Multi-Task Learning with BERT

Stanford CS224N Default Project

Naveen Kumar
Department of Computer Science
Stanford University
navee2@stanford.edu

Abstract

This project explores multi-task learning with BERT on three natural language processing tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. We employ a variety of techniques focusing on regularization techniques to improve the performance of the multi-task BERT model. We first establish a baseline using a LSTM model for each task. For model training, we start with individual models for each task and analyze the impact of various hyperparameters such as dropout and learning rate on each task. We then implement cosine annealing on the learning rate, use the AdamW optimizer, increase the training epochs, and vary the number of layers to be fine-tuned in the BERT model. For the semantic textual similarity task, we also experiment with different similarity structures to capture the semantic relationship between sentences. Finally, we train all three tasks together on a single BERT backbone and compare the performance with the individual task models. Our experiments demonstrate that multi-task learning with BERT can significantly improve the performance of some tasks.

1 Key Information to include

- Mentor: Cathy Yang

2 Introduction

Pre-training of language models has become a crucial step in natural language processing tasks, such as text classification, sentiment analysis, paraphrase detection, and many others. The goal of pre-training is to train a model on a large corpus of unannotated text data in an unsupervised manner. This process allows the model to learn the underlying patterns and structures in natural language, which can then be used to improve performance on downstream NLP tasks. Pre-training typically involves training a deep neural network on a large corpus of text data using masked language modeling in which model is trained to predict missing words in a sentence. Such pre-trained models are also called foundation models.

Once the model is pre-trained, it can be fine-tuned on a smaller labeled dataset for specific NLP tasks. Fine-tuning typically involves re-training the last few layers of the pre-trained model on the task-specific dataset.

In multi-task fine tuning different NLP tasks have different objectives and require different types of data and input representations. For example, sentiment analysis may require a model to identify and extract sentiment-related features from the text, while named entity recognition may require a model to identify and extract entities such as names, dates, and locations. This means that a single model that is trained to perform multiple tasks must be able to learn and extract different types of features from the text, which can be a difficult task.

For multi-task fine-tuning we use multiple datasets to build a robust network working across different problems. Different NLP tasks have different objectives and require different model structure. We

aim to create a robust backbone which supports these multiple tasks which adding layers on top of the backbone to support unique characteristics of the problem. Multi-task fine-tuning on BERT can be computationally expensive and requires a large amount of memory, especially when fine-tuning multiple tasks simultaneously. Such a trained model is expected to be robust for other tasks outside of the given tasks. We expect improved performance of all the base problems in the process in comparison to individual fine-tuning.

3 Related Work

Fine-Tuning and specifically Multi-Task fine-tuning on foundation models have been widely studied. Chi Sun et al provides three strategies for fine tuning - differentiate learning rate across layers, additional pretraining in target domain, and train on multiple tasks in target domain.[1]. Smoothness-Inducing Adversarial Regularization is an adverserial loss technique where we add loss corresponding to worst prediction difference in neighboring ϵ distance from inputs. This ensures that prediction is robust.[2]. In terms of loss function there is suggestion of simple addition of loss function from all the tasks.[3] A slight modification known as gradient surgery projects the gradient of the i-th task g_i onto the normal plane of another conflicting task's gradient g_j. We also looked into training of LLaMA model by meta for ways to improve training[4]. They have suggested training with AdamW optimizer with cosine learning rate schedule. Both of these provide some degree of regularization to the learning.

4 Approach

First we start with baseline establishment for each of the tasks. We have used Bi-directional LSTM for this purpose. We pass the tokens through an embedding layer to learn the embeddings. This is passed through two layers of Bidirectional LSTM. A feedforward layer is then used to produce the output. For sentiment analysis, feedforward layer produces 5 outputs corresponding to 5 classes. Paraphrase detection and semantic sentence similarity produces 1 output. For paraphrase detection and semantic sentence similarity problem we start with concatenation of 2 inputs. For sentiment analysis Cross entropy loss is used, for paraphrase analysis binary cross entropy loss is used and for semantic sentence similarity we use MSE on the produced output which is considered to be continuous prediction between 0-5. Figure 1 is an schematic diagram of the network.

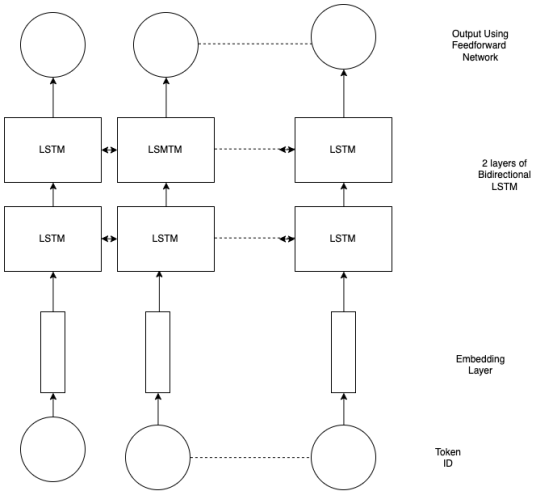

Figure 1: Baseline model with bi-directional LSTM

Next we observe effect of different hyperparameters on all the tasks. To do this we use simplest possible network. For sentiment classification we use a linear layer which has 5 outputs and loss of cross entropy is used. For paraphrase detection we first concatenate two embeddings and then use a linear layer with 1 output. Binary cross entropy is used as loss in this case. For semantic sentence similarity, we concatenate the embeddings from two sentences and then use a linear layer with 1

output. We consider the similarity to be a continuous metric and use a MSE loss. Figure 2 and Figure 3 show the two architectures.

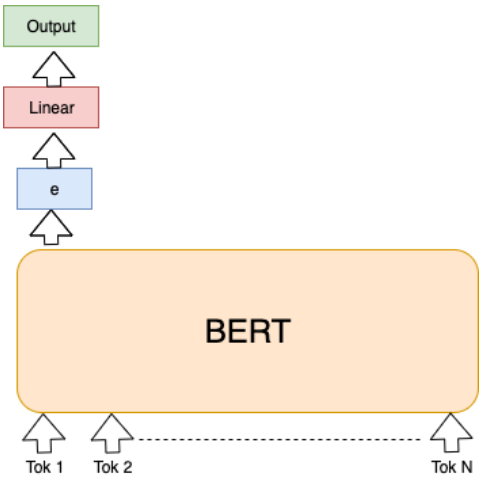

Figure 2: Base architecture for Sentiment classification

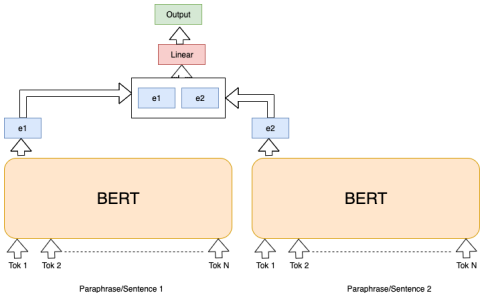

Figure 3: Base architecture for paraphrase detection and semantic sentence similarity

Semantic Textual similarity problem has unique evaluation in terms of correlation of predicted similarity score with ground truth similarity score. Pair of embedding were manipulated in different ways to obtain similarity score.

- **Concatenate and map to score**: Embeddings obtained from two sentences were concatenated. This was then passed through a linear layer to map to similarity score.
- **Take Multiply embeddings and map to score**: Embeddings were multiplied with each other (without addition) and then were mapped through a linear layer with similarity score
- **Take difference of embeddings and map to score**: Difference of embeddings was passed through a linear layer to map to similarity score.
- **Use scale cosine similarity**: Cosine similarity was evaluated between two embeddings. As the cosine similarity outputs between -1 to 1, it was scaled to 0 to 5 as (cosine_similarity + 1) * 2.5

A couple of optimization techniques were tried as well. AdamW is an optimization techniques which modifies Adam to include a weight decay term that reduces the magnitude of the weight values and helps prevent overfitting. Cosine Annealing is a technique which starts with a high learning rate, then reduces learning rate following a cosine curve to regualize the learning.

Once we had experimented with base tasks individually, we worked upon multi-task problem. We started with combining the loss function of all the tasks.

$$\mathcal{L} = \mathcal{L}_{sentiment} + \mathcal{L}_{paraphrase} + \mathcal{L}_{sts}$$

As the three datasets had different size, we used a round robin approach for training. For each epoch, first training was completed using data of sentiment analysis task. This was followed by training using paraphrase detection. Finally, Semantic textual similarity was used to propogate gradients through the network. As a modification to above network SMART loss was applied which added additional loss signifying different in prediction in neighborhood of a point.

5 Experiments

5.1 Data

As this is a multi-task problem there are 3 datasets involved:

1. **SST dataset** - The Stanford Sentiment Treebank consists of 11,855 single sentences from movie reviews extracted from movie reviews. The dataset was parsed with the Stanford parser and includes a total of 215,154 unique phrases from those parse trees, each annotated by 3 human judges. Each phrase has a label of negative, somewhat negative, neutral, somewhat positive, or positive.
2. **Quora Dataset** - 400,000 question pairs with labels indicating whether particular instances are paraphrases of one another.
3. **SemEval STS Benchmark Dataset** - 8,628 different sentence pairs of varying similarity on a scale from 0 (unrelated) to 5 (equivalent meaning). We have treated the similarity as a continuous variable in our experiments

5.2 Evaluation method

Following are individual evaluation metric for individual tasks:

1. SST dataset - Accuracy(primary), F1 of the multi-class classification
2. Quora Dataset - Accuracy of the binary classification
3. SemEval STS Benchmark Dataset - Pearson correlation of the true similarity values against the predicted similarity

For the multi-task problem the combined metric is average of the evaluation of three tasks.

5.3 Experimental details

We used dropout of 0.5, learning rate of 1e-5, finetuning all layers and 10 epochs as base configuration for all experiments. It was targeted that all 3 tasks should not take more than 30 minutes for initial experiments. As, paraphrase tasks had lots of data and was taking lots of time, it was sampled to make its size same as sentiment analysis problem. This help with quick iteration initially.We started with hyperparameter variation. Table 1 shows the hyper-parameter variation experimented for single task case.

Table 1: Hyperparameter variation

Hyperparameter	Base	Variation
Dropout	0.5	0.1, 0.3, 0.5
Learning Rate	1e-5	1e-4, 1e-5, 1e-6
Layer Freezing	Finetune all	Finetune all, last 2 finetune, last 1 finetune, pretrain
Epochs	10	10, 20, 30

In subsequent steps for multi-task learning we removed the sampling of paraphrase detection task and chose the most optimal hyperparameter setting.

5.4 Results

5.4.1 Baseline

We used a 2 layered Bi-directional LSTM to train all the 3 models. We used 10 epochs of training, learning rate of 10^{-5}, batch size of 64 and dropout of 0.5 after each of LSTM layers . Following table shows the performance for 3 problems

Table 2: Baseline for differnet Tasks

Task	Performance
Sentiment Analysis	25.34%
Paraphrase detection	37.5%
Semantic Textual Similarity	13.5%

We note that we trained the embedding layer. The performance of baseline could have been better if we had used a pretrained embedding layer instead of training embedding layer.

5.4.2 Hyperparameter variation on individual models

We varied dropout, learning rate, number of frozen layers and epochs. As a base dropout was kept at 0.5, learning rate at 1e-5, no frozen layer as base and epochs as 10. Table 3 Shows variation of tasks with dropout variation.

Table 3: Performance of different tasks with dropout variation

Task	Dropout=0.1	Dropout=0.3	Dropout=0.5
Sentiment Analysis	51.59%	**52.4%**	51.4%
Paraphrase detection	**76.32%**	76.02%	76.17%
Semantic Textual Similarity	**38.98%**	38.28%	36.74%

We don't see large variation in performance due to dropout in final layer.

Table 4 Shows variation of tasks with learning variation.

Table 4: Performance of different tasks with learning rate variation

Task	lr=1e-4	lr=1e-5	lr=1e-6
Sentiment Analysis	49.31%	**52.04%**	49.04%
Paraphrase detection	**78.83%**	75.31%	71.74%
Semantic Textual Similarity	37.96%	**39.15%**	33.95%

We observe that none of the tasks perform well with learning rate of 1e-6.

Table 5 Shows variation of tasks with epoch variation.

Table 5: Performance of different tasks with epoch variation

Task	epoch=10	epoch=20	epoch=30
Sentiment Analysis	52.4%	52.95%	**53.58%**
Paraphrase detection	75.18%	78.05%	**79.23%**
Semantic Textual Similarity	37.93%	38.34%	**38.55%**

We observe some benefit of training longer.

Table 6 Shows variation of tasks with varying number of freezing layers.

Table 6: Performance of different tasks with Number of freezing layers variation

Task	Fine Tune all	Fine Tune last 2	Fine Tune only last	PreTrain
Sentiment Analysis	51.68%	51.95%	**52.22%**	30.33%
Paraphrase detection	**76.18%**	74.21%	73.9%	62.84%
Semantic Textual Similarity	**38.34%**	38.08%	36.96%	17.58%

We observe a sharp reduction in performance of all tasks if we freeze all layers. In addition, we see similar or better performance if we fine tune only top 2 layers instead of all the layers.

5.4.3 Variation of fine tuning for Semantic Textual Similarity Task

We applied 4 kinds of similarity extraction for STS Task - concatenation, dot product, difference and cosine similarity as discussed in approach. Table 7 Shows variation of tasks with epoch variation.

Table 7: Performance of sts with variation in use of two embeddings

Fine-tune method	Performance
Concatenation and map	37.4%
Dot product and map	**38.93%**
Difference and map	3.69%
Scaled Cosine Similarity	38.76%

Dot product and map based method works best in this case.

5.4.4 Applying Pytorch AdamW and cosine Annealing

Table 8 Shows variation of tasks with epoch variation.

Table 8: Performance of different tasks with learning rate variation

Task	AdamW	CosineAnnealing
Sentiment Analysis	50.49%	52.04%
Paraphrase detection	75.64%	75.31%
Semantic Textual Similarity	38.16%	39.15%

We don't see improvement using AdamW and Cosine Annealing. These might be more useful with prolonged training as done in Llama paper.

5.4.5 Multi-task training

Now we apply round robin training of multi-task problem and its variation with smart. A major bump

Table 9: Performance of tasks with simultaneous optimization

Task	Additive Loss	Additive loss with SMART
Sentiment Analysis	49.4%	50.2%
Paraphrase detection	74.5%	73.62%
Semantic Textual Similarity	**46.7%**	39.4%

in performance of STS is seen which had lowest data among three providing evidence of benefit from other data sources. We didn't see performance bump from SMART regularization. Other tasks with larger dataset have shown slight degradation. This might be due to averaging of the three losses. One of the loss function got much importance then other.

6 Analysis

A major bump in performance of STS is seen which had lowest data among three providing evidence of benefit from other data sources We didn't see performance bump from SMART regularization Other tasks with larger dataset have shown slight degradation. This might be due to averaging of the three losses. One of the loss function got much importance then other Longer training provides some benefit across the tasks While fine-tuning all layers is preferred for STS and Paraphrase detection, we see the improvement from fine tuning last 2 is small. If BERT layers are not fine-tuned then there is large drop in performance Learning rate of 1e-5 worked best, but decrease in learning rate to 1e-6 leads to lower learning across tasks Lower dropout works better for two tasks. This is a bit counterintuitive as we see large overfitting. The train losses are almost always much higher than dev loss We don't see major benefits of optimization techniques such as AdamW and Cosine Annealing. These seem to be more appropriate in domain with higher data

7 Conclusion

In conclusion, our findings suggest that multi-task learning is a highly effective approach for improving the performance of NLP tasks. Our experiments demonstrated that training multiple tasks

simultaneously can lead to better results than training each task individually. Specifically, our results indicate that the benefits of multi-task learning are most pronounced for the task with the least amount of data. Additionally, our experiments also revealed that while varying hyperparameters for individual tasks did show some hyperparameters to be ineffective, we did not observe a significant improvement in overall performance across all tasks. This suggests that the benefits are not solely dependent on tuning hyperparameters tuning for each individual task. These findings have important implications for the design of NLP models and suggest that multi-task learning is a promising strategy for improving the accuracy and efficiency of natural language processing systems.

References

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

[2] 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. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2020.

[3] Asa Cooper Stickland and Iain Murray. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning, 2019.

[4] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.