Software Impacts 13 (2022) 100317

Contents lists available at ScienceDirect

Software Impacts

journal homepage: www.journals.elsevier.com/software-impacts

Original software publication

Joint learn: A python package for task-specific weight sharing for sequence classification

Shahrukh Khan *, Mahnoor Shahid

Saarland University, Germany

ARTICLE INFO

Keywords:
Deep learning
Text classification
Weight sharing
Transformers

ABSTRACT

Transfer Learning has enabled cutting-edge improvements in Deep Learning to attain state-of-the-art outcomes, notably in the domain of Natural Language Processing. Despite this, neural networks trained on the low-resource text classification corpora still face challenges because of the lack of pre-trained model checkpoints. In this paper, we introduce Joint Learn which is a PyTorch based comprehensive toolkit for weight sharing for text classification that leverages task-specific weight sharing to train a joint neural network for several sequence classification tasks and aids in the development of more generalized models while potentially eliminating the transfer learning issues that low-resource corpora encounter.

Code metadata

Current code version	v.1.0
Permanent link to code/repository used for this code version	https://github.com/SoftwareImpacts/SIMPAC-2022-36
Permanent link to Reproducible Capsule	https://codeocean.com/capsule/6910469/tree/v1
Legal Code License	MIT License
Code versioning system used	git
Software code languages, tools, and services used	Python
Compilation requirements, operating environments & dependencies	Pytorch
If available Link to developer documentation/manual	
Support email for questions	shkh00001@stud.uni-saarland.de

1. Introduction

There have been certain significant breakthroughs in the field of the Natural Language Processing paradigm with the advent of attention mechanism and its use in transformer sequence–sequence models coupled with different transfer-learning techniques have quickly become state-of-the-art in multiple pervasive Natural Language Processing tasks [1,2]. However, one potential problem arises from the lack of pre-trained model checkpoints for low-resource text classification corpora. Since training state-of-the-art neural networks require a significant amount of data when training from scratch [3]. In this paper, we propose *Joint Learn* which is an alternative solution for training neural networks from scratch on low-resource languages. It enables researchers to train a joint neural network instead of training separate models for multiple text classification tasks in multiple languages. This has multiple side benefits as weight-sharing allows to mitigate the complexity of neural networks [4]. It also enables better generalization and facilitates the robustness of the trained model. Joint learn is developed to aid researchers dealing with low-resource text classification multilingual corpora while bootstrapping their experiments with a minimal amount of code thereby it will facilitate the state-of-the-art Natural Language Processing research and development of novel and robust text classification applications across a range of different languages.

2. Functionalities and key features

Joint-Learn currently supports four different types of deep neural network topologies out of the box. Two of the model architectures, as shown in Fig. 1, utilize layers of vanilla LSTM [5] for weight sharing.

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

* Corresponding author.
E-mail addresses: shkh00001@stud.uni-saarland.de (S. Khan), mash00001@stud.uni-saarland.de (M. Shahid).

https://doi.org/10.1016/j.simpa.2022.100317
Received 22 March 2022; Received in revised form 21 April 2022; Accepted 11 May 2022

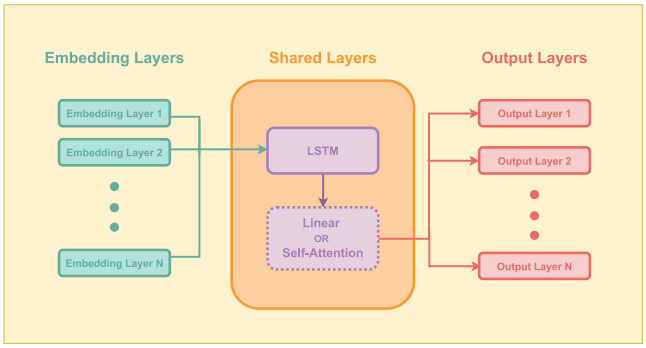

Fig. 1. Joint learning architecture with LSTM and linear/self attention.

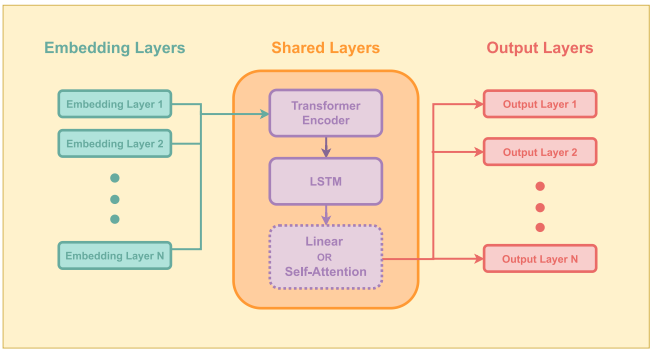

Fig. 2. Joint learning architecture with LSTM, transformer encoder and linear/self attention.

However, for the last block, we can either have a linear layer that generates the output or it can be replaced by Self-Attention block [6] that introduces sequence level attention mechanism to handle long-term dependencies in texts.

Furthermore, the other two variants include the Transformers Encoder [2] to attend to inputs using multi-head attention as depicted in Fig. 2. Likewise, the last block following LSTM, can either be Linear or Self Attention. These models can be trained directly after feeding in the datasets belonging to any language. Joint Learn also allows to load custom and pre-trained embeddings such as GloVe [7], FastText [8] and Word2Vec [9] independent of source language.

3. Impact overview

The proposed framework can be used by researchers and developers to train models on novel multi-lingual datasets and potentially build strong baselines and has already been used in *Hindi/Bengali Sentiment Analysis Using Transfer Learning and Joint Dual Input Learning with Self Attention* [10]. Joint Learn also provides features to load and save pre-trained checkpoints hence facilitating the opportunities to collaborate while using it. Moreover, it provides an interface to train language-agnostic models hence not limiting the users to specific languages. Furthermore, Joint Learn has a highly configurable implementation enabling the users to extend it to new neural architectures and other natural language processing tasks i.e. multi-label classification. Joint Learn is task agnostic for text classification tasks, which means the joint network can be trained for diverse nature of tasks simultaneously. For instance, the joint network can be used to train for sentiment, hate-speech, and intent classification using the same model hence resulting in models which are computationally efficient not only during training but also at inference. Finally, it also provides generic text pre-processing and PyTorch-based data-loader pipelines which facilitate instantiating an arbitrary number of data-loaders and pre-processors simultaneously which unifies and integrates well with the training of the joint network.

```python
## init jl lstm self-attention
jl_lstm = JLLSTMAttentionClassifier(
    batch_size=batch_size,
    hidden_size=hidden_size,
    lstm_layers=lstm_layers,
    embedding_size=embedding_size,
    dataset_hyperparams=dataset_hyperparams,
    bidirectional=bidirectional,
    fc_hidden_size=fc_hidden_size,
    self_attention_config=self_attention_config,
    device=device,
)

## define optimizer and loss function
optimizer = torch.optim.Adam(params=jl_lstm.parameters())

train_model(
    model=jl_lstm,
    optimizer=optimizer,
    dataloaders=jl_dataloaders,
    max_epochs=max_epochs,
    config_dict={
        "device": device,
        "model_name": "jl_lstm_attention",
        "self_attention_config": self_attention_config,
    },
)
```

Fig. 3. Example of training a joint self attention LSTM with PyTorch.

```python
## init jl transformer lstm
jl_lstm = JLLSTMTransformerClassifier(
    batch_size=batch_size,
    hidden_size=hidden_size,
    lstm_layers=lstm_layers,
    embedding_size=embedding_size,
    nhead=nhead,
    transformer_hidden_size=transformer_hidden_size,
    transformer_layers=transformer_layers,
    dataset_hyperparams=dataset_hyperparams,
    device=device,
    max_seq_length=max_seq_length,
)

## define optimizer and loss function
optimizer = torch.optim.Adam(params=jl_lstm.parameters())

train_model(
    model=jl_lstm,
    optimizer=optimizer,
    dataloaders=jl_dataloaders,
    max_epochs=max_epochs,
    config_dict={"device": device, "model_name": "jl_lstm"},
)
```

Fig. 4. Example of training a joint transformer LSTM with PyTorch.

4. Usage

Joint Learn provides an intuitive interface for jointly training models from scratch in PyTorch. An example of training a joint self attention LSTM with PyTorch is shown in Fig. 3 whereas Fig. 4 shows an example of jointly training a transformer-based LSTM in the joint learn package.

5. Conclusion and future work

Joint Learn provides a flexible and intuitive interface for researchers and users to train models with minimal boilerplate code. We intend to

extend the capabilities of Joint Learn to include other state-of-the-art neural architectures such as BERT [11]. Moreover, we also aim to add a visualization feature for visualizing self attention mechanism and multi-head attention from transformers. Lastly, we are also planning to make the Joint Learn package accessible through PyPI in order to ensure ease of use and installation.

CRediT authorship contribution statement

Shahrukh Khan: Conceptualization, Methodology, Software, Data curation, Writing – original draft, Software, Validation. **Mahnoor Shahid:** Visualization, Investigation, Reviewing and editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Sebastian Ruder, Matthew E. Peters, Swabha Swayamdipta, Thomas Wolf, Transfer learning in natural language processing, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials, Association for Computational Linguistics, Minneapolis, Minnesota, 2019, pp. 15–18.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, Attention is all you need, 2017.

[3] Aysu Ezen-Can, A comparison of LSTM and BERT for small Corpus, 2018.

[4] Dejiao Zhang, Haozhu Wang, Mário A.T. Figueiredo, Laura Balzano, Learning to share: simultaneous parameter tying and sparsification in deep learning, in: ICLR, 2018.

[5] Sepp Hochreiter, Jürgen Schmidhuber, Long short-term memory, Neural Comput. 9 (1997) 1735–1780.

[6] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, Yoshua Bengio, A structured self-attentive sentence embedding, 2017.

[7] Jeffrey Pennington, Richard Socher, Christopher Manning, GloVe: Global vectors for word representation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP, Association for Computational Linguistics, Doha, Qatar, 2014, pp. 1532–1543.

[8] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, Tomas Mikolov, FastText.zip: Compressing text classification models, 2016, arXiv preprint arXiv:1612.03651.

[9] Tomas Mikolov, Kai Chen, G.s Corrado, Jeffrey Dean, Efficient estimation of word representations in vector space, in: Proceedings of Workshop At ICLR, Vol. 2013, 2013.

[10] Shahrukh Khan, Mahnoor Shahid, Hindi/bengali sentiment analysis using transfer learning and joint dual input learning with self attention, BOHR Int. J. Res. Nat. Lang. Comput. 2022 (2022).

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: NAACL, 2019.