Multi-task BERT Classification

Stanford CS224N Default Project
Mentor: Xiaoyuan Ni

Shen Gao
Department of Computer Science
Stanford University
shengao@stanford.edu

Abstract

This paper explores strategies training BERT model for multi-task text classification tasks. We examine in-domain multi-task fine-tuning, gradient surgery and multiple negative loss as optimization strategies in addition to a base multi-task BERT model. We show that these techniques each contribute incrementally to heightened performance.

1 Introduction

Multitask learning is a promising approach to improve the performance of natural language processing (NLP) models on various tasks. The use of pre-trained models like BERT has shown significant improvement in various NLP tasks. In this context, multitask learning can further improve the performance by training models to perform multiple tasks simultaneously. This paper discusses the problem of multitask classification learning using BERT for three tasks, sentiment analysis, paraphrase identification, and semantic similarity. These tasks are challenging due to the complexity of natural language and the need to capture subtle nuances of semantic meaning. The paper explores how single-task BERT models may perform well on the task-specific data set but falls short on multi-task metrics. We propose a multi-task framework and a custom multiple negatives loss criterion for enhanced performance on multi-task objectives.

2 Related Work

2.1 BERT Finetuning for text classification

In Sun et al. (2019), the authors showed additional pre-training on BERT's original objectives, fine-tuning model directly on a single task and multi-task training on BERT all contribute to significant improvement on multi-task text classification performance. The motivation behind is the difference in data distribution between the data BERT is trained on and the data in the downstream tasks. These additional pretraining/finetuning warms up BERT parameters to the downstream task data distribution and provides better convergence as a result.

2.2 Gradient surgery for multi-task learning

Gradient directions of different tasks may conflict with one another. Yu et al. (2020) recommended a technique called Gradient Surgery that projects the gradient onto the normal plane of another when the two gradients are in opposite directions. The outcome of this method ensures pairwise gradients to always be less than or equal to 90, preventing gradient to descend into a particular task too far and unable to backtrack.

2.3 Multiple Negatives Ranking Loss Learning

Another effective way of improving embeddings would be to fine-tune the model with multiple negative ranking loss. For tasks where inputs are pairs of sentences, the training data consists of sets of K sentence pairs [(a_1, b_1), . . . ,(a_n, b_n)] where a_i, b_i are labeled as similar sentences and all (a_i, b_i) where $i \neq j$ are not similar sentences. Henderson et al. (2017) proposes a loss function that minimizes the distance between a_i, b_i while it simultaneously maximizing the distance (a_i, b_j). This loss function effectively include soft negatives (inferred from input data) to the learning objective making it magnitudes more efficient at convergence.

3 Approach

3.1 BERT Architecture

The BERT model uses scaled dot-product multi-head attention as described in the *default final project BERT handout* and *Attention is All You Need* paper, as outlined in Figure 1.

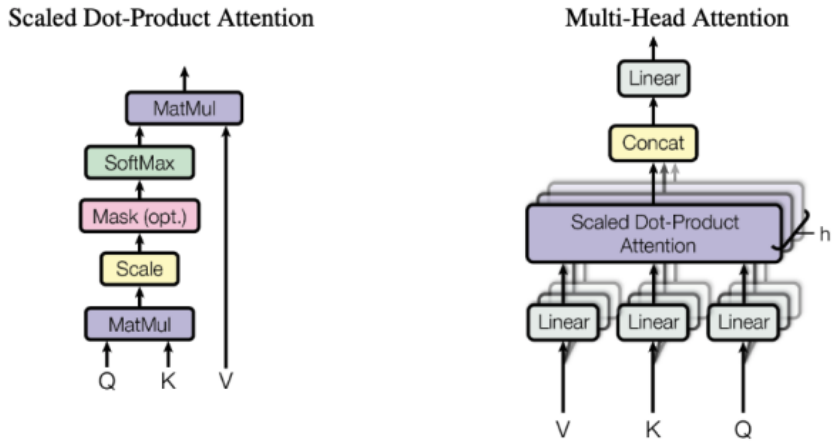

Figure 1: Multi-head attention architecture, sourced from the project handout.

The BERT model first computes the dot products of the query and key matrix, denoted Q and K, scaled by the key dimension, d_k, before applying a softmax function to obtain attention weights on the value, denoted V. Scaled dot-product attention is computed as:

$$Attention(Q, K, V) = Softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Multi-head attention improves upon single-head attention on allowing BERT to attend over subspaces at different positions. Multi-head attention is computed as:

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O$$
$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

where W_i^Q, W_i^K, W_i^V are learnable linear projections over Q, K, V.

In addition to the attention layer, each transformer layer also includes

1. An add-norm layer on the inputs and outputs of the attention layer
2. A feed forward layer on the outputs of the add-norm layer
3. A final add-norm layer on the inputs and outputs of the feed forward layer

The add-norm block in each transformer layer includes a residual connection, dropout and layer normalization for a more stabilized parameter update, robustness towards over-fitting, and a smooth parameter space for easier gradient descent. This structure is formulated as follows:

$$LayerNomalization(Dropout(Linear(output)) + input)$$

A word-embedding layer and a position-embedding layer are used for representing word tokens and positions of each token. They each map token ids and position ids to the corresponding embedding representation. The aggregate embedding layer is the sum of the two types of embeddings with dropout and layer normalization for robustness.

The BERT encoder consists of 12 BERT layers. Its output includes:

1. The last hidden state from the top BERT layer, representing the contextualized embedding for each token.
2. The pooler output, the embedding of the [CLS] token representing the class of each sequence.

3.2 AdamW Optimizer

We implemented the AdamW Optimizer as outlined in the default project. Note we implemented the decoupled weight decay update as follows: Below we describe the completed algorithm pseudo code:
Requires: : step size
Requires: β_1, β_2: exponential decay rates for moment estimates
Requires: $f(\theta)$: stochastic objective function with parameters θ
Requires: η: weight decay factor
Initialize: θ_0, m_0, v_0, t_0, where m, v are the exponentially weighted first and second moment of the gradient g.
In each gradient descent step, do

$$t = t + 1$$
$$g_t = \nabla f_t(\theta_{t-1})$$
$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$
$$\alpha_t = \alpha \sqrt{1 - \beta_2^t} / (1 - \beta_1^t)$$
$$\theta_t = \theta_{t-1} - \alpha_t m_t / (\sqrt{v_t} + \epsilon)$$
$$\theta_t = \theta_{t-1} - \eta \alpha \theta_{t-1}$$

3.3 Downstream Classification Tasks with BERT

For a base multi-task model, in addition to the shared BERT model, each task is furthered trained on layers unique to the task as follows.

Sentiment Analysis
We take the pooler output from the BERT model and pass it through a dropout layer then a linear layer with output channel 5. We then apply a softmax function to product probability logit as output. For training, we use cross entropy loss on the probability logits.

Paraphrase Detection
We concatenate the pair of sentences into a single sequence before passing it into BERT to obtain the pooler output. The output is then passed through a dropout and a dense layer to produce a single logit. For training, we apply sigmoid to the logit then use binary cross entropy loss as loss criterion.

Text Similarity
We concatenate the pair of sentences into a single sequence before passing it into BERT to obtain the pooler output. The output is then passed through a dropout and linear layer with output channel 6, before applying a softmax function to produce probability logits as output. For training, cross entropy loss is used as loss criterion.

In this base multi-task model setup, we train the model on all of the data in one dataset at a time in a sequential manner. We experiment a round robin style training pattern in the Experiments section.

3.4 Baseline

We establish four baselines in this project, one for each model trained solely on one task-specific dataset and a fourth baseline trained on all three datasets. We use our base multi-task model as described in the Section 3.3 for this purposes. The baseline metrics are as follows.

Table 1: Single-task baseline and Multi-task baseline

Baseline	Sentiment ACC	Paraphrase ACC	Similarity CORR	Avg Score
SST only	**0.52**	0.53	-0.09	0.322
Paraphrase only	0.154	**0.829**	0.143	0.375
STS only	0.128	0.605	**0.558**	0.43
Base all data	0.48	0.88	0.35	**0.576**

4 Experiments

4.1 Data

We leverage the datasets provided for the default project for training and testing purposes.

Stanford Sementiment Treebank (SST) dataset
We use the Stanford Sentiment Treebank (SST) for the sentiment task. The dataset consists of 17,086 training examples, 1,101 validation examples and 2,210 test examples. Each data includes one sentence extracted from movie reviews with a 5-class categorical label of negative (0), somewhat negative (1), neutral (2), somewhat positive (3) and positive (4).

Quora dataset
We use the Quora for the sentiment task. The dataset consists of 141,498 training examples, 20,212 validation examples and 40,431 test examples. Each data includes a pair of sentences with binary label indicating whether the pair is paraphrase (1) or not (0).

SemEval STS Benchmark (STS) dataset
We use the Stanford Sentiment Treebank (STS) for the semantic textual similarity task. The dataset consists of 6,040 training examples, 863 validation examples and 1,726 test examples. Each data includes a pair of sentences labeled on a scale from 0 (unrelated) to 5 (equivalent) representing their level of semantic similarity.

4.2 Evaluation method

We use the default evaluation method as provided in the default project, namely accuracy on the sentiment analysis and paraphrase detection tasks and Pearson correlation score on the semantic similarity task.

4.3 Experimental details

Hypterparameter Tuning
Through iterative experimentation, we find our model most sensitive to learning rate (LR) among other hyperparameters. We ran the following experiments using our best model as outlined in the next sections over 1 epoch and concluded 1e-5 to be the best learning rate.

Table 2: Hyperparameter tuning

Baseline	Sentiment ACC	Paraphrase ACC	Similarity CORR	Avg Score
LR=8e-6	0.463	0.792	0.207	0.487
LR=1e-5	0.431	0.878	0.533	**0.614**
LR=2e-5	0.338	0.875	0.528	0.58

Due to the virtual machine's memory limitations, we could not run experiments on batch size beyond 8; In our experiments, we found using the default dropout probability of 0.3 produces the best results so we kept it the same.

Extension to the base model
Since the focus of the project is on multi-task learning with pretrained BERT model, we are inter-

ested in exploring methods that iteratively increases our multi-task model performance especially benchmarking against single-task performance.

1. In-domain training

In the paper Sun et al. (2019), the authors presented in-domain training to be beneficial in tuning the BERT model. We attempted to use a similar approach in our three-task problem space. Since data is structurally similar between the paraphrase and STS task and the paraphrase (Quora) dataset is significantly larger than the STS data set, we think in-domain training could be most helpful with the STS task. We did not consider in-domain training for paraphrase since our model is already achieving 80+ accuracy on this task.

We noticed while in-domain improved learning on STS, Sentiment suffered a 10%+ degradation in performance. We think this is due to interference between the tasks. Specifically, the optimizer descents in completely opposite directions for Sentiment and STS in certain steps. The order and the amount of training data also matter in the case of interference. Since we use many more data for STS, we think the model progresses much more in the direction of STS. Since we train on all data in a task before training on a different task (sequential data pattern), we think this also causes issues when gradients descends too far in the direction of a particular task, make it impossible to reverse course when training the next data set.

2. Gradient surgery and round robin training

To alleviate the interference problem, we used the gradient surgery technique as proposed and implemented by Yu et al. (2020). The idea is that when the gradient of the i-th task g_i is in the opposite direction that of the j-th task g_j, g_i is modified as its projection onto g_j's normal plane as follows.

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

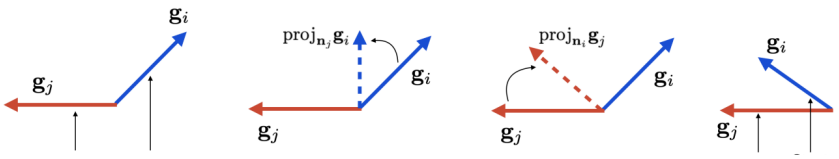

Figure 2: Gradient Surgery scenarios as illustrated in Yu et al. (2020)

Unlike the previous model where we only track the summed loss across three tasks, we first preserve losses for each task. When the angle of two task gradients are more than 90 degrees, we project one gradient to the normal plan of the other. This ensures gradient descent direction is less than 90 degrees with respect to each task-specific gradient.

In order to fit this pattern, we also updated our data ingestion from sequential to a round robin pattern where we train on a batch of all three datasets in each step. Compared to sequential training where the training loss descends continuously towards the space of a single task data set before switching to a different dataset, this pattern enables gradient surgery to modify each task's gradients in each training step, preventing loss to descend into task-specific space prematurely causing interference

We obtained a new high using this approach. Note that we are able to improve on paraphrase and STS but suffered significantly less interference from before.

3. Multiple negative ranking loss / contrastive learning

We had attempted to implement the original multiple negative ranking loss algorithm as proposed by Henderson et al. (2017), but our implementation performed poorly compared to our results up to this point so we implemented an original variant of the algorithm. Compared to Henderson et al. (2017) approach, our implementation doesn't have a concept of ranking negative losses but instead treating all negative cases the same level in terms of priority. In retrospect, this is closer to a form of contrastive learning.

For the sentences pairs used in Quora and STS data sets, we first compute a pairwise cosine similarities matrix, M, from the embedding matrix from the two sentences, denoted A and B.

$$M = \frac{AB^T}{max(\sqrt{(A \odot A)(B \odot B)^T} + \epsilon)}$$

whereas ϵ is a small value 1e-10 for numerical stability and ensure denominator is non-zero.

We then construct a contrastive example matrix from the original label by inflating the labels as the diagonal of a matrix N. The off-diagonal values of N are all zeroes. The (i,j) element of the matrix where $i \neq j$ represent the negative examples assuming the i-th first sentence have no similarity to the j-th second sentence. The (i,i) diagonal element represents the similarity of the original first and second sentence pair.

We then use an MSE loss criterion on the two matrices and reduces the losses to its mean.

$$J(\theta) = \frac{1}{b^2} \sum_{i,j} ||M(i,j) - N(i,j)||^2$$

where b represents the batch size, and both M and N have size b x b.

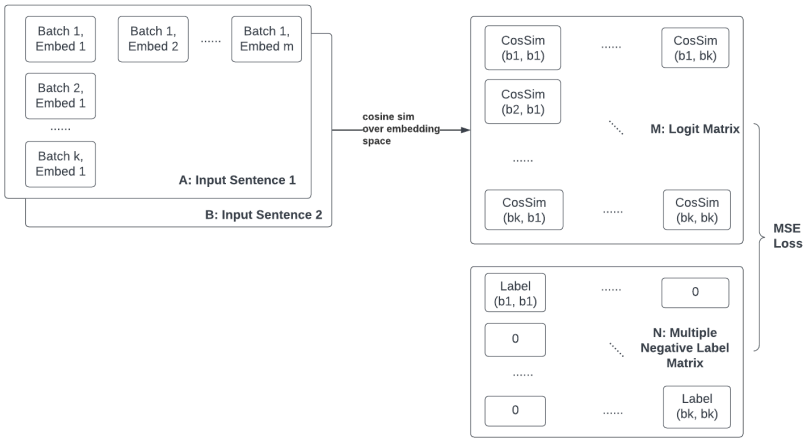

Figure 3: Novel multiple negative loss implementation

The motivation of this construction is that for any off-diagonal sentence i and j they should be completely uncorrelated and therefore implying zero similarity, significantly increasing the number of training examples used in a batch from batch size to batch size squared. The idea of using mismatching sentences within batch is inspired by and consistent with the Henderson et al. (2017) paper. We are able to achieve a new best using our implementation.

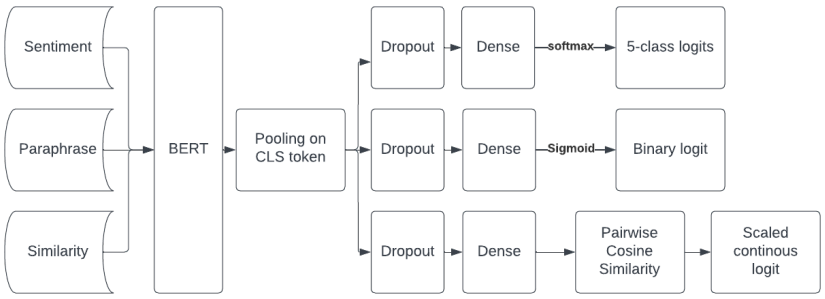

Figure 4: Model architecture (feed forward layers) after improvements

4. Data Augmentation with Back Translation

We also attempted to use back translation to augment training data. We used pretrained sentence transformers as provided by the nlpaug library to translate STS training data set from English to

French then translate it back to English. We noticed some back translations being identical to the original text therefore removing them from the augmented data set to overfitting on repeated examples.

5. Diagnosing model against STS state-of-the-art performance
SemEval maintains a leaderboard on which the SOTA results are in excess of 0.85 whereas our model best out max out at 0.579. This is a large difference considering performance against the other tasks are within 0.15 from SOTA. We attempted to resolve this issue by first increase model complexity with more layers and secondly increase dropout probability to reduce overfitting. We had attempted adding up to four MLP layers and/or the add norm layer we implemented inside BERT with residual connection. Adding layers increased our model's training accuracy up to 0.75 but made our model overfit even more (larger differences between training and dev losses). Overall we were not able to improve dev accuracy with these techniques.

4.4 Results

We report the following experiment results and submitted the multiple negative loss results to the test leaderboard.

Table 3: Experiments on model architecture

Experiments	Sentiment ACC	Paraphrase ACC	Similarity CORR	Avg Score
In-domain Training	0.336	0.845	0.43	0.537
Gradient Surgery	0.466	0.889	0.389	0.581
Multiple Negative Loss	0.474	0.886	0.579	**0.647**

5 Analysis

We focus the analysis effort solely on understanding why our STS model lags far behind SOTA. See prior discussions on experiments done in this effort. In Figure 2 below we show a scatter plot with predicted value versus true label from the STS data set. We notice the predicted values did not predict any values below 2. This is an intriguing result as there existed predictions below 2 in the first few epochs. The model somehow generalizes to only fit models of higher values. We think one potential reason is related to the negative examples during the in-domain training using the Quora data set to train STS objective. The negative Quora examples indicate two sentences are not paraphrase but in reality some sentences not completely dissimilar therefore making the model harder to generalize on low similarity examples. Another possible cause of this underperformance might be due to lack of cross attention between the two input sentences as we pass the two sentences separately into BERT and only allow interaction between the two sentences in the shallow fine-tune layers.

6 Conclusion

In this project, we were able to build a multi-task text classification model using BERT and iteratively improve upon it. We achieve an average of 0.647 across the three tasks. The accuracy metrics on sentiment and paraphrase tasks are within 15% of SOTA at the time of writing. Though our Pearson score on the similarity task lags more, we did gain experience analyzing a bias/variance model and contemplate resolutions for them.

Lastly, we'd like to thank the course staff for offering students new to NLP/deep learning the default project with thoughtful guidance on the base BERT model and potential extension ideas for a great learning experience.

References

Matthew Henderson, Rami Al-Rfou, Brian Strope, Yun hsuan Sung, Laszlo Lukacs, Ruiqi Guo, Sanjiv Kumar, Balint Miklos, and Ray Kurzweil. 2017. Efficient natural language response suggestion for smart reply.

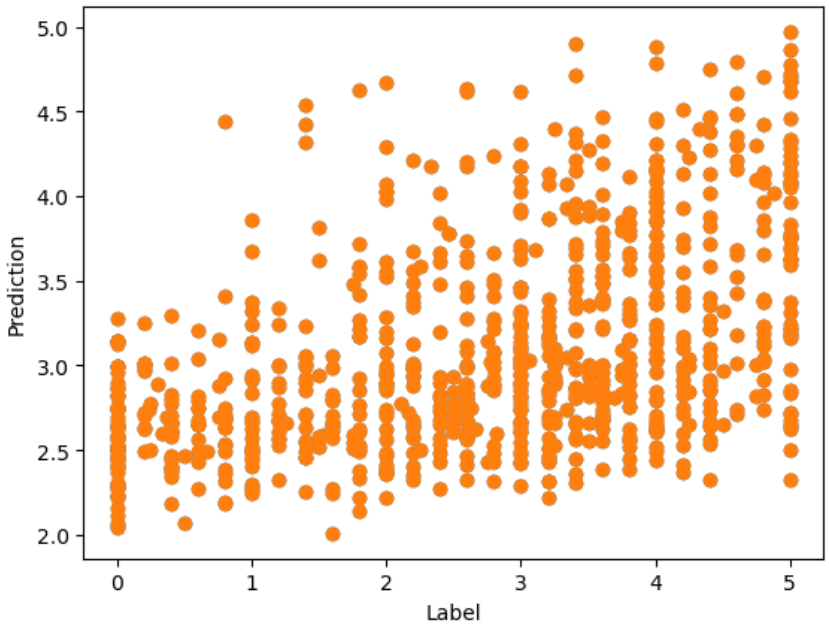

Figure 5: Multi-head attention architecture, sourced from the project handout.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune bert for text classification? In *Chinese Computational Linguistics: 18th China National Conference, CCL 2019, Kunming, China, October 18-20, 2019, Proceedings 18, pages 194-206*. Springer.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient surgery for multi-task learning. *arXiv preprint arXiv:2001.06782*.