Regularizing Data for Improving Execution Time of NLP Model

Thang Dang, Yasufumi Sakai, Tsuguchika Tabaru and Akihiko Kasagi
Fujitsu Limited
Fujitsu Laboratories
Kawasaki, Japan

Abstract

Natural language processing (NLP) is a very important part of machine learning that can be applied to different real applications. Several NLP models with huge training datasets are proposed. The primary purpose of these large-scale NLP models is the downstream tasks. However, because of the diversity and rapidly increasing the size of these datasets, they consume a lot of resources and time. In this study, we propose a state-of-the-art method to reduce the training time of NLP models on downstream tasks while maintaining accuracy. Our method focuses on removing unimportant data from the input data set and optimizing the padding of tokens to reduce the processing time for the NLP model. Experiments are conducted on many different GLUE benchmark datasets demonstrated that our method can reduce the most up to 57% in training time compared to other methods.

Introduction

The rapid development in the field of natural language processing (NLP) in recent years has realized many exceptional achievements (Devlin et al. 2019), supported by many powerful algorithms such as transformer (Vaswani et al. 2017). Many difficult tasks even for humans have been handled by NLP models with very competent results. Proportional to those developments, NLP models have become increasingly larger and more complex (Narayanan et al. 2021). A huge amount of resources are required to operate these NLP models. Besides, the increase in size and diversity of data sets cause many challenges for researchers with limited resources.

To be able to save budget and time consuming of training process, one of the appropriate and effective directions is to take advantage of these large pre-trained NLP models to fine-tune on different data sets with specific downstream tasks. However, this poses certain challenges when the time and size of the models are relatively large, especially if it is deployed on resource-constrained environments such as mobile devices, IoT, or real applications (Abdaoui, Pradel, and Sigel 2020). Several research efforts are focusing on optimizing the time and resources for training NLP models to reduce the size of pre-train models. Similarly another important research direction is the optimization of data processing. In this study, we focus on optimizing the preprocessing data for the pre-trained models with downstream tasks in the GLUE benchmark (Wang et al. 2018) to minimize fine-tuning time but still keep the accuracy of the system to a comparable rate.

Our contributions.

- We investigated comprehensive and efficient training strategies and provide the intuition of those methods.
- We proposed a novel method to reduce the number of tokens in the data process, decreasing the execution time of the NLP model while maintaining the accuracy of the system in reasonable values.
- We demonstrated the efficient approach to select the maximum length of the sentences when fine-tuning three different tasks in the GLUE benchmark.

The remainder of our paper is structured as follows: we provide the fundamental definition related our algorithm in the background section, the proposed system is described in the next section, and then our implementation and evaluation results are explained, then we give the literature review of related works and present a conclusion.

RELATED WORKS

Research on improving the effectiveness of the NLP model is attracting considerable attention. It can be divided in many different research directions in improving training efficiency for NLP such as model compression, efficient transformer, efficient training with distributed training, tokenization optimization, or data augmentation. In model compression, many studies have been proposed to find the best way to reduce the model size thereby speeding up the training process. Commonly used techniques are model compression (Li et al. 2020) found the bigger size of model sometimes converge faster than smaller transformer model.

In the research direction by efficient transformer, multiple proposed methods for reducing the complexity of the transformer algorithm have been proposed. (Kitaev, Kaiser, and Levskaya 2020) reduced the complexity of the transformer algorithm from $O\left(L^2\right)$ to $O\left(L \log L\right)$ by exchanging dot-product attention with locality-sensitive hashing. (Zhang

and He 2020) changed the architecture of the transformer block, thereby reducing the training time of the transformer-based model. In the research direction of tokenization optimization, there are a number of studies on word embedding optimization. (Kim, Kim, and Lee 2020) proposed a new algorithm to compress word embeddings while maintaining the performance of the model using the Gumbel-softmax strategy. (Provilkov, Emelianenko, and Voita 2020) introduced a new method named BPE-dropout for regularizing subwords to improve model performance. Our research is close to those of (Provilkov, Emelianenko, and Voita 2020) and (Kim, Kim, and Lee 2020). However, rather than proposing a new method to reduce the vocabulary size like (Kim, Kim, and Lee 2020), which requires to pre-train the model before applying it to downstream tasks, or (Provilkov, Emelianenko, and Voita 2020), which works well only with small dataset size in the source side and reduces performance for a large dataset if applied to both source and target languages, our model focuses on reducing the number of tokens and directly applying then to downstream tasks. Our method applies to both small (CoLA) and large datasets (MNLI).

Background

Tokenization and Padding

In general, an important process before training a NLP model is tokenization and padding. There are three common padding algorithms: Fixed padding, Dynamic padding and Uniform padding.

Fixed Padding

Given the input sentence $X = \{x_1, x_2, ..., x_n\}$, X is tokenized and inserted with two special tokens [CLS] and [SEP] to mark the start of the sequence and separate it from another sequence respectively. Next, these tokens map to an index matrix $D = d_1, d_2, ..., d_n$ with unique token IDs allocated for every x_i. Then, the attention masks are generated by inserting [PAD] tokens (in practice, we use the value 0) at the end of each d_i vectors.

Algorithm 1 Fixed Padding

Input: $X = \{x_1, x_2, ..., x_n\}, lb, bz, m$
Output: t_ID, att_mask, lbs

```
procedure FIXPADD
    for (i : 0; i < len(X); i+ = bz) do
        d_i = tokenizer(X[i:i+bz], padding=m)
        t_ID.append(d_i[input_ids])
        att_mask.append(d_i[attention_mask])
        lbs.append(tensor[lb[i:i+bz]])
```

In Algorithm 1, we provide input data X, ground truth label lb, batch-size bz, and max sentence length m as inputs of the function. Output of the function are token ID t_ID to index tokens from input X, attention mask att_mask, and list of label lbs to map t_ID with the input ground truth label lb.

Dynamic Padding

Unlike Fixed Padding creates a set of all d_i with the equal size as max-length, Dynamic Padding calculates the number of [PAD] that need to be added in the same batch. This method makes all d_i in each batch have the equal lengths but are different in batches. This method greatly reduces the number of [PAD] tokens that need to be inserted.

Uniform Padding

One improvement from the Dynamic Padding is the Uniform Padding. The method first sort the d_i in ascending order of size. Next, clusters of d_i are created by using the number of batches. Finally, the [PAD] tokens are inserted in the d_i to ensure that all d_i in the same batch are equal in size.

PROPOSED SYSTEM

We propose a new algorithm to reduce the number of input tokens required to execute the transformer-based NLP model in downstream tasks. As a consequence, it is feasible to reduce training time while maintaining the accuracy of the model. By analyzing previous methods used for padding tokens before training the model, we found redundancy of pad tokens. Moreover, the input data set contains several redundant data that does not help the model learn more effectively. By suggesting a solution to regularizing data before padding tokens, our method is effective in reducing the number of input tokens, thus reducing model execution time.

One basic but very important step before tokenization is removing trivial data for the training model. We found that in the three main tasks in the GLUE benchmark, using data in the forms such as: tag, html tag, xml, or emoticons. Several data in the GLUE benchmark are raw and not preprocessed. However, many studies (Kim, Kim, and Lee 2020) only focus on the optimization of tokenization and ignore this important step. From this data analysis, we proposed a method called regularizing data to completely remove unnecessary data before tokenization and padding to reduce the training tokens. Experiments on the GLUE benchmark have shown that our method significantly reduced the number of tokens, thereby reducing the training time while maintaining accuracy. Our method is described in Algorithm 2. Here we regularized the data before tokenization and padding. First we filter and remove all unnecessary data with special symbols. Next we continued to find emoticons in the data and remove them. Finally we obtained the cleaned data. The next step was to tokenize and use the Uniform Padding to handle samples before training the model.

EXPERIMENTS AND EVALUATIONS

Baseline model

In this study we use the pre-trained BERT-base (Devlin et al. 2019) as the base-line model for testing and evaluating our method. The pre-trained BERT-base has 110 million parameters. It consists of 12 encoder layers stacked with 12 attention heads. The Feedforward network uses 768 hidden units.

Algorithm 2 Our proposed algorithm

Input: $X = \{x_1, x_2, ..., x_n\}, lb, bz, m$
Output: t_ID, att_mask, lbs

```
procedure OurFunction
    for (x in X) do
        x = remove(find(< [∧ >]* >))
        emo = find((?::—;—=)(?:-)?(?:))
        x = remove(emo)
    for ( i:0 ; i< len(X); i+= 1 ) do
        D = tokenizer.batch_encode(X[i])
    G = group(D, bz)
    G = sort(G, ascending)
    for (i : 0; i < len(G); i+ = bz) do
        for (l in G) do
            n = m - max(l)
            t_ID = l + [tokenizer.pad_token_id]*n
            att_mask = [1]*len(l) + [0]*n
        lbs.append(tensor[lb[i]])
```

Experimental Settings

To evaluate and compare the effectiveness of our proposed algorithm with other methods we use a NVIDIA Tesla V100-SXM2 32GB computer. We use the datasets in the GLUE benchmark (Wang et al. 2018) for evaluation. GLUE consists of three main tasks: single-sentence task, similarity and paraphrase tasks, and inference tasks. In this study, we used the CoLA dataset (Warstadt, Singh, and Bowman 2019), MRPC (Dolan and Brockett 2005) and QQP (Iyer, Nikhil, and Kornel 2017) datasets, and MNLI dataset (Williams, Nangia, and Bowman 2018) in GLUE benchmark datasets. We fine-tuned the tasks using a pre-trained BERT-base model with the same parameters for all implementations. Our settings include batch-size: 16, learning rate of:5×10^{-5}. Next, the experiment was conducted with the epoch values ranging from 1 to 5 and the best obtained results have been reported herein. We report training time in the (hh:mm:ss) format for all implementation results. Another important parameter is the maximum length, many research works usually used the default settings in the set [128, 256, 512], and it is not clear why it was selected. We analyzed the histogram of the training datasets and found that the most maximum sentence length in 128. Thus, this setting is used for all our implementations.

Single Sentence Task

The CoLa includes 8,551 training samples and 527 testing samples. Table 1 shows that our method reduces the tokens around 92% to the Fix Padding method, 41% to the Dynamic and 11% compared to the Uniform Padding method. Our method is the fastest in training up to 57% than the Fixed Padding, 1% than the Dynamic and the Uniform Padding. Our method get the highest accuracy of 0.852 and MCC score of 0.639.

Similarity and Paraphrase Tasks

The QQP dataset comprises 255,024 samples marked as 0 and 149,263 samples marked as 1 in the "is_duplicate labels" column. Because we do not have the "is_duplicate labels" in our test set, we split the train dataset in proportion with 80% for training and 20% for testing. Table 1 shows that our method reduces the number of train tokens by 8,775,088 and Test Tokens by 2,196,33. Our method reduces around 79% to the Fix Padding method, 54% to the Dynamic Padding, and 11% compared to Uniform Padding method. Our method achieved the fastest in train time (0:39:19) while maintaining the accuracy at 87.8% which is lowered only by 0.6% to the highest score with the Dynamic method.

The MRPC is a paraphrase identity dataset includes 2,474 label "1" and 1,194 label "0" in the training set. From Table 1, our method reduced the number of train tokens by 167,800 and the test tokens by 78,939. Furthermore, our method reduced around 64%, 40%, and 14% using the Fixed Padding, Dynamic Padding, and Uniform Padding methods, respectively. Our training time speeds faster than the Fixed Padding 44%, the Dynamic Padding 21%, and the Uniform Padding 1%. Especially, our method got the highest scores at both accuracy 83.7% and F1 0.833.

Inference Tasks

The MNLI corpus has 433,000 samples. In particular, MNLI has two different test sets matched dataset and mismatched dataset. Thus we report m-accuracy for matched dataset and mm-accuracy for mismatched dataset. Table 2 shows our method reduces the number of train tokens by 13,996,944 and Test Tokens by 350,784. It reduces around 72% to the Fixed Padding method, 55% to the Dynamic Padding method, and 10% compared to Uniform Padding method. Our training time is the shortest at 2:05:12.

CONCLUSION

In this study, we proposed a simple and effective method for reducing NLP training time on the GLUE benchmark while maintaining the accuracy and metrics unchanged on most tasks. We first conducted data analysis on CoLA, QQP, MRPC, and MNLI datasets. We realized the presence of multiple raw data that have not been processed and are unuseful for model training. By exploring similar approaches, we reported that related studies have largely overlooked this basic but important step in improving and fine-tuning pre-trained models. By applying our method on three tasks, including single sentence task to check grammar, similarity and paraphrase task to check similarity between two sentences, and inference task to verify the hypothesis in GLUE benchmark, our method achieved very optimistic results in reducing the number of training tokens with the largest reduction in the CoLA dataset of 92%, 40%, and 11% compared to the fixed padding, dynamic padding, and uniform padding methods, respectively. Furthermore, our model achieved the biggest reduction in training time in CoLA dataset with a faster up 57% compared with the fixed padding method, while maintaining accuracy. Moreover, the implementation results in three of four datasets show that

Table 1: Implementation results on CoLA/QQP/MRPC dataset

Method	Train Tokens	Test Tokens	Train Time	Train Loss	Accuracy	MCC/F1
CoLA dataset						
Fixed Padding	1,094,528	67,456	0:06:11	0.07	0.837	0.599
Dynamic Padding	147,051	95,15	0:02:41	0.07	0.846	0.624
Uniform Padding	97,145	6,113	0:02:41	0.07	0.843	0.614
Our method	**86,907**	**5,459**	**0:02:39**	0.07	**0.852**	**0.639**
QQP dataset						
Fixed Padding	41,398,912	10,349,824	1:27:18	0.29	0.878	0.879
Dynamic Padding	19,165,121	4,821,342	0:52:12	0.28	**0.884**	**0.885**
Uniform Padding	9,830,944	2,460,480	0:39:55	0.29	0.877	0.878
Our method	**8,775,088**	**2,196,336**	**0:39:19**	**0.26**	0.878	0.879
MRPC dataset						
Fixed Padding	469,504	220,800	0:02:32	0.05	0.826	0.822
Dynamic Padding	279,676	130,829	0:01:47	0.04	0.828	0.822
Uniform Padding	195,916	92,024	0:01:26	0.05	0.834	0.83
Our method	**167,800**	**78,939**	**0:01:25**	**0.04**	**0.837**	**0.833**

Table 2: Implementation results on MNLI dataset

Method	Train Tokens	m-Test Tokens	mm-Test Tokens	Train Time	Train Loss	m-acc	mm-acc
Fixed Padding	50,265,856	1,280,000	1,280,000	4:24:57	**0.14**	0.803	0.810
Dynamic Padding	31,132,200	787,440	806,816	3:08:03	**0.14**	**0.812**	0.812
Uniform Padding	15,630,304	391,904	408,592	2:10:35	0.15	0.810	**0.814**
Our method	**13,996,944**	**350,784**	**361,984**	**2:05:12**	0.15	0.805	0.810

our method improves the accuracy of the model more than other methods. An exception is the MNLI dataset where our accuracy is only less than the dynamic padding method by 0.007 for the matched dataset and 0.004 for the mismatched dataset for the uniform padding method.

Acknowledgments. This work is supported by Fujitsu Japan Limited.

References

Abdaoui, A.; Pradel, C.; and Sigel, G. 2020. Load what you need: Smaller versions of mutililingual BERT. In *Proceedings of SustaiNLP*, 119–123.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the NAACL-HLT 2019*, 4171–4186.

Dolan, W. B., and Brockett, C. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the IWP 2005*.

Iyer, S.; Nikhil, D.; and Kornel, C. 2017. First quora dataset release: Question pairs.

Kim, Y.; Kim, K.-M.; and Lee, S. 2020. Adaptive compression of word embeddings. In *Proceedings of the ACL 2020*, 3950–3959.

Kitaev, N.; Kaiser, L.; and Levskaya, A. 2020. Reformer: The efficient transformer. In *ICLR*.

Li, Z.; Wallace, E.; Shen, S.; Lin, K.; Keutzer, K.; Klein, D.; and Gonzalez, J. 2020. Train big, then compress: Rethinking model size for efficient training and inference of transformers. In *ICML*, 5958–5968. PMLR.

Narayanan, D.; Shoeybi, M.; Casper, J.; et al. 2021. Efficient large-scale language model training on gpu clusters. *arXiv preprint arXiv:2104.04473*.

Provilkov, I.; Emelianenko, D.; and Voita, E. 2020. Bpe-dropout: Simple and effective subword regularization. In *Proceedings of the ACL 2020*, 1882–1892.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *NeurIPS*, 5998–6008.

Wang, A.; Singh, A.; Michael, J.; Hill, F.; Levy, O.; and Bowman, S. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop*, 353–355.

Warstadt, A.; Singh, A.; and Bowman, S. R. 2019. Neural network acceptability judgments. *Transactions of the ACL 2019* 7:625–641.

Williams, A.; Nangia, N.; and Bowman, S. R. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *NAACL-HLT*, 1112–1122.

Zhang, M., and He, Y. 2020. Accelerating training of transformer-based language models with progressive layer dropping. *CoRR* abs/2010.13369.