Task-Specific Training: The Benefits of Fine-Tuning BERT

Jack Hart
Georgetown University
csh87@georgetown.edu

Abstract

Fine-Tuning has been found to improve performance when working with pre-trained feature extractors. Pre-training is usually followed by fine-tuning the feature extractor for a particular problem. The down-stream, target task often requires additional layers added onto the pre-trained model for classification. However when performing simpler NLP tasks, such as sentiment analysis, and when training on limited data, it is unclear how much task-specific training of the feature extractor is helpful. This paper performs fine-tuning on BERT for a simple sentiment analysis task. Results are systematically compared to sparse feature baselines and BERT models without fine-tuning. There was no statistically significant difference between a fine-tuned BERT model and one without fine-tuning.

1 Introduction

In transfer learning for natural language processing (NLP), feature-based approaches use the outputs of a pre-trained feature extractor, like ELMo (2018), as static inputs into a separate model. Some feature extractors, such as ULMFiT (2018) and BERT (2018), can also be fine-tuned: trained on additional, domain specific data. Fine-tuning has improved model performance on a variety of NLP tasks (Radford, 2018); accordingly, it has become common practice to fine-tune feature extractors. With supervised fine-tuning, task-specific layers for classification need to be trained along with the feature extractor.

However when performing simpler NLP tasks, such as sentiment analysis, and when training on limited data, it is unclear how much task-specific training of the feature extractor is helpful. This paper investigates the degree to which fine-tuned dense embeddings, specifically BERT embeddings, improve performance (e.g. accuracy, F1). Specifically, we trained models on a sentiment analysis task with a relatively small dataset of tweets. Performance of BERT embedding without any fine-tuning were compared to fine-tuned embeddings. A regularization method of freezing specific layers within BERT was also implemented. As a baseline for performance, non-neural modelling was also performed.

BERT embeddings on their own appeared to only slightly increase accuracy over the baseline logistic regressions using TF-IDF, token count, and token occurrence feature representations. However, BERT models had much higher F1 scores than the baseline. Fine-tuning using baseline hyper-parameters did not result in a statistically significant increase accuracy. Performing layer-specific fine-tuning of BERT, however, does appear to be an effective regularization method.[1]

2 Background

Transfer-learning with pre-trained feature extractors has resulted in state-of-the-art performance on various NLP tasks. Pre-training is usually followed by fine-tuning the feature extractor for a particular problem. The down-stream, target task often requires additional layers added onto the pre-trained model for classification. Tuning feature extractors with additional layers creates unique and insufficiently understood challenges.

2.1 BERT

BERT (Devlin et al., 2018) is a language representation model that has achieved state-of-the-art performance on NLP datasets such as GLUE, MultiNLI, SQuAD v1.1, and SQuAD v2.0. BERT's performance is chiefly attributed to its use of bidirectional encoders, which can learn language context from the left and right. The

[1] This paper's GitHub repo can be found here.

amazing performance of BERT, discussed by (Devlin et al., 2018), has drastically changed the NLP world.

The BERT-base model has 12 encoder layers, each with a hidden size of 768 and self-attention heads. The model can take an input of 512 tokens, and outputs a representation of the sequence. Devlin et al. (2018) recommend using the concatenated output from the last four encoders as the model output. In regards to pre-training, BERT was trained on Masked Language Model and Next Sentence Prediction tasks with Wikipedia data. The optimizer was Adam with decoupled weight decay (Loshchilov and Hutter, 2017). The initial learning rate was 5e-5 and polynomial decay on the learning rate was applied.

2.2 Fine-Tuning

Research has found that unsupervised fine-tuning, tweaking model weights on a large corpus relevant to the target domain, has given word-embeddings better context, improving performance (Dai and Le, 2015; Peters et al., 2017). Unsupervised fine-tuning requires a large corpus in order to prevent catastrophic forgetting, which is the tendency of a neural network to forget previously learned information (Mccloskey and Cohen, 1989). Howbeit, supervised fine-tuning has been found to allow for smaller datasets and faster convergence without running into catastrophic forgetting (Howard and Ruder, 2018). Yet for simpler tasks and when data is relatively small, the utility of using complex feature extractors is unclear.

There has been a good amount of research regarding how one should actually perform supervised fine-tune of complex feature extractor such as BERT. For instance, Sun et al. (2019) applied supervised fine-tuning on individual BERT layers as of form of regularization for their text classification task, finding that models with tuned BERT layers closer to the end had lower error rates. Furthermore, they found that lower learning rates were essential in order for BERT to overcome the catastrophic forgetting problem.

3 The Dataset

The dataset for this task is a collection of Tweets originally collected by Data For Everyone, but now stored on Kaggle. Each Tweet was classified as either about a disaster or not. In total, before splitting into testing and training sets, there are 7,613 classified tweets. Out of all tweets, 4,242 (57%) were about non-disasters, and the remaining 3,271 (43%) were about disasters.

4 Methodology

Following the principle that we should always choose a simpler model if possible, research should address the relative usefulness of these complex models for simpler tasks. This paper implements two approaches for representing tweets: sparse features and dense features using BERT embeddings. The sparse features are used as input into a logistic regression, and the BERT embed dings are fed into a single dense layer with a logistic activation function. Theoretically, the performance of these two model structures should be equivalent, and the performance differences should be mainly attributed to the feature representations.

4.1 Sparse Features

Three different sparse feature representations are used: TF-IDF, token count, and token occurrence/hash vectors. All the sparse feature results are from 10-fold cross validation. The following is a break down of these three feature representations.

- **TF-IDF Features** This vector is created by scaling the number of times each word in a vocabulary appeared in a given Tweet by how often that word is used.

- **Token Count Features** This vector indicates the number of times each word in a vocabulary appeared in a given Tweet.

- **Hashing Features** This vector is created by applying a hashing function to word counts in a Tweet.

4.2 BERT Features

First, BERT features will be used as static input into the neural network and trained accordingly. In other words, BERT will be completely frozen, and no fine-tuning performed. Then this paper also applies fine-tuning to BERT by either completely unfreezing BERT or only unfreezing the first two encoders. Due to time constraints, no hyper-parameter tuning was implemented. Rather, the same hyper-parameters BERT was pre-trained

with are used. More specifically, BERT is fine-tuned and the additional layers are trained with an Adam optimizer and a learning weight of 5e-5.

Additionally, unlike the sparse-feature models, BERT models were trained on the same held-out dataset. This is due to the time-constraints and added complexity for implemented cross-validation with the native BERT model. Nonetheless, the BERT models were trained 10 times each, with different randomly initialized weights each time. For that reason there is still variance in their results.

5 Results

5.1 Baseline Sparse Features

Figure 1 depicts the accuracy and F1 score distributions for the three sparse models. There isn't much variance between these models, however the feature of only counts performed marginally better than the other two representations. Notably for the analysis with BERT embeddings, these models all have an average accuracy of about 0.79 and average F1 score of around 0.74 (refer to Table 1).

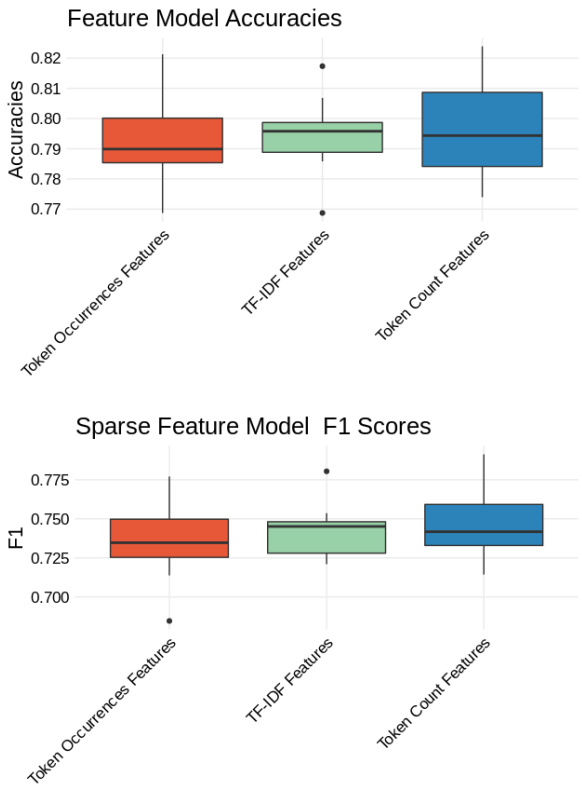

Figure 1: Logistic Regression Accuracy and F1 Scores Using Sparse Feature Representations

Model	Mean Acc.	Mean F1
TF-IDF	0.795	0.742
Count	0.796	0.746
Hash	0.793	0.735

Table 1: Mean accuracy and F1 scores for the baseline sparse feature models.

5.2 BERT Features

Figure 2 depicts the distributions of accuracy and F1 scores for the models using BERT embeddings. To no surprise, these models performed better overall than the baseline. However given the greater complexities of these models, the approximate 3 point increase in accuracy's over the baseline is not a lot. However when looking at the F1 scores, BERT models perform much better over the baseline, almost 10 points higher in some cases.

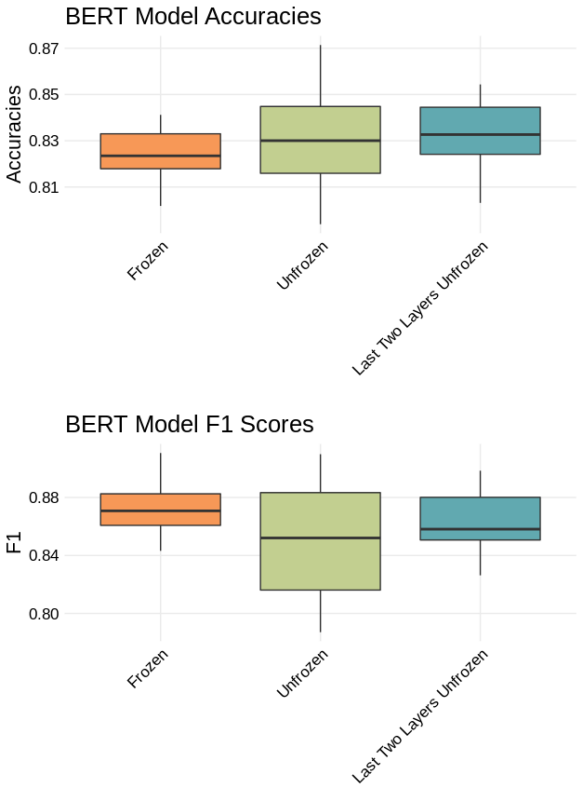

Figure 2: Accuracy and F1 Scores for various BERT models. Each model type was trained 10 times using the same train and test sets.

The average model accuracy's and F1 sores (refer to Table 2) are not very different for the fine-tuned and regular BERT models. Although the completely unfrozen BERT model had a slightly

higher average accuracy than the frozen, this difference was not statistically significant (t = -0.91, df = 13, p = 0.38). Even though the average accuracy of the model with only the last two layers of BERT unfrozen was slightly smaller than the unfrozen model's average accuracy, it's difference from the frozen model was more significant (t = -1.5, df = 16, p-value = 0.14).

Model	Mean Acc.	Mean F1
Frozen	0.824	0.872
Unfrozen	0.833	0.863
Last Two Layers	0.831	0.850

Table 2: Mean accuracy and F1 scores for the BERT feature models.

It is clear from the boxplots in Figure 2 that fine-tuning increased the maximum accuracy, but also increased the variance. However after applying regularization in the form of tuning less layers, this variance decreased. There is no indication here that fine-tuning is very helpful for this particular task, but it does appear that only tuning the last two layers helped regularize the tuning process.

6 Discussion

Future research should extend this analysis to a verity of datasets and applications, and should specifically look at popular baseline datasets such as CoNLL. Furthermore, a more systematic approach to fine-tuning BERT should be developed. For example, there should be an additionally methodology for applying a verity of learning rates and layer freezing combinations. This research is limited in its scope, and future work should build off these initial findings. This work provides substantial evidence that the benefits of fine-tuning on small datasets is limited, and at the very too irregular to be an efficient use of time.

References

Andrew M. Dai and Quoc V. Le. 2015. Semi-supervised sequence learning. *CoRR*, abs/1511.01432.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Jeremy Howard and Sebastian Ruder. 2018. Fine-tuned language models for text classification. *CoRR*, abs/1801.06146.

Ilya Loshchilov and Frank Hutter. 2017. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101.

Michael Mccloskey and Neil J. Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. *The Psychology of Learning and Motivation*, 24:104–169.

Matthew E. Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. 2017. Semi-supervised sequence tagging with bidirectional language models. *CoRR*, abs/1705.00108.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *CoRR*, abs/1802.05365.

Alec Radford. 2018. Improving language understanding by generative pre-training.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune bert for text classification? *ArXiv*, abs/1905.05583.