Natural Language Processing Project Report

Fuming Zhang, 118033910025

1 Introduction

In the past few years, the concept of artificial intelligence (AI) has received extensive attention under the influence of Alpha Go [1] [2]. Natural language processing (NLP) and computer vision (CV), as the two pillars of AI, have also developed rapidly in recent years.

Taking NLP as an example, word vectors have been the core representation technique of natural language processing for a long time. Word embedding for preprocessing a large amount of unmarked data by algorithms such as word2vec [3] and GloVe [4] is used to initialize the first layer of the neural network, and other layers are then trained on the data of specific tasks. However, word2vec and other related methods sacrifice expression for efficiency.

The recent rise of the pre-trained language representations partially relieved this problem. This method is mainly divided into two categories, feature-based and fine-tuning. Embeddings from language models (ELMo) [5], as one of feature-based approach, uses tasks-specific architecture, which takes pre-trained representations as extra features. While in generative pre-training (GPT) [6], as a representative of the fine-tuning approach, only a small number of additional parameters are introduced, and the pre-trained parameters can be fine-tuned simultaneously during the training process.

In this project, we are supposed to solve the question natural language inference (QNLI) task in the General Language Understanding Evaluation (GLUE) [7] benchmark. The QNLI task in GLUE is derived from the Stanford Question Answering Dataset (SQuAD) [8]. SQuAD is converted into a sentence pair classification task by forming a pair between a question and each sentence in the corresponding context. Thus the QNLI task is to determine whether the context sentence contains the answer to the question. SQuAD 2.0 [9] was released in 2018, and accordingly, the QNLI data set currently used by GLUE benchmark was also upgraded.

At the time when this project was released, the methods that ranked higher than GLUE baselines on the leaderboard all used the multi-layer bidirectional Transformer encoder (BERT) [10] method. So we decided to follow BERT at the very beginning. However, as of today (2019.6.20), the Multi-Task Deep Neural Networks for Natural Language Understanding (MT-DNN) [11] [12] and XLNet [13] methods have surpassed the BERT method, respectively, ranking first and second on the leaderboard.

The remainder of this report is organized as follows. In Section 2, we introduce the background knowledge of this project, mainly including Transformer, GPT, and BERT. Then, the implementation is described in detail in Section 3. The experimental results are described in Section 4. Finally we conclude the report in Section 5.

2 Background

In this section, we will briefly introduce the BERT model used in this project and the background knowledge of Transformer and GPT, which are closely related to it.

2.1 Transformer

In the field of natural language processing, recurrent neural networks such as long short-term memory [14] and gated recurrent neural networks [15] have dominated the state of art approaches for a long time. Among these methods, the encoder-decoder architecture and attention mechanism are also essential. However, Vaswani et al. [16] proposed a simple network architecture, the Transformer, which solely based on attention mechanisms and greatly improved the best results at the time.

The architecture of the Transformer is shown in Figure 1. It mainly follows the encoder-decoder architecture and uses stacked self-attention and point-wise, fully connected layers. The encoder is composed of a stack of identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. As for the decoder, it is also composed of a stack of identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack.

As we have mentioned in the previous paragraph, the Transformer uses a multi-head self-attention mechanism. In such a mechanism, it concat and project the outputs which yield by performing scaled dot-product attention function (in parallel) on the projected queries, keys, and values. As for the scaled dot-product attention function, it computes the dot products of the query with all keys and applies a softmax function to the weights on the values. The above two methods are shown in Figure 2.

- *Project link: https://github.com/AlexChang/NLU-QNLI*
- *E-mail: zhangfuming-alex@sjtu.edu.cn*

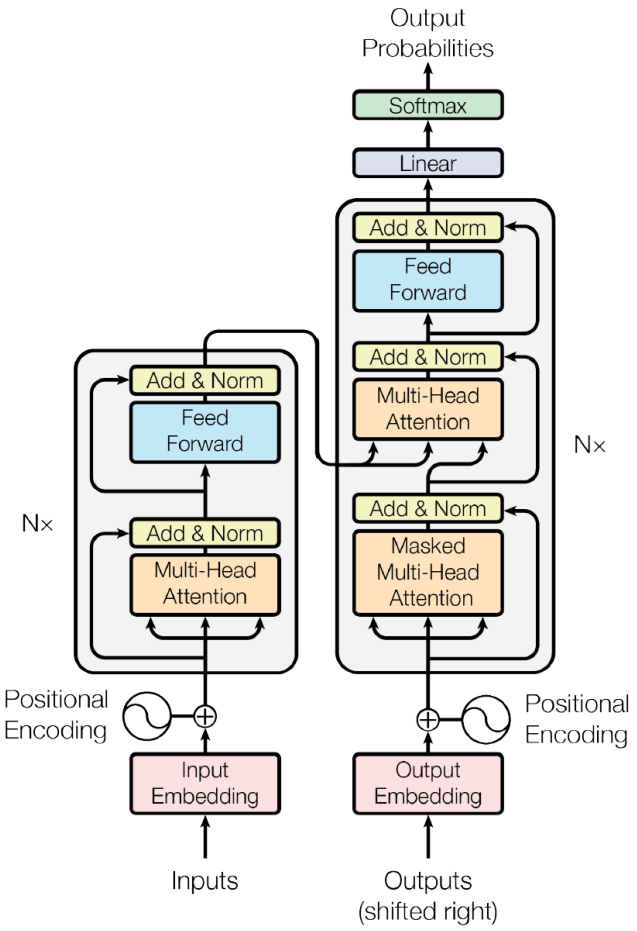

Fig. 1. Transformer Architecture

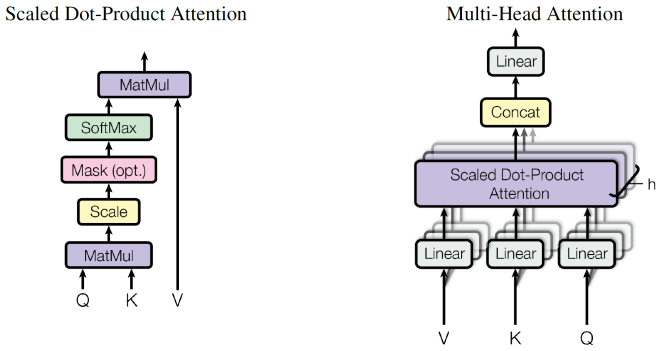

Fig. 2. (left) Scaled Dot-Product Attention; (right) Multi-Head Attention

2.2 GPT

Considering the wide variety of tasks in natural language processing, Radford et al. [6] proposed the approach of generative pre-training of a language model, to solve the challenge for discriminatively trained models to perform adequately due to the lack of labeled corpus.

In such a framework, it mainly consists of two stages. The first stage is the unsupervised pre-training. It learns a high-capacity language model, for which the authors use a multi-layer *Transormer decoder* [17], on a large corpus of text. Then in the following supervised fine-tuning stage, the final transformer block's activation is fed into an added linear output layer to predict the result.

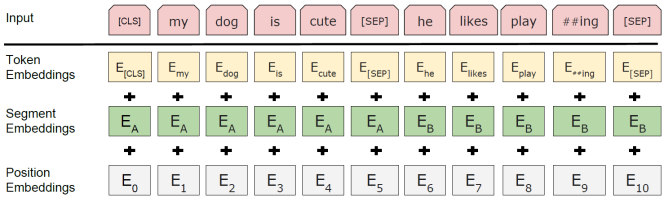

Fig. 3. BERT Input Representation

For those text classification tasks (such as QNLI in this project), we can directly fine-tune the model as described above. However, for some other tasks like question answering or textual entailment, some modifications like input transformation are required. Considering that this part of the content is not related to this project, we will not expand it in detail.

2.3 BERT

Although GPT has made significant improvements in many NLP tasks, Devlin et al. [10] argue that it severely restricts the power of the pre-trained representations since its language model is unidirectional, which means every token can only attend to previous tokens in the self-attention layers of the Transformer. Thus the authors proposed BERT: Bidirectional Encoder Representations from Transformers.

The architecture of BERT is a multi-layer bidirectional Transformer encoder based on the original implementation described in Vaswani et al. [16]. The input representation used in BERT is constructed by summing the corresponding token, segment, and position embeddings, as shown in Figure 3. Besides, to addresses the previously mentioned unidirectional constraints, BERT is pre-trained using two novel unsupervised prediction tasks. The first one is the masked language model, and the second one is the next sentence prediction.

As the authors have described in [10], BERT is designed to be very similar to GPT. The main difference between them is that BERT trains on larger corpora (BooksCorpus & Wikipedia), learns sentence separators and embeddings during pre-training and trains for 1M steps with a batch size of 128, 000 words.

3 Implementation

Considering the limited GPU resources we have, it is almost impossible to retrain the bert model. So we completed this project based on the existing open-source implementation and its pre-trained model. To be more specific, we use the PyTorch implementation provided by Hugging Face (https://github.com/huggingface/pytorch-pretrained-BERT).

Since we are solving QNLI, we used the provided example *run_classifier.py* to solve the GLUE's MRPC tasks. In this file, it supports the training and evaluation of tasks cola, mnli, mnli-mm, mrpc, sst-2, sts-b, qqp, qnli, rte and wnli, but does not support test set prediction. Thus, we extended the data processor for the QNLI task and added the code for the test set prediction in the main function. In addition,

for the sake of simplicity, we removed the code that is not related to the QNLI task and encapsulated the evaluation and testing into two separate functions.

Later, as we trained our model, we found that there is a problem in the original code to calculate the total number of training steps, which in turn will affect the learning rate adjustment. We corrected this error with reference to the following issue (https://github.com/huggingface/pytorch-pretrained-BERT/issues/556) on GitHub. However, this modification resulted in a slight decrease in the prediction accuracy of the fine-tuned model, whose submission name is QNLI_7 and later.

To visualize the training process, we introduced TensorBoard. At present, the stable version of PyTorch (1.1) provides the corresponding interface, but we need to install *TensorBoard* with version 1.14 or higher and *future* to run normally. At the time when we conducted the experiment, version 1.14 of TensorBoard was not released (the 1.14 version was released on 2019.6.15), so we used tb-nightly instead.

As we will see in the next Section 4, the highest prediction accuracy we can achieve is 92.1% without any changes to the original training set. In order to improve the prediction accuracy, We have made the following two improvements. On the one hand, we integrate the training set and the development set together for the model training. The relevant implementation code is in the file *merge_data.py*. On the other hand, we carry out a simple model ensemble. We combine the predictions of several models, conduct unweighted voting, and select the majority of the votes to be the final prediction. The relevant implementation code is in the file *merge_result.py*.

4 Experimental Results

We ran all of our experiments on a server with Ubuntu 16.04 (64bit), 2 E5-2630 v4 processors, 128GB RAM and 4 TITAN XP graphics cards (12GB memory per card). The programs we use and their versions are as follows, Python 3.5.2, CUDA 9.0.176, NVIDIA driver 410.79, and PyTorch 1.1.0. Our experimental results are shown in Table 1.

As shown in Table 1, our best performing single model achieved an accuracy of 92.2% (as shown in Figure 5). When training this model, we integrated the training set and the test set together. In order to further improve the prediction accuracy, we selected several models with relatively good performance and integrated their predictions. In submission QNLI_ENSEMBLE_1, we selected QNLI_2, QNLI_4, QNLI_11, and QNLI_12 in QNLI_ENSEMBLE_2, we selected QNLI_2, QNLI_4, QNLI_9, QNLI_11, QNLI_12, and QNLI_13. We combine the predictions of these models, conduct unweighted voting, and select the majority of the votes to be the final prediction. The accuracy of the prediction results after this procedure reached 92.8% (as shown in Figure 4).

TABLE 1
Prediction Results

Submission Name	Parameter	Accuracy Rate
QNLI_1	base_b32_e3	90.8%
QNLI_2	large_b32_e3	**92.1**%
QNLI_4	large_**train+dev**_b32_e3	**92.2%**
QNLI_7	large_b24_e3	91.6%
QNLI_8	large_b24_e4_wp0.075	91.3%
QNLI_9	large_b32_e4_wp0.075	91.9%
QNLI_10	large_b40_e3	91.0%
QNLI_11	large_b32_e3	91.9%
QNLI_12	large_b40_e4_wp0.075	91.8%
QNLI_13	large_**torch0.4.1**_b32_e3	91.6%
QNLI_ENSEMBLE_1	QNLI_2+4+11+12	**92.7%**
QNLI_ENSEMBLE_2	QNLI_2+4+9+11+12+13	**92.8%**

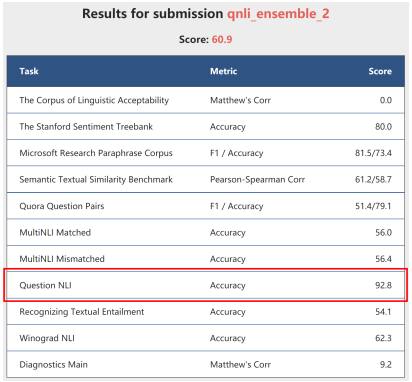

Fig. 4. Best Prediction Result(ensemble)

As we explained in the previous Section 3, we introduced TensorBoard in the program to visualize the training process. Figure 6 $\sim$ 8 shows the evaluation accuracy, evaluation loss and training loss during the training process, respectively. The smoothing coefficient in all these three figures is set to 0. During the training process, we record the loss every 100 steps and perform an evaluation every 1000 steps. If the current evaluation result is the best, we will save the current model in addition.

In addition to the above results, we also found a very strange thing, when we do not change the batch size and warmup proportion, only adjust the epoch, the model has a certain chance to "get lost" in the training process. The final trained model will give exactly the same output when predicting, that is, both entailment or not_entailment in our case. This is also why the submissions QNLI_3, QNLI_5, and QNLI_6 are not listed in Table 1. In order to solve this problem, we adjust the warmup proportion when the epoch is greater than 3. When epoch is 4, we set the warmup proportion to 0.075.

Finally, we also tested the impact of different software versions on the experimental results. As the records whose submission names are QNLI_8 and QNLI_13 shown in Table 1, even if the parameters are the same, the difference in the software version will lead to a gap in the experimental results.

5 Conclusion

In this report, we first reviewed and briefly introduced BERT and some other related methods like Transformer and GPT. Next, in the experiment, we tried a variety of parameter combinations and improved our prediction accuracy by integrating data sets and model ensemble. Finally, we

Results for submission qnli_4

Score: 60.8

Task	Metric	Score
The Corpus of Linguistic Acceptability	Matthew's Corr	0.0
The Stanford Sentiment Treebank	Accuracy	80.0
Microsoft Research Paraphrase Corpus	F1 / Accuracy	81.5/73.4
Semantic Textual Similarity Benchmark	Pearson-Spear...	61.2/58.7
Quora Question Pairs	F1 / Accuracy	51.4/79.1
MultiNLI Matched	Accuracy	56.0
MultiNLI Mismatched	Accuracy	56.4
Question NLI	Accuracy	92.2
Recognizing Textual Entailment	Accuracy	54.1
Winograd NLI	Accuracy	62.3
Diagnostics Main	Matthew's Corr	9.2

Fig. 5. Best Prediction Result(single)

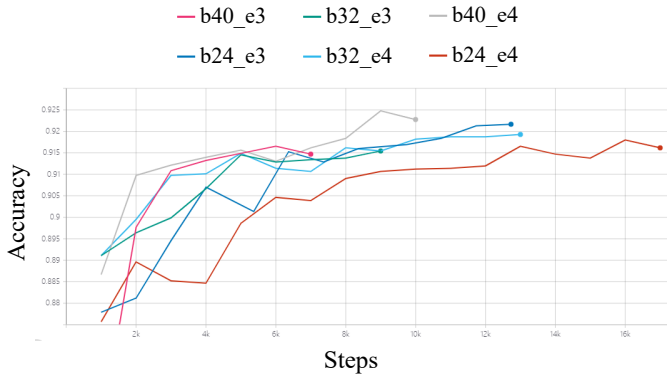

Fig. 6. Evaluation Accuracy

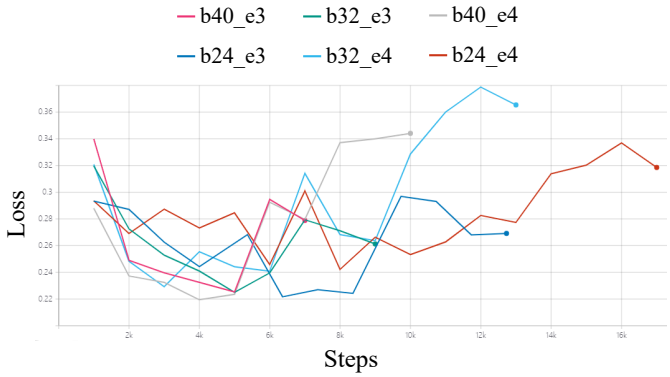

Fig. 7. Evaluation Loss

achieved a prediction accuracy of **92.8%** on the GLUE QNLI task based on the BERT model in this project.

Acknowledgments

The author would like to thank Prof. Hai Zhao and TA Shu Jiang and Zhuosheng Zhang for their instructions on this work.

References

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

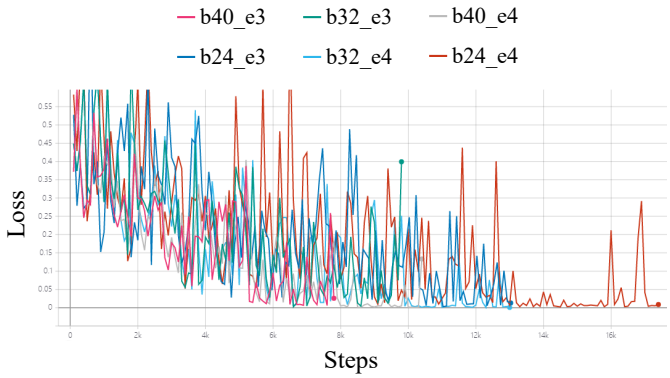

Fig. 8. Train Loss

[2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[3] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[4] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[5] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," *arXiv preprint arXiv:1802.05365*, 2018.

[6] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/languageunsupervised/language understanding paper. pdf*, 2018.

[7] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.

[8] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," *arXiv preprint arXiv:1606.05250*, 2016.

[9] P. Rajpurkar, R. Jia, and P. Liang, "Know what you don't know: Unanswerable questions for squad," *arXiv preprint arXiv:1806.03822*, 2018.

[10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[11] X. Liu, P. He, W. Chen, and J. Gao, "Multi-task deep neural networks for natural language understanding," *arXiv preprint arXiv:1901.11504*, 2019.

[12] ——, "Improving multi-task deep neural networks via knowledge distillation for natural language understanding," *arXiv preprint arXiv:1904.09482*, 2019.

[13] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and V. Q. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," *arXiv preprint arXiv:1906.08237*, 2019.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[17] P. J. Liu, M. Saleh, E. Pot, B. Goodrich, R. Sepassi, L. Kaiser, and N. Shazeer, "Generating wikipedia by summarizing long sequences," *arXiv preprint arXiv:1801.10198*, 2018.