Optimizing minBERT on Downstream Tasks Using Pretraining and Siamese Network Architecture

Stanford CS224N Default Project

Edwin Pua
Department of Computer Science
Stanford University
puaedwin@stanford.edu

Abstract

BERT is a remarkably powerful sequence-to-sequence model built on top of a transformer architecture that can output descriptive vector embeddings for sentences. In this paper, I plan to build on top of BERT and optimize BERT for three tasks - sentiment analysis, paraphrase detection, and similarity recognition between sentence pairs. To do so, I implement a Siamese-inspired network architecture, use back translation to augment the datasets, use SMART regularization, and pretrain BERT. In particular, the Siamese architecture and BERT pretraining were most effective at increasing performance in all three tasks.

1 Introduction

Due to the advent of the transformer model, multitask transfer learning on natural language processing tasks has exploded in popularity. Specifically, when models like Bidirectional Transformers for Language Understanding, or BERT for short, are trained to, for example, output the sentiment of a sentence as positive or negative, this knowledge that BERT is retained and can adapted for other tasks, like language translation and text summarization. In fact, we've seen that BERT smashed records on General Language Understanding Evaluation (GLUE) performance tasks and improve the record accuracy on datasets MultiNLI and SQuAD. [1]

This paper aims to explore ways to adapt BERT to three different tasks - sentiment analysis of a sentence, paraphrase detection between sentence pairs, and similarity recognition between sentence pairs. In other words, how can BERT and extensions of BERT be trained for, say, sentiment analysis, then transfer that knowledge for paraphrase detection and similarity recognition.

My approach involves refining BERT through a combination of modern techniques, including the implementation of a Siamese-inspired network architecture, dataset augmentation via back translation, integration of SMART regularization, and BERT pretraining. Through meticulous experimentation, I explored the efficacy of these methodologies, particularly highlighting the significant performance boosts achieved by the Siamese architecture and BERT pretraining across all targeted tasks.

2 Related Work

Because my extensions concern the Siamese network architecture, SMART regularization, and BERT pretraining, which I will go into more detail later, here are some papers that inspired the direction I took for multitask learning:

2.1 Pretrained Language Models

Devlin et al. introduced BERT, a transformer-based model pretrained on large amounts of unlabeled text using masked language modeling and next sentence prediction [1]. This has allowed BERT to

capture contextual information and semantic representations of text, which is precisely what enables transfer learning across various downstream tasks.

2.2 Siamese Network Architectures

Siamese network architectures have been widely used in tasks involving similarity assessment, such as paraphrase detection and sentence similarity recognition. Reimers and Gurevych introduced SBERT, which produces sentence embeddings that can detect semantic similarity between pairs of sentences [2].

2.3 Regularization Techniques

Regularization techniques are essential for preventing overfitting on training data and improving model generalization. SMART (Structured Model-Aware Regularization Technique) proposed by Jiang et al. introduces regularization based on task-specific losses [4]. SMART regularizes models by penalizing the model for having overly complex classification boundaries, which enhances robustness and generalization performance.

3 Approach

3.1 Baseline

For the baseline minBERT model, I implemented the multi-head attention layer of the transformer, coded the architecture for a single layer of the Bert model, defined the forward function of the BertSentimentClassifier, and implemented the step function of the Adam optimizer using momentum.

For the multi-head attention layer, I matrix multiplied the batches of key and query vectors, normalized by the square root of the size of the attention head, added the mask so that the padding tokens are not considered when calculating attention scores, and softmaxed across the dimension corresponding to each token, so that each key vector has a certain attention score probability associated with it, and all probabilities sum to one. Then I matrix multiplied by the value vectors to get the scaled dot product attention for one head - each head is then concatenated together.

The output of the multi-head attention layer is passed into a linear layer (i.e. multiplied by W_O). Dropout and a residual connection are applied to it before being normalized and passed into a feed-forward network (which is a linear layer followed by ReLU/GELU). This output is followed by yet another linear layer, dropout, residual connection, and normalization.

The forward function of the BertSentimentClassifier simply calls forward once on the BERT model, then the corresponding pooler output has a linear layer and dropout applied to it.

The Adam optimizer is implemented exactly as described in the handout, including bias-corrected first and second moments for a rolling average momentum and weight decay as regularization.

To implement the baseline MultitaskClassifier, I first implemented baseline versions of each classification layer. For instance, for the sentiment layer, I passed the given sentence through BERT to obtain its embedding, then passed that into a hidden layer and linear layer to obtain one of five different sentiment classes, with ReLU for nonlinearity and dropout for regularization. For the similarity and paraphrase layers, I passed both sentence one and sentence two through BERT to obtain the pooler output of their embeddings u and v, then concatenated u and v together and passed them through a hidden layer followed by a linear layer to obtain a logit, also with ReLU for nonlinearity and dropout for regularization. In the case of the similarity classification layer, I bounded the output of the final layer between 0 and 1 by applying a sigmoid, then multiplied that result by 5 to get a float between 0 and 5.

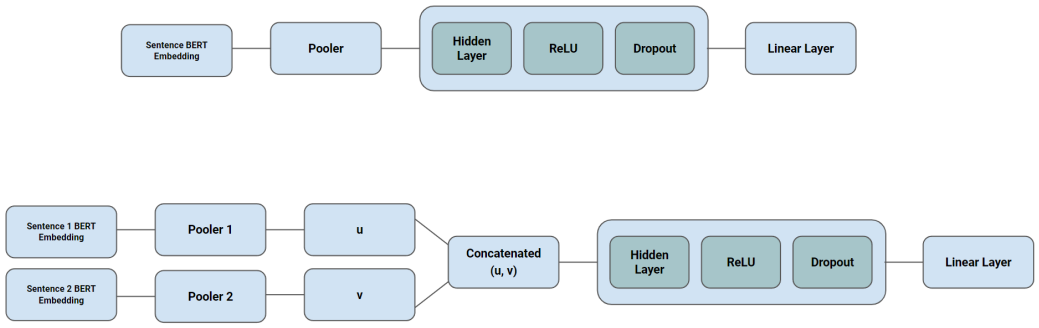

Figure 1: Sentiment Pipeline (Top) and Paraphrase/Similarity Pipeline (Bottom)

Because the output label for the semantic textual similarity task is a float between 0 and 5, I used mean square error loss. Because the paraphrase output label is a float of either 0 or 1, I used binary cross entropy loss with logits. Because the sentiment classification outputs one of five different sentiment classes, I used cross entropy loss.

3.2 Extension 1 - Siamese Architecture

I then improved upon the similarity and paraphrase layers by taking inspiration from the Siamese BERT network architecture by Reimers and Gurevych, modifying the input to both of these layers to be $(u, v, |u - v|)$ rather than just (u, v). [2]

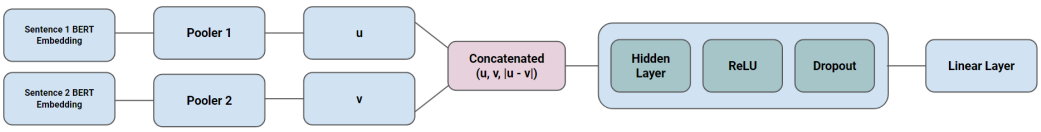

Figure 2: Updated Hidden Layer Input to Paraphrase/Similarity Pipeline

3.3 Extension 2 - Bigger Classification Architecture

To bring training loss down, I decided to increase the number of hidden layers from one to two for the paraphrase and similarity classification layers. My reasoning was that I would rather overfit my training data than underfit my training data, because overfitting on training data can be corrected through regularization techniques or additional training data. I chose not to increase the number of hidden layers for the sentiment classification layer because even just a single hidden layer was already overfitting on the training data.

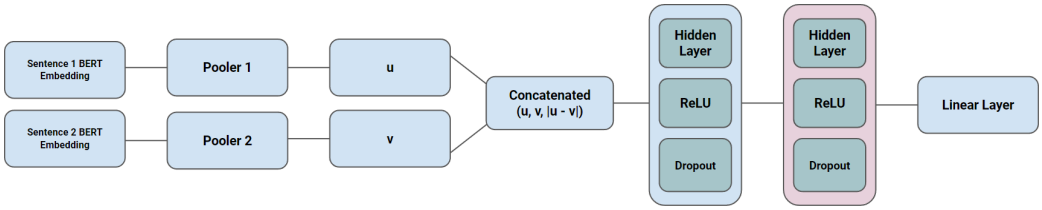

Figure 3: Increased Hidden Layers From One to Two for Paraphrase/Similarity Tasks

3.4 Extension 3 - Back Translation Augmentation

To help drive validation loss down, I also tried to augment the Semantic Textual Similarity dataset with back translation. For each sentence pair (s_1, s_2) in the training set of the STS dataset, I created a translated sentence pair (s_1', s_2'), where s_1' is equivalent to s_1 translated to Chinese, then back to English using EasyNMT. [3] My new training set consisted of four new sentence pairs for each original sentence pair: (s_1, s_2), (s_1', s_2), (s_1, s_2'), and (s_1', s_2'). My reasoning was that back translation would perhaps bolster the datasets with new sentence pairs that are distinct enough from their parents.

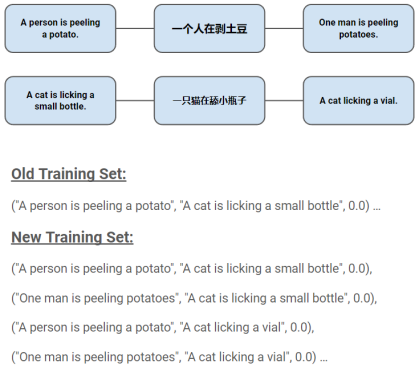

Figure 4: Quadrupled Size of Semantic Textual Similarity Dataset Using Chinese Back Translation

3.5 Extension 4 - SMART Loss Regularization

In addition to back translation, I also incorporated SMART Loss regularization for the sentiment classification task by Jiang et al., which minimizes $\mathcal{L}(\theta) + \lambda_S \mathcal{R}_S(\theta)$ as opposed to $\mathcal{L}(\theta)$, where $\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i;\theta), y_i)$ (ℓ is mean square error loss, binary cross entropy loss with logits, and cross entropy loss for the similarity, paraphrase, and sentiment, tasks respectively) and $\mathcal{R}_S(\theta) = \frac{1}{n}\sum_{i=1}^{n} \max_{||\tilde{x}_i - x_i||_p} \ell_S(f(\tilde{x}_i;\theta), f(x_i;\theta))$, where $\tilde{x}_i$ is a sampled perturbation of x_i and ℓ_S is chosen to be KL-divergence loss. [4]

3.6 Extension 5 - Additional Pretraining

Because I didn't want to exclusively train the classification layers, I also unfroze the BERT layers by setting each parameter 'requires_grad' in the BERT model to True. This way, BERT would give more robust and specialized embeddings for all three tasks.

4 Experiments

4.1 Data

For the sentiment classification task, I use the Stanford Sentiment Treebank, which contains 8544 movie reviews in the training set, with five possible corresponding sentiment labels of 0 (negative) to 4 (positive). [5]

For the paraphrase classification task, I use the Quora Question-Pair dataset, which consists of 141506 question pairs in the training set and corresponding label of 0 (questions are not duplicates of each other) or 1 (questions are duplicates of each other). [6]

For the similarity classification task, I use the SemEval STS Benchmark Dataset, which consists of 6041 sentence pairs in the training set and a corresponding label ranging from 0.0 (sentences are on different topics) to 5.0 (sentences are equivalent and mean the same thing). The spectrum of possible labels is all floats between 0 and 5. [7]

As mentioned earlier, I also created an augmented version of the SemEval STS Benchmark Dataset using back translation, which contains 6041×4 sentence pairs in the training set.

4.2 Evaluation method

Like the leaderboard, we evaluate the sentiment and paraphrase tasks using accuracy and evaluate the similarity task using Pearson correlation. Once Pearson correlation is normalized to be a number between 0 and 1, we can take an unweighted average of the three evaluation metrics to get an overall performance score.

4.3 Experimental details

I arbitrarily chose a learning rate of 1e-5, dropout probability of 0.5, batch size of 8, an ADAMW weight decay of 1e-5, and a SMART regularizer weight of .02. The size of each embedding and the hidden size for each the sentiment classification layer is 768. The hidden size and input size for the baseline similarity/paraphrase classification layers is 768×2 to accommodate (u, v), and that of the Siamese classification layers is 768×3 to accommodate $(u, v, |u - v|)$. For each extension, I ran for ten epochs - if after ten epochs, the dev accuracy/correlation was still exhibiting an upwards trend, then I extended the training period to one hundred epochs.

To reiterate, almost all of the hyperparameters were chosen arbitrarily. In hindsight, I would've liked to perform a random search over the possible hyperparameter combinations to see which exhibit the best results.

4.4 Results

	Sentiment (SST)		**Sentiment (CFIMDB)**	
Model Type	Accuracy	Benchmark	Accuracy	Benchmark
Pretrain	0.399	0.35	0.763	0.70
Finetune	0.529	0.45	0.938	0.90

Table 1: Dev accuracy of baseline minBERT model on sentiment classification compared to autograder benchmarks.

First, I implemented the baseline minBERT and trained a sentiment classifier to verify that all moving pieces were working properly (BERT model embeddings, AdamW optimizer, etc.) Because I pass all autograder benchmarks, I can be reasonably confident that all code blocks were implemented properly and start implementing extensions.

	Sentiment (SST)	**Paraphrase (Quora)**	**Similarity (STS)**	
Extension Type	Dev Accuracy	Dev Accuracy	Dev Correlation	Overall Dev Score
Baseline Model After Finetune	0.411	0.375	0.201	0.428
After Siamese	0.411	0.687	0.398	0.599
After Hidden Layer Increase	0.411	0.694	0.448	0.610
After Back Translation	0.411	0.694	0.449	0.610
After SMART	0.414	0.694	0.449	0.611
After Unfreezing BERT	**0.493**	**0.853**	**0.817**	**0.752**
	Test Accuracy	Test Accuracy	Test Correlation	Overall Test Score
After Unfreezing BERT	**0.494**	**0.858**	**0.794**	**0.750**

Table 2: Dev and test accuracies after each milestone.
Extensions were implemented in the order shown.

The extensions I decided to implement were not chosen arbitrarily, but chosen in response to the current dev scores and training losses. For instance, after implementing the baseline model, I noticed the dev scores for the paraphrase and similarity tasks were comparatively low compared to the sentiment task, so I implemented the Siamese network architecture changing the inputs to these classification layers from (u, v) to $(u, v, |u-v|)$. This led to a massive improvement in the paraphrase task accuracy (.375 to .687) and similarity correlation (.201 to .398).

Next, I noticed that training accuracy was staying relatively close to the dev accuracy for the paraphrase and similarity tasks. Since there wasn't any overfitting on these tasks yet, I decided to increase the number of hidden layers for the paraphrase and similarity classification networks from one to two, leading to a small improvement in paraphrase task accuracy (.687 to .694) and similarity correlation (.398 to .448).

I then tried to quadruple the size of the similarity training set with back translation to see if I could improve STS score further, but this did not noticeably impact the similarity correlation (.448 to .449). Although this is not listed in the table above, it is worth noting that the time per epoch increases from about 20 seconds to a little over a minute.

Then, I noticed that there was a massive discrepancy between the dev accuracy on the sentiment task (hovering around .4) and the training accuracy of the sentiment task (hovering around .8). So I decided to apply the SMART regularization technique to the sentiment training process, but only got a slight accuracy increase that may not even be attributable to SMART (.411 to .414).

Finally, I found a bug in my code that was always setting the BERT model parameters require_grad to False, so I resolved this bug and unfroze the BERT layer parameters, yielding the biggest improvements to the overall dev score since the Siamese architecture implementation (.414 to .493 on sentiment task, .694 to .853 on the paraphrase task, .817 to .858 on the similarity task).

Unsurprisingly, the final test scores were .494 accuracy on sentiment, .858 accuracy on paraphrase, and .794 on similarity, for a final test score of .75, staying relatively close on all metrics to the final dev scores.

5 Analysis

Initially, I was surprised that my sentiment dev accuracy (and everyone else's dev accuracy on the leaderboard) could never seem to exceed 0.6, but upon inspecting the sentiment dev set predictions, I noticed how difficult many of the examples were. For instance, one such example movie review is "Enormously entertaining for moviegoers of any age." Because the sentiment is positive but not strongly positive, both the model and I would predict a label of 3 - however, the true label is 4. Many of the erroneous dev outputs are off by one, because even though the emotional connotation might be obvious, the strength of a given emotion can be hard to detect using just small blurbs. Not to mention every critic and reviewer has their own personal rating scale with some rating stricter than others. As such, one interesting idea for an extension would be to normalize each rating in the dev set by subtracting it from the average rating of its respective reviewer, assuming that the number of reviews is non-trivially larger than the number of critics (i.e. one critic is responsible for multiple reviews). A rating of 3 from a strict reviewer has stronger positivity than a rating of 3 from a reviewer that gives nearly everything perfect scores.

Regarding the paraphrase task, many of the errors on the dev set are due to large amounts of word overlap between the two questions, meaning they have similar sentence embeddings and have hard-to-detect nuance. For instance, we see that the two questions "Is there any chance of medal for India in Rio Olympics 2016?" and "How many medals is India expected to win at the 2016 Rio Olympics?", we can see that the true label should be (and is) 0, but because there are a lot of similar words, the model will output 1, even though theoretically the word "many" should have a high attention score and therefore some non-insignificant influence on the embedding for that question.

Regarding my extensions - the likely reason why implementing a Siamese network architecture works so well for the paraphrase and similarity tasks is because the smaller in magnitude or closer to the zero vector that $|u - v|$ is, the likelier that the two sentences in a sentence pair are similar.

Unfreezing the BERT layers intuitively increases performance as well, since backpropagation now updates all parameters in the pipeline, not just the classification layers. In other words, unfreezing BERT allows the model to adapt its parameters to specific characteristics of the sentiment, paraphrase, and similarity tasks and learn task-specific nuances.

Regarding the extensions that didn't work well - I speculate the main reason that augmenting the STS dataset with back translation wasn't very effective was because the translated pairs didn't add enough diversity to the training set. Consider the sentence pair (s_1, s_2) = ("The woman is slicing tofu.", "The woman is putting on lipstick."). Here, the translated sentence pair using back translated Chinese is (s_1', s_2') = ("That woman is cutting tofu", "That woman's wearing lipstick."). Thus, s_1 and s_1' probably have nearly identical sentence embeddings, and s_2 and s_2' also have nearly identical sentence embeddings. Thus, the four combinations of concatenated embeddings (u, v, $|u - v|$), (u', v, $|u' - v|$), (u, v', $|u - v'|$), and (u', v', $|u' - v'|$) are all practically identical - it would be similar to copy pasting each sentence pair three additional times in the training set.

I am still unsure why SMART did not work well on SST - I double-checked to see if I had a bug in my code, but $\mathcal{R}_S(\theta)$ was returning nonzero values that weren't suspicious. If I had to guess why: let's imagine that sentiment was determined by two features (x, y), and each data point was plotted on a 2D plane. Then, the boundaries between sentiment classification would be 1D curves, separating the data points into clusters. What SMART would theoretically do to these boundaries is smooth them and make them less sharp - this mitigates overaggressive finetuning. If this regularization did not help the dev accuracy, then it's possible that the problem isn't that the classification boundaries are too sharp, but instead that the decision boundaries are placed inaccurately in the first place. This is likely due to training data that isn't fully representative of all possible sentiment movie reviews. If this is the case, then the better way to reduce validation loss for SST would be to find a way to obtain more data for the SST dataset.

6 Conclusion

One limitation of my project includes the lack of hyperparameter tuning using some sort of grid or randomized search on the appropriate number of hidden layers, hidden layer size, learning rate, batch size, etc., which would have probably allowed me to squeeze out some extra performance. Another limitation is that I don't know the efficacy of each extension in isolation, nor did I test specific combinations of extensions. For instance, it's possible that using SMART after unfreezing BERT would've shown that SMART is potent after all. It's also possible that there may be a specific performance difference between the baseline model with all the extensions implemented and the baseline model with only the Siamese and BERT unfreeze extensions implemented.

One avenue for future work: I noticed that unfreezing the BERT layers when training for the sentiment classification task would lower the dev score for the paraphrase and similarity tasks, and the same is true when training for other classification tasks. Thus, one way to improve the dev and test scores would be to implement gradient surgery.

Despite all this, I'm happy to have learned about the power of Siamese network architectures on similarity tasks, as well as the benefit of allowing gradient flow through all layers of a neural network pipeline versus only the final classification layers. I'm also happy to have learned that SMART is not a regularization panacea for all forms of overfitting, and that data augmentation using back translation is limited in its success.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. https://arxiv.org/abs/1810.04805

[2] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 3982–3992, 2019.

[3] Nils Reimers. EasyNMT. https://github.com/UKPLab/EasyNMT

[4] Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. arXiv preprint arXiv:1911.03437, 2019. https://aclanthology.org/2020.acl-main.197

[5] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In Proceedings of the 2013 conference on empirical methods in natural language processing, pages 1631–1642, 2013. https://aclanthology.org/D13-1170

[6] https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs

[7] Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. * sem 2013 shared task: Semantic textual similarity. In Second joint conference on lexical and computational semantics (*SEM), volume 1: proceedings of the Main conference and the shared task: semantic textual similarity, pages 32–43, 2013. https://aclanthology.org/S13-1004