Natural Language Processing Project Report

Fuming Zhang, 118033910025

1 Introduction

2 Background

2.1 Transformer

Transformer [1] [2]

2.2 GPT

GPT [3]

2.3 BERT

BERT [4]

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 example *run_classifier.py* it provided 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. 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*.

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

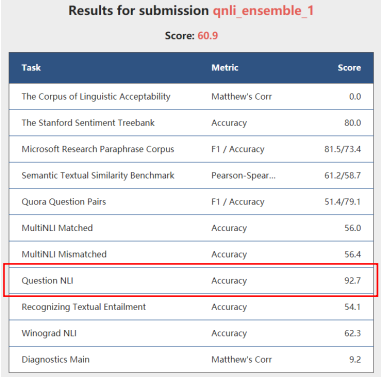
4 Experimental Results

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 memroy per card). The main program we use and its version 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 2). When training this model, we integrated the training set and the test set together. In order to further improve the prediction accuracy, we selected four models with relatively good performance. The submission names of the four models we selected are QNLI_2, QNLI_4, QNLI_11 and QNLI_12. 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 92.7% (as shown in Figure 1).

As we explained in the previous Section 3, we introduced TensorBoard in the program to visualize the training process. Figure 3 $\sim$ 5 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.

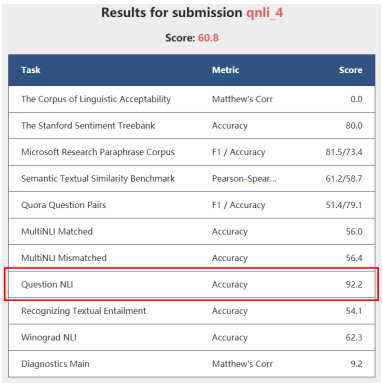


Results for submission qnli_ensemble_1

Score: 60.9

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	92.7
Recognizing Textual Entailment	Accuracy	54.1
Winograd NLI	Accuracy	62.3
Diagnostics Main	Matthew's Corr	9.2

Fig. 1. Best Prediction Result(ensemble)



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. 2. Best Prediction Result(single)

Finally, we also tested the impact of different software versions on the experimental results.

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_**torch_0.4.1**_b32_e3	91.6%
QNLI_ENSEMBLE_1	QNLI_2+4+11+12	**92.7%**

5 Conclusion

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] 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.

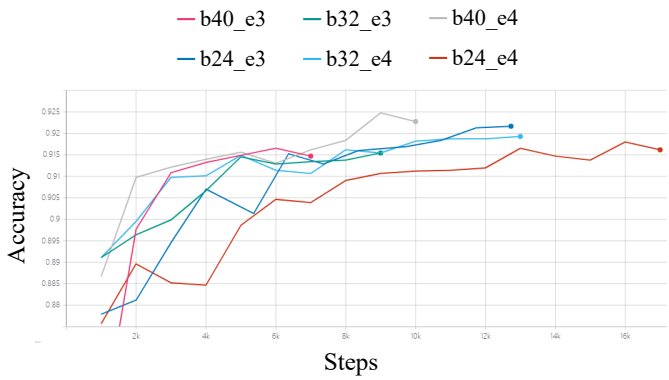


Fig. 3. Evaluation Accuracy

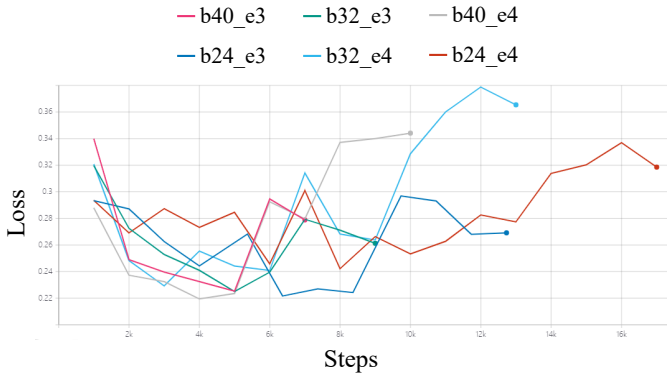


Fig. 4. Evaluation Loss

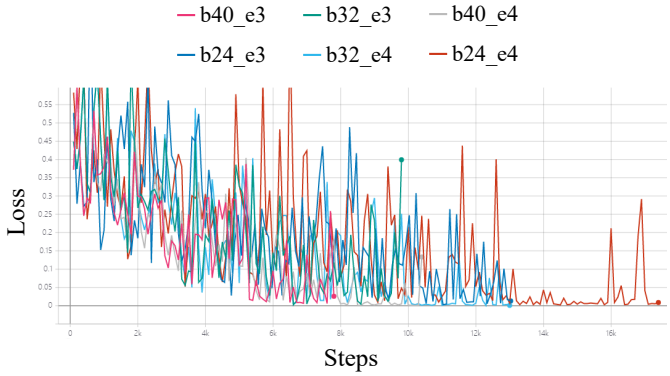


Fig. 5. Train Loss

[2] Z. Dai, Z. Yang, Y. Yang, W. W. Cohen, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context," *arXiv preprint arXiv:1901.02860*, 2019.

[3] 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.

[4] 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.