Ensemble and Multitask Learning with BERT for Question Answering on SQuAD 2.0

Stanford CS224N Default Project

Alvin Hou
Department of Computer Science
Stanford University
`alvinhou@stanford.edu`

Fang-I Hsiao
Department of Electrical Engineering
Stanford University
`fihsiao@stanford.edu`

Abstract

In this project, we aim to apply different ensemble models on BERT-based deep learning methods for question answering. We start from fine-tuning different types of BERT models such as BERT, RoBERTa, and ALBERT, including both base and large models. Our next step is to combine their outputs to produce the final results. In addition, we explore multitask learning by adding an auxiliary classification task to predict if a question has answers. Our experiment shows that it can slightly improve the performance of BERT.

1 Approach

Main Approach. Our approach is split into 2 phases. We are now on phase 1 and it is more like the experimental phase, where we fine-tune various past state-of-the-art models to see whether we could successively reproduce their results. We fine-tuned several BERT-based models on the SQuAD 2.0 tasks, including BERT-base , BERT-large [1], RoBERTa-base , RoBERTa-large [2], ALBERT-base-v2, ALBERT-large-v2 and ALBERT-xxlarge-v1 [3]. This phase is quite time consuming (some models took 3 days to train) and computationally intensive. We have almost finished fine-tuning 9 different BERT variants with promising F1 and EM scores.

Moreover, we try some experiments to see whether multitask learning could improve the model's performance on SQuAD. The task we pick is to classify whether a question has an answer.

We define $x_{\text{embeddings}}$ as the output of a model

$$x_{\text{embeddings}} = [e_{\text{CLS}}, e_1, e_2, ..., e_n] \in \mathbb{R}^{\text{max\_sequence} \times \text{hidden\_size}}$$

Then we take the embedding output of the `CLS` token and feed it into a linear layer

$$x_{\text{out}} = \mathbf{W}_{\text{proj}} e_{\text{CLS}} \in \mathbb{R}^2$$

where $\mathbf{W}_{\text{proj}} \in \mathbb{R}^{\text{hidden\_size} \times 2}$

Finally, we use $\text{softmax}(x_{\text{out}})$ to predict whether a question has an answer and train it jointly with the originally QA task. This learned task could come in handy since we could use it as a confidence measurement whether our *No_Ans* prediction is correct.

Our code is based on the provided starter code and a famous implementation called Hugging Face [1], which provides various pretrained BERT models and a training script [2] to fine-tune those models. We

[1]Huggingface Pytorch Transformers `https://github.com/huggingface/transformers/`
[2]Hugging Face Training Script `https://github.com/huggingface/transformers/blob/master/examples/run_squad.py`

modified the script so that it supports multitask learning and generating output embedding to prepare for applying ensemble methods.

Baseline. As we are training different types of BERT models, we use the BERT base model as our baseline which is the most basic model proposed by the initial BERT paper. This is a natural baseline when comparing different types of BERT models and we expect other BERT models are able to outperform it. It also serves as a baseline when we add the auxiliary classification task since we mainly implement the additional task and conduct the experiment on top of it. The fine-tuned result of the BERT base model on SQuAd2.0 is 73.46 EM and 76.67 F1.

2 Experiments

Data. We use the provided SQuAD dataset which is generated from the `setup.py` script.

Evaluation method. The two main metrics we use are **EM** and **F1**. One additional metric we focus on is the **classification accuracy** for whether a question has an answer or not.

Experimental details. We fine-tuned several BERT-based models on the SQuAD 2.0 tasks, including BERT-base , BERT-large [1], RoBERTa-base , RoBERTa-large [2], ALBERT-base-v2, ALBERT-large-v2 and ALBERT-xxlarge-v1 [3]. We also redesigned BERT-base and jointly train it with a classification task, which classifies whether a question has an answer or not.

We use learning rate 3e-5 for all models. For BERT-base, RoBERTa-base and ALBERT-base-v2, we set max sequence length to 384 and batch size to 8. For larger models such as BERT-large, RoBERTa-large, ALBERT-large-v2 and ALBERT-xxlarge-v1, we set max sequence length to 256 and the batch size to 4-8 due to hardware constraints. We fine tune the models for 2 to 3 epochs.

Results. RoBERTa-base and ALBERT-base-v2 yields similar results with around 78 EM and 82 F1, which is quite a large improvement over BERT-base. We found that for any BERT-family, the larger models completely outperform the smaller ones. We also observe that a lot of models may be undertrained, since the Roberta-large improved after we trained for 3 epochs instead of 2. The best score we get so far is **86.23** EM and **89.23** F1 from fine-tuning ALBERT-xxlarge-v1, which took almost 3 days to train. Our results will be evaluated on the PCE leaderboard. See Table 1 for the detailed evaluation results.

From Table 2 and Figure 1 (see the purple and blue line), we could observe that jointly train the BERT-base with a classification task yields a slightly better result. With this auxiliary task, the model seems to improve a bit. This seems reasonable since this auxiliary task behaves like an additional regularization which forces our model to learn useful information to classify if the question has an answer. However, we currently test it on only one model and the performance only improves slightly, which might not be enough for us to analyze its effect thoroughly. Therefore, our next step includes evaluating this additional task on other BERT models.

3 Future work

The next phase is to build an ensemble pipeline and combine these models to improve the F1 and EM scores. The challenge we might face is how to design a memory and computational efficient pipeline to overcome some hardware constraints. We plan to test 3 different ensemble methods, which are voting, weighted average ensemble and ensemble stacking. In addition, we plan to conduct experiments of the auxiliary task on different BERT models to analyze its effect completely. Finally, we might explore some data augmentation technique such as Easy Data Augmentation [4] to see if it can help with the fine-tuning phase and improve the performance.

4 Appendix

Table 1: SQuAD results on the Dev dataset

Model Type	EM	F1	Epochs	Training Time
BERT-base	73.46	76.67	2	3hr
BERT-large	81.56	84.92	2	15hr
RoBERTa-base	78.82	82.16	2	4hr
RoBERTa-large	82.82	86.32	2	1d 8hr
RoBERTa-large	83.35	86.62	**3**	1d 3hr
ALBERT-base-v2	78.38	81.50	2	3hr
ALBERT-large-v2	81.16	84.22	2	16hr
ALBERT-xxlarge-v1	**86.23**	**89.23**	**3**	**2d 13h**

Table 2: Performance with adding Classification Task

Model Type	EM	F1	No Ans F1	Training Time
BERT-base	73.46	76.67	74.62	3hr
BERT-base with CLS	**73.91**	**76.87**	**75.82**	4hr

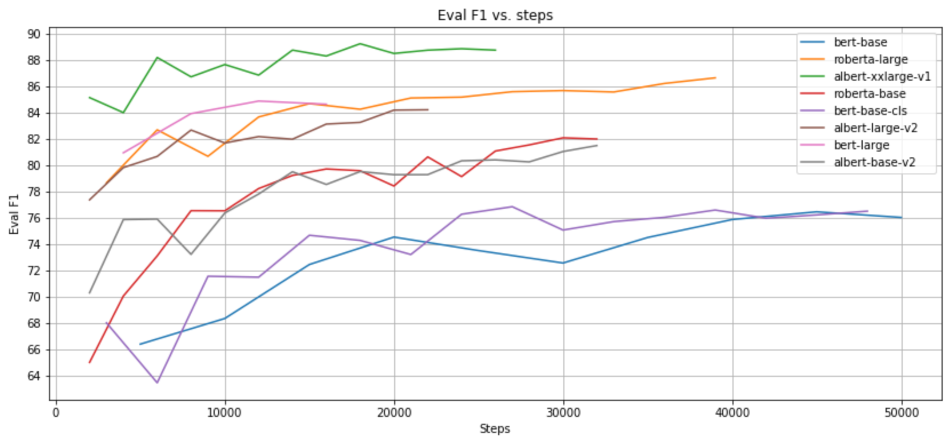

Figure 1: Plot of Evaluation F1 vs. number of steps

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[2] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[3] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2019.

[4] Jason W Wei and Kai Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196*, 2019.