CS230

Fine-tuning BERT On Fine Foods

Mengmeng Ji
Department of East Asian Languages and Cultures
Stanford University
mmj710@stanford.edu

1 Introduction

In the field of Natural Language Processing (NLP), text classification is a classic problem. Previous work has shown that pretrained models are beneficial in classification tasks and other NLP problems. Especially **B**idirectional **E**ncoder **R**epresentations from **T**ransformers (BERT), a multi-layer bidirectional Transformer which is trained on plain text or word prediction and next sentence prediction task. BERT have come to dominate the NLP landscape since their introduction by (Devlin et al. 2018). This largely replaced word embedding approaches such as word2vec (Mikolov et al. 2013) and Glove(Pennington, Socher, and Manning 2014). The transformer architecture that BERT uses was first introduced in (Vaswani et al. 2017) and has gained popularity on a multitude of tasks as it can train faster than previous RNN approaches such as (Liu, Qiu, and Huang 2016).
Although BERT has achieved great performance in many NLP tasks, its potential in text classification has not been fully explored. "How to Fine-Tune BERT for Text Classification" (Sun et al. 2019) provides effective methods for fine-tuning BERT on the text classification task. Fine-tuning BERT for different NLP tasks is a valuable skill for deep-learning practitioners but can present a daunting challenge for novices in the field. This project will reproduce the approach used in the (Sun et al. 2019) paper and extend it to the Amazon Fine Food Reviews text classification dataset. This project will apply these fine-tuning methods on the Amazon Fine Food Review dataset, which was not included in the original paper. In this way, I will confirm whether the methods from the paper have broader applicability across different datasets, and particularly if they can work on a new, more challenging dataset.

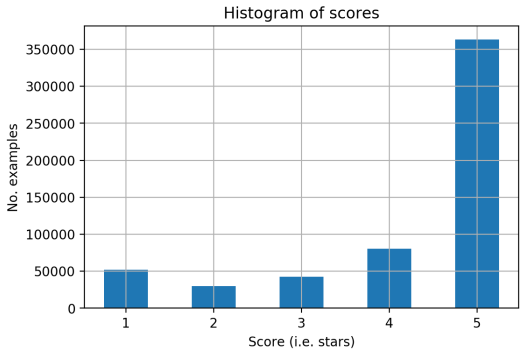
2 Dataset and Features

The Amazon Fine Food Reviews dataset is available through `https://www.kaggle.com/snap/amazon-fine-food-reviews`. The dataset contains 568,454 reviews of food written between 1999 and 2012. Each review contains both text and a score (1 to 5) for the product it reviews. Additional metainfo such as review "usefulness" (as judged by other users), timestamp, and summary are available but will not be used in the project.
The dataset predominantly contains positive 5 star reviews and a smaller proportion of 1-4 star reviews as is shown by the histogram in figure 1. Interestingly, there are more 1 star reviews than there are 2 or 3 star reviews.

As this is a relatively large dataset (500k+ reviews), it gives us a lot of flexibility to try a variety of model sizes and approaches. In particular it makes it possible to to perform in-task pretraining as has been suggested in (Sun et al. 2019).

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

Figure 1: Distribution of labels in the Fine Foods dataset



2.1 Creating the Fine-tuning Dataset

The 568,454 examples are split into a training set of 508,454 reviews, a development set with 40,000 samples, and test set with 20,000 samples.
The splits are then preprocessed independently (to prevent cross-contamination) following the classification example from the official BERT repository.[1]
Inputs to a BERT model are expected to have a fixed structure of [CLS, Text_a, SEP, Text_b]. In this classification example Text_a is the text of the review that we want to classify (first 128 tokens), Text_b will be left blank. Finally, the labels for each example (review) is the corresponding score (i.e. how many stars).
For the **baseline model** I performed the following pre-processing on each example: 1) lowercase the text, 2) tokenize the text, 3) break the word into WordPieces, 4) map the words to indexes using a vocab file that BERT provides, 5) add special "CLS" and "SEP" tokens, 6) append "index" and "segment" tokens to each input. Most of this functionality is readily accessible through tensorflow-bert. Furthermore the vocabulary file is part of the BERT-mini tensorflow hub module.
For the **final model**, instead of simply using the first 128 tokens of the review as text_a, I use the head+tail truncation methods mentioned in (Sun et al. 2019). This involves taking the first 64 and the last 64 tokens of the review. If the review is shorter than 128 tokens the entire text is used, if the review is longer, I throw out the middle tokens and make sure to keep the first and final (head and tail) 64 tokens. The motivation for this is quite intuitive as the start and end tokens of review tend to contain the most valuable information (Sun et al. 2019).
Finally, I write the datasets to the disk as a tfrecord file. Because the Fine Food dataset is relatively large this process takes approximately 20 minutes. By writing it to disk directly we also ensure that the train/dev/test splits is fixed, thus preventing cross-contamination.

2.2 Creating the In-task Pretraining Dataset

The BERT models provided on tfhub have been pretrained on a large corpus of Wikipedia articles. Therefore there is a mismatch between the training set (Wikipedia) and the target set (reviews). In (Sun et al. 2019) it is suggested to take the BERT model pretrained on Wikipedia, and adapt it to the on-task distribution by performing further pretraining with the on-task text. In this way we can hope to make the model more domain-specific and improve generalization.
Pretraining BERT involves two separate tasks: Mask Language Modeling (MLM) and Next Sentence Prediction (NSP) Devlin et al. 2018. These two tasks require us to preprocess the reviews differently than when doing fine-tuning. To create the in-task pretraining dataset I kept the same train/dev/test split of the Amazon Fine Foods Reviews dataset as the baseline model.
Much of the functionality to preprocess the data is available through tensorflow-bert, however the reviews still need to be converted into the format that the BERT package expects. For this I do the following work: 1) write the raw Reviews.csv data into files with one sentence per line and a blank line between different documents (i.e. reviews). Next I use the tokenizer (and vocabulary) that comes with the pretrained tfhub model. Finally, I provide the list of input files and the tokenizer to the

[1] See `https://github.com/google-research/bert/blob/master/predicting_movie_reviews_with_bert_on_tf_hub.ipynb/`

tensorflow-bert library. [2]
The resulting dataset is then written to disk as a single tfrecord file consisting of around 800k examples. Each examples has a maximum sequence length of 256, which is double the 128 sequence length used in the fine-tuning task. This is required for the next sentence prediction task where the goal is to determine whether text_a and text_b come from the same or different reviews (where each of the texts may have up to 128 tokens). For the masked language modelling task 15% of the tokens (40 tokens) are masked out. In this task the model must correctly "fill in the gaps" i.e. determine what token was there before masking.

3 Methodology

3.1 Baseline model

While the final model will use the fine-tuning tricks identified in (Sun et al. 2019), the milestone approach establishes what performance can be achieved with minimal time spent tuning hyperparameters. This will provide a the baseline against which the final model will be compared, both in terms of development time and performance. Therefore, for better comparison, instead of using LSTM, I use a pretrained version of BERT as baseline model.
Additionally, to keep the computational cost reasonable I work with a smaller pretrained version of BERT referred to as **BERT-mini**. This pretrained model is publicly available through tensorflow hub[3]. This version of BERT contains 4 encoder blocks with a hidden size of 256. The multi-head attention uses 4 attention heads. For contrast, **BERT-base** contains 12 layers with a hidden size of 768, and 12 attention heads. The availability of different sizes of pretrained models on tensorflow-hub makes it relatively easy to switch to a larger model later in the development process if higher accuracy is desired.
However in this project the focus is primarily on replicating and evaluating the fine-tuning tricks identified in (Sun et al. 2019). Specifically, whether they are applicable to the Fine Foods dataset. Working with a smaller model makes it easier and faster to iterate and run experiments.
To build the baseline model, I take the pretrained weights from BERT mini and add a single linear classifier layer on top. To reduce overfitting, I also use a dropout layer before the output layer. To train the model, I use mini batches of batch size 64. As the starting point of tuning hyperparameters, reasonable defaults are provided in the official github classification example.[4] With these defaults, the model trains well. The loss and accuracy on both the training and dev set are summarized in table 1. Despite training the model for a single epoch, there is evidence of some slight overfitting occurring such that performance on the train set is better than on the dev set. This gap could be reduced in the final model by increasing regularization, such as increasing the dropout rate before the classification layer.

3.2 Experiments on Fine Tuning BERT Model

3.2.1 In-task Pretraining Model

As suggested in the last section, first, I pretrain in-task dataset with the BERT model mini. The pretraining task typically contain 2 segments of text i.e. 2* 128, so the maximum sequence length is set to 256. The maximum amount of masked tokens per example in MLM task is 40, which conventionally is about 15% of the length of sequence.
Next, I set up pretrained BERT model for masked language modelling and next sentence prediction (MLM + NSP) task. The final loss is simply the sum of the MLM loss and NSP loss, therefore the model simultaneously learns to do well on both tasks. The cross-entropy loss and accuracy of MLM and NSP tasks in in-task pretraining run for a single epoch can be seen in figure3, respectively.
The NSP task appears to be relatively easy on the reviews dataset. This is likely because the reviews typically mention the food that they are reviewing, so it is easy to detect the mismatch between the first and second sequence. Similarly, the language used in the reviews is less sophisticated as

[2]The procedure of this step is similar to the creating dataset methods in BERT package, seehttps://github.com/google-research/bert/blob/master/create_pretraining_data.py

[3]https://tfhub.dev/google/small_bert/bert_uncased_L-4_H-256_A-4/1/

[4]https://colab.sandbox.google.com/github/tensorflow/tpu/blob/master/tools/colab/bert_finetuning_with_cloud_tpus.ipynb

compared to the Wikipedia articles on which BERT was initially pretrained. As such the model is able to achieve a >80% MLM accuracy in a single epoch.
For the subsequent fine-tuning run, the model weights are initialized from in-task pretraining run. To train the model, I again use mini batches of batch size 64, and set an initial learning rate of 0.00005. The fine-tuned classification accuracy on both training and dev sets are very similar to the baseline model.

3.2.2 Head+Tail Truncation Model

For this experiment, I use the head+tail truncation methods mentioned in (Sun et al. 2019) on BERT mini, choosing the first 64 and the last 64 tokens of the review.The start and end tokens of reviews tend to contains the most information.
The model weights are initialized from Google tfhub checkpoint. I add a single linear classifier layer on top. To train the model, I use mini batches of batch size 64, and set learning rate as 0.00005. To reduce the chance of overfitting, I train the model for one epoch. The performance of this model, as figure 2 and 1 shows, is slightly better than previous experiments.

3.2.3 Head+Tail on BERT Small Model

For this experiment, I implement the head+tail truncation method on a bigger model, BERT small. The BERT small model contains 4 layers with a hidden size of 512, and 8 attention heads. In contrast, my baseline model BERT mini contains 4 layers with a hidden size of 256, and 4 attention heads.
The model weights are initialized from Google tfhub checkpoint. I add a single linear classifier layer on top. I use mini batches of batch size 64, and set learning rate as 0.00005. To reduce the chance of overfitting, I train the model for one epoch. As figure 2 and 1 show, the fine tuned BERT small model gives the best performance.

Figure 2: Summary of fine-tuning training runs (1 epoch)

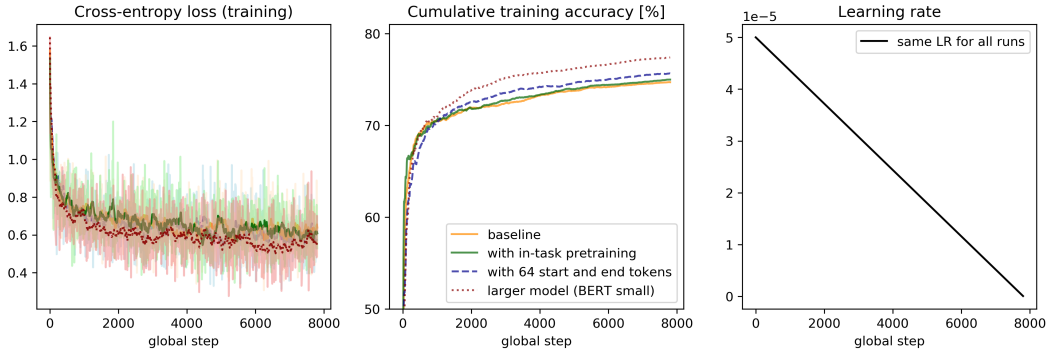
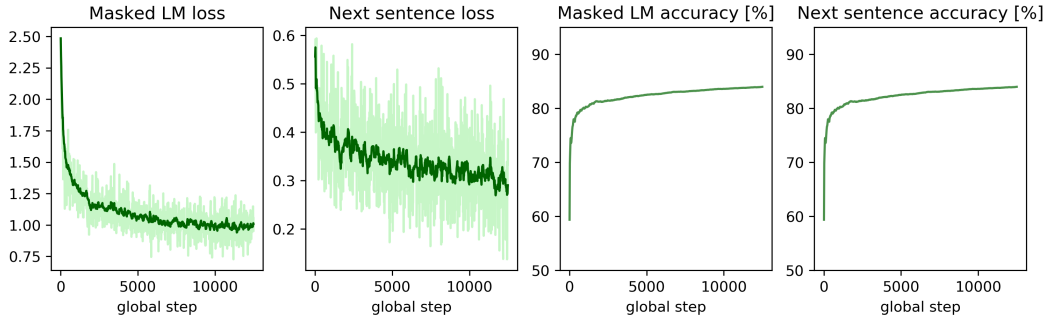


Figure 3: Summary of in-task pretraining run (1 epoch)



model	train loss	dev loss	train accuracy	dev accuracy
baseline (BERT mini)	0.5706	0.6089	0.787	0.771
with in task pretraining	0.5803	0.6187	0.782	0.764
with 64 first and final tokens	0.5505	0.5803	0.795	0.789
larger model (BERT small)	**0.4668**	**0.5328**	**0.830**	**0.804**

Table 1: Summary of performance on the training and development sets for all models

	precision	recall	f1-score	support
1 star review	0.75 / **0.79**	0.70 / **0.73**	0.72 / **0.76**	2010
2 star review	0.29 / **0.42**	0.49 / **0.58**	0.36 / **0.49**	643
3 star review	0.47 / **0.55**	0.49 / **0.57**	0.48 / **0.56**	1442
4 star review	0.31 / **0.41**	0.57 / **0.68**	0.40 / **0.51**	1513
5 star review	0.95 / **0.96**	0.84 / **0.86**	0.90 / **0.91**	14392
accuracy			0.77 / **0.80**	20000
macro avg	0.55 / **0.63**	0.62 / **0.69**	0.57 / **0.65**	20000
weighted avg	0.83 / **0.84**	0.77 / **0.80**	0.79 / **0.82**	20000

Table 2: **Test set** sklearn classification report for different review categories. Comparing: baseline / final model.

4 Results And Insights

Table 2 shows the result of both baseline model and the final model (final model result written in **bold**).

For baseline model result, as shown in table 2,we see that the F1 score is the highest on the over-represented 5-star review class, while it is significantly lagging for the 2-4 star review classes. Interestingly, the 1-star review category also achieves a relatively high F1 score. This suggests that reviews which will give exactly 1-star are easy to identify.

The biggest improvement in classification performance appears in 2-4 star reviews, which are under-represented categories. For instance, the 3 star review category sees a 8 point improvement in F1 score, while the 5 star review category's F1 score increases by only 1 point. Some of this improvement comes from using the head+tail approach suggested in Sun et al. 2019, but most of the benefits should be attributed to using a bigger model. The overall performance gain is encouraging, suggesting there may be further benefit to increasing the model size even more, or using a deeper model, such as BERT-medium (same hidden size as BERT small, but 8 layers instead of 4). This is left as an experiment for future work.

In contrast, in-task pretraining showed only limited benefit, with no significant improvement in overall dev set accuracy. To put things into perspective, running in-task pretraining for 1 epoch is as expensive as fine-tuning the model on the same dataset for 1 epoch, thus when using in-task pretraining, the total training time effectively doubles. When doubling the model hidden size from BERT mini to small, the total training time also doubled, but the benefits were much more significant.

In conclusion, while some of the tricks of fine tuning BERT from Sun et al. 2019 are applicable to new text classification datasets, the most reliable way to improve performance still appears to be using bigger and deeper models.

References

Devlin, Jacob et al. (2018). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805. arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

Liu, Pengfei, Xipeng Qiu, and Xuanjing Huang (2016). *Recurrent Neural Network for Text Classification with Multi-Task Learning*. arXiv: 1605.05101 [cs.CL].

Mikolov, Tomas et al. (2013). *Efficient Estimation of Word Representations in Vector Space*. arXiv: 1301.3781 [cs.CL].

Pennington, Jeffrey, Richard Socher, and Christopher Manning (Oct. 2014). "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in*

Natural Language Processing (EMNLP). Doha, Qatar: Association for Computational Linguistics, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://www.aclweb.org/anthology/D14-1162.

Sun, Chi et al. (2019). "How to Fine-Tune BERT for Text Classification?" In: *CoRR* abs/1905.05583. arXiv: 1905.05583. URL: http://arxiv.org/abs/1905.05583.

Vaswani, Ashish et al. (2017). *Attention Is All You Need*. arXiv: 1706.03762 [cs.CL].