Multitask Learning with BERT

Stanford CS224N {Default} Project

Sanjaye Elayattu
Department of Computer Science
Stanford University
sanjaye@stanford.edu

Abstract

The intent of this project is to develop and finetune a model that leverages the BERT architecture to perform three specific Natural Language Processing(NLP) tasks. The three tasks are Sentiment Analysis(SST), Paraphrase Detection(PARA) and Semantic Text Similarity(STS). In this project, I have created a multi task model that has some task specific layers, on top of a BERT model, to perform the three given NLP tasks. All the tasks share the base BERT model while having task specific output layers and corresponding loss functions.

Following are the key observations: i) To achieve improved performance on the NLP tasks, different, task specific networks are required. ii) The multiple NLP tasks also require their own loss functions based on the nature of the objective. iii) The model tends to over-fit on tasks that have smaller training data relative to the tasks having larger data set. iv) The multi-task model tends to perform better on some tasks than the others.

I tried several model architectures for the different tasks and selected the ones that provided best overall performance. For example sentence BERT representations(Reimers and Gurevych (2019)) improve the performance on the STS task. Paraphrase Detection tasks works better if use a neural network and binary classifier on top of the BERT model that take concatenated sentences as inputs. The sentiment classifer performed least among the three tasks as it contained less data and the nature of the task differed from the other two tasks.

For training the model, I experimented with round robin, square root and annealed sampling methods(Stickland and Murray (2019a)) and settled on square root sampling. To reduce over-fitting on SST and STS tasks, I used I used Fast Gradient Sign Method(FGSM)(Goodfellow et al. (2015)). I also implemented Multiple Negative Ranking Loss(MNRL) to reduce over fitting and improve sentiment classification accuracy. While I experimented with Gradient SurgeryYu et al. (2020) to adjust the diverging gradients from multiple tasks, I did not observe improvements in performance. Finally, I experimented with Projected Attention Layers(PAL)(Stickland and Murray (2019a)) and found out the additional observed that PAL did not significantly improve the accuracy, while it required an more than 6 million additional parameters.

1 Key Information to include

- Mentor: N/A
- External Collaborators (if you have any): N/A
- Sharing project: N/A

2 Introduction

The original BERTDevlin et al. (2019) model was created using transformer (Vaswani et al. (2023)) architecture with two objectives; (i) Masked Language Model(MLM) objective where the model learns prediction of randomly masked words in a sentence and (ii) Next Sentence Prediction(NSP) objective where the model predicts whether the second sentence provided to the model follows the first. BERT model uses a bidirectional encoder that leverages transformer architecture, to represent the input sentence.The representations provided by BERT model can be used for various NLP tasks.

The goal of multi-task learning is to use the same model to perform multiple tasks without having to create separate large language models for each task. Multi-task NLP models benefit from transfer learning and parameter sharing. In this scheme, a multi-task model is built using BERT as a base. These model benefits from transfer of knowledge due to training with multiple, related NLP training data and also share all the parameters of the base BERT. It is also efficient with respect to number of parameters when compared to single-task models.

In this project, I try to solve for 3 specific NLP tasks: SST, PARA and STS. The first step is to design the task specific additional layers (that run on top of the shared BERT model). The second step is to perform training the model on different tasks based on the provided training data. The key considerations are the handling of disparate training dataset volumes, identifying the right loss functions for the tasks and hyper parameter tuning.

In this project, more than ninety percent of the training data was for the paraphrase detection task. During training, the gradients from the different tasks can go in different directions which may cause model parameters to swing from one direction to another during training. This problem can be mitigated by a combination of interleaved training(Stickland and Murray (2019a)) and gradient surgery(Yu et al. (2020)). Over fitting is clearly another issue related multitask learning. In this project, the sentiment analysis task in particular was subject to over fitting. Regularization techniques such as dropout(Srivastava et al. (2014)), Weight decay and data augmentation techniques such as fast gradient sign method(Goodfellow et al. (2015) can be used to mitigate over-fitting issues.

3 Related Work

Singles Task NLP models using deep neural networks use a large number of parameters to perform specialized tasks. Stickland and Murray (2019b) discuss few options for multi task models that leverage transfer learning from different tasks. The paper discusses few architectural options such as task specific layers on top of BERT, additional BERT layer per task and Projected Attention Layers which add few projections within the stacked BERT layer. The new model is fine-tuned on specific tasks with the intent of getting comparable performance. Copper et al.Stickland and Murray (2019b) propose projected attention layers; a multi-task learning technique that reduces the number of additional parameters significantly, when compared with the parameter requirements for single task models.

Reimers and Gurevych (2019) show that sentence BERT (SBERT) where we take the mean pooling of the hidden layer to represent the whole sentence works Ill for sentence similarity tasks(STS). In this project, I used this approach to represent the sentences in the classification layers.

Over-fitting is a problem in multi task fine tuning. Goodfellow et al. (2015) propose introducing adversarial training data around decision boundary to reduce over fitting. I used Fast Gradient Sign Method for introducing adversarial data into training dataset. Stickland and Murray (2019a) propose multiple negative rank loss(MNRL) to improve accuracy on classification problems and reduce over fitting. I used this for training the SST task.

Yu et al. (2020) recommends gradient surgery, to manipulate diverging gradients from different tasks to improve the training speed as well as accuracy.

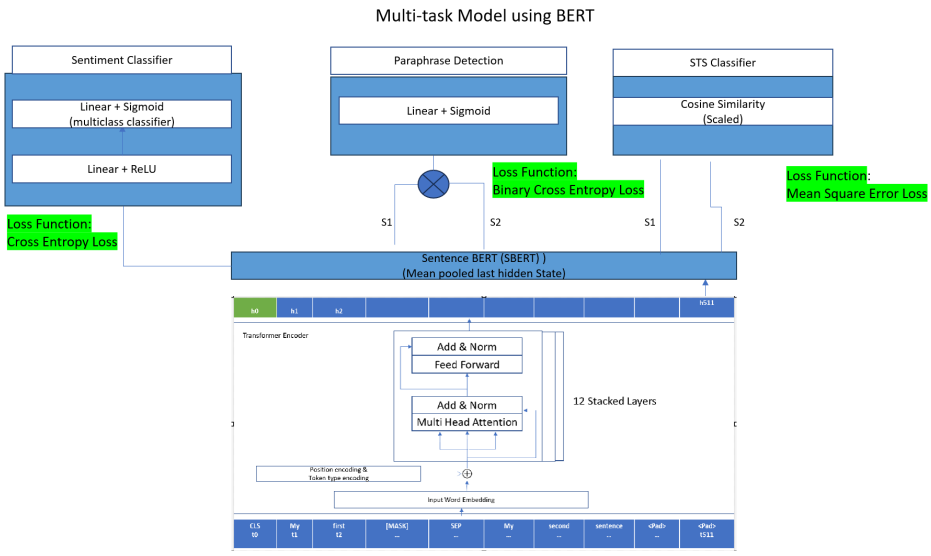

Figure 1: Multi Task model task specific classifiers sharing the BERT base model. (PAL layers are not shown as they Ire removed after some experimentation)

4 Approach

Figure 1 diagram shows the multi-task model that uses an implementation of BERT, I call it mini-BERT. The input is a total of 512 tokens including the CLS token that marks the start of the sentence and the SEP.

The embedding layer in the mini-BERT is a summation of word embedding (512 tokens), position embedding (512 positions and token type embedding(2 token types). The embedding dimension is 768.

Each BERT (Devlin et al. (2019)) Layer consists of multi head attention(Vaswani et al. (2023)) with a residual connection to the hidden inputs. This interim output is then fed into a feed forward network with a residual connection. Our mini-BERT model also uses 12 such BERT layers stacked on top of each other to complete the transformer encoder architecture. The output from the BERT model is fed into the task specific classifiers.

SBERT representation
For all the NLP tasks, I used SBERT representation as input for the sentences involved. In this implementation, I used mean pooling representation of the last hidden layer in BERT to represent the sentences. The sentence embedding is given by

$\mathbf{s}$. For example, mean pooling is represented as:

$$\mathbf{s} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{e}_i$$

4.1 Sentiment Classification

For Sentiment Classification (SST), I use the SBERT representation of the sentence as input to the task specific layer.The classification layer consists of a sequence of two layer multi-class classifier (Linear, ReLU,Linear and Sigmoid layers) . The prediction algorithm is shown below:

$$y = \text{argmax}\left(\text{softmax}\left(\mathbf{W}_2\max\left(0, \mathbf{W}_1\mathbf{x} + \mathbf{b}_1\right) + \mathbf{b}_2\right)\right)$$

- $\mathbf{x}$ is the embedding vector from SBERT. $\mathbf{W}_1, \mathbf{b}_1$ are the weights and biases of the first linear layer.$\mathbf{W}_2, \mathbf{b}_2$ are the weights and biases of the second linear layer, outputting class logits.
- softmax$(\cdot)$ converts logits to class probabilities.argmax$(\cdot)$ selects the class with the highest probability as the prediction.

I used cross entropy loss for this multi-class classification problem :

$$L = -\sum_{c=1}^{C} y_c \log(p_c)$$

- C is the number of classes.y_c is a binary indicator (0 or 1) if class label c is the correct classification for the observation.p_c is the predicted probability that the observation belongs to class c.

4.2 Paraphrase Detection

For paraphrase detection we first get the SBERT represetantion for the two sentences s1 and s2. The resulting two embedding are concatenated (2 * hidden size as the dimension).

$$\mathbf{y} = \sigma(\mathbf{z}) = \frac{1}{1 + e^{-\mathbf{z}}}$$

Where:

- $\mathbf{x}$ is the input vector to the linear layer.$\mathbf{W}$ is the weight matrix of the linear layer.$\mathbf{b}$ is the bias vector of the linear layer. $\mathbf{z}$ is the output of the linear transformation. $\mathbf{y}$ is the final output after applying the sigmoid activation function.

For paraphrase detection, I used binary cross entropy loss, which is defined as:

$$L = -\frac{1}{N} \sum_{i=1}^{N} \left[y_i \log(p_i) + (1 - y_i) \log(1 - p_i) \right]$$

- N is the number of observations in the dataset.y_i is the true label of the i-th sample, and can either be 0 or 1.p_i is the model's predicted probability that the i-th sample belongs to class 1.

4.3 Semantic Text Similarity

For Semantic Text Similarity, I used the SBERT representations for the two sentences (s1 and s2). The classification layer uses cosine similarity between the two SBERT sentence embedding. The result of the cosine simiarlity function (-1 and 1) are scaled to a numerical value between 0 and 5 as required by the training parameters.

The semantic text similarity task using SBERT embedding is computed as follows: Cosine similarity between the SBERT embedding of two sentences s_1 and s_2:

$$\text{cosine\_similarity}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \|\mathbf{v}\|}$$

Scale the cosine similarity result from the range of -1 to 1 to a range of 0 to 5 for compatibility with the training parameters:

$$\text{score} = 2.5 \times (1 + \text{cosine\_similarity}(\mathbf{u}, \mathbf{v}))$$

For this task, I used The Mean Squared Error (MSE) loss, which is given by:

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

where:

- N is the number of samples in the dataset. y_i is the actual value of the i-th sample. $\hat{y}_i$ is the predicted value by the model for the i-th sample.

This loss function quantifies the average squared discrepancies between predicted and actual outcomes, emphasizing larger errors more significantly than smaller ones due to the squaring of each error term.

5 Experiments

This section contains the following.

5.1 Data

I plan to use the provided data set for training. Specifically, I plan to use the SST data set for sentiment analysis, the Quora data set for paraphrase detection, and the SemEval data set for semantic textual analysis.

- The SST data set we have the following splits: train (8,544 examples), dev (1,101 examples) and test (2,210 examples).
- The Quora dataset has the following splits: train (283,010 examples), dev (40,429 examples) and test (80,859 examples)
- The semval STS dataset has the following splits: train (6,040 examples), dev (863 examples), test (1,725 examples).

5.2 Experimental details

In effort to match the best scores in the leader board, I ran around 15 experiments with a combinations of various model architectures, loss functions, hyper parameters and training methods. The key experiments are listed below

1. Single Task Training: The baseline working model used BERT pooler output for all the 3 tasks. the SST task used a multi-class classifier that used cross entropy loss. The PARA task used concatenated embedding for the two input sentences into a binary classifier with binary cross entropy loss. The STS classifier also used a concatenated embedding along with a scaled single layer perceptron. The entire model was fine tuned on one task at a time and we measured task level and overall accuracy in the dev dataset. I observed overall DEV accuracy in the range of 0.48,0.55 and 0.42 respectively for the SST,PARA and STS single task tuning. The STS task performed the worst on the model trained just on the other two tasks. The learning rate was left at 1e-05 and weight decay was 0 during these tests.

2. Baseline Multi Task Round Robin Fine Tuning: The model architectures remained same as the previous section. However, we trained all three datasets during each epochs and experimented with global (parameters are updated after one batch of all 3 tasks are trained) and local updates (parameters are updated per task per batch within an epoch).The other hyper parameters remained the same. This provided a baseline model with an overall accuracy of 0.53 for global update and 0.57 for local updates with STS task continuing to lag behind other tasks.

3. Multi Task Round Robin Fine Tuning with Siamese BERT for STS: I modified the classifier for Siamese to use cosine similarity as an objective function with no other changes to the remaining classifiers. The training method was similar to the previous section and parameters were updated locally for each task within an epoch. The dev accuracy improved to 0.62. I implemented gradient surgery Yu et al. (2020) to adjust the diverging gradients and noticed that this did not improve accuracy on the DEV data sets. The grad surgery also required global updates (one update after all three batches are trained for one batch in an epoch) since all three gradients are calculated and adjusted before the update is made.

Experiment	Overall	SST	PARA	STS
1) 3 Single Task Tuning*	0.55	0.48	0.653	0.25
2) Baseline: Multi Task Round Robin	0.57	0.467	0.653	0.163
3) Baseline With Siamese BERT - Cosine Similarity	0.62	0.448	0.676	0.501
4) SBERT inputs for all	0.71	0.494	0.712	0.786
5) With FGSM and MNRL loss	0.73	0.514	0.778	0.767
6) With PAL	0.71	0.489	0.766	0.723

Table 1: TEST and DEV Leaderboard final results. The results of single tasks training are not comparable with the rest of the experiments because the results are tabulated based on best result for a specific task when the model was trained for that task

The above configuration was repeated during increased number of epochs (10,20,50). Increasing number of epochs did not seem to improve the dev accuracy as the SST and STS tasks already had training accuracies over 0.9. Lower learning rate to 1e-06 and adding the weight decay of 0.1 slowed down the training speed and it did not seem to increase dev accuracy. The rest of the experiments were performed with a learning of 1e-05 and a weight decay of 0.01

4. Sentence BERT : In reviewing these two tasks in more detail, I used SBERT representation as an input to the STS tasks. SBERT representations used mean-pooling of the BERT embedding. The classifiers and loss functions remained the same. I noticed an increase in dev accuracy to 0.66 with this configuration. This change positively influenced the STS task more than the PARA and SST tasks; However I continued to use SBERT inputs for all these tasks since it did not have a negative impact on SST or PARA task.

5. MNRL loss for SST task: After noticing overfitting on the SST task, I implemented Multiple Negative Rank Loss (MNRL) for the SST task. A weighted (a hyper parameter witha default value of 0.5, used in the experiments) MNRL loss was added to the cross entropy loss as part of the training process. This process seemed to improve the SST task accuracy to 0/514 (from 0.494).
   Sampling Methods (Round Robin, Square Root and Annealed): In the training data, the PARA task contained considerably more data compared to the SST and STS training data. Even though I implemented annealed sampling (Stickland and Murray (2019a)), I noticed that the squared root sampling (sampling the data set using a probability proportional to the square root of the training data size for the specific task) was sufficient. I used square root sampling for this purpose.

6. The final experiment in this project was done by adding 3 PAL layers(Stickland and Murray (2019a)) within the BERT layer. This increased the total number of model parameters from 110 million to 116 million, even though the accuracy stay around 0.69.
   The following table shows the summary of the experiments and corresponding accuracy scores recorded

5.3 Results

- The following table shows the results of various experiments described in the previous section
- The following table shows the final score in the DEV and TEST leaderboards

6 Analysis

The mini-BERT model designed here works better with the the paraphrase detection and semantic text similarity tasks while it lags behind in the sentiment classification tasks. In the sentence " "if

Leaderboard	Model*	Overall	SST	PARA	STS
TEST	5	0.725	0.524	0.777	0.748
DEV	5	0.725	0.514	0.778	0.767

Table 2: TEST and DEV Leaderboard final results. Model number corresponds to the experiment numbers listed in Table 1

your taste runs into difficult films, you cannot absolutely miss this one", the model predicted a value 4 (it should have been 3). It appears that the model has difficulty in handling double negative language with a sarcastic tone, while it correctly classifies some straight forward sentences like "The best film about baseball to hit theatres since Field of Dreams". This could be because of the lack of enough training data for sentiment analysis compared to the other two tasks.

For Parapharse detection and Semantic Text similarity tasks are similar in certain way as they both depend on fully sentence representation of two BERT embeddings. The STS task, has comparable volume of training data as that of SST; However, it seems to benefit from learning related to paraphrase detection and performs very similar to paraphrase detection task.

The base BERT model used in the project has around 109.5 million parameters. Single task models would require similar number of parameters for each task. With the PAL implementation, explained in experiment 6, we would have 116.4 million total parameters). The final submitted model here, without the PAL contains approximately 111 million additional parameters.

7 Conclusion

Summarize the main findings of your project and what you have learned. Highlight your achievements, and note the primary limitations of your work. If you'd like, you can describe avenues for future work.

The mini-BERT model model performs fairly well on the three NLP tasks. I started with the implementation of the basic BERT model and built the task specific layers step by step on top of the BERT model. Development of this model required an iterative process with respect to model architecture, hyper parameter choices and training method. After running a series of 15 experiments, (six of them are listed in the experiments section of this paper) I observed the following

- The multiple tasks often come with different types of training data as we found in the training data sets of SST, PARA and STS tasks
- The training set data volume impacts the training process. In this scenarios, more than 90 percent of the total training data belonged to PARA task which appeared to cause overfitting issues for the SST task. In addition, I also wonder if it helped the STS (which had a small training data set task like SST) since the STS and PARA tasks are similar in nature.
- We would need task specific additional layers on top of BERT. The choice of these layer architectures and their loss functions are important
- Disparate training data volume among the NLP tasks may cause overfitting in some tasks. Few options to mitigate overfitting are dropout, adversarial training data injection (I used FGSM for this), MNRL loss are some of the options to reduce overfitting. During training, square root or annealed sampling can be used to balance the training iterations for various tasks based on their data volume.
- Gradient surgery can be used to align diverging gradients during the training. While I did not find it improving training accuracy in my experiments, I expect it to be useful in more divergent tasks involving longer iterations.
- PAL is a better option compared to single task training and task specific models. The multi-task model built here performed better without the PAL than with the PAL

Appendix A shows the list of options and hyper parameters that I used. It would have been interesting to train and test the model with the all combinations of hyper parameters and model configurations to arrive at an even better model. I also believe that the SST task would have benefited from another dataset such as restaurant reviews or business reviews.

8 Ethics Statement

Before deploying a model like this into the real world, several ethical and safety factors needs to be considered. We need to clearly define the use cases as in-scope and out-of-scope. The model should not only perform for the in-scope scenarios, but it should also be vetted on the out-of-scope scenarios. The experiments and results shown in this paper only addresses the in-scope use cases. It does not show the out-of-scope scenarios. If this model is deployed in a way where it is subject to out-of-scope scenarios, the model should respond by either avoiding the question or answer it in a neutral/safer way. The models must be evaluated for bias and fairness as well. For example, we need to test whether the model is biased in performing it tasks towards a specific gender,race, ethnicity etc.

To make sure that the NLP models are delivered for safe and ethical use, we need to incorporate Measurement and Mitigation strategies at every stage of the model creation even though it may be challenging to measure safety issues at the pretraining level rather than the finetuning level. There are several methods currently used to mitigate issues related to fairness and bias in NLP modelling. One of them is to inject the questions / prompts with more information (such as demographic/gender). Training the model with more diverse data is another option.

In the review of a case study, the speaker explained how I used adversarial data injection technique in my training to solve over fitting problem. The same technique of adding right type of perturbed data into the training can reduce the fairness and bias related issues with NLP.

Testing of NLP models for ethical and responsible behavior continues to be challenging and is an ongoing field of research. As in any risk management scenario, we can only mitigate what we can measure. Scoping the model based on the intended use and having a robust mechanism for out-of-scope use is important. It is also important to incorporate risk mitigation strategies such as removal/altering of questionable training data, injecting the inputs with more information to reduce bias, and measuring the output for bias and fairness tests

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

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

Asa Cooper Stickland and Iain Murray. 2019b. BERT and pals: Projected attention layers for efficient adaptation in multi-task learning. *CoRR*, abs/1902.02671.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention is all you need.

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

A Appendix (optional)

The following table shows the list of hyper parameters and configuration parameters used in the program.

1. Batch size parameters : I used 3 separate batch size parameters for the tasks to have finer control of batch size during training. I settled on using a batch size of 32 for all the three tasks after switching the training to annealed sampling
2. Learning Rate: I used two learning rates 1e-05 and 1e-06 and weight decays of 0 and 0.01 for various experiments. I ended up using learning rate of 1e-05 and weight decay of 0.01 at the end
3. grad surgery: This parameter dictates whether gradient surgery is enabled or not. If this is enabled, gradient surgery will be performed using the PCCGrad algorithm provided in that paper.
4. fgsm process flag and fgsm eps. These flags enable the introduction of adversarial data into the training process using Fast Gradient Sign Method (FGSM). The fgsm-eps flag controls the amount by which the training data would be perturbed. The default value is 1e-05
5. mnrl process and weight flag. These parameters enable the Multiple Negative Rank Loss(MNRL) for the sentiment task training. The weight (para-wt) determines how much of the calculated MNRL loss will be added to the cross entropy loss for the task. The default value is 0.5 for the MNRL weight
6. anneal flag. If this flag is enabled, training will use annealed sampling. If this is disabled, square root sampling is used. This can be changed to cube root or some other function based on another parameter called sample-wt (which is defaulted to 0.5 for square root sampling)
7. PAL is implemented in bert-PAL.py.