Fine-tuning minBERT on Downstream Tasks with Gradient Surgery and Weighted Losses

Stanford CS224N Default Project

Andrew Gan
Department of Computer Science
Stanford University
gans@stanford.edu

Tee Monsereenusorn
Department of Computer Science
Stanford University
teem@stanford.edu

Gareth Cockroft
Department of Computer Science
Stanford University
garethc@stanford.edu

Abstract

The Bi-Directional Encoder Representations from Transformers (BERT) model is known for its impressive results after pre-training on Masked Language Modeling and Next Sentence tasks over a large corpus of Wikipedia articles. In this project, we first implemented features of a baseline BERT model before exploring further fine-tuning methods on to improve BERT's multi-task performance. Our implemented extensions improve BERT fine-tuning in order to train the our model for sentiment analysis (SA), semantic textual similarity (STS), and paraphrase detection (PD). We find that using Pearson-similarity to predict Semantic Textual Similarity yielded better performances than using cosine-similarity. We also find that using a weighted sum of losses worked equally well compared to gradient surgery.

1 Introduction

Sentiment Analysis, Paraphrase Detection, and Semantic Textual Similarity are three specific text classification tasks in NLP. While we do not implement any original fine-tuning methods, this project was approached as an opportunity to explore how the different methods affect BERT's performance on the mentioned three classification tasks. Further, we had the advantage of the large Stanford Sentiment Tree Bank (SST), Quora, and SemEval STS Benchmark datasets to further train our model. Our work here differs from previous BERT fine-tuning research in its focus on these specific tasks and datasets.

Learning sentence representations underlies a model's assessment of semantic equivalence. Recent developments in NLP have shown that models pre-trained on a large, unsupervised corpus are highly effective for a variety of text classification and NLP tasks (Jacob Devlin and Toutanova, 2018). Such pre-trained models have shown higher performance than new models that are trained completely from scratch. Specifically, BERT has achieved state-of-the-art performance after pre-training on a large corpus of Wikipedia articles.

As an extension to BERT's already impressive performance on various text classification problems, significant work has been done to further optimize BERT performance. Previous work includes fine-tuning through further pre-training and multi-task fine-tuning (Sun et al., 2020), implementations of cosine similarity (Reimers and Gurevych, 2019), regularized optimization (Jiang et al., 2020), and much more. However, there is currently limited research on fine-tuning BERT for target tasks. In this project, we explore fine-tuning for the express purpose of improving performance on the three mentioned text-classification tasks.

The outline of our process throughout this project is as follows:

- We build and evaluate the performance of a base minBERT model. As a baseline, we fine-tune just our model as a single-task classifier on just Sentiment Analysis.
- We advance our BERT implementation to become a multi-task classifier built for Sentiment, Paraphrase Detection, and Semantic Textual Similarity.
- We then explore methods of fine-tuning the multi-task classifier with the goal of simultaneously improving performance on all three tasks. Here, we explore the affects of cosine similarity, Pearson coefficients, and MSE loss to improve Semantic Textual Similarity. To address multi-task performance, we experiment with weighted loss sum and gradient surgery to isolate relevant parameters to their respective tasks.
- We find BERT's performance across STS to be heavily improved by these fine-tuning methods, and that both weighted loss sum and gradient surgery offer modest multi-task improvements across all three tasks.

2 Related Work

Fine-tuning BERT for Target Tasks Work done by Sun et al. (2020) explores effects of various fine-tuning methods on BERT performance. Through experiments over eight widely studied text-classification datasets, they explore finetuing via further pre-training, multi-task learning, and target task fine-tuning. Their work was not meant to contribute new or unique NLP functionality, nor does it focus on optimizing BERT for single tasks. Rather, it focuses on finetuning BERT as a multi-task classifier. Given its broad approach to fine-tuning, there is room among the research for us to explore fine-tuning affects specific to our three desired text classification tasks. Sun et al. (2020) also mention the lack of significant research towards fine-tuning BERT for target tasks.

BERT on Semantic Textual Similarity In regards to specifically improving BERT performance on STS, Arase and Tsujii (2019) conduct significant work through a case study focused on improving semantic and paraphrase understanding. Rather than making any contributions that increase model size, their work investigates the addition of semantic relations into BERT. This addition was found to empirically increase BERT's sentence representation learning. These representation were shown to be especially useful for aiding in semantic equivalence tasks like STS. However, they find that performance over single sentence task sentiment classification on the Stanford Sentiment Treebank

dataset actually degrades. While this research is effective in demonstrating improved STS performance, our project hopes to improve multi-task classification performance, including both STS and single-sentence sentiment classification.

The Effects of Finetuning on BERT Substructure Determining the effectiveness of various fine-tuning methods also involves investigating how the underlying embedding structure of the model is changed by fine-tuning. Using classifier based probing and `DIRECTPROBE`, Zhou and Srikumar (2021) explore BERT's changes as a result of fine-tuning for five NLP tasks including text-classification. Their work concludes that fine-tuning affects classification performance by further distancing data examples with different examples. This would lead us to believe that explicit efforts to separate the features essential for each task could further improve BERT's classification perormance. Specifically, this encouraged our use of gradient surgery to improve performance across all three of our target tasks.

3 Approach

3.1 minBERT

We first built a standard minBERT implementation upon foundations provided by the CS224N course staff. Note, BERT normally uses learnable segmentation embeddings in order to differentiate between different sentences in the input. As we are only performing single-sentence tasks, our implementation does not use segmentation embeddings. minBERT is a minimal implementation of the BERT model presented by Jacob Devlin and Toutanova (2018). To complete the baseline implementation of minBERT, we built the Multi-Head Self-Attention and Transformer layers.

In addition to the standard minBERT implementation, we added an Adam Optimizer step function. At a time step t we use a parameter vector θ_t, biases β_1 and β_2 to estimate first and second gradients moments as m_t and v_t:

Algorithm 1 Adam Optimizer Step

$t \leftarrow t + 1$
$g_t \leftarrow \nabla f_t(\theta_{t-1})$ # Get gradients
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ # First raw moment estimate
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ # Second raw moment estimate
$\alpha_t \leftarrow \alpha_t \cdot \sqrt{1 - \beta_2^t}/(1 - \beta_1^t)$ # Bias-correct each moment estimate and compute new parameters
$\theta_t \leftarrow \theta_{t-1} - \alpha_t \cdot m_t/(\sqrt{v_t} + \epsilon)$

3.2 Extensions

Using the pre-trained minBERT model as our base model, we implemented a number of extension methods to fine-tune the model for performance on sentiment analysis, paraphrase detection, and semantic textual similarity. Below we describe the changes made relevant to each individual task, as well as those applied to all the resulting losses.

Sentiment Analysis. We first obtained a pooled representation of each sentence with BERT, encoded in the first [CLS] token. To classify the sentence, we applied dropout to the representation, then projected the result with a linear layer. We then calculated the categorical cross-entropy loss by first applying a softmax activation function, as defined by:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{1}$$

followed by a cross-entropy loss function, defined by:

$$CE = -\sum_{c=1}^{M} y_c \log(\sigma(z_i)) \tag{2}$$

where y_c is the true label for the category c.

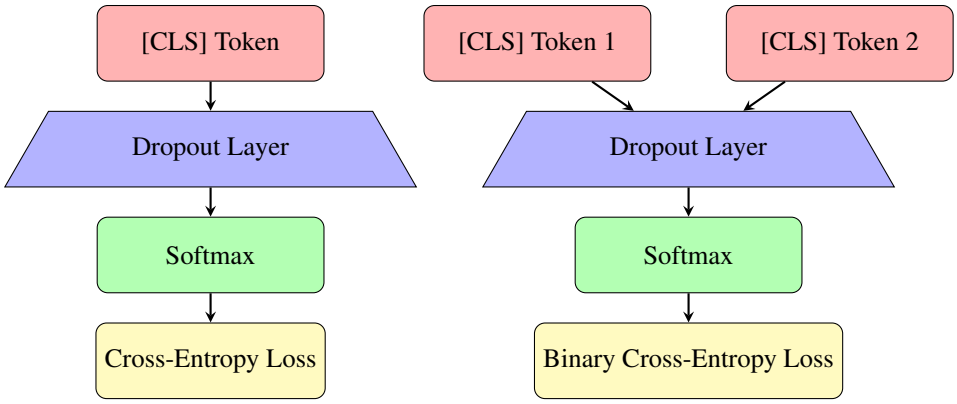

Figure 1: *Left*: Sentiment Analysis. *Right*: Paraphrase Detection

Paraphrase Detection. We first obtained the pooled representations of each of the two sentences to be compared using BERT, then concatenated the representations together before projecting it with a linear layer. To obtain the loss, we used a binary cross-entropy loss function, defined by:

$$BCE = -(y \log(p) + (1 - y) \log(1 - p)) \tag{3}$$

where y is the true label and p is the logit.

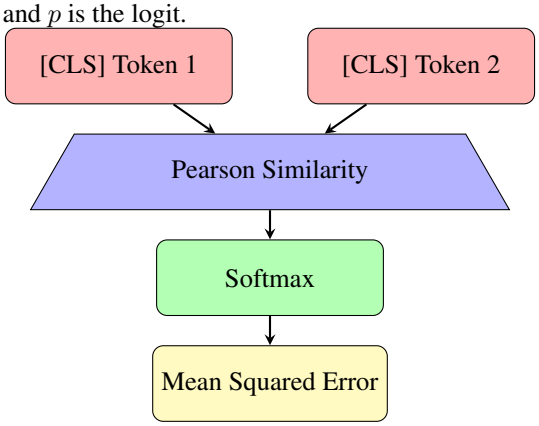

Figure 2: Calculating loss for Semantic Textual Similarity

Semantic Textual Similarity. We experimented with several fine-tuning techniques with mixed results. We began by evaluating the cosine similarity between the pooled representations of two sentences, where cosine similarity is defined by:

$$\cos(x, y) = \frac{x \cdot y}{|x||y|} \tag{4}$$

We soon found that cosine similarity was not a very effective prediction function for semantic textual similarity, prompting us to switch to Pearson correlation instead, defined by:

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2(y_i - \overline{y})^2}} \tag{5}$$

where the pairwise distance was used to compute the distance between the embeddings. To compute the loss, we first rescale the calculated Pearson correlation to a range of 0-5 by putting it through a Sigmoid and multiplying the output by five. We then use mean squared error to compute loss, defined by:

$$mse = \sum_{i=1}^{D}(x_i - y_i)^2 \tag{6}$$

Once we have the losses from each of the three fine-tuning tasks, we originally decided to combine the losses and train on the resulting sum. First, we implemented our own method of weighting the loss from each task. Before summing losses, each individual loss is weighted based on the proportion its dataset comprises of all three task's datasets. For example, STS loss is weighted by x/y where x is the number of examples in the SemEval dataset and y is our total number of examples. Lastly, as an alternative to the sum of weighted losses, we implemented gradient surgery (Yu et al., 2020) to project conflicting gradients from one task to the normal space of the conflicting task's gradient. This can be described by the equation:

$$g_i = g_i - \frac{g_i \cdot g_j}{||g_j||^2} \cdot g_j \tag{7}$$

where g_i is the gradient of the i'th task, and g_j is the gradient of the conflicting task.

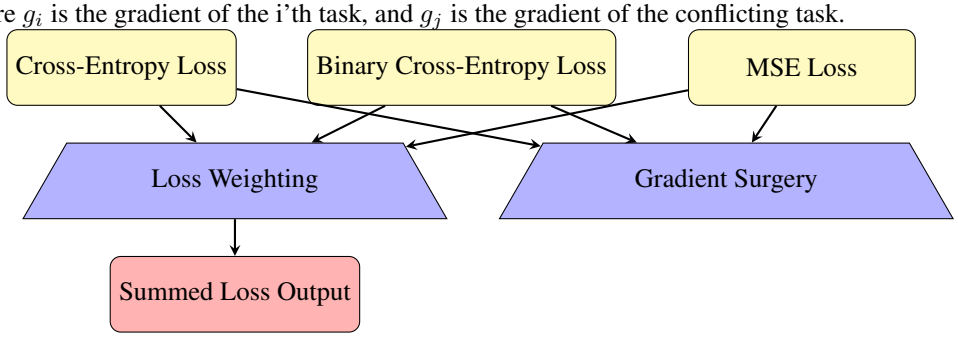

Figure 3: Multitask loss was trained on the weighted sum of losses or on gradient surgery-adjusted loss.

4 Experiments

4.1 Data

We trained and tested our BERT model's performance on the following three tasks across two datasets. Sentiment Analysis–classifying an input text as either **positive**, **negative**, or **neutral**. Paraphrase Detection—detecting if an input phrase is a paraphrasing of something else. Semantic Textual Similarity (STS)–where the model is given two input sentences, and rates their semantic similarity on a scale from 5, semantically equivalent, to 0, completely unrelated. For this paper, our model was fine-tuned and tested on all three tasks. Below are the details of the datasets used.

- **Sentiment Analysis –** *The Stanford Sentiment Treebank*[1] *(SST) dataset*: The SST dataset consists of 11,855 single-sentence movie reviews, comprised of 215,514 unique phrases. Each review is labeled one of five sentiments from the choices of **negative**, **somewhat negative**, **neutral**, **somewhat positive**, or **positive**. The dataset is split into 8,544 training examples, 1,101 dev examples, and 2,210 test examples.
- **Paraphrase Detection –** *Quora*: The Quora dataset used used in this project consists of 400,000 question pairs and binary labels denoting if text samples are paraphrases of each other. The dataset is split into 141,506 training examples, 20,215 dev examples, and 40,431 test examples.
- **Semantic Textual Similarity –** *SemEval STS Benchmark Dataset*: The SemEval dataset consists of 8,628 sentence pairs with simlarity rated from 0 to 5, meaning unrelated or equivalent, respectively. The dataset is split into the 6,041 training examples, 864 dev examples, and 1,726 test examples.

4.2 Evaluation Method

Our model's performances on sentiment and paraphrase tasks are measured through its accuracy, defined by whether or not each model prediction exactly matches the true classification of each input example. The percentage of predictions that match the label is kept as its accuracy score.

[1]1https://nlp.stanford.edu/sentiment/treebank.html

For evaluating the model on semantic textual similarity, we use the Pearson Coefficient, described in equation 5.

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2(y_i - \overline{y})^2}}$$

where x are our model predictions for the similarity and y is the true label. Higher values of the Pearson Coefficient indicate high correlations between the predicted similarity and the actual similarity.

4.3 Experimental details

We experimented with a number of different model configurations using different permutations of the extensions that we outlined in Section 3. The base hyperparameters that we used to train and compare results amongst the different configurations were 10 epochs, learning rate of 1×10^{-5}, and fine-tuning enabled. After training each model, the model's accuracies on the dev sets for Sentiment Analysis and Paraphrase Detection were obtained, along with Pearson Coefficients for Semantic Textual Similarity (as reported in Table 1). We then experimented with different hyperparameter configurations on our best performing model to attempt to further optimize the model.

4.4 Results

Model	**Sentiment Analysis** (Stanford Sentiment Treebank)	**Paraphrase Detection** (Quora)	**Semantic Textual Similarity** (SemEval STS)
BERT	0.344	0.512	-0.029
BERT-COS-WL	0.482	0.738	0.375
BERT-COS-GRAD	0.478	0.736	0.401
BERT-PEARS-WL	**0.483**	0.731	**0.517**
BERT-PEARS-GRAD	0.469	**0.747**	0.509

Table 1: All listed scores are the Dev accuracies achieved by each model on the respective task and dataset. BERT is the base BERT implementation. Suffixes COS and PEARS denote models using cosine similarity and Pearson correlation, respectively. Suffixes WL and GRAD denote models using weighted losses and gradient surgery, respectively.

Fine-tuning First, using Pearson-similarity to make semantic textual similarity predictions demonstrated a significant improvement over using cosine-similarity. This is to be expected as Pearson-similarity is the evaluation metric used to determine semantic textual similarity and thus was a more accurate training method for the downstream task. However, we were surprised to see that gradient surgery did not offer significant improvements over weighted loss, despite conflicting gradients across the three tasks. It may be the case that training with just weighted loss already separates certain parameters to be of more importance for different tasks.

Hyperparameters			Performance per Task (Dataset)		
Epochs	**Learning Rate**	**Dropout**	**Sentiment Analysis** (Stanford Sentiment Treebank)	**Paraphrase Detection** (Quora)	**Semantic Textual Similarity** (SemEval STS)
10	1×10^{-5}	0.3	**0.479**	**0.731**	**0.517**
20	1×10^{-5}	0.3	0.469	0.657	0.436
10	1×10^{-3}	0.3	0.473	0.698	0.499
10	1×10^{-7}	0.3	0.405	0.632	0.412
10	1×10^{-5}	0.4	0.460	0.722	0.501

Table 2: All listed scores are the Dev accuracies achieved by each model on the respective task and dataset.

Experimenting With Hyperparameters. Additionally, we took our best model (BERT-PEARS-WL) and experimented with different hyperparameters to determine the best combination. Our findings are reported in Table 2 (found at the end of Section 4).

Our findings show that the hyperparameters best suited for our model are the base numbers that we started with (10 epochs, lr=1e-5, dropout=0.3). Increasing the number of epochs led to overfitting, where the training losses were much smaller and training set accuracies were very high, however yielded substantially worse results on the validation set. Observing our performance after each epoch made it clear that the best results came within the 8-12 epoch range before the model overfits. Adjusting the learning rate also yielded worse results on the dev set. A higher learning rate of 1×10^{-3} achieved good results much faster than the lower learning rate, however was never able to optimize further. This is likely because the higher learning rate was causing the model to overshoot optimal values. A lower learning rate took far longer to converge and achieve good results, and did not perform well in 10 epochs.

After experimenting with the different model and hyperparameter configurations, we settled on the model that computed STS with pearson coefficient, trained on the weighted sum of the losses. The hyperparameters we used were a learning rate of 1×10^{-5}, trained for 10 epochs with fine-tuning turned on. Our results on the test set were:

Sentiment Analysis (Stanford Sentiment Treebank)	**Paraphrase Detection** (Quora)	**Semantic Textual Similarity** (SemEval STS)	**Overall test score**
0.487	0.751	0.447	0.562

Table 3: All listed scores are the Test accuracies achieved the optimal model on the respective task and dataset.

5 Analysis

5.1 Sentiment Analysis

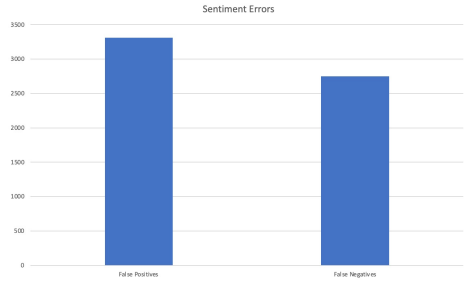

Figure 4: *Sentiment errors* – 3,310 false positives and 2,748 false negatives.

Our model's performance on sentiment analysis averaged the worst on the dev set amongst our three tasks. Here, "false positive" denotes a prediction that was incorrectly more positive than the true label while "false negative" is a prediction incorrrectly more negative than the true label. From the slight disparity in types of incorrect predictions, it is difficult to conclude that our model is overly sensitive to positive or negative words. Here are two example predictions:

Correct Label: 3 **Predicted Label:** 1
A movie that successfully crushes a best selling novel into a timeframe that mandates that you avoid the Godzilla sized soda .

A review like this has multiple words that would cause opposite interpretations if understood in isolation. "Successfully" implies a positive review, while "crushes" and "avoid" imply negative. Perhaps our model interprets certain words more strongly than others, possibly using a strong word like a "avoid" to identify a negative review.

Correct Label: 3 **Predicted Label:** 4 Richard Gere and Diane Lane put in fine performances as does French actor Oliver Martinez .

In contrast, while the correct label for this review is the same as the previous false negative label, our model predicted too positive here. Notably, this this example contains few words that are indicative of the sentiment. Aside from the subjects and predicates, there is a single sentiment modifier, "fine", which causes a positive sentiment review. This shows perhaps our model is too sensitive to single salient words.

5.2 Paraphrase Detection

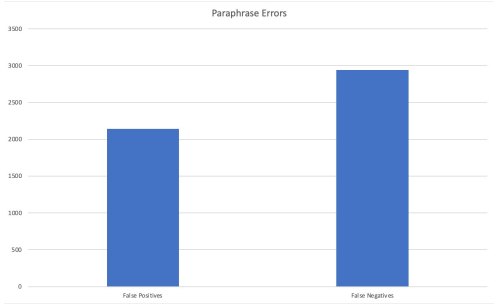

Figure 5: *Paraphrase errors* – 2,139 false positives and 2,943 false negatives.

Of the three downstream tasks that our model was finetuned for, the model's performance on paraphrase was by far the best at nearly 75% accuracy. However, of the 25% errors that it made, the majority were false negatives as reported in Figure 3.

For instance, the model incorrectly classified that the sentences, "How does diet affect hair health?" and "How does diet affect hair (appearance, color, loss, etc)?" were not paraphrases of each other. One possible explanation for this could be that the model's embeddings are unable to encode the meaning of health as it relates to hair, such that it is unable to recognize that appearance, color, and loss are all facets of hair health. This points to a larger issue in the model's ability to differentiate the meaning of words in different contexts.

5.3 Semantic Textual Similarity

On the test set, our models performance on STS was the worst of our three tasks. Looking at the results across many of our predictions, our model frequently predicted lower similarity than the true labels indicated. For example:

Correct Label: 4.5 **Predicted Label:** 1.2
Sentence 1: A cat is drinking milk. **Sentence 2:** A kitten is drinking milk.

Despite these two sentences having identical words save for "kitten" and "cat," our model greatly underestimated a similarity of 1.2. From this we can conclude that it must have extremely disparate understandings of the words "cat" and "kitten." Possibly, it understands these two words are the *subject* of each sentence, and the fact that they differ has enough weight to profoundly affect the predicted label.

6 Conclusion

Our findings show that the different optimizations we implemented to fineitune for the three downstream tasks had varying levels of success. Our implementation for weighted loss sums performed equally well compared to gradient surgery, while predicting semantic textual similarity using Pearson correlation substantially increased performance compared to cosine similarity. Additionally, experiments with different hyperparameters showed that the base hyperparameters we used were optimal for our model.

However, there were some limitations to our model. Firstly, our semantic textual similarity scores were much lower on the test set than the dev set, meaning our model may not be as generalizable as we thought. Further research can be done to find more generalizable fine-tuning techniques beyond Pearson similarity and MSE loss, perhaps by adding learnable parameters for STS on top of computing scores with just BERT embeddings.

Specific to these downstream tasks, our performance issues indicate some possible avenues for target-task finetuning. To adjust for over predictions based on single salient words in sentiment analysis, future work might include further training around a corpus of sentiment-indicative words. In regards to paraphrase detection and STS, it seems our model would benefit from better syntactic understanding. With further syntax-focused fine-tuning, our model could avoid over-weight subject sentence components, and better understanding nuanced meanings of words.

References

Yuki Arase and Junichi Tsujii. 2019. Transfer fine-tuning: A BERT case study. volume abs/1909.00931.

Kenton Lee Jacob Devlin, Ming-Wei Chang and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. In *arXiv preprint arXiv:1810.04805*.

Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2020. SMART: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. volume abs/1908.10084.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2020. How to fine-tune bert for text classification?

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient surgery for multi-task learning.

Yichu Zhou and Vivek Srikumar. 2021. A closer look at how fine-tuning changes BERT. volume abs/2106.14282.