network neurons. We search in each layer of f_0 for suitable neurons that best match a target neuron in f, starting from the first hidden layer. Given that we matched all neurons in layer $l-1$, we try to establish their connections to neurons in the current layer l. A best match is decided by minimizing the l2-distance to its potential input parameters thereby adjusting for an optimal scaling factor. For example, let neuron i in Layer l of the target f have nonzero parameters $(b, \boldsymbol{w})$ that point to already matched neurons in Layer $l-1$. Each of the matched neurons j' in Layer $l-1$ of f_0 has been previously associated to a scaling factor $\lambda_{\text{old},j'}$ such that the corrected $\boldsymbol{\theta} = (b, \boldsymbol{w}\boldsymbol{\lambda}_{\text{old}})$ parameters would compute the correct neuron in f_0. In Layer l of f_0, each neuron j that we have not matched yet could be a potential match for i. Let the corresponding parameters of j be $\boldsymbol{m}$. The match quality between i and j is assessed by

$$q_{\boldsymbol{\theta}}(\boldsymbol{m}) = \|\boldsymbol{\theta} - \lambda(\boldsymbol{m})\boldsymbol{m}\|_2,$$

where $\lambda(\boldsymbol{m}) = \boldsymbol{\theta}^T\boldsymbol{m}/\|\boldsymbol{m}\|_2^2$ is the optimal scaling factor. The best matching parameters $\boldsymbol{m}^* = \text{argmin}_{\boldsymbol{m}}\, q_{\boldsymbol{\theta}}(\boldsymbol{m})$ are replaced by rescaled target parameters $\boldsymbol{\theta}/\lambda(\boldsymbol{m}^*)$ in f_0 and we remember the scaling factor $\lambda(\boldsymbol{m}^*)$ to consider matches of neurons in Layer $l+1$. Note that this rescaling is necessary to ensure that the neuron is properly hidden and attains similar values as other non-planted neurons in f_0. We provide pseudocode as Algorithm 8.1.

Matching neurons thoroughly can be computational resource intensive, if the target network f consists of a high number of neurons, because each neuron needs to be compared with most of the neurons in the mother network or at least a significant share of candidate neurons. A fast alternative is to pick a random neuron in the mother network as a match and choose an appropriate scaling factor (see Algorithm 8.2).

We use this algorithm to plant the following three exemplary targets that expose pruning algorithms to different but common challenges related to high sparsity.

Algorithm 8.1: Planting

input : target f, larger neural network f_0
output: f_0 with planted f ($f \subset f_0$), output scaling factors $\boldsymbol{\lambda}$

Initialize $\boldsymbol{\lambda}_{\text{old}} = [1]^{n_0}$ // scaling factors for input are 1
for l = 1 to $L - 1$ **do**
  for all neurons i of f in Layer l **do**
    $\boldsymbol{\theta} := (b, \boldsymbol{w}\boldsymbol{\lambda}_{\text{old}})$ // scaled parameters of neuron i in f
    $\boldsymbol{m}^* = \text{argmin}_{\boldsymbol{m}}\, q_{\boldsymbol{\theta}}(\boldsymbol{m})$ // find best match for i in f_0
    Replace $\boldsymbol{m}^*$ in f_0 by $\boldsymbol{\theta}/\lambda(\boldsymbol{m}^*)$
    $\lambda_i = \lambda(\boldsymbol{m}^*)$ // remember scaling factor of i in f_0
  end
  $\boldsymbol{\lambda}_{\text{old}} = \boldsymbol{\lambda}$
end
return $f_0, \boldsymbol{\lambda}$

Construction of targets for planting

Based on the proposed planting algorithm, we generate sparse tickets for three problems that expose general pruning algorithms to common challenges in machine learning: a basic classification problem, regression problem, and manifold learning problem. On purpose, these are designed to avoid high computational burdens and, most importantly, have sparse neural network architectures with variable depth.

ReLU unit Apart from a trivial function $f(x) = 0$, a univariate ReLU unit $f(x) = \phi(x) = \max(x, 0)$ is the most sparse lottery ticket that is possible. Assuming a mother network f_0 of depth L, a ReLU can be implemented with a single neuron per layer. Any path through the network with positive weights $\prod_{l=1}^{L} \phi(w_{i_{l-1}i_l} x)$ defines a ReLU with scaling factor $\lambda = \prod_{l=1}^{L} w_{i_{l-1}i_l}$ for indices i_l in Layer l with $w_{i_{l-1}i_l} > 0$.

Note that each random path fulfills this criterion with probability 0.5^L so that even random pruning could have a considerable chance to find an optimal ticket. A winning path exists with probability $\prod_{l=1}^{L}(1 - 0.5^{n_{l,0}})$, which is almost 1 even in small mother networks. Thus, planting is not really necessary in this case. Since not all pruning algorithms set biases to zero, however, we still set all randomly initialized biases along a winning path to zero to make the problem easier.

As we see in experiments, despite this simplification, pruning algorithms

Algorithm 8.2: Faster planting by random matching

input : target f, larger neural network f_0
output: f_0 with planted f ($f \subset f_0$), output scaling factors $\boldsymbol{\lambda}$

Initialize $\boldsymbol{\lambda}_{\text{old}} = [1]^{n_0}$ // scaling factors for input are 1
for l = 1 to $L - 1$ **do**
  for all neurons i of f in Layer l **do**
    $\boldsymbol{\theta} := (b, \boldsymbol{w}\boldsymbol{\lambda}_{\text{old}})$ // scaled parameters of i in f
    $\boldsymbol{m}^* =$ parameters of random unmatched neuron in f_0
    Replace $\boldsymbol{m}^*$ in f_0 by $\boldsymbol{\theta}/\lambda(\boldsymbol{m}^*)$
    $\lambda_i = \lambda(\boldsymbol{m}^*)$ // remember scaling factor of i in f_0
  end
  $\boldsymbol{\lambda}_{\text{old}} = \boldsymbol{\lambda}$;
end
return $f_0, \boldsymbol{\lambda}$

are severely challenged in finding an optimally sparse ticket. Even though basic, a ReLU unit seems to be a suitable benchmark that is a common building block of other tickets.

CIRCLE For simplicity, we restrict ourselves to a 4-class classification problem with 2-dimensional input. The output is therefore 4-dimensional, where each output unit $f_c(x)$ corresponds to the probability $f_c(x)$ that an input $(x_1, x_2) \in [-1, 1]^2$ belongs to the corresponding class c with $c = 0, 1, 2, 3$. As common, this probability is computed assuming softmax activation functions in the last layer. The decision boundaries are defined in the last layer based on inputs of the form $g(x_1, x_2)$. The role of the first layers with ReLU activation functions of a `Circle` target f is to compute the function $g(x_1, x_2) = x_1^2 + x_2^2$, which is fundamental to many problems, in particular to the computation of radial symmetric functions.

The high symmetry of $g(x_1, x_2)$ allows us to construct a particularly sparse representation by mirroring data points along axes as visualized in Figure 8.2 (a). With the first two layers ($l = 1, 2$), we map each input vector (x_1, x_2) to the first quadrant by defining $x_1^{(1)} = \phi(x_1) + \phi(-x_1)$ and $x_2^{(1)} = \phi(x_2) + \phi(-x_2)$. Thus, Layer $l = 1$ consists of 4 neurons, i.e., $x_1^{(1)} = \phi(x_1)$, $x_2^{(1)} = \phi(-x_1)$, $x_3^{(1)} = \phi(x_2)$, $x_4^{(1)} = \phi(-x_2)$, while Layer $l = 2$ consists of 2 neurons, i.e., $x_1^{(2)} = \phi\left(x_1^{(1)} + x_2^{(1)}\right)$, $x_2^{(2)} = \phi\left(x_3^{(1)} + x_4^{(1)}\right)$.

Each consecutive layer l mirrors the previous layer $(x_1^{(l-1)}, x_2^{(l-1)})$ along

the axis $\boldsymbol{a}^{(l)} = (\cos(\pi/2^{l-1}), \sin(\pi/2^{l-1}))$. It achieves this by mapping the neurons of the previous layer to three neurons, one representing the component of $\boldsymbol{x}^{(l-1)}$ that is parallel to $\boldsymbol{a}^{(l)}$, and two neurons that each represent the positive or negative signal component that is perpendicular to the axis $\boldsymbol{a}^{(l)}$. The last two neurons could be added to a single neuron in the next layer if we want to decrease the width of some layers to 2 in between. To take more advantage of the allowed depth L, we map three neurons immediately to the next three neurons that represent the mirroring by defining

$$\begin{aligned} x_1^{(l)} &= \phi\left(a_1^{(l)} x_1^{(l-1)} + a_2^{(l)} x_2^{(l-1)} - a_2^{(l)} x_3^{(l-1)}\right), \\ x_2^{(l)} &= \phi\left(a_2^{(l)} x_1^{(l-1)} - a_1^{(l)} x_2^{(l-1)} + a_1^{(l)} x_3^{(l-1)}\right), \\ x_3^{(l)} &= \phi(-h_2^{(l)}). \end{aligned}$$

If the depth of our network is high enough, we could use the parallel component $x_1^{(l)}$ as estimate of the radius of the input. To enable higher precision for networks of smaller depth, however, we also apply to each remaining component a piecewise linear approximation of the univariate function $h(x) = x^2$ and add those two components. Note that any univariate function can be easily approximated by a neural network of depth $L = 2$. The precise approach is explained in our next example.

HELIX To test the ability of pruning algorithms to detect lower dimensional submanifolds, we approximate a helix with three output coordinates $f_1(x) = (5\pi + 3\pi x) * \cos(5\pi + 3\pi x)/(8\pi)$, $f_2(x) = (5\pi + 3\pi x) * \sin(5\pi + 3\pi x)/(8\pi)$, and $f_3(x) = (5\pi + 3\pi x)/(8\pi)$ for 1-dimensional input $x \in [-1, 1]$. As we have observed that many pruning algorithms have the tendency to keep a higher number of neurons closer to the input (and sometimes also the output layer), we construct a ticket that has similar properties. This should ease the task for pruning algorithms to find the planted winning ticket.

Each of the components $f_i(x)$ is an univariate function that we can approximate by an univariate deep neural network $n_i(x)$ that encodes a piece-wise linear function (see Figure 8.2 (c) for an explanation). As neural networks are generally overparameterized, we have multiple options to represent $n_i(x)$. For simplicity, we write it as composition of the identity with a depth $L = 2$ univariate network $g_i(x)$ of width N in the hidden layer, which can be written as

$$g_i(x) = \sum_{j=1}^{N} a_j^{(i)} \phi(p_j(x - s_j)) + b^{(i)},$$

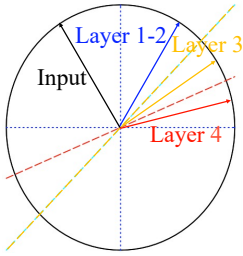
where the signs $p_j \in \{-1, 1\}$ can be chosen arbitrarily (and we chose alternating signs to create diversity). The knots $\boldsymbol{s} = (s_j)_{j\in[N]}$ mark the boundaries of the linear regions and $\boldsymbol{a} = (a_j^{(i)})_{j\in[N]}$ indicate changes in slopes $m_j^{(i)} = \big(f_i(s_{j+1}) - f(s_j)\big) \,/\, \big(s_{j+1} - s_j\big)$ (with $s_{N+1} := s_N + \epsilon$) from one linear region to the next. $a_j^{(i)} = m_j^{(i)} - m_{j-1}^{(i)}$ for $2 \leq i \leq N$, $a_1^{(i)} = m_1^{(i)}$, and $b^{(i)} = f_i(s_1) - \sum_{j=1}^{N} a_j^{(i)} \phi(p_j(s_1 - s_j))$. Note that only the outer parameters $a_j^{(i)}$ are function specific, while the inner parameters p_j and s_j can be shared among the functions f_i.

We thus create a helix ticket by first mapping the input $x \in [-1, 1]$ to $[0, 2]$. This allows us to represent the identity in the later layers by $\phi(x) = x$, as $x \geq 0$. We can always compensate for the bias $+1$ by subtracting a bias -1 when needed. $f_3(x) = (5\pi + 3\pi x)/(8\pi)$ can therefore be represented by a path from the input to the output that only contains a single neuron per layer. We concatenate this path with a neural network that consists of layers that approximate $f_1(x)$ and $f_2(x)$ and otherwise identity functions. At Layer $l = 2$, this network creates neurons of the form $\phi(p_j(x - s_j))$, where the knots s_j mark an equidistant grid of $[0, 2]$. Layer $l = 3$ creates two neurons, one corresponding to $x_1^{(2)} = f_1(x)$ and one corresponding to $x_2^{(2)} = f_2(x)$. These can be computed by linear combination of the previous neurons using the parameters $a_j^{(i)}$ and $b^{(i)}$. All the remaining layers basically encode the identity.

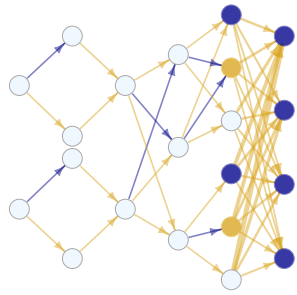
Strong tickets based on trained neural networks Even though we cannot expect to construct sparse baseline solutions for benchmark image classification tasks, we can leverage the fact that weak LTs can currently be identified at lower sparsity levels than strong LTs (see our experiments). To answer the question whether state-of-the-art pruning algorithms can find sparse strong LTs in the setting of standard benchmark data, we plant a trained weak LT in a randomly initialized (VGG like) neural network. Note that the proposed pruning algorithm can also be applied to convolutional layers in addition to fully connected ones.

8.4 EXPERIMENTS

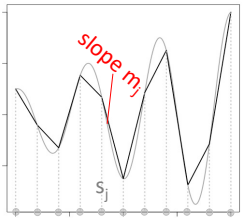
We utilize our planting framework to answer the question whether LT pruning algorithms that identify subnetworks of randomly initialized neural networks are able to identify highly sparse LTs, ideally in a strong sense but we also analyze weak LTs. Hypothetically, it could be possible that pruning algorithms for weak LTs only have to resort to training the identified LT because a highly sparse strong LT does not exist with high probability. Yet, if we guarantee the



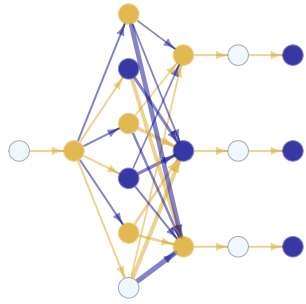
(a) Mirroring along axes for `Circle`.



(b) `Circle` architecture with depth $L = 5$.



(c) Univariate function approximation.



(d) `Helix` architecture with depth $L = 5$.

Figure 8.2: (a) Visualization of the first layers of `Circle` representing $g(x_1, x_2) = x_1^2 + x_2^2$. (c) Univariate deep neural network parametrization with outer weights $a_j^{(i)} = \Delta m_j = m_j - m_{j-1}$. (b+d) Ticket architectures, edge width is proportional to the absolute weight value, blue indicates a negative sign, yellow a positive sign. Neurons are colored by bias sign, gray indicates zero biases.

existence of a sparse strong LT, the algorithms would be able to find it and would not require further training. Similarly, if we insist on finding extremely sparse architectures, it might be necessary to give up the search for initial LTs (Frankle et al., 2020; Renda et al., 2020; Liu et al., 2021a). If this were true, we should be able to find highly sparse LTs with the original pruning algorithm if we ensure the existence of a solution by planting.

We reject these hypotheses with our experiments, in which we randomly initialize a large neural network of width $n_l = 100$ and depth $L = 5$ by He initialization with nonzero biases (Fischer et al., 2021a) and plant one of our constructed targets into the initial network. To show that our experiments reflect realistic conditions, we also compare the general trends to results on standard image classification. We compare only pruning methods that identify lottery tickets as subnetworks of randomly initialized neural networks, as these could potentially find our planted solution or an equally performing one. We thus consider GraSP, SNIP, Synflow, Magnitude pruning, and Random pruning (Wang et al., 2020; Lee et al., 2019b; Tanaka et al., 2020; Frankle and Carbin, 2019), which are algorithms to discover weak tickets, and Edge-Popup, which was designed to find strong tickets (Ramanujan et al., 2020).

We distinguish two different pruning approaches, singleshot (see Fig. 8.3) and multishot (see Fig. 8.5) experiments. In singleshot pruning, which is originally applied in SNIP, and GraSP, edges are scored in a single pass and then pruned to the desired sparsity. Compared to multishot pruning, this saves significant amount of resources by preventing training entirely if a strong ticket is found. If a weak ticket is found, only a small subnetwork needs to be trained once. Multishot pruning leads usualy to better results (Frankle et al., 2021), because it relies on updated gradient information. Analogous to iterative magnitude pruning, for each round r, we iteratively reduce the sparsity to $\rho^{r/10}$, where ρ is the desired network sparsity. Within each round, the current subnetwork is first trained, then pruned to the current target sparsity, and then reset to initial parameters for the next round. We analyze the performance of tickets before training to assess whether they qualify as strong LTs and after training to evaluate whether at least pruning for weak LTs is feasible and can identify LT of sparsities that can compete with our planted baseline ticket. Our code is made publicly available[1].

8.4.1 Hyperparameters and Data

For each experiment, we generate $n = 10000$ samples, where input data is sampled from $[-1, 1]$, for `ReLU` and `Helix` and from and $[-1, 1]^3$ for `Circle`. The output for `ReLU` is computed by $f(x) = max(0, x)$. For `Helix`, we compute the three output coordinates as $f_1(x) = (5\pi + 3\pi x) * \cos(5\pi + 3\pi x)/(8\pi)$,

[1] https://github.com/RelationalML/PlantNSeek/releases/tag/v1.0-beta

$f_2(x) = (5\pi + 3\pi x) * \sin(5\pi + 3\pi x)/(8\pi)$, and $f_3(x) = (5\pi + 3\pi x)/(8\pi)$. For `Circle`, we consider circles centered at the origin with radius $\sqrt{0.2}$, $\sqrt{0.5}$, and $\sqrt{0.7}$ as decision boundaries for the classes. We additionally introduce a small amount of noise to simulate real world data more closely. For `Circle` we flip approximately 1% of samples to the next closest class, and for the two regression problems we introduce additive noise drawn from $\mathcal{N}(0, 0.01)$ to each output dimension. To assess the accuracy respectively mean squared error of the tickets and trained models, we split off 10% of the data that acts as a hold out test set.

In general, all initial networks for each specific task are generated using the algorithm explained in the previous section. For `Circle`, we use 10 knots for the piecewise linear approximation, and 30 knots for the piecewise lienar approximations done in `Helix`. To prune by GraSP, SNIP, Synflow, Magnitude, and Random and train the derived tickets, we use `Adam` (Kingma and Ba, 2015) with a learning rate of 0.001. We found that this learning rate performed well over all experiments, and leads to accurate models when there is no pruning. It also corresponds to the default settings suggested by the authors of Synflow (Tanaka et al., 2020). Training of the discovered tickets was done for 10 epochs across all experiments, where we could observe a convergence of the respective loss. We measured loss by cross entropy respectively MSE and used a batch size of 32 for all experiments. We report mean and obtained intervals (i.e., minimum and maximum) across 10 repetitions for multishot, and across 25 repetions for the main singleshot experiments measured on the hold out test set.

Singleshot pruning For singleshot pruning, we considered networks of depth $3, 5, 10$ each with layer width 100 for all three data sets on target sparsities $\{0.01, 0.1, 0.5, 1\}$ and the sparsity of the ground truth ticket. Additionally, we tested for a network of depth 6 and width 1000 on `Circle` for the same sparsity levels. As suggested by Tanaka et al. (2020), we also test Synflow in combination with 100 rounds of pruning for a network of depth 6 and width 100 on `Circle`. For all additional singleshot experiments, we provide results in the next section.

Multishot pruning For multishot pruning, we alternated pruning and training for 10 rounds, where each training step was carried out for 5 epochs, which consistently led to convergence of accuracy on the considered `Circle` data set. Similar to singleshot pruning, we considered target sparsities $\{0.01, 0.1, 0.5, 1\}$ and ground truth ticket sparsity.

Edge-Popup pruning To prune with Edge-Popup, we here used the parameters suggested in the original code of Ramanujan et al. (2020), which is SGD with

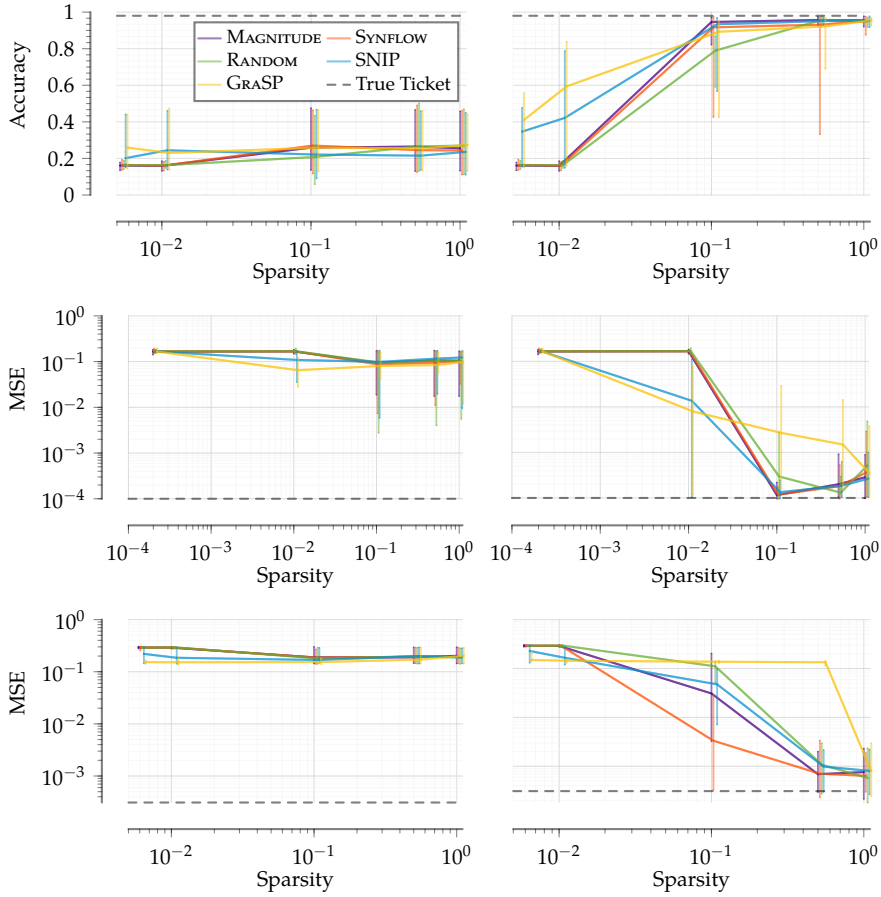


Figure 8.3: *Singleshot results depth* 5. Performance of discovered tickets for `Circle`, `ReLU`, and `Helix` against target sparsities as mean and obtained intervals (minimum and maximum) across 25 runs. In order of appearance from top to bottom: `Circle`, `ReLU`, and `Helix` post pruning (left) and post training performance (right). Baseline ticket with leftmost sparsity and performance given by black dashed line.

momentum of 0.9 and weight decay 0.0005, combined with cosine annealing of the learning rate. To establish a comparison to the multishot pruning results, we train the scores for 10 epochs. Additionally, for the experiment extending Edge-Popup by annealing the sparsity level, we slowly reduce the sparsity over time by $\rho^{i/10}$, where ρ is the desired network sparsity, and i is the current epoch.

8.4.2 hand designed ground truth

Singleshot pruning

For our benchmark data we test the ability of algorithms to discover both strong and weak tickets. The key results for singleshot pruning are visualized in Fig. 8.3, reporting performance of the algorithms trying to discover tickets at varying sparsity levels. Results for varying network depths and widths can be found in App. A.7.4 noting that they are consistent also with larger depth, but the methods fail to find any ticket in more shallow networks of depth 3.

First, we note that training the full network (at sparsity level 1.0) can solve each of our tasks. Thus, planting does not destroy the general trainability of the initial mother network. In fact, we would observe the same performances without planting. Second, we find that none of the approaches is able to discover strong tickets, in particular not the planted ticket, *before* training in a single shot. Furthermore, the performance of all methods on the simpler `ReLU` is considerably better than on `Helix`. We find that all approaches, including Random pruning, are able to find weak tickets for moderate sparsity levels for `Circle` and `ReLU`, but fail to recover sparse weak tickets on the manifold learning task `Helix`. Interestingly, although Magnitude pruning was originally not designed for this pruning strategy, it is on par with state-of-the-art singleshot methods. For more extreme sparsity levels ≤ 0.01, in particular baseline ticket sparsity, all methods fail to recover good subnetworks.

Dissecting the singleshot results further, we identify layer collapse as the main source of failure for more extreme sparsities, meaning that entire layers are masked, thus disrupting flow through the network. While layer-wise pruning could prevent this collapse, i.e., have a fixed target sparsity per layer, in practice an interruption of flow is observed nevertheless. Despite that Synflow was proposed as a solution to this issue, we observe that it also experiences layer collapse for extreme sparsities, even when pruning for 100 rounds as suggested in the original paper, performing only slightly better than with one round of pruning (see App. A.7.4). In summary, with only a single pruning round, most pruning algorithms discover weak tickets at moderate sparsity, however fail to recover weak tickets of low sparsity and any strong tickets.

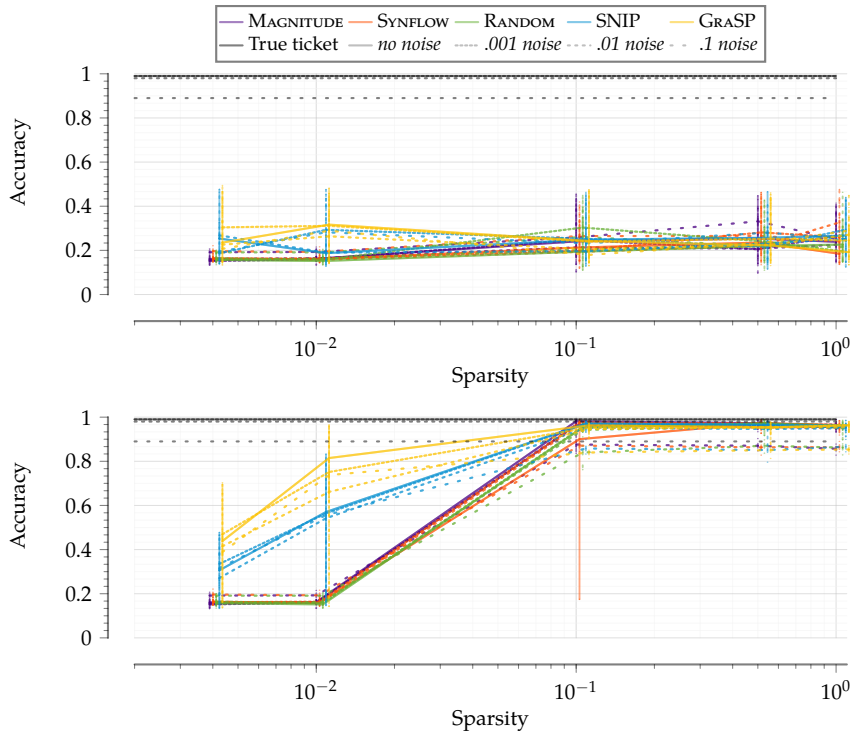


Figure 8.4: *Varying noise.* Performance of methods for `Circle` with varying noise for 10 rounds of alternating pruning and training. We report mean and obtained intervals (minimum and maximum) of accuracies of the final pruned network across 10 repetitions before (left) and after (right) the final training. The noise level is indicated by line type. Ground truth ticket accuracy is indicated by black lines.

Comparison to results on image data The reported results are in line with experiments on image data as reported in the literature (Tanaka et al., 2020). In particular, for all these methods a similar drop around 0.01 sparsity is observed for different VGG and Resnet architectures and image data sets. Similarly, layer collapse has been reported for image data. The main difference is that for our data, we know the obtainable sparsity as well as performance of tickets, setting these results into a context beyond trendline differences of methods for selected sparsity values.

Robustness to noise To model real-world settings, all our datasets contain small amounts of noise. To rule out that noise in the data is the primary source for issues with discovering tickets, we generated `Circle` datasets with varying

levels of noise. Hence, to test the robustness of pruning algorithms to noise in the data, we considered `Circle` with a network of depth 6 and width 100 and varied the amount of noise in the data to be $\{0, 0.001, 0.01, 0.1\}$. We report the results before and after training in Fig. 8.4. The results indicate that on the one hand, without noise we do not see much of an improvement in terms of discovered tickets, but on the other hand observe that the algorithms are robust to even large amounts of noise, finding tickets with almost similar performance as with no noise at all.

Multishot pruning

While much more resource intensive, iteratively training followed by pruning and resetting to initial weights, slowly annealing to the desired sparsity, has been proven a successful approach to discover lottery tickets. Here we extend this pruning scheme to other approaches beyond magnitude pruning.

To investigate the effect of multishot pruning, we run each of the previous methods iteratively for 10 rounds on our benchmark datasets (see Fig. 8.5). Analogous to iterative magnitude pruning, for each round r, we iteratively reduce the sparsity to $\rho^{r/10}$, where ρ is the desired network sparsity. Within each round, the current subnetwork is first shortly trained, then pruned to the current target sparsity, and then reset to initial parameters for the next round. Compared to the singleshot results, we observe that for the classification task, Magnitude, SNIP, and Synflow are able to retrieve weak tickets of much higher sparsity. Furthermore, these three approaches are now able to recover weak tickets of moderate sparsities also for the challenging `Helix` dataset. Overall, Synflow consistently performs best in discovering weak tickets, even recovering the extremely sparse baseline ticket for `Circle`. We also observe that GraSP performs poor overall, noting that it is a method designed for singleshot pruning. Examining the results, we see that GraSP experiences layer collapse already in early iterations with large target sparsities. None of the above approaches is able to discover strong tickets reliably, only in individual instances of the `ReLU` task, Synflow and SNIP could retrieve a strong ticket of high sparsity.

To discover strong tickets, Ramanujan et al. (2020) proposed Edge-Popup, which falls in the same category of multishot pruning approaches. Edge-Popup assigns each model parameter a score value, which is then actively trained for several rounds while freezing all original parameters, requiring a similar computational effort as multishot pruning, yet leaving the original weight and bias parameters untouched. Training Edge-Popup for 10 rounds, we observe that it discovers a strong ticket of sparsity 0.5 for `Circle`, however, fails to discover tickets of different sparsity, which is in line with their original results (Ramanujan et al., 2020). Similar to other algorithms, we observe layer collapse. We can extend their original algorithm using annealing as in multishot prun-

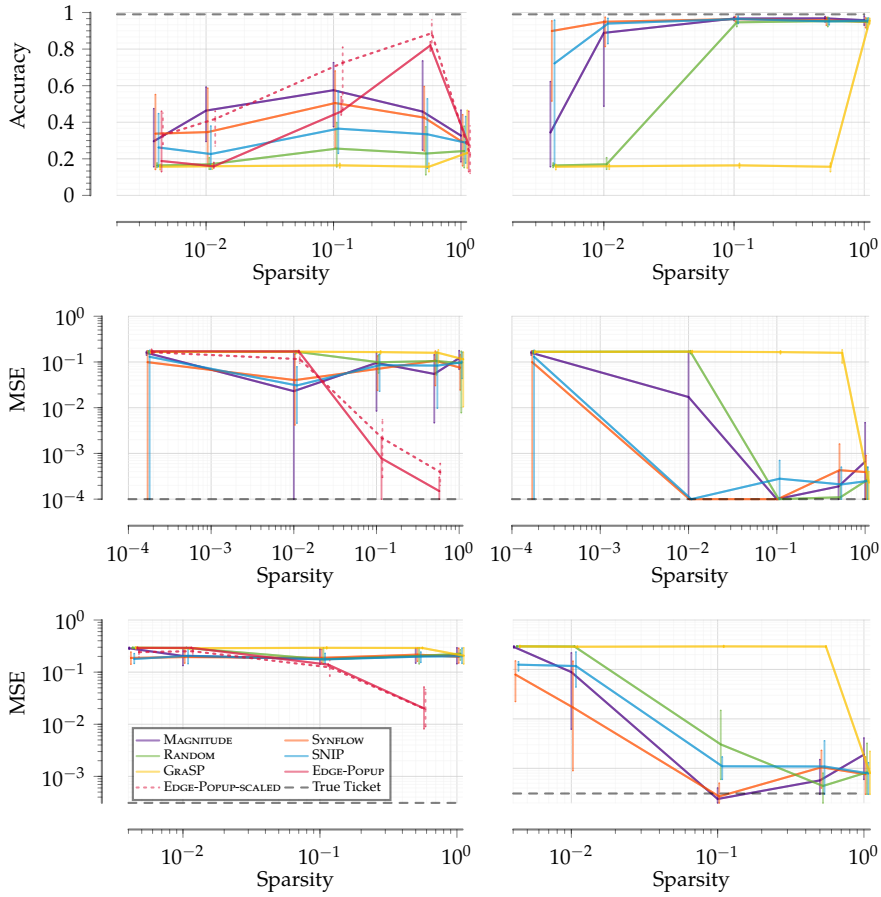


Figure 8.5: *Multishot results.* Performance for `Circle` (top), `ReLU` (middle), and `Helix` (bottom) for 10 rounds of alternating pruning and training. For Edge-Popup we report the performance for 10 rounds of training (solid line), as well as training combined with iterative reduction of the sparsity (dashed line). We provide mean and obtained intervals (minimum and maximum) of accuracies of the final pruned network across 10 repetitions before (left) and after (right) final training. Baseline ticket accuracy is indicated in black.

ing by slowly decreasing the target sparsity in every round, which increases performance allowing it to discover a subnetwork with reasonable accuracy at 0.1 sparsity. Edge-Popup is not able to find any good subnetwork for the `Helix` task.

GraSP with local sparsity constraints As observed before, GraSP seems unsuitable for multishot pruning due to early layer collapse. Several works (You et al., 2020; Tanaka et al., 2020) considered local sparsity constraints, having a target sparsity for each layer, or even channel. These however impose unrealistic architecture constraints as layer sparsity is usually imbalanced (Tanaka et al., 2020), which also holds true for our `Circle` and `Helix` benchmarks. With the goal to avoid layer collapse, we still equipped GraSP with local sparsity constraints per layer (see App. A.7.5). Yet, the flow through the layers stays interrupted. A possible explanation is that GraSP incorporates information about weight couplings via the hessian in its pruning strategy, which makes it more sensitive to removing individual connections from the mask as it happens during iterative pruning.

Comparison to results on image data Our results are coherent with the reported relative trends on image tasks both for strong and weak tickets (Ramanujan et al., 2020; Tanaka et al., 2020). We further reproduced results for VGG16 on CIFAR10 with non-zero bias initialization (Fig. 8.6 left). In particular, for strong tickets we observe the same trends for Edge-Popup spiking at 0.5 sparsity, performing less well at sparsity 0.1, and not recovering tickets for higher and lower sparsity. Again consistent with previously reported results, other methods are neither suited nor designed to find strong tickets. For weak tickets, Synflow performs best, with only a slight margin towards SNIP and Magnitude. This margin is however much tighter than originally reported in Synflow, as we allow both SNIP and Magnitude to learn in a multishot fashion, which closely resembles the approach of iterative magnitude pruning and prevents layer collapse to a large extent.

8.4.3 VGG with strong tickets

While no ground truth solution is known for image classification tasks on standard benchmark datasets, we can still use our planting framework to answer meaningful questions in this context, as we demonstrate next. To test the hypothesis whether Edge-Popup is limited to discover strong tickets of suboptimal sparsity of around 0.5, we investigate its capabilities to recover a planted baseline ticket from VGG16. For that, we use Synflow to discover a weak ticket of sparsity 0.01 from VGG16 with multishot pruning, train the weak ticket on CIFAR10, and plant it back into the network. Running Edge-Popup on this

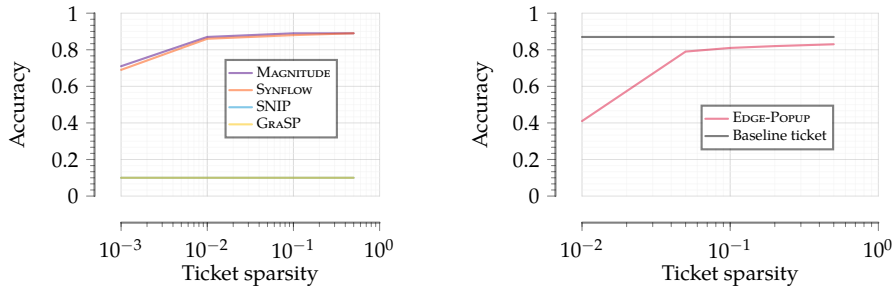


Figure 8.6: *VGG16 CIFAR10 results.* (Left) Performance for learned weak tickets. (Right) Performance of strong tickets discovered by Edge-Popup for VGG with planted baseline ticket of sparsity 0.01. Baseline ticket performance is indicated by black line.

network, we observe that it indeed cannot retrieve the baseline ticket of desired sparsity in this real world setting (see Fig. 8.6 right).

8.4.4 Structure learning and lottery tickets

Our planting framework is designed for the analysis of LT algorithms that seek for subnetworks of randomly initialized NNs. Structure learning methods follow a different approach, yet also yield subnetworks that perform well on a given task. While planting is less relevant in this case, we can still compare their results with our hand-crafted solutions. We show in App. A.7.3 that neither dynamic sparse training (Evci et al., 2020) nor LT fine-tuning techniques (Liu et al., 2021a; Renda et al., 2020) find architectures that are competitive with our constructed ground truth tickets.

8.5 Discussion & conclusion

We investigated the optimality of existing lottery ticket pruning methods and their potential for improvement, both regarding the discovery of strong tickets – subnetworks that perform well at initialization, as well as weak tickets – subnetworks that perform well after training. Recent works, in particular by Frankle et al. (2021), evaluated lottery ticket pruning methods and showed that no single best method across considered settings and sparsities exists, and raised the issue of missing baselines in the field. To tackle this issue, we here proposed an algorithm that plants and hides target networks within a larger network, thus allowing to generate baseline tickets for rigorous benchmarking. For three common challenges in machine learning, a classification, regression, and manifold learning problem, we hand-crafted extremely sparse network

topologies, planted them in large, randomly initialized neural networks, and evaluated the state-of-the-art pruning methods in combination with different pruning strategies.

Our results indicate that state-of-the-art lottery ticket pruning methods achieve in general sub-optimal sparsity levels, and are not able to recover lottery tickets that are competitive with a planted ground truth. This suggests that previous challenges to identify highly sparse winning tickets as subnetworks of randomly initialized dense networks (Frankle et al., 2020; Ramanujan et al., 2020) can be explained by algorithmic limitations rather than fundamental problems with lottery ticket existence. While slightly discouraging, these result on our benchmark data are coherent with reported as well as reproduced classification results on image data. This shows that our benchmarks, while artificial in nature, reflect realistic conditions that result in similar trends as real world image data sets would. Moreover, we have shown that our planting framework can also be used in a real data setting to answer a limited set of questions. For instance, by planting a trained weak ticket back into a CNN, we established that the failure of Edge-Popup to discover extremely sparse strong lottery tickets is likely an algorithmic rather than a fundamental limitation. This exemplifies how our framework enables experiments beyond relative method comparisons, as typically conducted on standard image benchmark data. As our results indicate, several major questions pertaining to neural network pruning are still open: How can pruning approaches for weak tickets be improved to discover tickets of best possible sparsity? How can we find weak tickets of high sparsity that match the performance of the large network without intermediate training rounds? And, how can we discover highly sparse strong lottery tickets? We anticipate that our contribution can be used and extended to measure progress regarding these questions against independent sparse and well-performing baseline tickets.

CONCLUSION

9

In this thesis, we focused on key research questions pertaining to pattern mining and neural networks. We further showed how these two fields complement each other, leveraging the interpretability of patterns to better understand neural networks, and the efficiency of neural network learning for pattern discovery. In particular, we considered the questions of how to find expressive patterns in binary data, and how to do so efficiently, an answer to both being crucial for the discovery of insightful patterns in data. Furthermore, we considered the questions of what neural networks learn to successfully arrive at a decision, and what regularities in the input data leads to systematic errors in their prediction. These questions are relevant to better understand the decision process of these opaque models and improve their robustness, both important clues of models in safety-critical environments such as the medical domain. Finally, we considered the question of how to learn neural networks in a resource-efficient way, as state-of-the-art models are largely over-parameterized, making them in many settings prohibitively expensive to train. As our work was cumulative, we will first briefly summarize all contributions to these open questions and then provide a joint discussion with future directions of research.

SUMMARY OF CONTRIBUTIONS

Related to the first set of questions about pattern mining, we proposed richer languages of rules and patterns that express co-occurrences and mutual exclusivity, along with algorithms to find such patterns effectively in real-world binary data. We put special emphasis on the discovery of insightful, non-

redundant, and human-interpretable patterns in the data, such that our results can aid domain-experts in better understanding the process underlying their data, aid them in postulating new hypotheses, or guide their decision-making. We showed on several synthetic benchmarks that our contributions have the among existing methods unique ability to find the relevant regularities in data without picking up spurious results, and showed on real-worl data that discovered results provide relevant insights. On a recent large-scale transcriptomics dataset, we demonstrated that such algorithms – in contrast to state-of-the-art – scale to the challenging biological and biomedical settings. Moreover, the rich pattern language allows to capture biologically meaningful properties and provide novel insights that domain experts can leverage for future research. Our contributions, hence, enable pattern mining with powerful languages at scale, which are of relevance in such fields as biology that has a high demand for exploratory methods for knowledge discovery.

We further showed how to transfer and extend our pattern mining approaches to better understand how neural networks work internally. In particular, we proposed to examine networks by discovering rules across neuron activations, extending the language of rule tails to also capture disjunctive statements. Our results revealed novel insights into what a neural network deems interesting for image prediction tasks, how information flows through a network to arrive at an outcome, and what it learns to distinguish within and between classes. The suggested approach, hence, allows to peek into the black box of neural networks to better understand their decision process and to learn from how it uses information, both of which are crucial to understand their success, but also the biases it might learn.

It is important to understand the success of neural networks, but also their systematic errors. To understand those, we proposed to discover differential patterns that explain which regularities in the input data make neural networks err. Our solution is readily scalable to describe misclassifications of modern natural language processing models in terms of a rich language of conjunctions and mutual exclusivity. Our results not only provide interesting clues about systematic issues in the data that are challenging for the networks, but are also actionable.

To scale to extremely high dimensional data such as encountered in typical retail transaction data or in population-scale genomic studies, we proposed a new framework to pattern mining, rethinking the classical approach of discrete optimization of patterns and pattern sets. Connecting classical pattern mining with neural networks, we proposed a new type of binarized autoencoder that allows to find pattern sets, linking continuous optimization with the exploration of the discrete search space of patterns. We showed that our approach scales to orders of magnitude higher dimensional data than what was possible before, enabling pattern mining in data of the current age. On the 1000 Genomes data

set, data of genetic variation in a human population, we further showed that the discovered patterns provide interesting insights into human variation that suggest a potential role of so far unknown mutations in human individuals.

Finally, we turned to the problem of training small, but well-performing neural networks. Such networks not only forego the prohibitively expensive training process of their larger counterparts, but are also inherently more interpretable. Models that are easier to train and to understand not only benefit classical neural network learning, but, through the link to pattern mining established in this thesis, also directly benefit the discovery process of patterns. We, here, considered one of the recent, arguably most promising avenues to find such networks, which is described by the lottery ticket hypothesis. This hypothesis states that there exist small, well trainable sub-networks in randomly initialized neural networks that can match the performance of their larger counterparts. Even more promising are strong lottery tickets, which are such sub-networks that do not require any further training and have been formally proven to exist.

We showed that standard initialization schemes of neural networks are, however, not sufficient to equip the strong lottery tickets with the universal approximation property. To overcome this issue, we proposed new initializations schemes that are well trainable and equip initialized neural networks with the ability of universal approximation. We then formally showed that strong lottery tickets exist in such networks, hence equipping strong lottery tickets with the ability of universal approximation. Based on insights from the proof we further suggested a parameter scaling procedure that lead to better signal preservation in tickets. Both of these contributions enable finding orders of magnitude sparser tickets in synthetic benchmark data. We are convinced that our theoretical contributions facilitate research for the discovery of sparse strong lottery tickets.

So far, approaches that discover any lottery ticket – weak or strong – are only evaluated against their dense counterpart and existing methods. This is due to a lack of benchmarks for lottery tickets. Hence, whether there exist sparser or better tickets could never be answered. This raised the question if the current limitations regarding sparsity and performance are due to algorithmic or fundamental constraints. In our last contribution we derived a framework that allows to plant and hide a target network, e.g. a lottery ticket, in a large, randomly initialized neural network. Building up on the results of our previous contribution, we posed three challenging problems and constructed extremely sparse neural networks solving these tasks, which we then hid in large neural networks. These small planted subnetworks serve as benchmarks to evaluate the state-of-the-art in lottery ticket pruning with respect to sparsity and performance. The results indicate that the current limitations are, in fact, algorithmic constraints, opening up room for new proposals for the discovery of lottery

tickets. Our contribution, for the first time, allows to measure progress in lottery ticket discovery and enables the evaluation of future progress in the field.

OUTLOOK

Our proposed richer pattern languages, such as robust rules or patterns of mutual exclusivity, enable to discover important types of regularities that so far were hard to find. With our formulation of differential pattern mining, we furthermore extended this line of work to explaining labels of a dataset, where these labels implicitly induce a decomposition of the data. While this is of high value in settings where label descriptions offer insights into an underlying process, such as for misclassification labels, it would also be interesting to study how to find data decompositions automatically in a purely unsupervised setting, for which Dalleiger and Vreeken (2020) offer a solution in the context of conjunctive statements. It would furthermore be interesting to extend on the idea of statistical testing of discovered patterns as introduced in the third chapter also for other pattern languages and models, for example building on the work of Hämäläinen (2012) for rules. Assessing the statistical significance is in high demand, especially in biological exploratory studies.

Our suggested approach to analyze networks through rules was the first of its kind and offered unique insights into the behaviour of neural networks. Following up on this, it would be interesting to equip the pattern language with the concept of negations, as networks, just as humans, make use of absence of features. It, hence, would likely provide valuable insights to capture this usage of information through rules to further understand the decision making process in neural networks.

For the analysis of systematic issues of neural networks beyond misclassification, such as unfair racial or societal biases in a prediction, it would make for engaging future work to consider multiple labels at once. A solution to this problem was proposed by Budhathoki and Vreeken (2015), which is, however, limited to conjunctive statements and small-scale data. Patterns across multiple labels or label outcomes could for example show what in the input causes a bias of the network prediction against minorities, with the found patterns serving as a basis to counteract this behaviour of networks. Analyzing input patterns describing labels being interesting on its own, analyzing patterns of network activations in the context of labels might not only serve for insights into the network reasoning, but also provide actionable patterns where a domain expert could for example directly interact with the neurons leading to false predictions or a bias.

Taking a step back, we discussed pattern mining and neural networks and how we can bridge these two fields. In particular, we showed how these two

can benefit from each other's strength, on the one hand by exploring networks through patterns, offering the interpretability of patterns to better understand the networks, and on the other hand discovering patterns through networks, offering the fast, continuous learning strategies of neural networks to drastically speed up pattern discovery. With these foundations laid, there are several directions opening up for future work.

It would first of all be interesting for both, large-scale biological applications as well as the exploration of modern deep and wide neural networks, to study how to construct binarized autoencoders in a way that allow for the discovery of the richer pattern languages proposed in this thesis or the concept of differential patterns. In other words, how can we adapt the new paradigm of discovering patterns to our more expressive languages to leverage the power of neural network training for pattern mining. On a similar note, it would be interesting to study different languages for entirely different data types such as graphs or sequences. Utilizing recent advances in architectures for graph neural networks (Scarselli et al., 2009; Duvenaud et al., 2015; Veličković et al., 2018) and transformers (Vaswani et al., 2017; Devlin et al., 2019) might be an interesting avenue to approach this line of research. Reversely, it would also make for engaging future work to analyze these types of networks – graph neural networks and transformers – through the lense of patterns. This would require a different type of pattern language that is able to express statements over graphs or sequences, where we can draw from contemporary work in graph (Goebl et al., 2016; Coupette and Vreeken, 2021; Coupette et al., 2022) and sequence mining (Bhattacharyya and Vreeken, 2017; Wiegand et al., 2021).

With the connection of pattern mining and neural networks, studying the problem of overparametrization and expensive training of neural networks becomes twice beneficial. Having a solution to this problem pertaining to neural networks would make their training more efficient, and at the same time improve efficiency in the discovery of patterns. Our foundational work on lottery tickets provides an important step into this direction. Knowing where we stand in the quest for lottery tickets, identifying several issues of current network pruning algorithms, and being able to evaluate success all serves as a great starting point to develop new algorithmic approaches to this problem. Noting that this line of research was conducted relatively late in the PhD studies, we are looking forward to engage on this research problem.

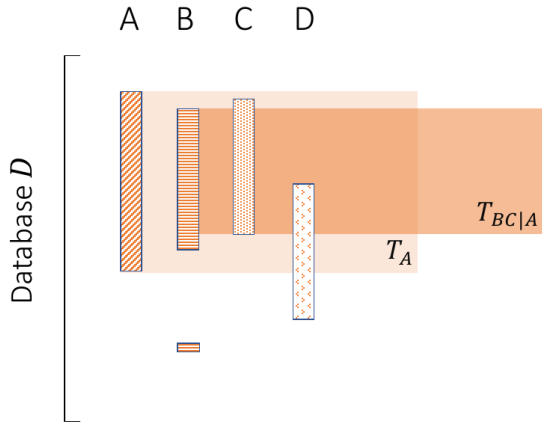
Slightly disconnected at first glance, lottery tickets and pattern mining also turn out to offer plenty of opportunities for future research to connect them. For example, the properties of lottery tickets, in particular what they learn, is largely unknown. Mining patterns or rules across the activations of these tickets, similar to what we proposed for fully connected and convolutional neural networks, might deliver interesting insights into what makes a good lottery ticket. Considering our binarized autoencoders, discovering lottery

tickets in them might correspond to finding the patterns in the data and hence offer an even more efficient solution to pattern mining. This approach, due to the constraints on the weight matrices, is, however, non-trivial and hence also makes for engaging future work.

APPENDICES

A.1 ROBUST RULES

A.1.1 AN EXAMPLE COMPUTATION OF MDL FOR RULES



$$R_{ind} = \{\emptyset \to A, \emptyset \to B, \emptyset \to C, \emptyset \to D\}$$

$$R = R_{ind} \cup \{A \to BC\}$$

$$|D| = n = 100$$
$$|T_A| = 40$$
$$|T_B| = 35$$
$$|T_C| = 33$$
$$|T_D| = 35$$
$$|T_{BC|A}| = 30$$

Figure 1: *Example database and model.* A toy database D with blocks indicating where the items A, B, C, D occur in D, margins and relevant joint counts are given on the right. A sensible rule set $M \cup M_{ind} = A \to BC \cup M_{ind}$ is given on the right, the part of the database where the rule applies and holds is indicated by a light respectively dark orange area.

For the given example in Fig. 1, we now compute the codelength $L(D, M) = L(M) + L(D \mid M)$ of transmitting the whole database D using $M \cup M_{ind}$. Here,

we will stick with the simple encoding without error matrices, to make the process of computation more understandable. For reference, we first compute the baseline model, which is given by

$$\begin{aligned} L(D, M_{ind}) =& |\mathcal{I}| \times L_{pc}(|D|) + \sum_{I \in \mathcal{I}} \log \binom{|D|}{|T_I|} \\ =& 4 \times L_{pc}(100) + \log \binom{100}{40} + 2 \log \binom{100}{35} + \log \binom{100}{33} \\ \approx& 14.88 + 93.47 + 179.64 + 87.93 = \mathbf{375.92}. \end{aligned}$$

Thus, sending the data with just the baseline model costs 375.92 bits. Now, we will compute $L(D, M \cup M_{ind})$, we will start with the costs of sending the data $L(D \mid M \cup M_{ind})$

$$\begin{aligned} L(D \mid M \cup M_{ind}) =& \left(\sum_{X \to Y \in M} \log \binom{|T_X|}{|T_{Y|X}|} \right) + \left(\sum_{I \in \mathcal{I}} \log \binom{|D|}{|T'_I|} \right) \\ =& \log \binom{40}{30} + \log \binom{100}{40} + \log \binom{100}{5} + \log \binom{100}{3} + \log \binom{100}{35} \\ \approx& 29.66 + 93.47 + 26.17 + 17.30 + 89.82 = \mathbf{256.42}. \end{aligned}$$

The model costs are composed of the parametric complexities for the (adapted) baseline rules, plus the costs of transmitting what the rule is composed of along with its parametric complexity. We thus get

$$\begin{aligned} L(M \cup M_{ind}) =& |\mathcal{I}| \times L_{pc}(|D|) + \left(\sum_{X \to Y \in M} L_{\mathbb{N}}(|X|) \right. \\ & \left. + L_{\mathbb{N}}(|Y|) + L(X) + L(Y) + L_{pc}(T_X) \right) \\ =& 4 \times L_{pc}(100) + L_{\mathbb{N}}(1) + L_{\mathbb{N}}(2) \\ & - \log \frac{40}{143} - \log \frac{35}{143} - \log \frac{33}{143} + L_{pc}(40) \\ \approx& 14.88 + 1.52 + 2.52 + 1.84 \\ & + 2.03 + 2.12 + 3.11 \\ =& \mathbf{28.02}. \end{aligned}$$

Hence, the model with the complex rule has a smaller codelengh than the baseline, with $L(D, M \cup M_{ind}) = \mathbf{284.44}$ bits.

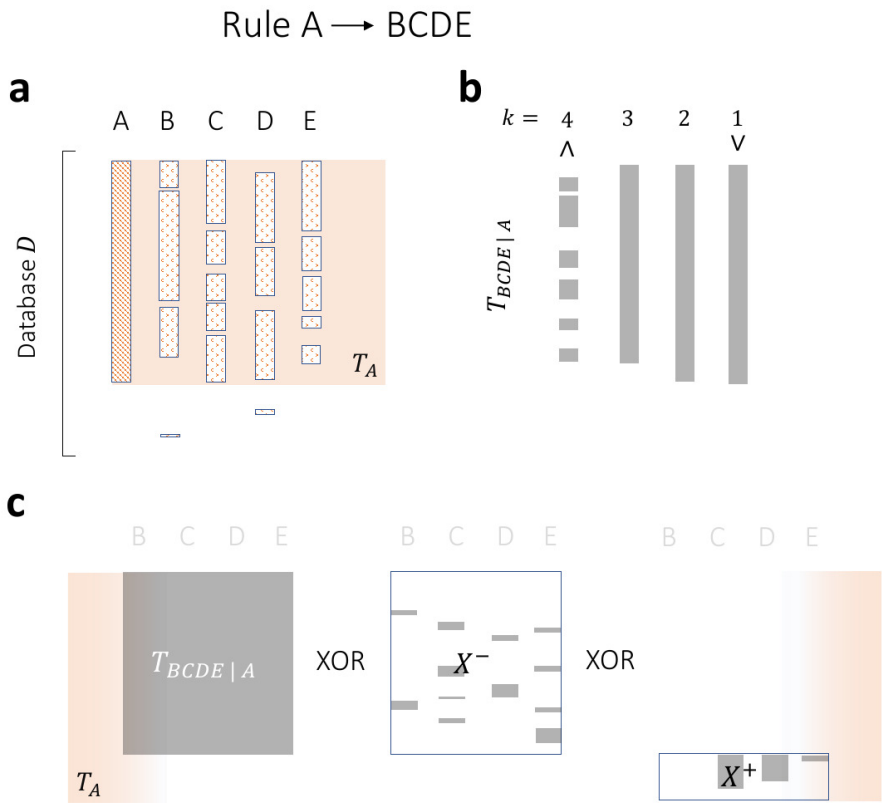


Figure 2: *Example of tail error encoding.* For a given database D given in a, where blocks indicate the occurrence of items, a good rule is given by$A \rightarrow BCDE$. The part of the database where the rule applies is indicated by the orange area. In b we show the part of the transaction were the rule holds for varying number k of tail items that have to be present in a transaction, from all items on the left – corresponding to a conjunction – towards just a single item on the right, which corresponds to a disjunction. In c we visualize the error encoding used to transmit the data for $k = 3$. We first transmit the data where the rule holds, resulting in the area that is indicated by the gray block. XORing the error matrix X^- with this block, it is possible to reconstruct the original data for the part where the rule holds. Using X^+, we reconstruct the original data in the area where the rule applies but does not hold.

For the error encoding for tails, which allow to discover rules in noisy settings (compare Fig. 2a,b), we send where a rule $X \rightarrow Y$ approximately holds according to some parameter k, which defines the number of items of the tail that

have to be present in the transaction. The errors made by this approximation are then accounted for by sending error correcting matrices $\mathcal{X}^-_{X\to Y}$ and $\mathcal{X}^+_{X\to Y}$, which account for the destructive, respectively additive noise in the are where the rule applies (compare Fig. 2c).

Let us first assume we are given a k, we will later show how we can optimize for k. We redefine the transaction sets $T_{Y|X} = \{t \in D \mid (X \subset t) \wedge (|Y \cap t| \geq k)\}$, which corresponds to the transactions where the rule approximately holds. We will now slightly abuse notation and indicate the binary input matrix that correspond to D by $\mathcal{D}$, and we subset this matrix using the transaction id lists and item subsets. Both of these are sets of indices that indicate which rows, respectively columns to use of the matrix. For example, the submatrix where X holds is given by $\mathcal{D}[T_X, X]$. We can now define the error correcting matrices to be $\mathcal{X}^-_{X\to Y} = \mathcal{D}[T_{Y|X}, Y] \otimes \mathbb{1}^{|T_{Y|X}|\times|Y|}$, and $\mathcal{X}^-_{X\to Y} = \mathcal{D}[T_X \setminus T_{Y|X}, Y]$, where $\otimes$ is the element-wise XOR operator and $\mathbb{1}^{i\times j}$ is a matrix of size $i \times j$ filled with ones. The receiver, knowing T_X and $T_{Y|X}$, can then reconstruct the original data $\mathcal{D}[T_{Y|X}, Y] = \mathbb{1}^{|T_{Y|X}|\times|Y|} \otimes \mathcal{X}^-_{X\to Y}$, respectively $\mathcal{D}[T_X \setminus T_{Y|X}, Y] = \mathcal{X}^+_{X\to Y}$.

While this explains the concept of how error correcting matrices can be used to reconstruct the original input, which hence define a lossless encoding, we are mainly interested in the codelength functions. To adapt the data costs, we now additionally send the two error matrices, which we can do using binomial codes. Hence, we get

$$\begin{aligned} L(D \mid M) = &\left(\sum_{X\to Y\in M} \log \binom{|T_X|}{|T_{Y|X}|} \right) + \left(\sum_{I\in\mathcal{I}} \log \binom{|D|}{|T'_I|} \right) \\ &+ \log \binom{|T_{Y|X}| \times |Y|}{|\mathcal{X}^-_{X\to Y}|} + \log \binom{|T_X \setminus T_{Y|X}| \times |Y|}{|\mathcal{X}^+_{X\to Y}|}, \end{aligned}$$

with the second line providing the codelength of the error matrices, and $|\mathcal{X}|$ indicating the number of ones in $\mathcal{X}$.

Our model M now not only consists of rules $M \cup M_{ind}$, but also of the set of error correcting matrices. As the submatrix to which we need to apply the matrix is fully defined by $T_X, T_{Y|X}$, and Y of the corresponding rule, also defining its size, the only adaptation we need for the model costs is the parametric complexities induced by the codes for transmitting the data. This yields

$$\begin{aligned} L(M) = &|\mathcal{I}| \times L_{pc}(|D|) + \Bigg(\sum_{X\to Y\in M} L(X \to Y) \\ &+ L_{pc}(|T_{Y|X}| \times |Y|) + L_{pc}(|T_X \setminus T_{Y|X}| \times |Y|) \Bigg). \end{aligned}$$

This completes the MDL costs for rules robust to noise in the tail for a given k. To optimize k, the crucial insight is that the codelength of individual complex rules are independent, as is the data cost. That means we can optimize a k for each rule separately. Thus, for a given rule $X \to Y$ we can enumerate all $|Y|$ many models for the different thresholds k and let MDL decide which one fits the data best.

A.1.2 SUBMODULARITY AND MONOTONICITY OF THE SEARCH SPACE

In this section we provide counterexamples to disprove submodularity and monotonicity of the codelength function on the search space of all rule sets.

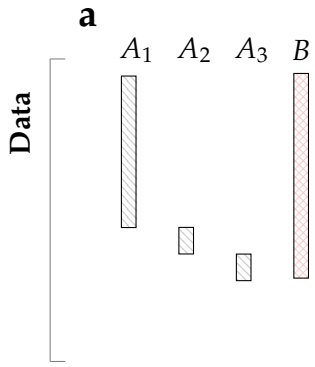
SUBMODULARITY Let us first define submodularity:

Definition 2 (Submodularity)**.** *Given an alphabet* Ω *and a set function* $f : 2^{\Omega} \to \mathbb{R}$*, a set function is called submodular iff* $\forall X \subset Y \subseteq \Omega, z \in \Omega.\ f(X \cup \{z\}) - f(X) \geq f(Y \cup \{z\}) - f(Y)$.

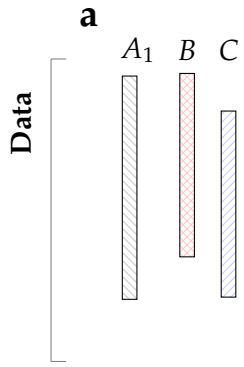
This definition of submodularity inherently shows the diminishing returns property of submodular functions, that is the larger the set grows, adding a particular element to it give less increase in the function value. Theoretically, this property can be used to guide the search, but here we show that if we define the alphabet to be all possible rules and the function to be the codelength defined before, $\Omega = \{X \to Y \mid X \subseteq \mathcal{I}, Y \subseteq \mathcal{I} \setminus X\}$ and $f(R) = -L(D, R)$, the codelength is neither submodular nor monotone w.r.t the search space of rules.

For the counterexample, visualized in Fig. 3a, let $n = 1000$ be the database size and $n_{A_1} = 500, n_{A_2} = 100, n_{A_3} = 100, n_B = 700$ the respective usages of the variables. Now let us set $X = \{A_1 \to B\}, Y = \{A_1 \to B, A_2 \to B\}$, and $z = A_3 \to B$ and compute the codelengths:

$$\begin{aligned} f(X) = -\Bigg(&\log \binom{n}{n_{A_1}} + \log \binom{n}{n_{A_2}} + \log \binom{n}{n_{A_3}} + \log \binom{n}{n_B - n_{A_1}} \\ &+ L_{pc}(n_{A_1}) + L_{pc}(n_{A_1} \times |B|) + 4L_{pc}(n) \Bigg), \\ f(X \cup \{z\}) = -\Bigg(&\log \binom{n}{n_{A_1}} + \log \binom{n}{n_{A_2}} + \log \binom{n}{n_{A_3}} + \log \binom{n}{n_B - n_{A_1} - n_{A_3}} \\ &+ L_{pc}(n_{A_1}) + L_{pc}(n_{A_1} \times |B|) + L_{pc}(n_{A_3}) + L_{pc}(n_{A_3} \times |B|) + 4L_{pc}(n) \Bigg), \end{aligned}$$



(a) Submodularity counterexample.



(b) Monotonicity counterexample.

Figure 3: Data used as counterexamples for submodularity and monotonicity of $L(D, M)$. For the submodularity counterexample (a), $n = 1000, n_{A_1} = 500, n_{A_2} = 100, n_{A_3} = 100, n_B = 700$. For the monotonicity counterexample (b), $n = 1000, n_A = 850, n_B = 700, n_C = 700$.

$$
\begin{aligned}
f(Y) = -\Bigg(&\log \binom{n}{n_{A_1}} + \log \binom{n}{n_{A_2}} + \log \binom{n}{n_{A_3}} + \log \binom{n}{n_B - n_{A_1} - n_{A_2}} \\
&+ L_{pc}(n_{A_1}) + L_{pc}(n_{A_1} \times |B|) + L_{pc}(n_{A_2}) + L_{pc}(n_{A_2} \times |B|) + 4L_{pc}(n) \Bigg), \\
f(Y \cup \{z\}) = -\Bigg(&\log \binom{n}{n_{A_1}} + \log \binom{n}{n_{A_2}} + \log \binom{n}{n_{A_3}} + L_{pc}(n_{A_1}) + L_{pc}(n_{A_1} \times |B|) \\
&+ L_{pc}(n_{A_2}) + L_{pc}(n_{A_2} \times |B|) + L_{pc}(n_{A_3}) + L_{pc}(n_{A_3} \times |B|) + 3L_{pc}(n) \Bigg).
\end{aligned}
$$

Note that the error matrices are basically zero cost, because in this counterexample we do not have any errors. If we use the definition of submodularity from above and plug in the numbers given above, we get:

$$
\begin{aligned}
f(X \cup \{z\}) - f(X) &\geq f(Y \cup \{z\}) - f(Y) \\
\log \binom{n}{n_B - n_{A_1}} - \log \binom{n}{n_B - n_{A_1} - n_{A_3}} &\geq \log \binom{n}{n_B - n_{A_1} - n_{A_2}} + L_{pc}(n) \\
716.9 - 464.4 &\geq 464.4 + 5.33 \; \lightning,
\end{aligned}
$$

which is a contradiction.

Monotonicity

The next question that can be raised is if the codelength function is monotonically decreasing with respect to the search space. Let us first define Monotonicity formally.

Definition 3 (Monotonicity)**.** *Given an alphabet* Ω *and a set function* $f : 2^{\Omega} \to \mathbb{R}$*, a set function is called monotonically decreasing iff* $\forall X \subset Y \subseteq \Omega.\ f(X) \geq f(Y)$*. Analogously, the function is called monotonically increasing iff* $\forall X \subset Y \subseteq \Omega.\ f(X) \leq f(Y)$*.*

It is easy to see that the function is not monotonically decreasing, as if we add a rule that only covers parts of the data already explained by a different rule, there are no bits saved because all rules are send independently and thus we just pay additional bits to send the part where the new rule holds. Consider the example database given in Figure 3b. If we start with a model $X = \{A \to B, A \to C\}$, and add a rule such that $Y = X \cup \{A \to B \cup C\}$, it is easy to see that the codelength is increasing, as we redundantly encode the data B and C with the new rule. The exact calculations are left as an exercise.

A.1.3 Apriori on mushroom

To illustrate the (well-known) impracticality of using Apriori to mine useful association rules, for different support and confidence parameters we show in Fig. 4 the number of rules Apriori discovers on the *Mushroom* data. We see that even at 90% confidence and 10% frequency, i.e. approx. 800 out of 8024 rows of data) we discover close to $10M$ rules, which hardly is a useful result on a dataset of 8 024 transactions over 124 items.

A.1.4 Gain estimates

In this section we provide all formulas necessary to compute first and second tier estimates of the gain. The first and second tier estimate computations are explained throughout the next subsections.

Merging singletons As we encode singleton rules $\emptyset \to A, A \in \mathcal{I}$ and more complex rules $X \to Y,\ |X| > 0 \vee |Y| > 1$ with two different models, we need to have different gain estimates for all combinations of merging different types of rules. In this section, we provide definitions of all estimates for all combinations.

Suppose we have rules $\emptyset \to A, \emptyset \to B,\ A, B \in \mathcal{I}$, we get the gain estimate for $r = \emptyset \to AB$ by considering all possible splitpoints for the error matrices,

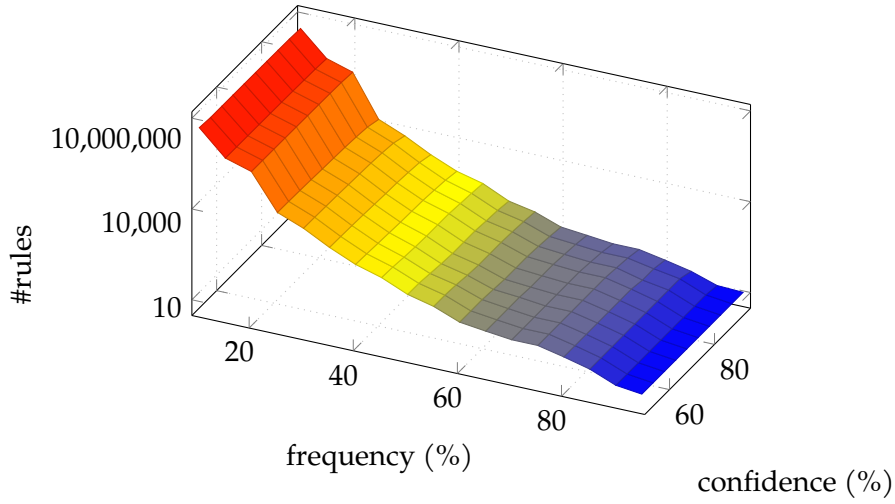


Figure 4: *Rule discovery with* APRIORI. The number of rules APRIORI discovers on the *Mushroom* data for varying levels of frequency and confidence.

assuming perfect overlap for the first tier estimate and partial overlap given by n_{est} for the second tier estimate:

$$\begin{aligned}\widehat{\Delta}_1(r) = &- \log \binom{n}{n_{\emptyset \to A}} - \log \binom{n}{n_{\emptyset \to B}} - 2 \times L_{pc}(n) \\ &+ \min \Bigg(\log \binom{2n}{n_A + n_B} + L_{pc}(n) + L_{pc}(2n), \\ &\log \binom{n}{n_A} + \log \binom{2n_A}{n_A + n_B} + L_{pc}(n) + L_{pc}(2n_A) + L_{pc}(2(n - n_A)), \\ &\log \binom{n}{n_B} + \log \binom{2(n - n_B)}{n_A - n_B} + L_{pc}(n) + L_{pc}(2n_B) + L_{pc}(2(n - n_B)) \Bigg),\end{aligned}$$

$$
\begin{aligned}
\widehat{\Delta}_2(r) = &- \log \binom{n}{n_{\emptyset \to A}} - \log \binom{n}{n_{\emptyset \to B}} - 2 \times L_{pc}(n) + \hat{L}(M) \\
&+ \min \Bigg(\log \binom{2n}{n_A + n_B} + L_{pc}(n) + L_{pc}(2n), \\
&\log \binom{n}{n_{est}} + \log \binom{2(n - n_{est})}{n_A + n_B - 2n_{est}} \\
&+ L_{pc}(n) + L_{pc}(2n_{est}) + L_{pc}(2(n - n_{est})), \\
&\log \binom{n}{n_A + n_B - n_{est}} + \log \binom{2(n_A + n_B - n_{est})}{n_A + n_B} \\
&+ L_{pc}(n) + L_{pc}(2(n_A + n_B - n_{est})) + L_{pc}(2(n - n_A - n_B + n_{est})) \Bigg),
\end{aligned}
$$

with $n_I = |T_I|$ the support of item I, $n_{X \to Y} = |T^M_{Y|X}|$ the current usage of the singleton rule, and $n_{est} = |T_A \cap T_B|$ the estimated support of the new rule r. We compute the change in the model costs regarding sending information about a rule, that is the terms appearing in $L(M)$, exactly and abbreviate it here with $\hat{L}(M)$.

Similarly, we get the gain estimates for $r = A \to B$:

$$
\begin{aligned}
\widehat{\Delta}_1(r) = &- \log \binom{n}{n_{\emptyset \to B}} - L_{pc}(n) \\
&+ \log \binom{n_A}{n_B} + L_{pc}(n_A) + L_{pc}(n_B) + L_{pc}(n_A - n_B)\,,
\end{aligned}
$$

$$
\begin{aligned}
\widehat{\Delta}_2(r) = &- \log \binom{n}{n_{\emptyset \to B}} - I(T^M_{B|\emptyset} \subseteq T_A) \times L_{pc}(n) + \hat{L}(M) \\
&+ \log \binom{n_A}{n_{est}} + \log \binom{n}{|T^M_{B|\emptyset} \setminus T_A|} \\
&+ L_{pc}(n_A) + L_{pc}(n_{est}) + L_{pc}(n_A - n_{est})\,.
\end{aligned}
$$

Note that in the first line of the second tier gain estimate, we have the indicator variable I that evaluates to 1 iff $T^M_{B|\emptyset} \subseteq T_A$. That is, we only substract the model costs for the old singleton rule $\emptyset \to B$ if it is not used anywhere else.

Merging a singleton with a pattern If we combine a rule $\emptyset \to X, |X| > 1$ with a singleton rule $\emptyset \to A$, we need to consider three different cases, each of which has a different gain estimate. First, let us consider the case of merging the tails to $r = \emptyset \to XA$.

For the first tier gain estimate we assume a perfect overlap between the

dense error matrix of $\emptyset \to X$ and the transactions supporting A. In essence, we assume that A overlaps with the most dense part of $\emptyset \to X$. We can get the splitpoint (Eq. 2.1) by estimating the new count vector for the first tier gain estimate as follows. We generate a new count vector of size equal to the size of the new tail plus one for the cases where none of the items in the tail hold. Starting from the largest bin (i.e. all items of the tail hold), for each bin we merge all combinations of the count bins of the generating rules that would yield this number of items. E.g. for rules $r_1 = \emptyset \to AB$ and $r_2 = \emptyset \to C$ we have counts $B_1 = [2, 10, 47]$ and $B_2 = [1, 55]$, we merge these two count vectors into $B' = [1, 3, 8, 47]$. With these counts we can then estimate the new splitpoint k' and the corresponding transaction set and error matrix counts $|T^M_{XA|\emptyset}|'$, and $\mathbb{1}'(T^M_{XA|\emptyset})$, as well as $|T^M_{\cancel{XA}|\emptyset}|'$, and $\mathbb{1}'(T^M_{\cancel{XA}|\emptyset})$ for the number of rows and number of 1s in a matrix, respectively. Be aware of the prime as a notation for this approximation of the overlap. For the second tier gain estimate we compute error matrix counts by intersecting the corresponding tid lists for each level according to Eq. 2.1. We indicate these counts by $k*$ superscript. The estimates are hence given as

$$
\begin{aligned}
\widehat{\Delta}_1(r) = &- L(\emptyset \to X) - \log \binom{n}{n_{\emptyset \to A}} + \log \binom{n}{|T^M_{XA|\emptyset}|'} \\
&+ \log \binom{|T^M_{XA|\emptyset}|' \times (|X|+1)}{\mathbb{1}'(T^M_{XA|\emptyset})} + \log \binom{|T^M_{\cancel{XA}|\emptyset}|' \times (|X|+1)}{\mathbb{1}'(T^M_{\cancel{XA}|\emptyset})} \\
&+ L_{pc}(n) + L_{pc}\left(|T^M_{XA|\emptyset}|' \times (|X|+1)\right) + L_{pc}\left(|T^M_{\cancel{XA}|\emptyset}|' \times (|X|+1)\right), \\
\widehat{\Delta}_2(r) = &- L(\emptyset \to X) - \log \binom{n}{n_{\emptyset \to A}} + \log \binom{n}{|T^M_{XA|\emptyset}|^{k*}} + \hat{L}(M) \\
&+ \log \binom{|T^M_{XA|\emptyset}|^{k*} \times (|X|+1)}{\mathbb{1}(T^M_{XA|\emptyset})^{k*}} + \log \binom{|T^M_{\cancel{XA}|\emptyset}|^{k*} \times (|X|+1)}{\mathbb{1}(T^M_{\cancel{XA}|\emptyset})^{k*}} \\
&+ L_{pc}(n) + L_{pc}\left(|T^M_{XA|\emptyset}|^{k*} \times (|X|+1)\right) + L_{pc}\left(|T^M_{\cancel{XA}|\emptyset}|^{k*} \times (|X|+1)\right).
\end{aligned}
$$

The second case for merging $\emptyset \to X$ and $\emptyset \to A$ is that the singleton becomes the new head, i.e. $r = A \to X$. Here, as before, we assume that tids with X are a subset of tids of A. For the second tier estimate we get the overlap of X with A by intersecting the tid list of the destructive noise matrix of X and the tid list of A to find out where X holds. Furthermore, we estimate increase

of the singleton costs for all singletons $I \in X$ that are not covered any longer by looking at transactions $T^M_{I|\emptyset} \cup (T_I \setminus T_A)$. With n_{est} given as the number of transactions where A is present and where $\emptyset \to X$ holds, we thus obtain

$$\begin{aligned}\widehat{\Delta}_1(r) = &+ \log \binom{n_A}{|T^M_{X|\emptyset}|} + \log \binom{|T^M_{X|\emptyset}| \times |X|}{\mathbb{1}(T^M_{X|\emptyset})} + \log \binom{(n_A - |T^M_{X|\emptyset}|) \times |X|}{\mathbb{1}(T^M_{\not X|\emptyset})} \\ &- L(\emptyset \to X) + L_{pc}(n_A) + L_{pc}(|T^M_{X|\emptyset}| \times |X|) + L_{pc}((n_A - |T^M_{X|\emptyset}|) \times |X|)\,,\end{aligned}$$

$$\begin{aligned}\widehat{\Delta}_2(r) = &- L(\emptyset \to X) + \hat{L}(M) + \sum_{I \in X} \left(- \log \binom{n}{|T^M_{I|\emptyset}|} + \log \binom{n}{|T^M_{I|\emptyset} \cup (T_I \setminus T_A)|} \right) \\ &+ \log \binom{n_A}{n_{est}} + \log \binom{n_{est} \times |X|}{\min\left(\mathbb{1}(T^M_{X|\emptyset}), n_{est} \times |X|\right)} \\ &+ \log \binom{(n_A - n_{est}) \times |X|}{\min\left(\mathbb{1}(T^M_{\not X|\emptyset}) + \max\left(0, \mathbb{1}(T^M_{X|\emptyset}) - n_{est} \times |X|\right), (n_A - n_{est}) \times |X|\right)} \\ &+ L_{pc}(n_A) + L_{pc}(n_{est} \times |X|) + L_{pc}((n_A - n_{est}) \times |X|)\,.\end{aligned}$$

The third case of the mixed rule estimates is that X becomes the new head, we thus get $r = X \to A$. The first estimate is straightforward, assuming that the tids of the singleton rule is a subset of occurences of tids of X. For the second tier gain estimate we obtain the usage of the new rule by intersecting the tid lists of the two merged rules as before, we only need to take care of the case that not all singletons occur together with X. In that case we have to send an adapted code for the singleton. For this singleton code, the regret term is taken care of by the indicator variable in the first line of the second tier estimate, the data costs are covered by the estimate $|T_A \cap T_X|$ of how much less of A we have to encode with a singleton using r. Note that, as before, we still have to send complexity terms for the error matrices, hence we get

$$\begin{aligned}\widehat{\Delta}_1(r) = &- \log \binom{n}{n_{\emptyset \to X}} - L_{pc}(n) + \log \binom{n_X}{n_A} \\ &+ L_{pc}(n_X) + L_{pc}(n_A) + L_{pc}(n_X - n_A)\,,\end{aligned}$$

$$\begin{aligned}\widehat{\Delta}_2(r) = &- \log \binom{n}{n_{\emptyset \to X}} - I(T^M_{X|\emptyset} \subseteq T_X) \times L_{pc}(n) + \hat{L}(M) + \log \binom{n_X}{n_{est}} \\ &+ \log \binom{n}{|T^M_{X|\emptyset} \setminus T_X|} - \log \binom{n}{|T^M_{A|\emptyset}|} + \log \binom{n}{|T^M_{A|\emptyset}| - |T_A \cap T_X|} \\ &+ L_{pc}(n_X) + L_{pc}(n_{est}) + L_{pc}(n_X - n_{est}) \,,\end{aligned}$$

where $I(l)$ is the indicator variable evaluation to 1 if the statement l is true, and 0 otherwise.

Merging general rules What remains are the estimates for merging two "proper" rules $X \to Y$ and $X \to Z$. To obtain the first tier gain estimate of merging the tails to $r = X \to YZ$, we assume that the most dense parts of the destructive noise matrices overlap. Hence we can use our approach of before and estimate the splitpoint k' and counts B' by merging always the counts $B_1[i]$ and $B_2[j]$ with i, j as large as possible, thus getting estimates for the error matrices $|T^M_{YZ|X}|'$, $\mathbb{1}'(T^M_{YZ|X})$ and so on. For the second tier gain estimate we obtain a splitpoint k^* by intersecting the corresponding tid lists for each level pair. This yields

$$\begin{aligned}\widehat{\Delta}_1(r) = &- L(X \to Y) - L(X \to Z) + \log \binom{n_X}{|T^M_{YZ|X}|'} \\ &+ \log \binom{|T^M_{YZ|X}|' \times |Y \cup Z|}{\mathbb{1}'(T^M_{YZ|X})} + \log \binom{|T^M_{\not{YZ}|X}|' \times |Y \cup Z|}{\mathbb{1}'(T^M_{\not{YZ}|X})} \\ &+ L_{pc}(|T^M_{YZ|X}|' \times |Y \cup Z|) - L_{pc}(|T^M_{\not{YZ}|X}|' \times |Y \cup Z|) \,,\end{aligned}$$

$$\begin{aligned}\widehat{\Delta}_2(r) = &- L(X \to Y) - L(X \to Z) + \hat{L}(M) + \log \binom{n_X}{|T^M_{YZ|X}|^{k^*}} \\ &+ \log \binom{|T^M_{YZ|X}|^{k^*} \times |Y \cup Z|}{\mathbb{1}(T^M_{YZ|X})^{k^*}} + \log \binom{|T^M_{\not{YZ}|X}|^{k^*} \times |Y \cup Z|}{\mathbb{1}(T^M_{\not{YZ}|X})^{k^*}} \\ &+ L_{pc}(|T^M_{YZ|X}|^{k^*} \times |Y \cup Z|) + L_{pc}(|T^M_{\not{YZ}|X}|^{k^*} \times |Y \cup Z|) + L_{pc}(n_X) \,.\end{aligned}$$

For the other case, where we merge the head to $r = XY \to Z$, we assume for $\widehat{\Delta}_1$ that the transactions where $X \to Z$ holds overlaps perfectly with the area where $X \cup Y$ holds. For the second tier gain estimate we get the exact overlap counts by intersecting the tid lists of each level. We estimate the singleton costs by adding up the potential costs of increasing the singleton counts by the number of transaction not covered anymore, i.e. estimating the new costs by

$\sum_{I \in Z} \left(\log \binom{n}{|T^M_{I|\emptyset} \cup (T_I \setminus T_{XY})|} \right)$. With notation as before, we obtain

$$\begin{aligned}
\widehat{\Delta}_1(r) = &- L(X \to Z) + \log \binom{n_{XY}}{\min\left(n_{XY}, |T^M_{Z|X}|\right)} + \hat{L}(M) \\
&+ \log \binom{\min\left(n_{XY}, |T^M_{Z|X}|\right) \times |Z|}{\min\left(\min\left(n_{XY}, |T^M_{Z|X}|\right) \times |Z|, \mathbb{1}(T^M_{Z|X})\right)} \\
&+ \log \binom{\max\left(0, n_{XY} - |T^M_{Z|X}|\right) \times |Z|}{\min\left(\mathbb{1}(T^M_{\not Z|X}), \max\left(0, n_{XY} - |T^M_{Z|X}|\right) \times |Z|\right)} \\
&+ L_{pc}(n_{XY}) + L_{pc}(\min\left(n_{XY}, |T^M_{Z|X}|\right) \times |Z|) \\
&+ L_{pc}(\max\left(0, n_{XY} - |T^M_{Z|X}|\right) \times |Z|),
\end{aligned}$$

$$\begin{aligned}
\widehat{\Delta}_2(r) = &- L(X \to Z) + \hat{L}(M) + \log \binom{|T^M_{Z|XY}|^{k^*} \times |Z|}{\mathbb{1}(T^M_{Z|XY})^{k^*}} + \log \binom{|T^M_{\not Z|XY}|^{k^*} \times |Z|}{\mathbb{1}(T^M_{\not Z|XY})^{k^*}} \\
&+ \sum_{I \in Z} \left(- \log \binom{n}{|T^M_{I|\emptyset}|} + \log \binom{n}{|T^M_{I|\emptyset} \cup (T_I \setminus T_{XY})|} \right) \\
&+ L_{pc}(n_{XY}) + L_{pc}(|T^M_{Z|XY}|^{k^*} \times |Z|) + L_{pc}(|T^M_{\not Z|XY}|^{k^*} \times |Z|).
\end{aligned}$$

A.1.5 DATA

In this section, we briefly summarize the composition of the real datasets used to analyze GRAB.

- **Abstracts** A collection of $m = 3933$ stemmed words (features) for $n = 859$ abstracts of ICDM 2001-2007 (samples).
- **Accidents** A dataset of roughly $n = 340000$ records of traffic accidents in Belgium[1] with binarized meta information about the accident's location and circumstances summing up to $m = 468$ features.
- **Adult** A collection of Census income data with $m = 97$ binary variables providing information about $n = 10830$ persons and if their income exceeds 50K.
- **Covtype** Data about forest cover types for $n = 581012$ small areas along with meta information about the area, such as soil type and slope, building a data set with $m = 105$ binary features.

[1] http://fimi.ua.ac.be

- **DNA** Microarray measurements of DNA amplification comprising data about copy number amplification for $n = 4590$ cases and $m = 391$ sites (binarized).
- **Mammals** A record of $n = 2183$ geo locations across europe along with longitude an latitude stating for $m = 121$ different mammals if it has been sighted at this location.
- **Mushrooms** A collection of hypothetical samples of $n = 8124$ Mushrooms generated from The Audubon Society Field Guide to North American Mushrooms, with $m = 119$ binary attributes such as cap shape or spore color.
- **Plants** For $m = 70$ states in the United States and Canada, for $n = 22632$ plants information has been gathered, whether the plant appears in the state or not.

A.2 WHAT'S IN THE BOX?

In this section we will give extended examples on how to compute the MDL score for a given database and set of rules, elaborate on the error encoding for the rule tails, and give a visual toy example on the impact of the extended pattern language for the rule head.

A.2.1 THE IMPACT OF THE EXTENDED PATTERN LANGUAGE

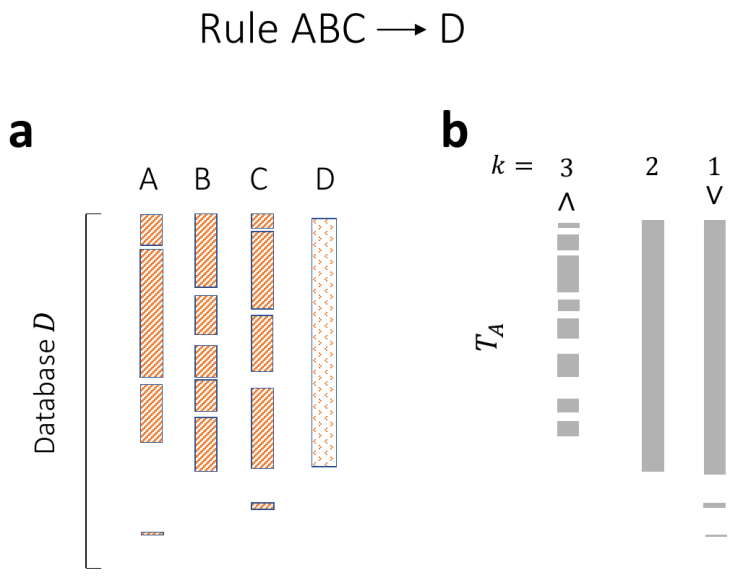


Figure 5: *Example of the impact of noise.* For a given database D given in a, where blocks indicate the occurrence of items, a good rule is given by $ABC \to D$. Due to high noise, the simple conjunctive pattern language results in a bad representation on where the rule should apply, visualized on the left of b. More relaxed definitions towards disjunctions, where we only require l items of the head to be present in the transaction, result in much more stable representation on where the rule applies.

Extending the pattern language for rule heads is crucial to be applicable for tracing activation patterns through a neural network. We here continue the example of the first chapter (see Appendix A.1.1) highlighting one reason why we need the extended pattern language. First of all, we need to start from labels, which are inherently activated mutually exclusive, as we only have a single label as classification. To find shared features of labels, it is essential to

be able to express disjunctions with rule heads. Furthermore, the data as well as activation patterns across the data are very noisy. Thus, determining where a rule applies just based on conjunctions of features can give a very twisted look of the data at hand, as visualized in Fig. 5. That is the reason to introduce a more flexible language similar to approximate rule tails, which solves these issues.

A.2.2 EXPERIMENTS AND DATA

Here, we detail the setup and training of the individual networks, and provide further experimental results. In particular, we first discuss the training setup for MNIST and highlight key results in App. Sec. A.2.2, and then provide additional insights into *ImageNet* prototypes. For *ImageNet*, we first shortly discuss prototypes obtained for GoogLeNet – a different network architecture than VGG – in App. Sec. A.2.2 and then proceed to show additional results on VGG-S for *ImageNet* in App. Sec. A.2.2.

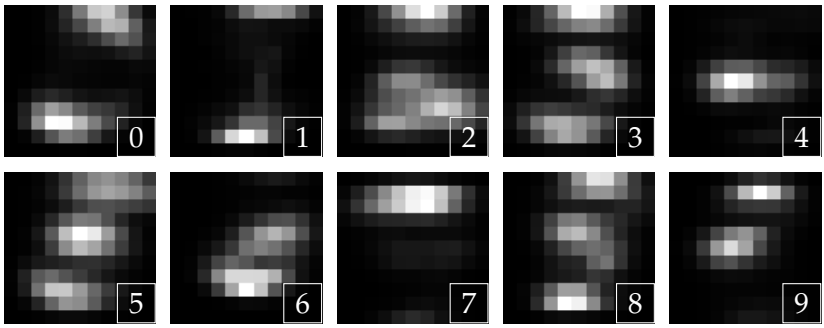
MNIST TRAINING

We trained a CNN on MNIST using the Keras framework, using 60000 images for training and 10000 images as hold out test set for evaluation. The network consists of 2 convolutional layers, with 20 filters in the first layer and 40 filters in the second layer, each using 3x3 kernels and 2x2 maxpooling. The convolutional layers are followed by a Dropout layer with dropout rate .25, and the flattened outputs are passed on to a fully connected layer with 64 nodes with *ReLU* activations. Then follows a dropout layer with rate .5 and the output layer of size 10 with softmax activations. The network was trained using AdaDelta with default parameters based on categorical cross entropy loss over 12 epochs using a batch size of 128. We gathered binarized activations across all filters and applied ExplaiNN to build rules from the output layer to the first respectively second convolutional layer. Additionally to the examples in the main thesis, we provide images for filter 36 in layer 2 in Fig. 6. The discovered rules show that it detects horizontal edges in a class specific manner, whereas prototyping and activation maps again fail to reveal this information. Interestingly, these edge-detectors are long known to play an important role in convolutional layers for image tasks.

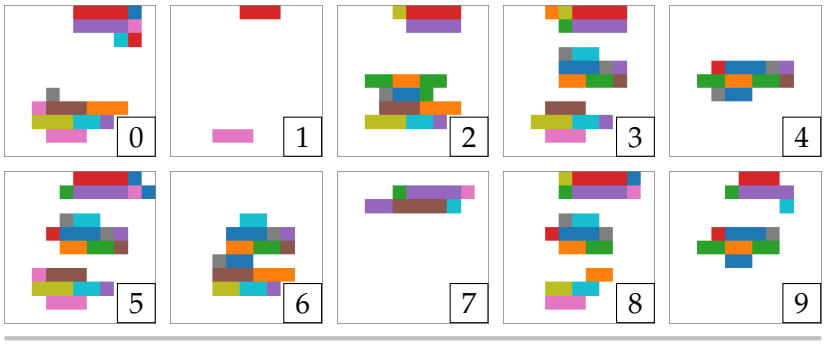
IMAGENET FURTHER RESULTS

Here, we present additional results on the *ImageNet* data set. If not specified directly, pretrained models were obtained from the references indicated in the main manuscript.

GOOGLENET RESULTS To examine if rule mining also works on different network architectures and prototyping methods, we ran GoogLeNet pretrained on *ImageNet* and gathered activation values across the network. Here, we only focus on rules from output to last hidden layer for brevity. Similar to VGG-S, we find expressive rules that span multiple classes and multiple neurons in the last layer, capturing typical structures of the classes (see Fig. 7). We observe,



(a) *MNIST Average activations.* Average activation maps across a class for filter 36.

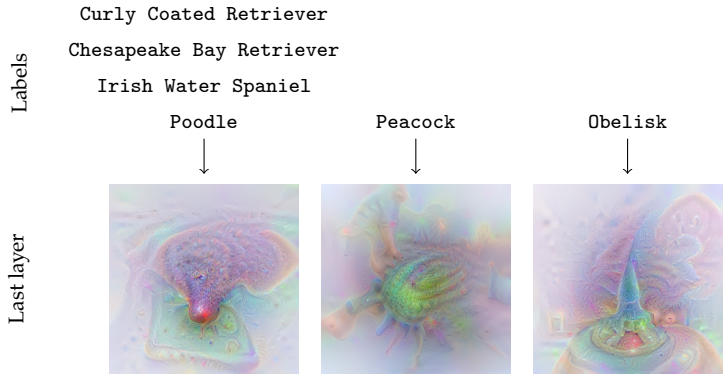


(b) *Horizontal edge detector.* Discovered rules, feature groups found across classes share the same colour.

Figure 6: *Filter visualizations.* Activation maps (a) for the classes, the prototype of the filter (b), and discovered rules (c), over the whole dataset for filter 36 in the second convolutional layer.

however, that this particular prototyping method yields harder to interpret images, which is known to be an issue and not due to the rules.

VGG SHARED NEURONS One key result for the VGG-S network for *ImageNet* is that, similar to the previous MNIST network, traits that are shared between classes are encoded by the same set of neurons. We discovered many such shared traits that the network is able to pick up across classes, which are encapsulated in groups of neurons in the last layer. For example, there are neurons that capture the red beaks of different birds, arch like structures of buildings, tusks of elephants, and the ugly face of a whole group of different dog breeds (see App. Fig. 8).



(a) Prototypes for rules from output label to last hidden layer.

(b) Samples for curly haired dog breeds. From left: `Curly Coated Retriever`, `Chesapeake Bay Retriever`, `Irish Water Spaniel`, `Poodle`

Figure 7: *GoogLeNet results on ImageNet*. (a)Visualizations for the rules found between the labels and the last hidden layer in GoogLeNet. The labels in the rule heads are written above the prototype images of the tail unit groups. Each rule tail captures some interesting features of the corresponding classes: In the first rule the characteristic curly hair of different dog breeds is captured, the second group encapsulates information about the typical colourful plumage of peacocks, the third captures the shape of obelisks. We provide example images of the curly haired dog breeds in (b).

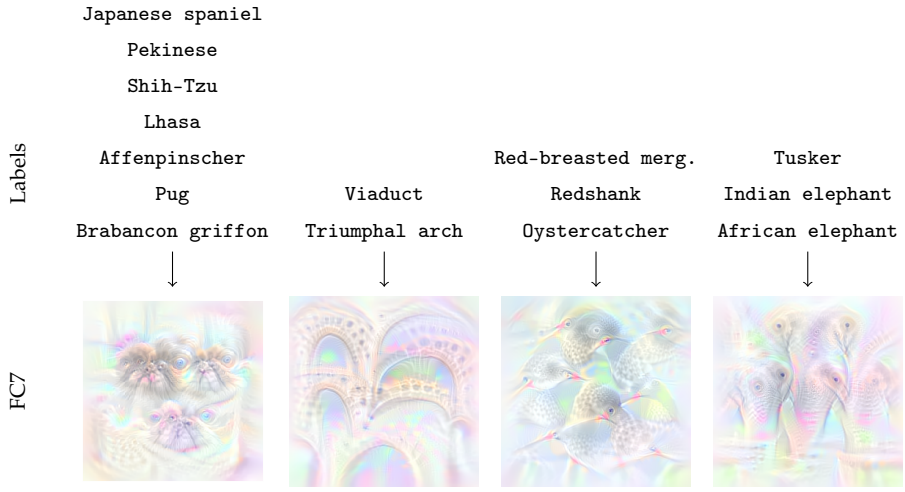


Figure 8: *Shared information across labels.* Visualizations for the rules found between the labels and the last fully connected layer (FC7). The labels in the rule heads are written above the prototype images of the tail unit groups. Each rule tail captures some interesting features of the corresponding classes: In the first rule the characteristic face of different dog breeds is captured, the second group encodes information about the arch structures present for both `Viaduct` and `Triumphal arch`, the third captures the red beaks surrounded by blackish feather that are shared between different birds, and the fourth shows typical heads and tusks of elephants.

A.3 PATTERNS OF MUTUAL EXCLUSIVITY AND CO-OCCURRENCE

THEORY

A.3.1 FISHER'S RECURRENCE

We now proof by induction that the following recurrence relation holds for the terms $\hat{p}_i$ of the summation in $\hat{p}$

$$\hat{p}_0 = \frac{\binom{n-n_{\tilde{X}}}{n_{\tilde{Y}}}}{\binom{n}{n_{\tilde{Y}}}},$$

$$\hat{p}_i = \hat{p}_{i-1} \cdot \frac{(n_{\tilde{X}} - i)(n_{\tilde{Y}} - i)}{(i+1)(n - n_{\tilde{X}} - n_{\tilde{Y}} + i + 1)}.$$

Proof. Induction base. Assume $n_{\tilde{X}\tilde{Y}} = 0$. Obviously holds as $\hat{p}_0$ is the original equation.
Induction step. Assume that the equation holds for i. Then we get

$$\begin{aligned}
&\binom{n_{\tilde{X}}}{i+1}\binom{n-n_{\tilde{X}}}{n_{\tilde{Y}}-i-1} \Big/ \binom{n}{n_{\tilde{Y}}} \\
&= \frac{n_{\tilde{X}}-i}{i+1}\binom{n_{\tilde{X}}}{i}\binom{n-n_{\tilde{X}}}{n_{\tilde{Y}}-i-1} \Big/ \binom{n}{n_{\tilde{Y}}} \\
&= \frac{n_{\tilde{X}}-i}{i+1}\frac{n_{\tilde{Y}}-i}{n-n_{\tilde{X}}-n_{\tilde{Y}}+i+1}\binom{n_{\tilde{X}}}{i}\binom{n-n_{\tilde{X}}}{n_{\tilde{Y}}-i} \Big/ \binom{n}{n_{\tilde{Y}}}
\end{aligned}$$

by using the well known identity for binomials $\binom{n}{k} = \frac{n-k+1}{k}\binom{n}{k-1}$. □

EXPERIMENTS

A.3.2 PROGRAM CALLS

MEXICAN To rule out exploring unreasonable patterns, we constrain MEXICAN to candidates with a minimum overlap of 50% for $\bigwedge$, resp. a maximum overlap of 10% for $\bigotimes$. Furthermore we set the minimum support to 0.1% of the number of rows in the data and a conservative significance threshold $\alpha = 0.001$. To keep the results interpretable for the *SC* data, we set a minimum support threshold for a pattern to 3.

KINGFISHER We set a minimum confidence of 50% for the rule mining and a minimum support of 0.1% of the number of rows in the data and a conservative significance threshold $\alpha = 0.001$, similar to MEXICAN.

LEMINER To somewhat limit the number of patterns that LEMINER finds, we set the maximum number of bits to 1.5 and the dependency threshold to 0.7 and only mine attribute sets (`-s`).

A.3.3 DATA PREPROCESSING

DLQ DATA

The arXiv data base was queried with keywords *Deep Learning*, respectively *Quantum Computing* to retrieve paper abstracts. Words in all abstracts were lemmatized, and words that are not nouns, verbs or adjectives got removed. Five thousand abstracts of each of the two categories were put together in a data set. The words of all abstracts were treated as bag of words and encoded as separate binary columns, words with a support of less than 10 were removed from the bag.

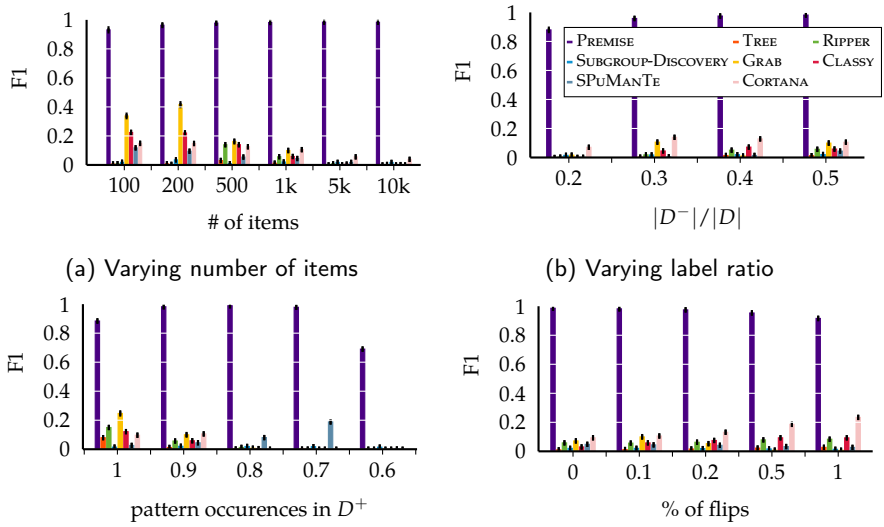
INSTACART

The Instacart dataset originally has many items of the same type but different vendors (e.g. tens of different toilet papers), while transactions are usually very short. This makes the data very sparse, driving the support for even small patterns down to single digits, which in context of a database with more than 2 million rows is nothing. Hence we decided to process the data by hand to summarize similar items into buckets, and focussing on the food data.

SINGLE CELL DATA

The original single cell data was processed to have a transcript level normalization, by using the established TPM (Transcripts Per Million) values. Furthermore, we carried out a per cell normalization, setting the 75% quantile of gene expression levels to 0, the upper quantile to 1, as common in the literature. Columns containing only zeroes were removed and the resulting data was converted into a transaction file format.

A.4 LABEL-DESCRIPTIVE PATTERNS AND THEIR APPLICATION TO CHARACTERIZING CLASSIFICATION ERRORS



(a) Varying number of items

(b) Varying label ratio

(c) Varying label shift

(d) Varying background noise

Figure 9: *Synthetic data results (F1 score).* We visualize results on synthetic data with varying number of items (a), label ratio (b), label shift (c) and amount of background noise (d). The results are in terms of F1 score with respect to the ground truth.

A.5 DIFFERENTIABLE PATTERN SET MINING

A.5.1 TRAINING PARAMETERS

For SLIM, we use the proposed default candidate ordering from their manuscript and provide a minimum support threshold of 10 for pruning. For ASSO, we test $T = \{0.1, 0.2, \cdots, 1\}$, as suggested in their example experiment setup in the codebase. We implemented BINAPS in Pytorch, and for all experiments train the networks using Adam Kingma and Ba (2015), with an initial learning rate of 0.01, and an adaptive learning rate schedule lowering the base learning rate to 0.001 and 0.0001 after epoch 5 and 7, respectively, and train for overall 10 epochs. The exception is *Instacart*, with relatively few features but comparatively many samples. There, we observe a saturation of loss in the third epoch and hence stop after that. As discussed in the main text, Asso is trained for rank selection testing k up to the number of features in the data, similarly we set the capacity c of BINAPS to the number of features. For *Kosarak* and *Genomes*, we provide the methods with $k = c = 1000$. For medium sized data with less than $20k$ features, such as the synthetic data and the *DNA* data set, we use a batch size of 64, for all other data we use a batch size of 32. In general, we recommend these as default values as this setup proved robust about the wide range of experiments we carried out. It is however easy to carry out a parameter grid search evaluating reconstruction loss on a test set. To extract pattern sets from the network, we binarize the weights at a threshold of .2.

A.5.2 BINAPS WITH SMALL n

Here we provide a derivation of the claim that already a single pattern is likely ($> 23\%$) to co-occur with other patterns when planting 100 patterns in 1000 samples with a density of .05 uniformly at random over the transactions. For a patterns p and any other pattern q with marginal frequencies $n_p = 50$ and $n_q = 50$, we are interested in the minimum joint frequency n_{pq} such that the pattern occur statistically significant. To test the hypothesis assuming indpendence between patterns, we use Fisher's exact test $\mathcal{F}$, setting the significance threshold to $\alpha = 0.01$. Searching for the smallest joint frequency n_{pq} such that $\mathcal{F} < \alpha$, we obtain $n_{pq} \geq 8$, meaning that if the patterns co-occur in at least 8 samples their relation is likely to be statistically significant. The next question is how likely an event of p co-occuring with any of the other 99 pattern is, which are planted in the data set. Hence we compute this probability P using the hypergeometric distribution now taking into account the overall number of patterns, yielding

$$P = 99 * \sum_{i=8}^{50} \frac{\binom{50}{i}\binom{950}{50-i}}{\binom{1000}{50}} \approx 0.233.$$

Hence, the chance of observing even just one pattern co-occuring significantly with another pattern is larger than 23%.

A.5.3 INSTACART DATA

We obtained the instacart dataset from the official Kaggle challenge[2] and merged food items of the same type (e.g. all sugar of different brands) each into a single item. This allows us to circumvent problems induced by the extreme sparsity of the database, where many items only occur extremely infrequent, even just once, and thus do not expose any statistical significant relationships, and to be able to find actual patterns such as e.g. *Milk* and *Cookie*, which would not be possible if we would consider all combinations of e.g. brands and types of chocolate cookies. Treating each transaction separately, independent of time and customer id, we obtain a dataset of 1236 food items appearing in 2704831 transactions.

A.5.4 1000 GENOMES DATA

We processed the variant calls of all individuals available in phase 3 of the 1000 Genomes project[3], filtering for autosomal single nucleotide variants (SNVs) with an allel frequency of at least .01. For all protein coding genes specified for the reference genome, we define windows from the transcription start site (TSS) to 1000 base pairs downstream of the TSS. We then filter for SNPs that appear in such a window, and define features in our binary matrix M for all cases where at least one of the alleles show the rare variant ("1|0", "0|1", "1|1"). Thus, the data matrix is of size 3·#filtered variants $\times$ #individuals. For each individual i, we set the data entry $M_{ij} = 1$ if the individual shows genotype j.

[2] https://www.kaggle.com/c/instacart-market-basket-analysis/data
[3] ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502/

A.6 Pruning for lottery tickets with non-zero biases

In the following section, we present the missing proofs and additional experimental results.

A.6.1 ReLU networks without bias are not universal approximators

We, here, provide the necessary derivations for the counter-examples used in Theorem 3. For the example $g(x) = 0.5, x \in [-1, 1]$, by minimizing the mean squared error (MSE), we get a loss of

$$\mathcal{L}(x; w_+, w_-) = \int_{-1}^{1} (g(x) - f(x))^2 \, dx,$$

where $f(x)$ is the neural network, which from Lemma 7.1 we know is only dependent on $w_+, w_- \in \mathbb{R}$. Solving this integral, we obtain

$$\begin{aligned}
\mathcal{L}(x; w_+, w_-) &= \int_{-1}^{1} (g(x) - f(x))^2 \, dx \\
&= \int_{-1}^{0} (0.5 - w_- \phi(-x))^2 dx \\
&\quad + \int_{0}^{1} (0.5 - w_+ \phi(x))^2 dx && \text{(Lemma 7.1)} \\
&= \int_{0}^{1} 0.5 - w_- \phi(x) + w_-^2 \phi(x)^2 \\
&\quad - w_+ \phi(x) + w_+^2 \phi(x)^2 dx \\
&= \int_{0}^{1} 0.5 - w_- x + w_-^2 x^2 - w_+ x + w_+^2 x^2 dx && \text{(Def. } \phi(x) \text{ for } x > 0) \\
&= \Bigg[0.5x - 0.5 w_- x^2 + \frac{1}{3} w_-^2 x^3 \\
&\quad - 0.5 w_+ x^2 + \frac{1}{3} w_+^2 x^3 + C \Bigg]_0^1 . && \text{(Primitive function)}
\end{aligned}$$

We are interested in the network, and hence parameters w_+^*, w_-^*, that minimize the loss. Thus, we solve $\frac{\mathcal{L}(x; w_+, w_-)}{dw_+} \stackrel{!}{=} 0$, which yields $w_+^* = \frac{3}{4}$, and $\frac{\mathcal{L}(x; w_+, w_-)}{dw_-} \stackrel{!}{=}$

0, which yields $w_-^* = \frac{3}{4}$. We can directly see from the shape of the function that these values are indeed a minimum, and not a maximum. Plugging this back into the primitive function above, we obtain $\mathcal{L}(x; w_+^*, w_-^*) = \frac{1}{8}$. Hence, for any $\epsilon < \frac{1}{8}$ there does not exist a network that can approximate the function to this error.

For a slightly more complicated function, without explicit offset, we consider $g(x) = e^x, x \in [-1, 1]$. Analogue to above, we first minimize the MSE

$$\begin{aligned}
\mathcal{L}(x; w_+, w_-) &= \int_{-1}^{0} (e^x - w_- \phi(-x))^2 dx \\
&\quad + \int_{0}^{1} (e^x - w_+ \phi(x))^2 dx \\
&= \int_{0}^{1} e^{-2x} - 2e^{-x} w_- x + w_-^2 x^2 \\
&\quad + e^{2x} - 2e^x w_+ x + w_+^2 x^2 dx \\
&= \Bigg[-0.5e^{-2x} + 2e^{-x} w_- x + 2e^{-x} w_- + \frac{1}{3} w_-^2 x^3 \\
&\quad + 0.5e^{2x} - 2e^x w_+ x + 2e^x w_+ + \frac{1}{3} w_+^2 x^3 + C \Bigg]_0^1 .
\end{aligned}$$

For $\frac{\mathcal{L}(x; w_+, w_-)}{dw_+} \stackrel{!}{=} 0$, we get $w_+^* = 3$, and $\frac{\mathcal{L}(x; w_+, w_-)}{dw_-} \stackrel{!}{=} 0$, which gives $w_-^* = -3(2e^{-1} - 1)$. Again, we can observe from the shape of the function that is given by $f(x)$ that these values are indeed a minimum and not a maximum. Plugging back in to the primitive function, we get $\mathcal{L}(x; w_+^*, w_-^*) = 11.5e^{-1} - 12e^{-2} + 0.5e^2 - 6$. Hence, for any smaller ϵ we do not have a network that can approximate to this error.

A.6.2 Existence of lottery tickets: proof of theorem 7.1

Statement. *Assume that $\epsilon, \delta \in (0, 1)$ and a target network f with depth L and architecture $\bar{n}$ are given. Each weight and bias of a larger deep neural network f_0 with depth $2L$ and architecture $\bar{n}_0$ is initialized independently, uniformly at random according to $w_{ij}^{(l)} \sim U([-\sigma_{w,l}, \sigma_{w,l}])$ and $b_i^{(l)} \sim U([-\prod_{k=1}^{l} \sigma_{w,k}, \prod_{k=1}^{l} \sigma_{w,k}])$. Then, with probability at least $1 - \delta$, f_0 contains an approximation $f_\epsilon \subset f_0$ so that $||f - \lambda f_\epsilon||_\infty \leq \epsilon$ if for $l = 1, ..., L$*

$$n_{2l-1,0} = C n_{l-1} \log\left(\frac{1}{\min\{\epsilon_l, \delta_l\}}\right) \text{ and } n_{2l,0} = n_l,$$

where ϵ_l is defined in Eq. (7.4), $\delta_l = \delta/(Lk_{l-1,max}n_l)$, and the output is scaled by $\lambda = \prod_{l=1}^{2L} \sigma_{w,l}^{-1}$.

Proof. Lemma 7.2 simplifies the above parameter initialization to an equivalent setting, in which each parameter is distributed as $w_{ij}, b_i \sim U[-1,1]$, while the overall output is scaled by the stated scaling factor λ. We assume that all parameters are bounded by 1 so that we can find them within the range $[-1,1]$. Otherwise, we would need to increase n_{2l-1} by a factor that is proportional to the maximum parameter value $\theta_{\max}$, which is integrated into our constant C.

Every layer l of f corresponds in our construction to two layers of f_0, i.e., layers $2l-1$ and $2l$. The neurons in layer $2l$ correspond directly to the output neurons in layer l of f. Thus, we only need width $n_{2l,0} = n_l$ in f_0. Layer $2l-1$ serves the construction of intermediary neurons of in-degree 1. Using the identity $\phi(x) = \phi(x) - \phi(-x)$, we see that all neurons in layer l of f can indeed be represented by a two-layer neural network consisting of $3n_{l-1}$ intermediary neurons of degree 1, as

$$\begin{aligned} x_i^{(l)} &= \phi\left(\sum_j w_{ij}^{(l)} x_j^{(l-1)} + b_i^{(l)}\right) \\ &= \phi\left(\sum_j w_{ij}^{(l)} \phi\left(x_j^{(l-1)}\right) - \sum_j w_{ij}^{(l)} \phi\left(-x_j^{(l-1)}\right) + b_i^{(l)} \phi\left(1\right)\right). \end{aligned}$$

According to Lemma 7.4, we need to approximate each $w_{ij}^{(l)}$ and $-w_{ij}^{(l)}$ up to error $\epsilon_l/2$ and $b_i^{(l)}$ up to error ϵ_l to guarantee our overall approximation objective. Since we have to do this for every parameter, our overall approximation can only be successful with probability $1-\delta$ if we increase our success probability for each parameter, $1-\delta_l$, accordingly. In total, we have $m_{l,\max}$ of such nonzero parameters in layer l with $m_{l,\max} \leq 2n_l k_{l,\max}$. (To be precise, $m_{l,\max}$ denotes the number of parameters that are bigger than ϵ_l). The successes of finding different parameters are not necessarily independent but we can identify a sufficient δ_l with the help of a union bound. Accordingly, $1-\delta \geq 1 - \sum_{l=1}^{L} \delta_l m_{l,\max}$ is fulfilled for $\delta_l \leq \delta/(2Ln_l k_{l,\max})$. Note that we later integrate the factor 2 in the constant related to the layer width.

With probability at least $1-\delta_l$, we can approximate each single parameter by solving the subset sum problem for the corresponding neuron. As outlined in Cor. 2 by Pensia et al. (2020), which is based on Cor. 3.3 by Lueker (1998), we need $C \log\left(\frac{1}{\min(\delta_l,\epsilon_l)}\right)$ neurons in layer $2l-1$ per neuron of the form $\phi\left(\pm x_j^{(l-1)}\right)$ or $\phi(1)$. Since we have to represent $3n_{l-1}$ of these neurons, in total

we require layer $2l-1$ of f_0 to have width

$$n_{2l-1,0} \geq C n_{l-1} \log\left(\frac{1}{\min(\delta_l, \epsilon_l)}\right).$$

Next, we briefly explain the main ideas that lead to this result. The main difference of our situations in comparison with Pensia et al. (2020) is that we additionally create neurons of the form $\phi(1)$ to represent nonzero biases. Let $\phi(y)$ be our target neuron, where y is either $y = x_j^{(l-1)}$ or $y = 1$ depending on which neuron we want to represent. Note that we can construct multiple candidates for a neuron $\phi(y)$ by pruning neurons in layer $2l-1$. We achieve that by setting all weights that do not lead to y in the previous layer and the bias term of a neuron to zero or, if $y = 1$, we set all weights to zero and keep the nonzero bias term if the bias is positive. Let the index set of the such pruned neurons corresponding to y be I. This leaves us with neurons of the form $w_{2,i}\phi(w_{1,i}y)$ with $\text{sign}(y)w_{1,i} \sim U[0,1]$ and $w_{2,i} \sim U[-1,1]$ for $i \in I$. For the probability distribution of $w_{1,i}w_{2,i}$, Cor. 2 of Pensia et al. (2020) states that it contains a uniform distribution. It follow that the subset sum problem has a solution. Thus, for any parameter $\theta \in [-1,1]$ there exists a subset $S \subset I$ so that with probability at least $1-\delta_l$

$$|\theta - \sum_{i \in S} w_{1,i}w_{2,i}| \leq \epsilon_l.$$

if $|I| \geq C \log\left(\frac{1}{\min(\delta_l, \epsilon_l)}\right)$, which was to be shown.

Note that the same result also holds for normally distributed $w_{1,i}$, $w_{2,i}$, as their product contains a uniform distribution. This follows from the fact that the normal distribution contains a uniform distribution (Pensia et al., 2020). Thus, the product of two normal distributions contains a product of two uniform distributions and this product of uniform distributions contains a uniform distribution as stated by Cor. 2 of Pensia et al. (2020). □

'Looks-linear' initializations are also covered by this proof. When we can construct a parameter $w_{ij}^{(l)}$ by solving a subset sum problem, we can construct $-w_{ij}^{(l)}$ in the same way just with the negative correspondents of the parameters that construct $w_{ij}^{(l)}$.

A.6.3 ALGORITHM FOR EDGE-POPUP-SCALED

In this section we outline the proposed `edge-popup-scaled` algorithm. Here, the $getMask(.)$ function returns a binary mask M, where the weights corre-

Algorithm A.1: `edge-popup-scaled`

input : Data $(\mathbf{X}, \boldsymbol{y})$, sparsity ρ, levels e_a, epochs T, mother network f_0 with depth L, loss $\mathcal{L}$

output: Ticket f_ϵ, scaling factor λ_{opt}

Initialize parameters $\boldsymbol{W}_0, \boldsymbol{b}_0, w_{ij}^{(l)} \sim U([-\sigma_{w,l}, \sigma_{w,l}])$ and $b_i^{(l)} \sim U([-\prod_{k=1}^{l}\sigma_{w,k}, \prod_{k=1}^{l}\sigma_{w,k}])$, scores $\boldsymbol{S}, s_{w_{ij}^{(l)}} \sim Const(0.5)$ and $s_{b_i^{(l)}} \sim Const(0.5)$

for $i = 1$ **to** e_a **do**

  $\rho' = \rho^{i/e_a}$

  for t in $[1, .., T]$ **do**

    $\boldsymbol{M} = getMask(f_0, \boldsymbol{S}, \rho')$ // mask largest scores

    $\mathcal{L}(f(\boldsymbol{x}_i, \boldsymbol{M} \cdot [\boldsymbol{W}_0, \boldsymbol{b}_0]), y_i)$ // compute loss

    $\tilde{s}_{w_{ij}^{(l)}} = s_{w_{ij}^{(l)}} - \alpha \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}^{(l-1)}} w_{ij}^{(l)} \phi(\boldsymbol{h}_i^{(l-1)})$ // gradient step

    $\tilde{\lambda} = \text{argmin}_{\lambda>0} \mathcal{L}(y, \lambda x^{(L)})$ // compute optimal scaling

    $w_{ij}^{(l)} = w_{ij}^{(l)} \tilde{\lambda}^{1/L}, \quad b_i^{(l)} = b_i^{(l)} \tilde{\lambda}^{l/L}$ // apply scaling

$\lambda_{opt} = \tilde{\lambda}, \boldsymbol{S}_{opt} = \tilde{\boldsymbol{S}}, \boldsymbol{M}_{opt} = getMask(f_0, \boldsymbol{S}_{opt}, \rho)$

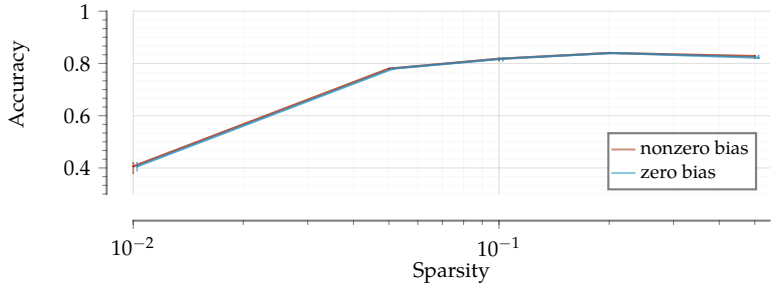
$f_\epsilon = f_0(\boldsymbol{M}_{opt} \cdot [\boldsymbol{W}_0, \boldsymbol{b}_0])$ // final ticket

return $f_\epsilon, \lambda_{opt}$

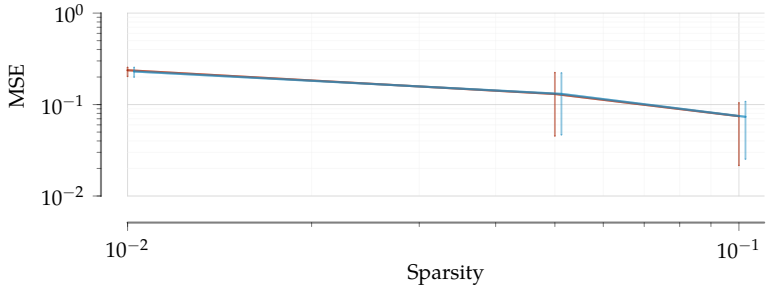
sponding to the largest scores (according to the sparsity ρ) are retained in the mask.

A.6.4 Experiments on imaging data

We conducted experiments on the benchmark CIFAR10 for classification with CNNs and reconstruction with autoencoders. In particular, we pruned He initialized VGG16 models for classification with Edge-Popup with annealing for 5 levels, 50 epochs, and optimized via SGD with momentum of .9, learning rate of .1, and cosine annealing of the learning rate, as discussed in the original paper (Ramanujan et al., 2020). Experiments were repeated 5 times. For reconstruction, we pruned a convolutional autoencoder as detailed in Table. 1, with annealing for 5 levels, 50 epochs, optimized with Adam and a constant learning rate of .001. These set of experiments were repeated 4 times. From the results, reported in Fig. 10, our first observation is that we can success-



(a) VGG16 results.



(b) Autoencoder results.

Figure 10: *CIFAR10 results*. Comparison of EDGE-POPUP results on CIFAR10 with models with zero vs nonzero bias initializations. Reported are mean as well as minimum and maximum obtained from 5 repetitions for classifiation (a) and reconstruction (b) across different sparsity levels.

fully prune for strong lottery tickets for both zero bias as well as nonzero bias initialized networks. Second, the achieved results do not differ significantly. Upon further investigation of the autoencoder tickets, this might be the case due to insufficient pruning of bias parameters, as most of the layers in the ticket do not contain any bias. This could be a systematic property of image data, i.e. convolutions do not require bias parameters for efficient prediction or reconstruction, or could be a shortcoming of current strong lottery ticket pruning to be parameter-inefficient and hence unable to draw advantage from the bias parameters. In recent literature, it was shown that EDGE-POPUP is indeed unable to recover a ground truth ticket of high sparsity for CIFAR10 (Fischer and Burkholz, 2022). Only once more efficient SLT pruners are available, we can give a definite answer to the question of whether biases are necessary for efficent SLTs for imaging data.

Layer name	Description
Encoder 1	Conv(3, 64, 3)
Encoder 2	Conv(64, 64, 3)
Encoder 3	Conv(64, 128, 3)
Encoder 4	Conv(128, 128, 3)
Encoder 5	Conv(128, 128, 3)
Encoder 6	Linear(2048, 512)
Decoder 1	Linear(512, 2048)
Decoder 2	ConvTranspose(128, 128, 3)
Decoder 3	Conv(128, 128, 3)
Decoder 4	ConvTranspose(128, 64, 3)
Decoder 5	Conv(64, 64, 3)
Decoder 6	ConvTranspose(128, 3, 3)

Table 1: *Autoencoder architecture*. Description of stacked autoencoder architecture. Each inner layer is followed by a ReLU activation, the final layer by tanh activation.

A.7 PLANT 'N' SEEK

In the following section, we present the proofs of the theorems and lemmas of the main manuscript.

A.7.1 ERROR PROPAGATION: PROOF OF LEMMA 8.1

Statement. *Assume $\epsilon > 0$ and let the target network f and its approximation f_ϵ have the same architecture. If every parameter θ of f and corresponding θ_ϵ of f_ϵ in layer l fulfils $|\theta_\epsilon - \theta| \leq \epsilon_l$ for*

$$\epsilon_l := \epsilon \left(L\sqrt{m_l} \left(1 + \sup_{x \in [-1,1]^{n_0}} \left\| \boldsymbol{x}^{(l-1)} \right\|_1 \right) \prod_{k=l+1}^{L} \left(\left\| \boldsymbol{W}^{(l)} \right\|_\infty + \epsilon / L \right) \right)^{-1},$$

then it follows that $\|f - f_\epsilon\|_\infty \leq \epsilon$.

Proof. Our objective is to bound $\|f - f_\epsilon\|_\infty \leq \epsilon$. We repeatedly use the triangle inequality and that $|\phi(x) - \phi(y)| \leq |x - y|$ is Lipschitz continuous with Lipschitz constant 1 to derive

$$\begin{aligned}
\left\| \boldsymbol{x}^{(l)} - \boldsymbol{x}_\epsilon^{(l)} \right\|_2 &\leq \left\| \boldsymbol{h}^{(l)} - \boldsymbol{h}_\epsilon^{(l)} \right\|_2 \\
&\leq \left\| \left(\boldsymbol{W}^{(l)} - \boldsymbol{W}_\epsilon^{(l)} \right) \boldsymbol{x}^{(l-1)} \right\|_2 \\
&\quad + \left\| \boldsymbol{b}^{(l)} - \boldsymbol{b}_\epsilon^{(l)} \right\|_2 + \left\| \boldsymbol{W}_\epsilon^{(l)} \left(\boldsymbol{x}^{(l-1)} - \boldsymbol{x}_\epsilon^{(l-1)} \right) \right\|_2 \\
&\leq \epsilon_l \sqrt{m_l} \sup_{x \in [-1,1]^{n_0}} \left\| \boldsymbol{x}^{(l-1)} \right\|_1 + \epsilon_l \sqrt{m_l} \\
&\quad + \left(\left\| \boldsymbol{W}^{(l)} \right\|_\infty + \epsilon_l \right) \left\| \left(\boldsymbol{x}^{(l-1)} - \boldsymbol{x}_\epsilon^{(l-1)} \right) \right\|_2
\end{aligned}$$

with $\epsilon_l \leq \epsilon / L$. m_l denotes the number of parameters in layer l that are smaller than ϵ_l and $\|\boldsymbol{W}\|_\infty = \max_{i,j} |w_{i,j}|$. Note that $m_l \leq n_l k_{l,\max}$. The last inequality follows from the fact that all entries of the matrix $\left(\boldsymbol{W}^{(l)} - \boldsymbol{W}_\epsilon^{(l)} \right)$ and of the vector $(\boldsymbol{b}^{(l)} - \boldsymbol{b}_\epsilon^{(l)})$ are bounded by ϵ_l and maximally m_l of these entries are non-zero. Furthermore, $\left\| \boldsymbol{W}_\epsilon^{(l)} \right\|_\infty \leq \left(\left\| \boldsymbol{W}^{(l)} \right\|_\infty + \epsilon_l \right)$ follows again from the fact that each entry of $\left(\boldsymbol{W}^{(l)} - \boldsymbol{W}_\epsilon^{(l)} \right)$ is bounded by ϵ_l.

Thus, at the last layer it holds for all $\boldsymbol{x} \in [-1,1]^{n_0}$ that

$$\begin{aligned}\|f(x) - f_\epsilon(x)\|_2 &= \left\|\boldsymbol{x}^{(L)} - \boldsymbol{x}_\epsilon^{(L)}\right\|_2 \\ &\leq \sum_{l=1}^{L} \epsilon_l \sqrt{m_l} \left(1 + \sup_{x\in[-1,1]^{n_0}} \left\|\boldsymbol{x}^{(l-1)}\right\|_1\right) \prod_{k=l+1}^{L} \left(\left\|\boldsymbol{W}^{(l)}\right\|_\infty + \epsilon/L\right) \\ &\leq L\frac{\epsilon}{L} = \epsilon,\end{aligned}$$

using the definition of ϵ_l in the last step. □

A.7.2 Lower bound on existence probability: proof of Theorem 8.1

Next, we prove the following lower bound on the probability that a very sparse lottery ticket exists.

Statement. *Assume that $\epsilon \in (0,1)$ and a target network f with depth L and architecture $\bar{n}$ are given. Each parameter of the larger deep neural network f_0 with depth L and architecture $\bar{n}_0$ is initialized independently, uniformly at random with $w_{ij}^{(l)} \sim U\left(\left[-\sigma_w^{(l)}, \sigma_w^{(l)}\right]\right)$ and $b_i^{(l)} \sim U\left(\left[-\prod_{k=1}^{l}\sigma_w^{(k)}, \prod_{k=1}^{l}\sigma_w^{(k)}\right]\right)$. Then, f_0 contains a rescaled approximation f_ϵ of f with probability at least*

$$\mathbb{P}\left(\exists f_\epsilon \subset f_0 : \|f - \lambda f_\epsilon\|_\infty \leq \epsilon\right) \geq \prod_{l=1}^{L}\left(1 - \sum_{i=1}^{n_l}(1-\epsilon_l^{k_i})^{n_{l,0}}\right),$$

where ϵ_l is defined as in Eq. (7.4) and the scaling factor is given by $\lambda = \prod_{l=1}^{L} 1/\sigma_w^{(l)}$.

Proof. As shown by Fischer et al. (2021a), the scaling of the output by λ simplifies the above parameter initialization to an equivalent setting, in which each parameter is distributed as $w_{ij}, b_i \sim U[-1,1]$, while the overall output is scaled by the stated scaling factor λ, again assuming that all parameters are bounded by $1-\epsilon$. Each parameter in Layer l needs to be approximated up to error ϵ_l according to Lemma 7.4. To match the same sparsity level of f, for each neuron i in each Layer l of f, we have to find exactly one neuron in the same layer (Layer l) of f_0 that represents i. We start with matching neurons at Layer 1 (given the input in Layer 0) and proceed iteratively by matching neurons in Layer l given the already matched neurons in Layer $l-1$.

Let us pick a random neuron in Layer l of f_0. How high is the probability that it is a match with a given target neuron i in layer l of f? The neuron i consists of k_i parameters that have to be matched. Since the corresponding neurons in layer $l-1$ of f and f_0 have already been matched according to

our assumption, we only have one possible candidate θ_0 for each of the k_i parameters θ_i. For uniformly distributed parameters, we have $|\theta_i - \theta_0| \leq \epsilon_l$ with probability ϵ_l. For normally distributed $\theta_0 \sim \mathcal{N}(0,1)$, the probability is at least $\epsilon_l/2$ (as long as $|\theta_0 \pm \epsilon| \leq 1$. This can be seen by Taylor approximation of the cdf of a standard normal $\Phi(z+\Delta z) - \Phi(z-\Delta z)$ in z. For the remainder of the proof, however, we assume uniformly distributed parameters. Thus, all k_i independent parameters are a match with probability $\epsilon_l^{k_i}$. Accordingly, none of the available $n_{l,0}$ neurons in Layer l is a match with probability $\left(1-\epsilon_l^{k_i}\right)^{n_{l,0}}$.

With the help of a union bound we can deduce that the probability that at least one of the neurons i in Layer l of f has no match in f_0 is smaller or equal to $\sum_{i=1}^{n_l}\left(1-\epsilon_l^{k_i}\right)^{n_{l,0}}$. Therefore, the converse probability that we find a match for every single neuron in Layer l of f is at least $1-\sum_{i=1}^{n_l}\left(1-\epsilon_l^{k_i}\right)^{n_{l,0}}$.

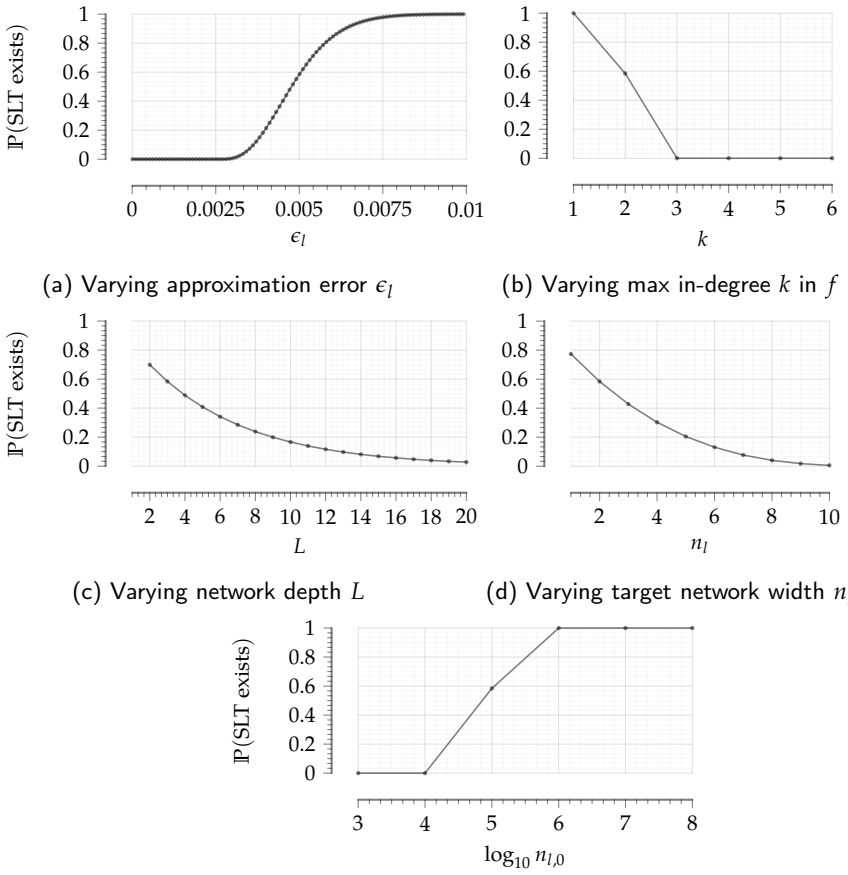
Since we have to guarantee a match for each single layer and the matching probability of a new layer is conditional on the previous layer, we obtain a lower bound on the existence probability of a lottery ticket by multiplying the layerwise bounds. □

This bound is only practical for very sparse target networks f with neurons of small in-degrees k_i. It still shows that the existence of very sparse lottery tickets is possible under the right conditions. To provide an intuition how this bound on the existence probability depends on the relevant parameters, we visualize the bound when one component is varied in a simple example, in which all nodes and layers are homogeneous so that they have identical properties like degree, width, etc. Fig. 11 shows that the bound most critically depends on the degree of a node and the width of the mother network. Yet, all parameters matter and can make the existence of a LT unlikely. Extreme sparsity can pose significant challenges to pruning algorithms, as we also see in our experiments with planted solutions.

Next, we discuss all relevant parameters and set-ups to reproduce the experimental results. Furthermore, we provide additional results obtained for singleshot learning spanning different architectures and pruning schemes. All source code to run pruning algorithms and to generate the data is made publicly available.

A.7.3 OTHER PRUNING APPROACHES

Finetuning Recent results indicated that finetuning discovered subnetworks yields better models than training these subnetworks from scratch (Liu et al., 2021a). In particular, they proposed to use iterative (magnitude) pruning and skip the resetting of the parameters to their initial values, but rather continue training the current parameter set, which incurs no additional computational



(a) Varying approximation error ϵ_l

(b) Varying max in-degree k in f

(c) Varying network depth L

(d) Varying target network width n_l

(e) Varying mother network width $n_{l,0}$

Figure 11: *Visualization of lower bound.* Bound on SLT existence probability for $\epsilon_l = 0.005$, $k_i = 2$, $L = 3$, $n_l = 2$, and $n_{l,0} = 10^5$. In each plot, one single variable is varied while the remaining ones are kept fixed.

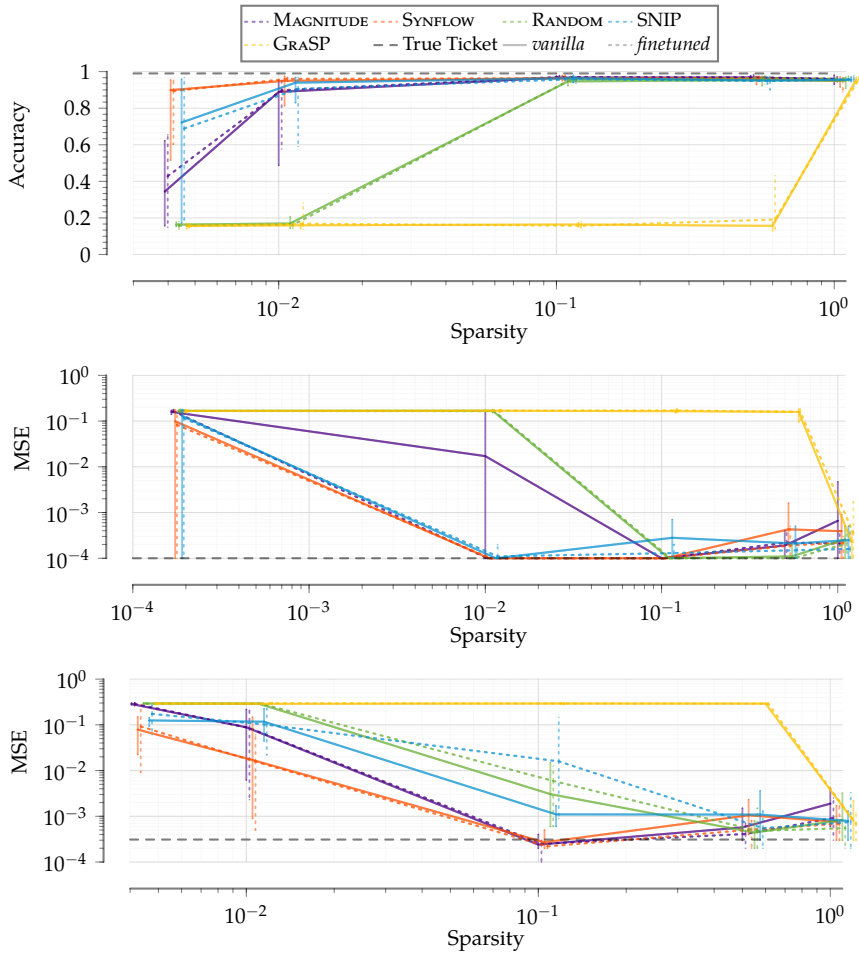


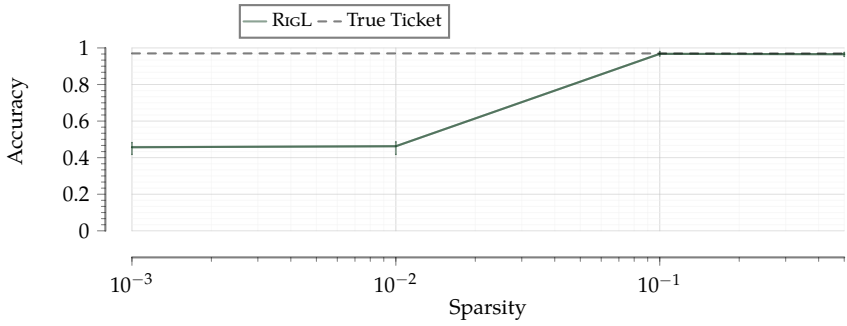
Figure 12: *Finetuning vs initialization.* Performance for `Circle` (top), `ReLU` (middle), and `Helix` (bottom) for 10 rounds of alternating pruning and training. Visualized are weak ticket performances (i.e. training on initialized weights) against finetuned subnetworks (i.e. no reset of weights after final pruning round). Baseline ticket performance in black.

overhead. Here, we investigate how finetuning affects results on our benchmark data for all considered methods. We repeat the multishot learning experiments in combination with finetuning and visualize the results in Fig. 12. For the considered classification task, finetuning matches the performance of classical 'weak ticket'-training with parameter reset, achieving almost perfect accuracy levels for up to .01 target sparsity. For even sparser target networks, we see a marginal improvement of MAGNITUDE and SYNFLOW by using finetuning, which is however within the confidence of the original prediction. For the regression tasks we get a slightly different view: here, the original weak tickets already detoriate in performance for relatively dense target networks of .5 or .1 target sparsity. Notably, for the `ReLU` task, finetuning does improve ticket performance for those sparsity levels, such that the algorithms can match the performance of the ground truth ticket. For more extreme sparsity levels, we see a drastic improvement for MAGNITUDE pruning, but no improvement for other algorithms or at ground truth ticket performance. For `Helix`, the results are mixed, where we observe an improvement at dense target sparsities of .5 similar to before, but no improvement or even a decrease of performance for smaller target sparsity levels. In summary, on our benchmarks, finetuning helps to improve performance at relatively large target sparsity levels ($> .1$), but does not provide an advantage compared to the common training after parameter reset for extreme sparsities.

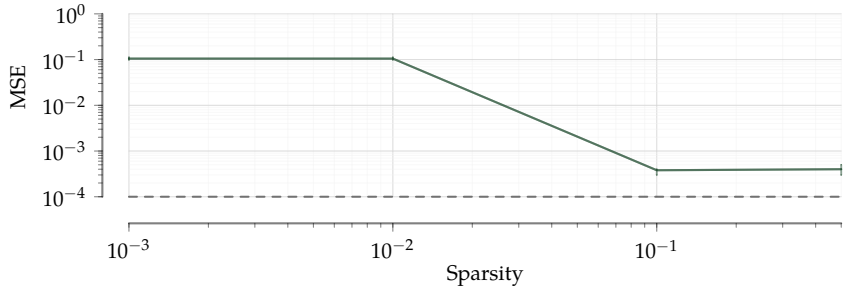
RIGL While the main focus of our paper are lottery tickets, we here briefly discuss results for RIGL (Evci et al., 2020), a state-of-the-art dynamic sparse training approach, which results in sparsified and trained network architectures which are comparable to trained 'weak' tickets. For our benchmark data, we construct similar networks as for the multishot experiments – i.e. depth 6 and width 100 fully connected networks – and run the available implementation of RIGL with default parameters as suggested in the paper, and Adam optimization with the same parameter settings as for all other considered methods. We train networks for 60 epochs, which results in a comparable amount of training time as in the multishot experiments, and note that good performance is reached much earlier.

The original results reported in Evci et al. (2020) indicate that for their considered (classification) tasks, RIGL outperforms other dynamic sparse training methods, and that for target sparsity levels of .1 and lower, performance quickly deteriorates for all methods.[4] In the original paper, there was no exploration of the more extreme sparsities considered here, nor a comparison to ticket pruning other than SNIP. Our results on `Circle` match those results, with RIGL being able to match ground truth ticket performance for sparsity .5 and .1. The performance decreases quickly with extremer sparsity levels (see

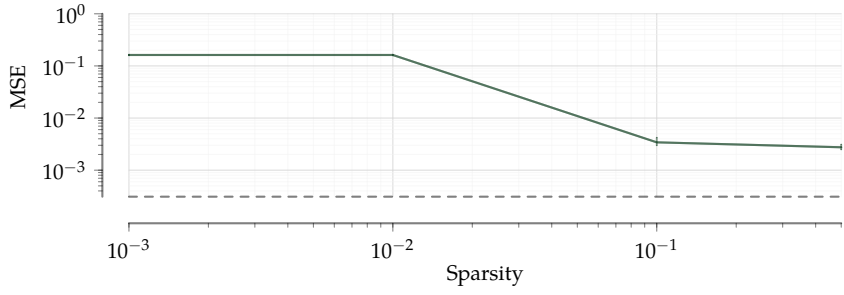
[4]Note that Evci et al. (2020) use percentage of pruned parameters for their plots, i.e. $1-$sparsity in our paper.



(a) RigL on `Circle`.



(b) RigL on `ReLU`



(c) RigL on `Helix`.

Figure 13: RigL. Performance for `Circle` (top), `ReLU` (middle), and `Helix` (bottom) for 60 rounds of training with default parameters. We report mean and minimum and maximum values across 10 repetitions. Baseline ticket performance is indicated in black.

Fig. 13). In comparison with the multishot results, we observe that Synflow, SNIP, and Magnitude pruning outperform RigL on this task for the extreme sparsity levels (compare Fig. 8.5). Note that the version of SNIP used in the original paper is essentially the singleshot approach, which indeed performs worse than RigL (compare Fig. 8.3). For regression tasks, we see a similar trend, with RigL performing comparably good as SNIP for sparsity levels $\geq .1$, but the performance decreases rapidly for more extreme target sparsitiy levels. Generally, for the regression tasks we observe that RigL is outperformed by the state-of-the-art ticket pruner Synflow and iterative Magnitude pruning. Yet, RigL allows for efficient computations, saving FLOPS by only infrequently updating gradients, which render it the method of choice for target sparsities of $\geq .1$ in specific applications.

A.7.4 Additional results on singleshot learning

Model depth In this section, we provide all results from the singleshot learning for depths 3, 10 and width 100 in Figure 14 and 15, respectively. We observe that all methods have problems to deal with smaller networks, while the results for the larger networks are consistent.

Model width To investigate the effect of layer size, we run an experiment with a network of depth 6 and width 1000 on `Circle` – as the layer size is an important factor for the theoretical probability of the existence of tickets – and report the results in Figure 16. We observe that although the network is much larger, there is barely any change in comparison to the previous results. Note that the results after training of tickets of individual sparsities cannot be directly compared directly to the other singleshot results, as the tickets are much larger (due to the much larger number of parameters) and hence easier to train.

Multiple pruning rounds We report the results of running Synflow with 100 rounds of pruning on a network of depth 6 and width 100 for `Circle` in Figure 17. We find that there is again barely any change to the original singleshot results after pruning, but there is a slight increase in performance after training compared to the single-round singleshot results.

A.7.5 Additional results on multishot learning

To test whether we can reach an improvement of the performance of tickets discovered by GraSP using a multishot pruning approach and avoid layer collapse, we ran additional experiments using a local pruning rate. In particular, we used layer-wise pruning setting the target sparsity of the parameters of each individual layer to the global sparsity level. We report results in Fig. 18.

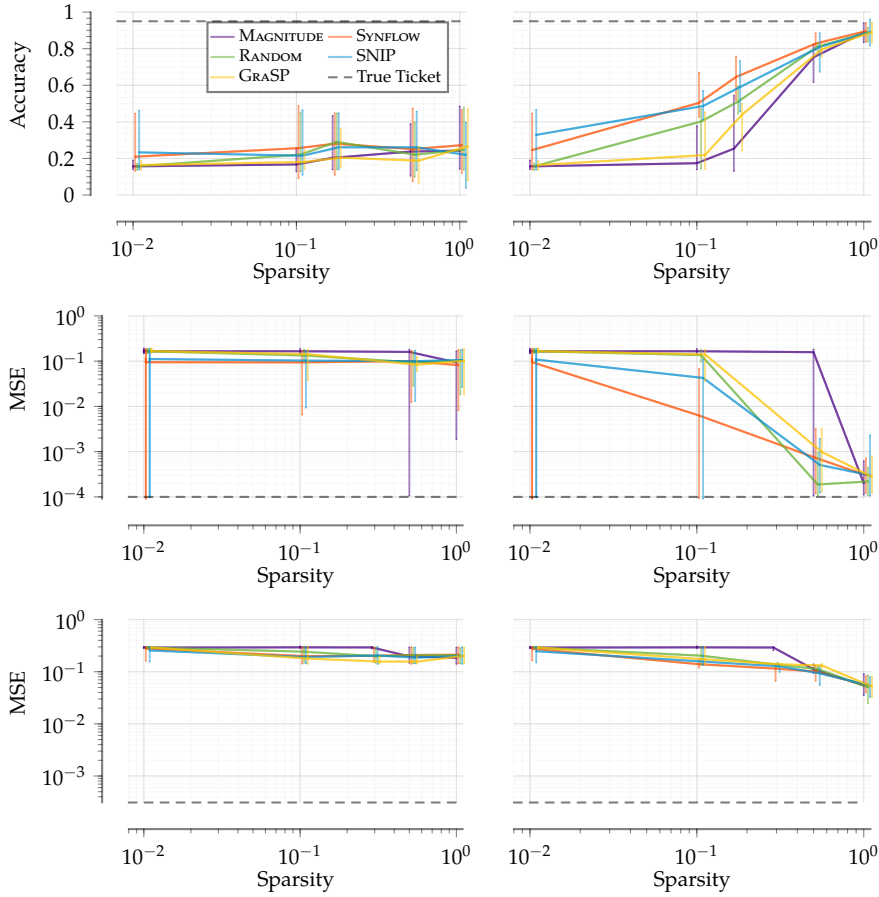


Figure 14: *Singleshot results depth* 3. Performance of discovered tickets for `Circle`, `ReLU`, and `Helix` against target sparsities as mean and obtained intervals (minimum and maximum) across 25 runs. In order of appearance from top to bottom: `Circle`, `ReLU`, and `Helix` post pruning (left) and post training performance (right). Baseline tickets have sparsities of .16, .01, and .29, and their performance is given by black dashed line.

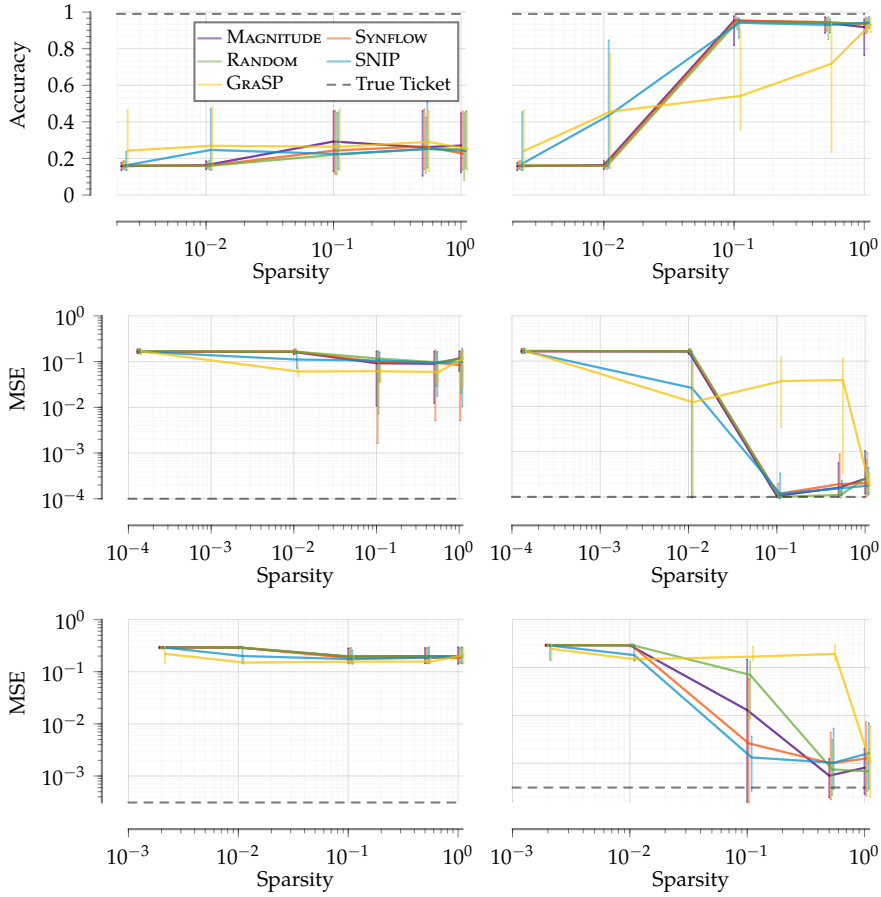


Figure 15: *Singleshot results depth* 10. Performance of discovered tickets for `Circle`, `ReLU`, and `Helix` against target sparsities as mean and obtained intervals (minimum and maximum) across 25 runs. In order of appearance from top to bottom: `Circle`, `ReLU`, and `Helix` post pruning (left) and post training performance (right). Baseline ticket has leftmost sparsity and its performance given by black dashed line.

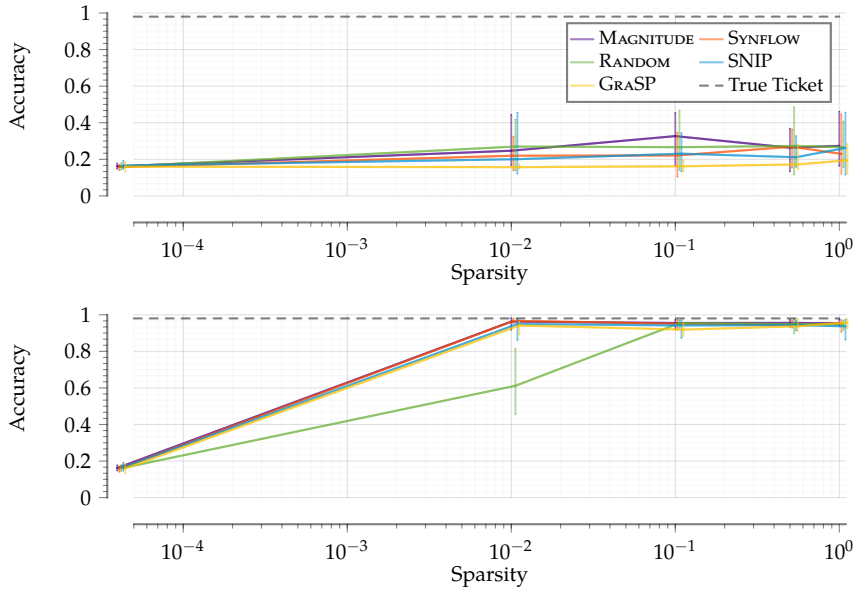


Figure 16: *Singleshot results, depth* 6 *width* 1000. Performance on test data are plotted for `Circle` against target sparsities. We report mean and obtained intervals (minimum and maximum) across 10 repetitions of ticket performance right after pruning (left) and after training (right). The baseline ticket performance is indicated by the black line, leftmost sparsity correspond to planted ticket sparsity.

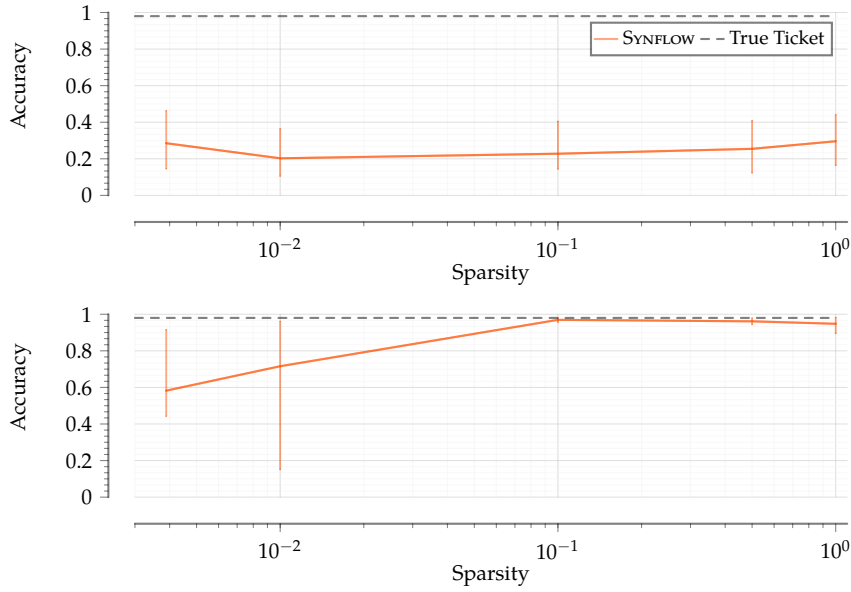


Figure 17: *Singleshot results for* SynFlow *with* 100 *pruning rounds.* Performance on test data are plotted for `Circle` against target sparsities. We report mean and obtained intervals (minimum and maximum) across 10 repetitions of ticket performance right after pruning (left) and after training (right). The original network is of depth 6 and width 100. The baseline ticket performance is indicated by the black line, leftmost sparsity correspond to planted ticket sparsity.

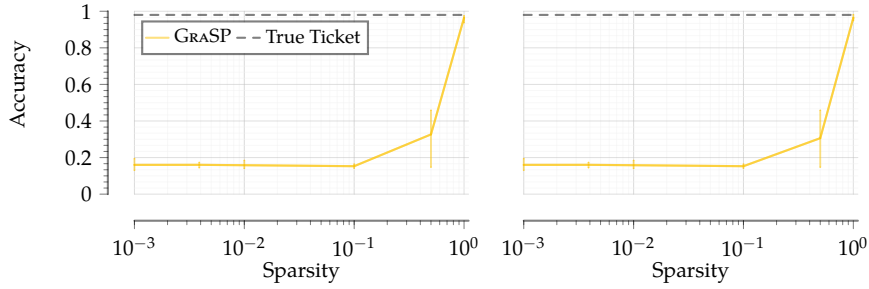


Figure 18: *Multishot with local pruning.* Performance on test data are plotted for `Circle` against target sparsities. We report mean and obtained intervals (minimum and maximum) across 10 repetitions of ticket performance right after pruning (left) and after training (right). The ground truth ticket performance is indicated by the black line, second to left sparsity correspond to planted ticket sparsity.

REFERENCES

A. Adadi and M. Berrada. Peeking inside the black-box: A survey on Explainable Artificial Intelligence (XAI). *IEEE Access*, 6:52138–52160, 2018.

R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of International Conference on Very Large Data Bases*, pages 487–499, 1994.

R. Agrawal, T. Imielinksi, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the ACM International Conference on Management of Data* (*SIGMOD*), pages 207–216. 1993.

D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, E. Elsen, J. H. Engel, L. Fan, C. Fougner, A. Y. Hannun, B. Jun, T. Han, P. LeGresley, X. Li, L. Lin, S. Narang, A. Y. Ng, S. Ozair, R. Prenger, S. Qian, J. Raiman, S. Satheesh, D. Seetapun, S. Sengupta, C. Wang, Y. Wang, Z. Wang, B. Xiao, Y. Xie, D. Yogatama, J. Zhan, and Z. Zhu. Deep speech 2 : End-to-end speech recognition in english and mandarin. In *Proceedings of the International Conference on Machine Learning* (*ICML*), pages 173–182, 2016.

M.-L. Antonie and O. R. Zaïane. Mining positive and negative association rules: An approach for confined rules. In *Proceedings of the European Conference on Principles and Practice of Knowledge Discovery in Databases* (*ECML PKDD*), pages 27–38, 2004.

F. B. Ardakani, K. Kattler, K. Nordström, N. Gasparoni, G. Gasparoni, S. Fuchs, A. Sinha, M. Barann, P. Ebert, J. Fischer, B. Hutter, G. Zipprich, C. D. Imbusch, B. Felder, J. Eils, , B. Brors, T. Lengauer, T. Manke, P. Rosenstiel, J. Walter, and M. H. Schulz. Integrative analysis of single-cell expression data reveals

distinct regulatory states in bidirectional promoters. *Epigen. & chrom.*, 11(1): 66, 2018.

R. Arora, A. Basu, P. Mianjy, and A. Mukherjee. Understanding deep neural networks with rectified linear units. In *International Conference on Learning Representations* (*ICLR*), 2018.

M. Atzmueller. Subgroup discovery. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 5(1):35–49, 2015.

S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

D. Balduzzi, M. Frean, L. Leary, J. P. Lewis, K. W.-D. Ma, and B. McWilliams. The shattered gradients problem: If resnets are the answer, then what is the question? In *Proceedings of the International Conference on Machine Learning* (*ICML*), volume 70, pages 342–350, 2017.

N. Barakat and J. Diederich. Eclectic rule-extraction from support vector machines. *International Journal of Computational Intelligence*, 2(1):59–62, 2005.

O. Bastani, C. Kim, and H. Bastani. Interpreting blackbox models via model extraction. *arXiv preprint arXiv:1705.08504*, 2017.

D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), pages 6541–6549, 2017.

S. D. Bay and M. J. Pazzani. Detecting group differences: Mining contrast sets. *Data Mining and Knowledge Discovery*, 5:213–246, 2001.

R. Bayardo. Efficiently mining long patterns from databases. In *Proceedings of the ACM International Conference on Management of Data* (*SIGMOD*), pages 85–93, 1998.

Y. Bengio, N. Léonard, and A. C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

A. Bhattacharyya and J. Vreeken. Efficiently summarising event sequences with rich interleaving patterns. In *Proceedings of the SIAM International Conference on Data Mining* (*SDM*), pages 795–803, 2017.

K. Budhathoki and J. Vreeken. The difference and the norm – characterising similarities and differences between databases. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases* (*ECML PKDD*), *Porto, Portugal*. Springer, 2015.

R. Burkholz and A. Dubatovka. Initialization of ReLUs for dynamical isometry. In *Proceedings of the Annual Conference on Neural Information Processing Systems* (*NeurIPS*), pages 2382–2392, 2019.

T. Calders and B. Goethals. Non-derivable itemset mining. *Data Mining and Knowledge Discovery*, 14(1):171–206, 2007.

S. F. Carr, E. Papp, J. C. Wu, and Y. Natsumeda. Characterization of human type I and type II IMP dehydrogenases. *J. Biol. Chem.*, 268(36):27286–27290, 1993.

W. L. Chang, D. C. Lee, S. Leu, Y. M. Huang, M. C. Lu, and P. Ouyang. Molecular characterization of a novel nucleolar protein, pNO40. *Biochem. Biophys. Res. Commun.*, 307(3):569–577, 2003.

K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *British Machine Vision Conference* (*BMVC*), 2014.

X. Chen, Y. Cheng, S. Wang, Z. Gan, J. Liu, and Z. Wang. The elastic lottery ticket hypothesis. In *Proceedings of the Annual Conference on Neural Information Processing Systems* (*NeurIPS*), 2021.

L. Chu, X. Hu, J. Hu, L. Wang, and J. Pei. Exact and consistent interpretation for piecewise linear neural networks: A closed form solution. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining* (*SIGKDD*), pages 1244–1253, 2018.

Y. Chung, T. Kraska, N. Polyzotis, K. H. Tae, and S. E. Whang. Automated data slicing for model validation: A big data - AI integration approach. *IEEE Transactions on Knowledge and Data Engineering*, 32(12):2284–2296, 2020.

W. W. Cohen. Fast effective rule induction. In *Proceedings of the International Conference on Machine Learning* (*ICML*), pages 115–123. 1995.

C. Coupette and J. Vreeken. Graph similarity description: How are these graphs similar? In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining* (*SIGKDD*), pages 185–195, 2021.

C. Coupette, S. Dalleiger, and J. Vreeken. Differentially describing groups of graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence* (*AAAI*), 2022. URL https://arxiv.org/abs/2201.04064.

M. Courbariaux, Y. Bengio, and J.-P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. *Proceedings of the Annual Conference on Neural Information Processing Systems* (*NeurIPS*), pages 3123–3131, 2015.

S. Dalleiger and J. Vreeken. Explainable data decompositions. In *Proceedings of the AAAI Conference on Artificial Intelligence* (*AAAI*), pages 3709–3716, 2020.

T. De Bie. Maximum entropy models and subjective interestingness: an application to tiles in binary databases. *Data Mining and Knowledge Discovery*, 23 (3):407–446, 2011.

J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de Las Casas, C. Donner, L. Fritz, C. Galperti, A. Huber, J. Keeling, M. Tsimpoukelli, J. Kay, A. Merle, J. M. Moret, S. Noury, F. Pesamosca, D. Pfau, O. Sauter, C. Sommariva, S. Coda, B. Duval, A. Fasoli, P. Kohli, K. Kavukcuoglu, D. Hassabis, and M. Riedmiller. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897): 414–419, 2022.

L. Deng, P. Jiao, J. Pei, Z. Wu, and G. Li. Gxnor-net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework. *Neural Networks*, 100:49 – 58, 2018.

J. Denker, D. Schwartz, B. Wittner, S. Solla, R. Howard, L. Jackel, and J. Hopfield. Large automatic learning, rule extraction, and generalization. *Complex Syst.*, 1(5), 1987.

J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT*, pages 4171–4186, 2019.

G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining* (*SIGKDD*), pages 43–52, 1999.

X. Dong, S. Chen, and S. J. Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Proceedings of the Annual Conference on Neural Information Processing Systems* (*NeurIPS*), pages 4857–4867, 2017.

D. Dua and C. Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

W. Duivesteijn and J. Thaele. Understanding Where Your Classifier Does (Not) Work – The SCaPE Model Class for EMM. In *Proceedings of the IEEE International Conference on Data Mining* (*ICDM*), pages 809–814, 2014.

D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Proceedings of the Annual Conference on Neural Information Processing Systems* (*NeurIPS*), pages 2224–2232, 2015.

U. Evci, T. Gale, J. Menick, P. S. Castro, and E. Elsen. Rigging the lottery: Making all tickets winners. In *Proceedings of the International Conference on Machine Learning* (*ICML*), pages 2943–2952, 2020.

J. Fischer and R. Burkholz. Plant 'n' Seek: Can You Find the Winning Ticket? In *International Conference on Learning Representations* (*ICLR*), 2022.

J. Fischer and J. Vreeken. Sets of robust rules, and how to find them. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases* (*ECML PKDD*), pages 38–54. Springer, 2019.

J. Fischer and J. Vreeken. Discovering succinct pattern sets expressing co-occurrence and mutual exclusivity. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining* (*SIGKDD*), pages 813–823, 2020.

J. Fischer and J. Vreeken. Differentiable pattern set mining. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining* (*SIGKDD*), pages 383–392, 2021.

J. Fischer, A. Gadhikar, and R. Burkholz. Towards strong pruning for lottery tickets with non-zero biases. *arXiv preprint arXiv:2110.11150*, 2021a.

J. Fischer, A. Oláh, and J. Vreeken. What's in the box? Exploring the inner life of neural networks with robust rules. In *Proceedings of the International Conference on Machine Learning* (*ICML*), pages 3352–3362, 2021b.

R. A. Fisher. On the interpretation of χ^2 from contingency tables, and the calculation of P. *Journal of the Royal Statistical Society*, 85(1):87–94, 1922.

J. Fowkes and C. Sutton. A subsequence interleaving model for sequential pattern mining. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining* (*SIGKDD*), pages 835–844, 2016.

J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations* (*ICLR*), 2019.

J. Frankle, G. K. Dziugaite, D. Roy, and M. Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *Proceedings of the International Conference on Machine Learning* (*ICML*), pages 3259–3269, 2020.

J. Frankle, G. K. Dziugaite, D. Roy, and M. Carbin. Pruning neural networks at initialization: Why are we missing the mark? In *International Conference on Learning Representations* (*ICLR*), 2021.

N. Frosst and G. Hinton. Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784*, 2017.

Y. Fu, Q. Yu, Y. Zhang, S. Wu, X. Ouyang, D. Cox, and Y. Lin. Drawing robust scratch tickets: Subnetworks with inborn robustness are found within randomly initialized networks. In *Proceedings of the Annual Conference on Neural Information Processing Systems* (*NeurIPS*), 2021.

A. García-Vico, C. Carmona, D. Martín, M. García-Borroto, and M. del Jesus. An overview of emerging pattern mining in supervised descriptive rule discovery: taxonomy, empirical study, trends, and prospects. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(1):e1231, 2018.

M. Gardner, Y. Artzi, V. Basmova, J. Berant, B. Bogin, S. Chen, P. Dasigi, D. Dua, Y. Elazar, A. Gottumukkala, N. Gupta, H. Hajishirzi, G. Ilharco, D. Khashabi, K. Lin, J. Liu, N. F. Liu, P. Mulcaire, Q. Ning, S. Singh, N. A. Smith, S. Subramanian, R. Tsarfaty, E. Wallace, A. Zhang, and B. Zhou. Evaluating models' local decision boundaries via contrast sets. In *Proceedings of the Association for Computational Linguistics* (*EMNLP*), pages 1307–1323, 2020.

M. R. Garey and D. S. Johnson. *Computers and intractability : a guide to the theory of NP-completeness*. Freeman, 22. pr. edition, 2000. ISBN 0716710447.

L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

F. Geerts, B. Goethals, and T. Mielikäinen. Tiling databases. In *Proceedings of Discovery Science*, pages 278–289, 2004.

K. Geurts, G. Wets, T. Brijs, and K. Vanhoof. Profiling high frequency accident locations using association rules. In *Proceedings of the 82nd Annual Transportation Research Board, Washington DC. (USA)*, page 18pp, 2003.

L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal. Explaining explanations: An overview of interpretability of machine learning. In *IEEE International Conference on data science and advanced analytics (DSAA)*, pages 80–89, 2018.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 249–256, 2010.

S. Goebl, A. Tonch, C. Böhm, and C. Plant. Megs: Partitioning meaningful subgraph structures using minimum description length. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, pages 889–894, 2016.

E. Grave, P. Bojanowski, P. Gupta, A. Joulin, and T. Mikolov. Learning word vectors for 157 languages. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC)*, 2018.

P. Grünwald. *The Minimum Description Length Principle*. MIT Press, 2007.

P. Grünwald and T. Roos. Minimum description length revisited. *International Journal of Mathematics for Industry*, 11(01):1930001, 2019.

W. Hämäläinen. Kingfisher: an efficient algorithm for searching for both positive and negative dependency rules with statistical significance measures. *Knowledge and Information Systems*, 32(2):383–414, 2012.

J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 1–12, 2000.

S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2015.

B. Hanin and D. Rolnick. Deep relu networks have surprisingly few activation patterns. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 359–368, 2019.

B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems (NIPS)*, 1992.

S. Hayou, J.-F. Ton, A. Doucet, and Y. W. Teh. Robust pruning at initialization. In *International Conference on Learning Representations* (*ICLR*), 2021.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the International Conference on Computer Vision* (*ICCV*), pages 1026–1034, 2015.

M. A. Hedderich, D. Adelani, D. Zhu, J. Alabi, U. Markus, and D. Klakow. Transfer learning and distant supervision for multilingual transformer models: A study on African languages. In *Proceedings of the Association for Computational Linguistics* (*EMNLP*), pages 2580–2591, 2020.

M. A. Hedderich, J. Fischer, D. Klakow, and J. Vreeken. Label-descriptive patterns and their application to characterizing classification errors. In *Proceedings of the International Conference on Machine Learning* (*ICML*), pages 8691–8707, 2022.

H. Heikinheimo, J. K. Seppänen, E. Hinkkanen, H. Mannila, and T. Mielikäinen. Finding low-entropy sets and trees from binary data. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining* (*SIGKDD*), pages 350–359, 2007.

A. Henelius, K. Puolamäki, H. Boström, L. Asker, and P. Papapetrou. A peek into the black box: Exploring classifiers by randomization. *Data Mining and Knowledge Discovery*, 28(5–6):1503–1529, 2014.

Y. J. Hsieh, T. K. Kundu, Z. Wang, R. Kovelman, and R. G. Roeder. The TFIIIC90 subunit of TFIIIC interacts with multiple components of the RNA polymerase III machinery and contains a histone-specific acetyltransferase activity. *Mol. Cell. Biol.*, 19(11):7697–7704, 1999.

J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), pages 7132–7141, 2018.

I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. *Proceedings of the Annual Conference on Neural Information Processing Systems* (*NeurIPS*), pages 4107–4115, 2016.

E. Inoue and J. Yamauchi. AMP-activated protein kinase regulates PEPCK gene expression by direct phosphorylation of a novel zinc finger transcription factor. *Biochem. Biophys. Res. Commun.*, 351(4):793–799, 2006.

S. Jaroszewicz and D. A. Simovici. Interestingness of frequent itemsets using bayesian networks as background knowledge. In *Proceedings of the ACM*

International Conference on Knowledge Discovery and Data Mining (*SIGKDD*), pages 178–186, 2004.

J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, 2021.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations* (*ICLR*), 2015.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations* (*ICLR*), 2014.

P. Kontkanen and P. Myllymäki. MDL histogram density estimation. In *Proceedings of the International Conference on Artificial Intelligence and Statistics* (*AISTATS*), 2007.

M. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *Aiche Journal*, 37:233–243, 1991.

J. Krungkrai, N. Wutipraditkul, P. Prapunwattana, S. R. Krungkrai, and S. Rochanakij. A nonradioactive high-performance liquid chromatographic microassay for uridine 5′-monophosphate synthase, orotate phosphoribosyltransferase, and orotidine 5′-monophosphate decarboxylase. *Anal. Biochem.*, 299(2):162–168, 2001.

H. Lakkaraju, E. Kamar, R. Caruana, and J. Leskovec. Interpretable & explorable approximations of black box models. *arXiv preprint arXiv:1707.01154*, 2017.

Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems* (*NIPS*), pages 598–605, 1990.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

G. Lee, S. Kim, and S. Hwang. Qadiver: Interactive framework for diagnosing QA models. In *Proceedings of the AAAI Conference on Artificial Intelligence* (*AAAI*), pages 9861–9862, 2019a.

N. Lee, T. Ajanthan, and P. H. S. Torr. Snip: single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations (ICLR)*, 2019b.

N. Lee, T. Ajanthan, S. Gould, and P. H. S. Torr. A signal propagation perspective for pruning neural networks at initialization. In *International Conference on Learning Representations (ICLR)*, 2020.

F. Lemmerich and M. Becker. pysubgroup: Easy-to-use subgroup discovery in python. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, pages 658–662, 2018.

F. Li and B. Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.

H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations (ICLR)*, 2017.

M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer, 1993.

N. Liu, G. Yuan, Z. Che, X. Shen, X. Ma, Q. Jin, J. Ren, J. Tang, S. Liu, and Y. Wang. Lottery ticket preserves weight correlation: Is it desirable or not? In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 7011–7020, 2021a.

S. Liu, T. Chen, X. Chen, Z. Atashgahi, L. Yin, H. Kou, L. Shen, M. Pechenizkiy, Z. Wang, and D. C. Mocanu. Sparse training via boosting pruning plasticity with neuroregeneration. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021b.

F. Llinares-López, M. Sugiyama, L. Papaxanthos, and K. Borgwardt. Fast and memory-efficient significant pattern mining via permutation testing. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 725–734, 2015.

C. Lucchese, S. Orlando, and R. Perego. Mining top-k patterns from binary datasets in presence of noise. In *Proceedings of the SIAM International Conference on Data Mining (SDM)*, pages 165–176, 2010.

G. S. Lueker. Exponentially small bounds on the expected optimum of the partition and subset sum problems. *Random Structures & Algorithms*, 12(1): 51–62, 1998.

X. Ma and E. Hovy. End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics* (*ACL*), pages 1064–1074, 2016.

X. Ma, G. Yuan, X. Shen, T. Chen, X. Chen, X. Chen, N. Liu, M. Qin, S. Liu, Z. Wang, and Y. Wang. Sanity checks for lottery tickets: Does your winning ticket really win the jackpot? In *Proceedings of the Annual Conference on Neural Information Processing Systems* (*NeurIPS*), 2021.

E. Malach, G. Yehudai, S. Shalev-Schwartz, and O. Shamir. Proving the lottery ticket hypothesis: Pruning is all you need. In *Proceedings of the International Conference on Machine Learning* (*ICML*), pages 6682–6691, 2020.

M. Mampaey, J. Vreeken, and N. Tatti. Summarizing data succinctly with the most informative itemsets. *ACM Transactions on Knowledge Discovery from Data*, 6:1–44, 2012.

H. Mannila, H. Toivonen, and A. I. Verkamo. Efficient algorithms for discovering association rules. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining* (*SIGKDD*), pages 181–192, 1994.

M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of english: The penn treebank. *Comp. Ling.*, 19(2):313–330, June 1993.

J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *Artificial Neural Networks and Machine Learning* (*ICANN*), pages 52–59, 2011.

M. Meeng and A. J. Knobbe. Flexible enrichment with Cortana – software demo. In *Proceedings of Benelearn*, pages 117–119, 2011.

H. Mhaskar, Q. Liao, and T. Poggio. When and why are deep networks better than shallow ones? In *Proceedings of the AAAI Conference on Artificial Intelligence* (*AAAI*), pages 2343–2349, 2017.

P. Miettinen. Sparse Boolean matrix factorizations. In *Proceedings of the IEEE International Conference on Data Mining* (*ICDM*), pages 935–940, 2010.

P. Miettinen and J. Vreeken. Model order selection for Boolean matrix factorization. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining* (*SIGKDD*), pages 51–59, 2011.

P. Miettinen and J. Vreeken. MDL4BMF: Minimum description length for Boolean matrix factorization. *ACM Transactions on Knowledge Discovery from Data*, 8 (4):A18:1–31, 2014.

P. Miettinen, T. Mielikäinen, A. Gionis, G. Das, and H. Mannila. The discrete basis problem. *IEEE Transactions on Knowledge and Data Engineering*, 20(10): 1348–1362, 2008.

G. A. Miller. Wordnet: A lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.

T. Mitchell-Jones. Societas europaea mammalogica, 1999. URL http://www.european-mammals.org.

F. Moerchen, M. Thies, and A. Ultsch. Efficient mining of all margin-closed itemsets with applications in temporal knowledge discovery and classification by compression. *Knowledge and Information Systems*, 29(1):55–80, 2011.

P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations (ICLR)*, 2017.

A. Mordvintsev, C. Olah, and M. Tyka. Inceptionism: Going deeper into neural networks. 2015. https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html.

S. Myllykangas, J. Himberg, T. Böhling, B. Nagy, J. Hollmén, and S. Knuutila. DNA copy number amplification profiling of human neoplasms. *Oncogene*, 25(55):7324–7332, 2006.

R. Naffouje, P. Grover, H. Yu, A. Sendilnathan, K. Wolfe, N. Majd, E. P. Smith, K. Takeuchi, T. Senda, S. Kofuji, and A. T. Sasaki. Anti-Tumor Potential of IMP Dehydrogenase Inhibitors: A Century-Long Story. *Cancers (Basel)*, 11 (9), 2019.

A. A. Nanavati, K. P. Chitrapura, S. Joshi, and R. Krishnapuram. Mining generalised disjunctive association rules. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*, pages 482–489, 2001.

M.-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729, 2008.

P. K. Novak, N. Lavrac, and G. I. Webb. Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining. *Journal of Machine Learning Research*, 10:377–403, 2009.

C. Olah, A. Mordvintsev, and L. Schubert. Feature visualization. *Distill*, 2017.

C. Olah, A. Satyanarayan, I. Johnson, S. Carter, L. Schubert, K. Ye, and A. Mordvintsev. The building blocks of interpretability. *Distill*, 2018.

C. Olah, N. Cammarata, L. Schubert, G. Goh, M. Petrov, and S. Carter. Zoom in: An introduction to circuits. *Distill*, 2020.

L. Orseau, M. Hutter, and O. Rivasplata. Logarithmic pruning is all you need. *Proceedings of the Annual Conference on Neural Information Processing Systems* (*NeurIPS*), pages 2925–2934, 2020.

M. E. Otey, C. Wang, S. Parthasarathy, A. Veloso, and W. M. Jr. Mining frequent itemsets in distributed and dynamic databases. In *Proceedings of the IEEE International Conference on Data Mining* (*ICDM*), pages 617–620, 2003.

A. M. Øygard. Peeking inside convnets. 2016. https://www.auduno.com/2016/06/18/peeking-inside-convnets/.

L. Özbakır, A. Baykasoğlu, and S. Kulluk. A soft computing-based approach for integrated training and rule extraction from artificial neural networks: Difaconn-miner. *Applied Soft Computing*, 10(1):304–317, 2010.

L. Papaxanthos, F. Llinares-López, D. A. Bodenham, and K. M. Borgwardt. Finding significant combinations of features in the presence of categorical covariates. In *Proceedings of the Annual Conference on Neural Information Processing Systems* (*NeurIPS*), pages 2271–2279, 2016.

N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *Proceedings of the International Conference on Database Theory* (*ICDT*), pages 398–416. ACM, 1999.

L. Pellegrina and F. Vandin. Efficient mining of the most significant patterns with permutation testing. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining* (*SIGKDD*), pages 2070–2079, 2018.

L. Pellegrina, M. Riondato, and F. Vandin. SPuManTE: Significant pattern mining with unconditional testing. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining* (*SIGKDD*), pages 1528–1538, 2019.

J. Pennington, S. S. Schoenholz, and S. Ganguli. Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice. In *Proceedings of the Annual Conference on Neural Information Processing Systems* (*NeurIPS*), pages 4785–4795, 2017.

J. Pennington, S. S. Schoenholz, and S. Ganguli. The emergence of spectral universality in deep networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics* (*AISTATS*), pages 1924–1932, 2018.

A. Pensia, S. Rajput, A. Nagle, H. Vishwakarma, and D. Papailiopoulos. Optimal lottery tickets via subset sum: Logarithmic over-parameterization is sufficient. In *Proceedings of the Annual Conference on Neural Information Processing Systems* (*NeurIPS*), pages 2599–2610, 2020.

H. M. Proença and M. van Leeuwen. Interpretable multiclass classification by MDL-based rule lists. *Inf. Sci.*, 512:1372–1393, 2020.

V. Ramanujan, M. Wortsman, A. Kembhavi, A. Farhadi, and M. Rastegari. What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), pages 11890–11899, 2020.

G. Ras, M. van Gerven, and P. Haselager. Explanation methods in deep learning: Users, values, concerns and challenges. In *Explainable and Interpretable Models in Computer Vision and Machine Learning*, pages 19–36. Springer, 2018.

M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Proceedings of the European Conference on Computer Vision* (*ECCV*), pages 525–542, 2016.

A. Renda, J. Frankle, and M. Carbin. Comparing rewinding and fine-tuning in neural network pruning. In *International Conference on Learning Representations* (*ICLR*), 2020.

M. T. Ribeiro, S. Singh, and C. Guestrin. "Why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining* (*SIGKDD*), pages 1135–1144, 2016.

M. T. Ribeiro, S. Singh, and C. Guestrin. Anchors: High-precision model-agnostic explanations. In *Proceedings of the AAAI Conference on Artificial Intelligence* (*AAAI*), pages 1527–1535, 2018.

M. T. Ribeiro, T. Wu, C. Guestrin, and S. Singh. Beyond accuracy: Behavioral testing of NLP models with CheckList. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics* (*ACL*), pages 4902–4912, 2020.

J. Rissanen. Modeling by shortest data description. *Automatica*, 14(1):465–471, 1978.

J. Rissanen. A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, 11(2):416–431, 1983.

M. Robnik-Šikonja and I. Kononenko. Explaining classifications for individual instances. *IEEE Transactions on Knowledge and Data Engineering*, 20(5):589–600, 2008.

M.-A. Rondeau and T. J. Hazen. Systematic error analysis of the Stanford question answering dataset. In *Workshop on Machine Reading for Question Answering*, pages 12–20, 2018.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing*, chapter 8. 1986.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3): 211–252, 2015.

D. Saad and E. Marom. Training feed forward nets with binary weights via a modified chir algorithm. *Complex Systems*, 4(5), 1990.

Y. Sakamoto, H. Inoue, P. Keshavarz, K. Miyawaki, Y. Yamaguchi, M. Moritani, K. Kunika, N. Nakamura, T. Yoshikawa, N. Yasui, H. Shiota, T. Tanahashi, and M. Itakura. SNPs in the KCNJ11-ABCC8 gene locus are associated with type 2 diabetes and blood pressure levels in the Japanese population. *Journal of Human Genetics*, 52(10):781–793, 2007.

P. Savarese, H. Silva, and M. Maire. Winning the lottery with continuous sparsification. In *Proceedings of the Annual Conference on Neural Information Processing Systems* (*NeurIPS*), pages 11380–11390, 2020.

A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *International Conference on Learning Representations* (*ICLR*), 2014.

F. Scarselli and A. C. Tsoi. Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results. *Neural Networks*, 11(1):15–37, 1998.

F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1): 61–80, 2009.

J. Schmidt-Hieber. Nonparametric regression using deep neural networks with ReLU activation function. *The Annals of Statistics*, 48(4):1875–1897, 2020.

S. S. Schoenholz, J. Gilmer, S. Ganguli, and J. Sohl-Dickstein. Deep information propagation. In *International Conference on Learning Representations* (*ICLR*), 2017.

D. C. Schultz, K. Ayyanathan, D. Negorev, G. G. Maul, and F. J. Rauscher. SETDB1: a novel KAP-1-associated histone H3, lysine 9-specific methyltransferase that contributes to HP1-mediated silencing of euchromatic genes by KRAB zinc-finger proteins. *Genes Dev.*, 16(8):919–932, 2002.

T. J. Sejnowski, P. K. Kienker, and G. E. Hinton. Learning symmetry groups with hidden units: beyond the perceptron. *Physica D: Nonlinear Phenomena*, 2:260–275, 1986.

I. Sharma, R. K. Dutta, N. K. Singh, and Y. S. Kanwar. High Glucose-Induced Hypomethylation Promotes Binding of Sp-1 to Myo-Inositol Oxygenase: Implication in the Pathobiology of Diabetic Tubulopathy. *Am. J. Pathol.*, 187 (4):724–739, 2017.

Y. Shima, S. Mitsuishi, K. Hirata, and M. Harao. Extracting minimal and closed monotone DNF formulas. In *Proceedings of Discovery Science*, pages 298–305, 2004.

A. Shrikumar, P. Greenside, and A. Kundaje. Learning important features through propagating activation differences. In *Proceedings of the International Conference on Machine Learning* (*ICML*), pages 3145–3153, 2017.

T. Simons and D.-J. Lee. A review of binarized neural networks. *Electronics*, 8 (6), 2019.

K. Smets and J. Vreeken. Slim: Directly mining descriptive patterns. In *Proceedings of the SIAM International Conference on Data Mining* (*SDM*), pages 236–247, 2012.

S. Srinivas and R. V. Babu. Generalized dropout. *arXiv preprint arXiv:1611.06791*, 2016.

J. Su, Y. Chen, T. Cai, T. Wu, R. Gao, L. Wang, and J. D. Lee. Sanity-checking pruning methods: Random tickets can win the jackpot. In *Proceedings of the Annual Conference on Neural Information Processing Systems* (*NeurIPS*), 2020.

M. Sundararajan, A. Taly, and Q. Yan. Axiomatic attribution for deep networks. In *Proceedings of the International Conference on Machine Learning* (*ICML*), pages 3319–3328, 2017.

C. Sutton, M. Boley, L. Ghiringhelli, M. Rupp, J. Vreeken, and M. Scheffler. Identifying domains of applicability of machine learning models for materials science. *Nature Communications*, 11:1–9, 2020.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations* (*ICLR*), 2014.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), pages 1–9, 2015.

I. A. Taha and J. Ghosh. Symbolic interpretation of artificial neural networks. *IEEE Transactions on knowledge and data engineering*, 11(3):448–463, 1999.

H. Tan and M. Bansal. LXMERT: Learning cross-modality encoder representations from transformers. In *Proceedings of the Association for Computational Linguistics* (*EMNLP*), pages 5100–5111, 2019.

S. Tan, R. Caruana, G. Hooker, P. Koch, and A. Gordo. Learning global additive explanations for neural nets using model distillation. *arXiv preprint arXiv:1801.08640*, 2018.

H. Tanaka, D. Kunin, D. L. Yamins, and S. Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. In *Proceedings of the Annual Conference on Neural Information Processing Systems* (*NeurIPS*), 2020.

N. Tatti. Maximum entropy based significance of itemsets. *Knowledge and Information Systems*, 17(1):57–77, 2008.

N. Tatti and J. Vreeken. Finding good itemsets by packing data. In *Proceedings of the IEEE International Conference on Data Mining* (*ICDM*), pages 588–597, 2008.

The 1000 Genomes Project Consortium. A global reference for human genetic variation. *Nature*, 526(7571):68–74, 2015.

S. N. Tran and A. S. d'Avila Garcez. Deep logic networks: Inserting and extracting knowledge from deep belief networks. *IEEE Transactions on Neural Networks and Learning Systems*, 29(2):246–258, 2018.

K. Tunyasuvunakool, J. Adler, Z. Wu, T. Green, M. Zielinski, A. Žídek, A. Bridgland, A. Cowie, C. Meyer, A. Laydon, S. Velankar, G. Kleywegt, A. Bateman, R. Evans, A. Pritzel, M. Figurnov, O. Ronneberger, R. Bates, S. Kohl, and

D. Hassabis. Highly accurate protein structure prediction for the human proteome. *Nature*, 596:1–9, 08 2021.

H. van Eenennaam, D. Lugtenberg, J. H. Vogelzangs, W. J. van Venrooij, and G. J. Pruijn. hPop5, a protein subunit of the human RNase MRP and RNase P endoribonucleases. *J. Biol. Chem.*, 276(34):31635–31641, 2001.

M. van Leeuwen and A. J. Knobbe. Diverse subgroup set discovery. *Data Mining and Knowledge Discovery*, 25(2):208–242, 2012.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 5998–6008, 2017.

R. Velik. Discrete fourier transform computation using neural networks. In *Proceedings of the International Conference on Computational Intelligence and Security*, pages 120–123, 2008.

P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018.

A. Veloso, W. M. Jr., M. de Carvalho, B. Pôssas, S. Parthasarathy, and M. J. Zaki. In *Proceedings of the SIAM International Conference on Data Mining (SDM)*, pages 494–510, 2002.

S. Verdenius, M. Stol, and P. Forré. Pruning via iterative ranking of sensitivity statistics. *arXiv preprint arXiv:2006.00896*, 2020.

R. Vimieiro. *Mining disjunctive patterns in biomedical data sets*. PhD thesis, The University of Newcastle, Australia, 2012.

J. Vreeken and N. Tatti. *Interesting Patterns*, pages 105–134. Springer, 2014.

J. Vreeken, M. van Leeuwen, and A. Siebes. Krimp: Mining itemsets that compress. *Data Mining and Knowledge Discovery*, 23(1):169–214, 2011.

C. Wang and S. Parthasarathy. Summarizing itemset patterns using probabilistic models. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 730–735, 2006.

C. Wang, G. Zhang, and R. B. Grosse. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations (ICLR)*, 2020.

F. Wang and C. Rudin. Falling rule lists. In *Proceedings of the International Conference on Artificial Intelligence and Statistics* (*AISTATS*), 2015.

G. I. Webb. OPUS: an efficient admissible algorithm for unordered search. *J. Artif. Intell. Res.*, 3:431–465, 1995.

G. I. Webb. Discovering significant patterns. *Machine Learning*, 68(1):1–33, 2007.

G. I. Webb. Self-sufficient itemsets: An approach to screening potentially interesting associations between items. *ACM Transactions on Knowledge Discovery from Data*, 4(1):1–20, 2010.

G. I. Webb. Filtered-top-k association discovery. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(3):183–192, 2011.

G. I. Webb and S. Zhang. K-optimal rule discovery. *Data Min. Knowl. Discov.*, 10 (1):39–79, 2005.

A. Weigend, D. Rumelhart, and B. Huberman. Generalization by weight-elimination with application to forecasting. In *Advances in Neural Information Processing Systems* (*NIPS*), 1991.

B. Wiegand, D. Klakow, and J. Vreeken. Mining easily understandable models from complex event logs. In *Proceedings of the SIAM International Conference on Data Mining* (*SDM*), pages 244–252, 2021.

S. Wrobel. An algorithm for multi-relational discovery of subgroups. In *Proceedings of the European Conference on Principles and Practice of Knowledge Discovery in Databases* (*ECML PKDD*), pages 78–87, 1997.

T. Wu, M. T. Ribeiro, J. Heer, and D. Weld. Errudite: Scalable, reproducible, and testable error analysis. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics* (*ACL*), pages 747–763, 2019.

Y. Xiang, R. Jin, D. Fuhry, and F. F. Dragan. Succinct summarization of transactional databases: an overlapped hyperrectangle scheme. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining* (*SIGKDD*), pages 758–766, 2008.

N. Xie, G. Ras, M. van Gerven, and D. Doran. Explainable deep learning: A field guide for the uninitiated. *arXiv preprint arXiv:2004.14545*, 2020.

K. Xu, J. Li, M. Zhang, S. S. Du, K. ichi Kawarabayashi, and S. Jegelka. What can neural networks reason about? In *International Conference on Learning Representations* (*ICLR*), 2020.

G. Yang, J. Pennington, V. Rao, J. Sohl-Dickstein, and S. S. Schoenholz. A mean field theory of batch normalization. In *International Conference on Learning Representations* (*ICLR*), 2019.

D. Yarotsky. Optimal approximation of continuous functions by very deep relu networks. In *Proceedings of the Conference On Learning Theory* (*COLT*), pages 639–649, 2018.

H. You, C. Li, P. Xu, Y. Fu, Y. Wang, X. Chen, R. G. Baraniuk, Z. Wang, and Y. Lin. Drawing early-bird tickets: Toward more efficient training of deep networks. In *International Conference on Learning Representations* (*ICLR*), 2020.

H. Yu, S. Edunov, Y. Tian, and A. S. Morcos. Playing the lottery with rewards and multiple languages: lottery tickets in RL and NLP. In *International Conference on Learning Representations* (*ICLR*), 2020.

M. Zaki, N. Ramakrishnan, and L. Zhao. Mining frequent boolean expressions: Application to gene expression and regulatory modeling. *Int. J. Knowl. Discov. Bioinform.*, 1(3):68–96, 2010.

M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining* (*SIGKDD*), 1997.

Q. Zhang, R. Cao, F. Shi, Y. N. Wu, and S. Zhu. Interpreting CNN knowledge via an explanatory graph. In *Proceedings of the AAAI Conference on Artificial Intelligence* (*AAAI*), pages 4454–4463, 2018.

Z. Zhang, J. Jin, Z. Zhang, Y. Zhou, X. Zhao, J. Ren, J. Liu, L. Wu, R. Jin, and D. Dou. Validating the lottery ticket hypothesis with inertial manifold theory. In *Proceedings of the Annual Conference on Neural Information Processing Systems* (*NeurIPS*), 2021.

B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), pages 2921–2929, 2016.

H. Zhou, J. Lan, R. Liu, and J. Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Proceedings of the Annual Conference on Neural Information Processing Systems* (*NeurIPS*), pages 3597–3607, 2019.

Y. Zhu, O. Groth, M. Bernstein, and L. Fei-Fei. Visual7W: Grounded Question Answering in Images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), pages 4995–5004, 2016.

A. Zimmermann and S. Nijssen. Supervised pattern mining and applications to classification. In *Frequent Pattern Mining*, pages 425–442. 2014.