UER: An Open-Source Toolkit for Pre-training Models

Zhe Zhao[1,2,♠] **Hui Chen**[2,♣] **Jinbin Zhang**[2,♣] **Xin Zhao**[1,♠] **Tao Liu**[1,♠]
Wei Lu[1,♠] **Xi Chen**[3,♦] **Haotang Deng**[2,♣] **Qi Ju**[2,♣,*] **Xiaoyong Du**[1,♠]
[1] School of Information and DEKE, MOE, Renmin University of China, Beijing, China
[2] Tencent AI Lab
[3] School of Electronics Engineering and Computer Science, Peking University, Beijing, China
♠{helloworld, zhaoxinruc, tliu, lu-wei, duyong}@ruc.edu.cn
♣{chenhuichen, westonzhang, haotangdeng, damonju}@tencent.com
♦{mrcx}@pku.edu.cn

Abstract

Existing works, including ELMO and BERT, have revealed the importance of pre-training for NLP tasks. While there does not exist a single pre-training model that works best in all cases, it is of necessity to develop a framework that is able to deploy various pre-training models efficiently. For this purpose, we propose an assemble-on-demand pre-training toolkit, namely Universal Encoder Representations (UER). UER is loosely coupled, and encapsulated with rich modules. By assembling modules on demand, users can either reproduce a state-of-the-art pre-training model or develop a pre-training model that remains unexplored. With UER, we have built a model zoo, which contains pre-trained models based on different corpora, encoders, and targets (objectives). With proper pre-trained models, we could achieve new state-of-the-art results on a range of downstream datasets.

1 Introduction

Pre-training has been well recognized as an essential step for NLP tasks since it results in remarkable improvements on a range of downstream datasets (Devlin et al., 2018). Instead of training models on a specific task from scratch, pre-training models are firstly trained on general-domain corpora, then followed by fine-tuning on downstream tasks. Thus far, a large number of works have been proposed for finding better pre-training models. Existing pre-training models mainly differ in the following three aspects:

1) Model encoder.

Commonly-used encoders include RNN (Hochreiter and Schmidhuber, 1997), CNN (Kim, 2014), AttentionNN (Bahdanau et al., 2014), and their combinations (Zhou et al., 2016). Recently, Transformer (a structure based on attentionNN) is shown to be a more powerful feature extractor compared with other encoders (Vaswani et al., 2017).

2) Pre-training target (objective).

Using proper target is one of the keys to the success of pre-training. While the language model is most commonly used (Radford et al., 2018), many works focus on seeking better targets such as masked language model (cloze test) (Devlin et al., 2018) and machine translation (McCann et al., 2017).

3) Fine-tuning strategy.

Using a proper fine-tuning strategy is also important to the performance of pre-training models on downstream tasks. A commonly-used strategy is to regard pre-trained models as feature extractors (Kiros et al., 2015).

Table 1 lists 8 popular pre-training models and their main differences (Kiros et al., 2015; Logeswaran and Lee, 2018; McCann et al., 2017; Conneau et al., 2017; Peters et al., 2018; Howard and Ruder, 2018; Radford et al., 2018; Devlin et al., 2018). In additional to encoder, target, and fine-tuning strategy, corpus is also listed in Table 1 as an important factor for pre-training models.

There are many open-source implementations of pre-training models, such as Google BERT[1], ELMO from AllenAI[2], GPT and BERT from HuggingFace[3]. However, these works usually focus on the designs of either one or a few pre-training models. Due to the diversity of the downstream tasks and the computational resources constraint, there does not exist a single pre-training model that works best in all cases. BERT is one of the most widely used pre-training models. It exploits

* Corresponding author.

[1] https://github.com/google-research/bert
[2] https://github.com/allenai/bilm-tf
[3] https://github.com/huggingface

Proceedings of the 2019 EMNLP and the 9th IJCNLP (System Demonstrations), pages 241–246
Hong Kong, China, November 3 – 7, 2019.

Model	Corpus	Encoder	Target
Skip-thoughts	Bookcorpus	GRU	Conditioned LM
Quick-thoughts	Bookcorpus+UMBCcorpus	GRU	Sentence prediction
CoVe	English-German	Bi-LSTM	Machine translation
Infersent	Natural language inference	LSTM;GRU;CNN;LSTM+Attention	Classification
ELMO	1billion benchmark	Bi-LSTM	Language model
ULMFiT	Wikipedia	LSTM	Language model
GPT	Bookcorpus; 1billion benchmark	Transformer	Language model
BERT	Wikipedia+bookcorpus	Transformer	Cloze+sentence prediction

Table 1: 8 pre-training models and their differences. For space constraint of the table, fine-tuning strategies of different models are described as follows: Skip-thoughts, quick-thoughts, and infersent regard pre-trained models as feature extractors. The parameters before output layer are frozen. CoVe and ELMO transfer word embedding to downstream tasks, with other parameters in neural networks uninitialized. ULMFit, GPT, and BERT fine-tune entire networks on downstream tasks.

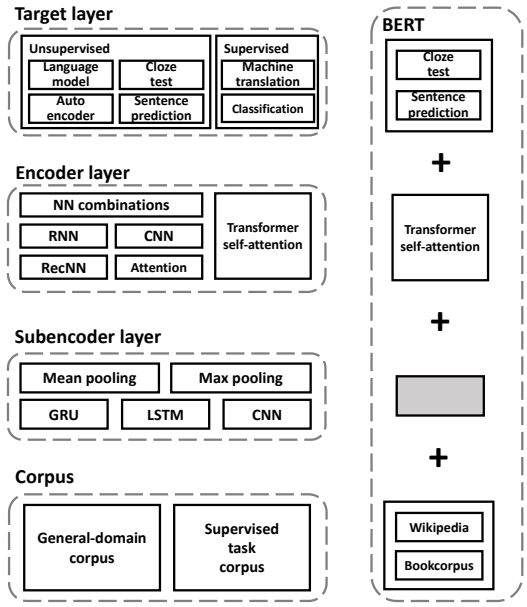

Figure 1: The architecture of UER (pre-training part). We can combine modules in UER to implement BERT model.

two unsupervised targets for pre-training. But in some scenarios, supervised information is critical to the performance of downstream tasks (Conneau et al., 2017; McCann et al., 2017). Besides, in many cases, BERT is excluded due to its efficiency issue. Based on above reasons, it is often the case that one should adopt different pre-training models in different application scenarios.

In this work, we introduce UER, a general framework that is able to facilitate the developments of various pre-training models. UER maintains model modularity and supports research extensibility. It consists of 4 components: subencoder, encoder, target, and downstream task fine-tuning. The architecture of UER (pre-training part) is shown in Figure 1. Ample modules are implemented in each component. Users could assemble different modules to implement existing models such as BERT (right part in Figure 1), or develop a new pre-training model by implementing customized modules. Clear and robust interfaces allow users to assemble (or add) modules with as few restrictions as possible.

With the help of UER, we build a Chinese pre-trained model zoo based on different corpora, encoders, and targets. Different datasets have their own characteristics. Selecting proper models from the model zoo can largely boost the performance of downstream datasets. In this work, we use Google BERT as baseline model. We provide some use cases that are based on UER, and the results show that our models can either achieve new state-of-the-art performance, or achieve competitive results with an efficient running speed.

UER is built on PyTorch and supports distributed training mode. Clear instructions and documentations are provided to help users read and use UER codes. The UER toolkit and the model zoo are publicly available at `https://github.com/dbiir/UER-py`.

2 Related Work

2.1 Pre-training for deep neural networks

Using word embedding to initialize neural network's first layer is one of the most commonly used strategies for NLP tasks (Mikolov et al., 2013; Kim, 2014). Inspired by the success of word embedding, some recent works try to initialize entire networks (not just first layer) with pre-trained parameters (Howard and Ruder, 2018; Radford et al., 2018). They train a deep neural network upon large corpus, and fine-tune the pre-trained model on specific downstream tasks. One of the most influential works among them is BERT (Devlin et al., 2018). BERT extracts text features with 12/24 Transformer layers, and exploits

masked language model task and sentence prediction task as training targets (objectives). The drawback of BERT is that it requires expensive computational resources. Thankfully, Google makes its pre-trained models publicly available. So we can directly fine-tune on Google's models to achieve competitive results on many NLP tasks.

2.2 NLP toolkits

Many NLP models have tens of hyper-parameters and various tricks, and some of which exert large impacts on final performance. Sometimes it is unlikely to report all details and their effects in research paper. This may lead to a huge gap between research papers and code implementations. To solve the above problem, some works are proposed to implement a class of models in a framework. This type of work includes OpenNMT (Klein et al., 2017), fairseq (Ott et al., 2019) for neural machine translation; glyph (Zhang and LeCun, 2017) for classification; NCRF++ (Yang and Zhang, 2018) for sequence labeling; Hyperwords (Levy et al., 2015), ngram2vec (Zhao et al., 2017) for word embedding, to name a few.

Recently, we witness many influential pre-training works such as GPT, ULMFiT, and BERT. We think it could be useful to develop a framework to facilitate reproducing and refining those models. UER provides the flexibility of building pre-training models of different properties.

3 Architecture

In this section, we firstly introduce the core components in UER and the modules that we have implemented in each component. Figure 1 illustrates UER's framework and detailed modules (pre-training part). Modularity design of UER largely facilitates the use of pre-training models. At the end of this section, we will give some case studies to illustrate how to use UER effectively.

3.1 Subencoder

This layer learns word vectors from subword features. For English, we use character as subword features. For Chinese[4], we use radical and pinyin as subword features. As a result, the model can be aware of internal structures of words. Subword information has been explored in many NLP tasks such as text classification (Zhang and LeCun, 2017) and word embedding (Joulin et al., 2016). In the pre-training literature, ELMO exploits subencoder layer. In UER, we implement RNN and CNN as subencoders, and use mean pooling or max pooling upon hidden states to obtain fixed-length word vectors.

[4]We don't do word segmentation on Chinese corpus. We regard each Chinese character as a word. Internal structures such as radical and pinyin are regarded as Chinese subword features.

3.2 Encoder

This layer learns features from word vectors. UER implements a series of basic encoders, including LSTM, GRU, CNN, GatedCNN, and AttentionNN. Users can use these basic encoders directly, or use their combinations. The output of an encoder can be fed into another encoder, forming networks of arbitrary layers. UER provides ample examples of combining basic encoders (e.g. CNN + LSTM). Users can also build their custom combinations with basic encoders in UER.

Currently, Transformer (a structure based on multi-headed self-attention) becomes a popular text feature extractor and is proven to be effective for many NLP tasks. We implement Transformer module and integrate it into UER. With Transformer module, we can implement models such as GPT and BERT easily.

3.3 Target (objective)

Using suitable target is the key to the success of pre-training. Many papers in this field propose their targets and show their advantages over other ones. UER consists of a range of targets. Users can choose one of them, or use multiple targets and give them different weights. In this section we introduce targets implemented in UER.

- **Language model (LM).** Language model is one of the most commonly used targets. It trains model to make it useful to predict current word given previous words.
- **Masked LM (MLM, also known as cloze test).** The model is trained to be useful to predict masked word given surrounding words. MLM utilizes both left and right contexts to predict words. LM only considers the left context.
- **Autoencoder (AE).** The model is trained to be useful to reconstruct input sequence as close as possible.

The above targets are related with word prediction. We call them word-level targets. Some works

show that introducing sentence-level task into targets can benefit pre-training models (Logeswaran and Lee, 2018; Devlin et al., 2018).

- **Next sentence prediction (NSP).** The model is trained to predict if the two sentences are continuous. Sentence prediction target is much more efficient than word-level targets. It doesn't involve sequentially decoding of words and softmax layer over entire vocabulary.

Above targets are unsupervised tasks (also known as self-supervised tasks). However, supervised tasks can provide additional knowledge that raw corpus can not provide.

- **Neural machine translation (NMT).** CoVe (McCann et al., 2017) proposes to use NMT to pre-train model. The implementation of NMT target is similar with autoencoder. Both of them involve encoding source sentences and sequentially decoding words of target sentences.

- **Classification (CLS).** Infersent (Conneau et al., 2017) proposes to use natural language inference task (three-way classification) to pre-train model.

Most pre-training models use above targets individually. It is worth trying to use multiple targets at the same time. Some targets are complementary to each other, e.g. word-level target and sentence-level target (Devlin et al., 2018), unsupervised target and supervised target. In experiments section, we demonstrate that proper selection of target is important. UER provides the flexibility to users in trying different targets and their combinations.

3.4 Fine-tuning

UER exploits similar fine-tuning strategy with ULMFiT, GPT, and BERT. Models on downstream tasks share structures and parameters with pre-training models except that they have different target layers. The entire models are fine-tuned on downstream tasks. This strategy performs robustly in practice. We also find that feature extractor strategy produces inferior results on models such as GPT and BERT.

Most pre-training works involve 2 stages, pre-training and fine-tuning. But UER supports 3 stages: 1) pre-training on general-domain corpus; 2) pre-training on downstream dataset; 3) fine-tuning on downstream dataset. Stage 2 enables models to get familiar with the distributions of downstream datasets (Howard and Ruder, 2018; Radford et al., 2018). It is also called semi-supervised fine-tuning strategy in the work of Dai and Le (2015) since stage 2 is unsupervised and stage 3 is supervised.

3.5 Case Studies

In this section, we show how UER facilitates the use of pre-training models. First of all, we demonstrate that UER can build most pre-training models easily. As shown in the following code, only a few lines are required to construct models with the interfaces in UER.

```
# Implementation of BERT.
embedding = BertEmbedding(args, vocab_size)
encoder = BertEncoder(args)
target = BertTarget(args, vocab_size)

# Implementation of GPT.
embedding = BertEmbedding(args, vocab_size)
encoder = GptEncoder(args)
target = LmTarget(args, vocab_size)

# Implementation of Quick-thoughts.
embedding = Embedding(args, vocab_size)
encoder = GruEncoder(args)
target = NspTarget(args, None)

# Implementation of InferSent.
embedding = Embedding(args, vocab_size)
encoder = LstmEncoder(args)
target = ClsTarget(args, None)
```

In practice, users can assemble different subencoder, encoder, and target modules without any code work. Users can specify modules through options *–subencoder*, *–encoder*, and *–target*. More details are available in quickstart and instructions of UER's github project. UER provides ample modules. Users can try different module combinations according to their downstream datasets. Besides trying modules implemented by UER, users can also develop their customized modules and integrate them into UER seamlessly.

4 Experiments

To evaluate the performance of UER, experiments are conducted on a range of datasets, each of which falls into one of four categories: sentence classification, sentence pair classification, sequence labeling, and document-based QA. BERT-base uncased English model and BERT-base Chinese model are used as baseline models. In section 4.1, UER is tested on several evaluation benchmarks to demonstrate that it can produce models as intended. In section 4.2, we ap-

Implementation	SST-2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	WNLI
HuggingFace	93.0	83.8	89.4	90.7	84.0/84.4	89.0	61.0	53.5
UER	92.4	83.0	89.3	91.0	84.0/84.0	91.5	66.8	56.3

Table 2: The performance of HuggingFace's implementation and UER's implementation on GLUE benchmark.

Implementation	XNLI	LCQMC	MSRA-NER	ChnSentiCorp	nlpcc-dbqa
ERNIE	77.2	87.0	92.6	94.3	94.6
UER	77.5	86.6	93.6	94.3	94.6

Table 3: The performance of ERNIE's implementation and UER's implementation on ERNIE benchmark.

ply pre-trained models in our model zoo to different downstream datasets. Significant improvements are witnessed when proper encoders and targets are selected. For space constraint, we put some contents in UER's github project, including dataset and corpus details, system speed, and part of qualitative/quantitative evaluation results.

4.1 Reproducibility

This section uses English/Chinese benchmarks to test BERT implementation of UER. For English, we use sentence and sentence pair classification datasets in GLUE benchmark (dev set) (Wang et al., 2019). For Chinese, we use five datasets of different types: sentiment analysis, sequence labeling, question pair matching, natural language inference, and document-based QA (provided by ERNIE[5]). Table 2 and 3 compare UER's performance to other publicly available systems. We can observe that UER could match the performance of HuggingFace's and ERNIE's implementations. Results of HuggingFace and ERNIE are reported on their github projects. Results of UER can be reproduced by scripts in UER's github project.

4.2 Influence of targets and encoders

In this section, we give some examples of selecting pre-trained models given downstream datasets. Three Chinese sentiment analysis datasets are used for evaluation. They are Douban book review, Online shopping review, and Chnsenticorp datasets.

First of all, we use UER to pre-train on large-scale Amazon review corpus with different targets. The parameters are initialized by BERT-base Chinese model. The target of original BERT consists of MLM and NSP. However, NSP is not suitable for sentence-level reviews (we have to split reviews into multiple parts). Therefore we remove NSP target. In addition, Amazon reviews are attached with users' ratings. To this end, we can exploit CLS target for pre-training (similar with InferSent). We fine-tune these pre-trained models (with different targets) on downstream datasets. The results are shown in Table 4. BERT baseline (BERT-base Chinese) is pre-trained upon Chinese Wikipedia. We can observe that pre-training on Amazon review corpus can improve the results significantly. Using CLS target achieves the best results in most cases.

Dataset	Douban.	Shopping.	Chn.
BERT baseline	87.5	96.3	94.3
MLM	88.1	**97.0**	95.0
CLS	**88.3**	**97.0**	**95.8**

Table 4: Performance of pre-training models with different targets.

BERT requires heavy computational resources. To achieve better efficiency, we use UER to substitute 12-layers Transformer encoder with a 2-layers LSTM encoder (embedding size and hidden size are 512 and 1024). We still use the above sentiment analysis datasets for evaluation. The model is firstly trained on mixed large corpus with LM target, and then trained on large-scale Amazon review corpus with LM and CLS targets. Table 5 lists the results of different encoders. Compared with BERT baseline, LSTM encoder can achieve comparable or even better results when proper corpora and targets are selected.

Dataset	Douban.	Shopping.	Chn.
BERT baseline	**87.5**	96.3	94.3
LSTM	80.3	94.0	88.3
LSTM+pre-training	86.5	**96.9**	**94.5**

Table 5: Performance of pre-training models with different encoders.

For space constraint, this section only uses sentiment analysis datasets as examples to analyze the influence of different targets and encoders. More tasks and pre-trained models are discussed

[5]https://github.com/PaddlePaddle/ERNIE

in UER's github project.

5 Conclusion

This paper describes UER, an open-source toolkit for pre-training on general-domain corpora and fine-tuning on downstream tasks. We demonstrate that UER can largely facilitate implementations of different pre-training models. With the help of UER, we pre-train models based on different corpora, encoders, targets and make these models publicly available. By using proper pre-trained models, we can achieve significant improvements over BERT, or achieve competitive results with an efficient training speed.

Acknowledgments

This work is supported by National Natural Science Foundation of China Grant No.U1711262 and No.61472428, 2018 Tencent Rhino-Bird Elite Training Program.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. *EMNLP*.

Andrew M Dai and Quoc V Le. 2015. Semi-supervised sequence learning. In *NIPS*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8).

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. *EMNLP*.

Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-thought vectors. In *NIPS*.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M Rush. 2017. Opennmt: Open-source toolkit for neural machine translation. *ACL*.

Omer Levy, Yoav Goldberg, and Ido Dagan. 2015. Improving distributional similarity with lessons learned from word embeddings. *TACL.*, 3.

Lajanugen Logeswaran and Honglak Lee. 2018. An efficient framework for learning sentence representations. *arXiv preprint arXiv:1803.02893*.

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. 2017. Learned in translation: Contextualized word vectors. In *NIPS*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. *NAACL*.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR*.

Jie Yang and Yue Zhang. 2018. Ncrf++: An open-source neural sequence labeling toolkit. *arXiv preprint arXiv:1806.05626*.

Xiang Zhang and Yann LeCun. 2017. Which encoding is the best for text classification in chinese, english, japanese and korean?

Zhe Zhao, Tao Liu, Shen Li, Bofang Li, and Xiaoyong Du. 2017. Ngram2vec: Learning improved word representations from ngram co-occurrence statistics. *EMNLP*.

Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, and Bo Xu. 2016. Attention-based bidirectional long short-term memory networks for relation classification. In *ACL.*, volume 2.