minBERT for Sentiment Analysis, Paraphrase Detection, and Semantic Textual Similarity

Stanford CS224N Default Project
Mentor: Shai Limonchik

Shelly Goel
Department of Computer Science
Stanford University
shelly23@stanford.edu

Haya Hidayatullah
Department of Statistics
Stanford University
hayah@stanford.edu

Yoko Nagafuchi
Institute for Computational and Mathematical Engineering
Stanford University
yokongf@stanford.edu

Abstract

In this project, we focus on improving the performance and generalization of natural language processing (NLP) tasks such as sentiment analysis (SST), paraphrase detection, and semantic textual similarity (STS) by implementing a multitask-based model. We proposed using a BERT-based architecture called minBERT adapted for multitask learning and experimented with various additional model architectures, pre-training, and fine-tuning techniques to improve the performance of the baseline model. We found that using Cosine similarity (Cos) and increasing the model size through hidden MLP layers and ReLU activation (ReLU) were particularly effective for the STS and Paraphrase tasks, respectively. We also used GradNorm and PCGrad to balance gradients across diverse tasks and address conflicting gradients between tasks in multitask learning. Additionally, we employed data engineering techniques (such as looping over smaller datasets (Loop) and normalizing labels) and used ensembling to determine the most optimal model architecture. The best overall model incorporated the Cos, ReLU, and Loop extensions, improving the baseline average score by 0.13 on the Dev set and significantly improving the task-wise scores for Paraphrase and STS by 0.22 and 0.17, respectively. Our results demonstrated that a BERT-based multi-task learning architecture, combined with various pre-training, fine-tuning, and ensembling techniques, can significantly improve the performance of NLP tasks such as our three downstream tasks.

1 Introduction

In the field of NLP, there are various tasks such as sentiment analysis, paraphrase detection, and semantic textual similarity which help us understand the nuances of language and semantic relationships better. These tasks can be challenging as they require an understanding of the underlying relationships of different pieces of text with each other, which can be very context-dependent, have complex meanings, or be ambiguous. The goal of our project was to implement a multi-task-based model to improve performance and generalize well across all three tasks as we recognize that all three tasks are inherently related to each other in how we understand language and the relationships that texts have with each other semantically. Many existing methods for these tasks rely on separate models that perform each task independently. However, we believe that training a model that learns a shared encoding can help improve the performance of each task and also generalize well across all three tasks.

We pre-trained the model using masked language modeling on the SST dataset and fine-tuned it with Cosine similarity to capture the semantic similarity between similar sentence pairs. We used both GradNorm and PCGrad to balance gradients across diverse tasks and address conflicting gradients between tasks in multitask learning. We also modified the number of MLP hidden layers and ReLU activations, employed data engineering techniques, and used ensembling to determine the most optimal model architecture. Our best overall model incorporated the Cos, ReLU, and Loop extensions, improving the baseline average score by 0.13 on the Dev set and significantly improving the task-wise scores for Paraphrase and STS by 0.22 and 0.17, respectively.

Our project aims to contribute to the development of more effective NLP models that can better understand the semantics of the language by making use of shared relationships between similar NLP tasks.

2 Related Work

Our project aims to refine BERT embeddings for multiple downstream tasks using multi-task learning and gradient optimization techniques. Several recent papers have explored similar approaches and have shown improvements in model performance.

The 'MTRec: Multi-task Learning over BERT for News Recommendation' (Bi et al. (2022)) introduced a novel multi-task learning framework that incorporates downstream tasks as auxiliary tasks. This approach results in deeper and more representative BERT embeddings, leading to improved performance over baseline single-task learning. The paper also successfully utilizes the Gradient Surgery (Tianhe Yu (2020)) technique to resolve conflicting gradients and further improve model performance. The 2019 state-of-the-art model on 10 NLU tasks Liu et al. (2019) also uses multi-task learning to improve performance across tasks and text data in different domains. This paper shows the potential of combining language pre-training and multi-task learning to overcome major challenges in NLP, such as language representation and generalization across tasks.

Other papers combine gradient optimization techniques with multi-task learning to improve overall model performance; GradNorm (Zhao Chen (2018)) normalizes gradient magnitudes and task learning rates whereas Gradient Surgery (Tianhe Yu (2020)) resolves conflicting gradients between tasks.

For our specific downstream tasks, various papers suggest extensions that improve single-task performance as well. The original BERT paper showed how additional pretraining helps in fine-tuning for text classification problems (Devlin et al. (2018)). Another paper on single task finetuning (Nils Reimers (2019)) show how Cosine similarity improves task performance for the semantic textual similarity task.

Overall, these papers highlight the potential of multi-task learning, along with ensemble mulitask models combining gradient optimization techniques and single task finetuning techniques to improve model performance and refine BERT embeddings.

3 Approach

3.1 Tasks

We build a multitask model for three downstream tasks. **Sentiment Analysis (SST)** The model takes in single sentences and predicts their sentiments as negative, somewhat negative, neutral, somewhat positive, or positive. **Paraphrase Detection (Para)** The model takes in pairs of sentences and predicts whether each pair is a paraphrase of one another or not. **Semantic Textual Similarity (STS)** The model takes in pairs of sentences and predicts how similar they are, on a scale from 0 to 5.

3.2 Models

Our baseline is a BERT-based multitask model called minBERT CS224N. This backbone BERT consists of 12 transformer layers with multiheaded self-attention and outputs bidirectional word embeddings(Devlin et al., 2018). To adapt to multitask learning, we add a prediction head for each downstream task; each head consists of a single linear layer mapping the embeddings of length 768 to five logits or a scalar, depending on the task. The losses for the three tasks are summed before

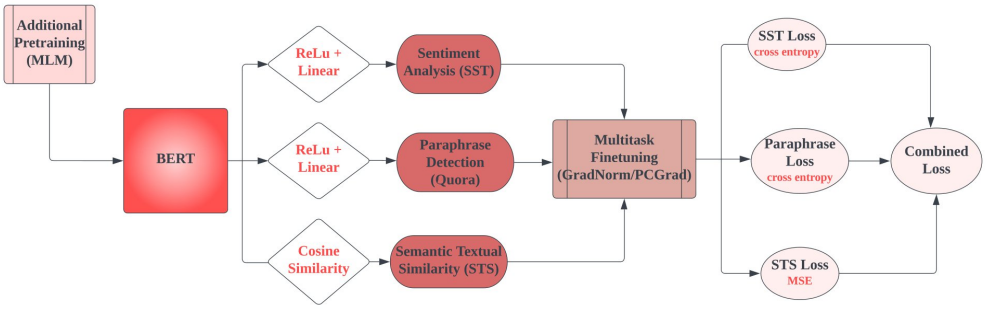

Figure 1: Overview of Workflow and Extensions

backpropagating through the model 3. To improve our baseline model, the following extensions were implemented and applied to the baseline individually or combined as ensembles.

Additional Pretraining with Masked Language Modeling We chose to further pre-train our model on the SST dataset to particularly focus on improving upon the sentiment analysis task. Our model was trained using the masked LM objective, similar to the training in the original paper Devlin et al. (2018), predicting the 'masked' word piece tokens using the surrounding context. The 'masked' tokens were chosen from the input sequence with a 15% chance, 80% of which were then replaced with the [MASK] token, 10% with random tokens from the vocabulary, and 10% left untouched. The masked embedding was passed into a two-layer network with ReLU activation to predict the tokens under the mask, i.e. the logits corresponding to each token in the vocabulary of size 30,522. The logits were then passed through a softmax function to obtain the predicted token for each of the masked positions. Cross entropy loss was used to evaluate the model performance. The Dev. accuracy was 5% after 25 epochs with lr=1e-3.

Cosine-Similarity Fine-Tuning We experimented with using Cosine similarity for both the STS and Paraphrase detection because both tasks involve semantic similarity, and therefore we can measure the similarity between the two embedding vectors of the sentences in a high-dimensional space. We experimented with using Cosine Embedding loss for the paraphrase task since its labels are discrete and Cosine Similarity for the STS task which has continuous labels. We found that Cosine similarity was most effective on the STS task. Specifically, we passed each sentence in a pair through the BERT model to obtain the corresponding embedding vector of size 768. We then passed these embeddings through the Cosine_similarity method from PyTorch to calculate the Cosine similarity score between the two sentence embeddings, as shown below. Since Cosine similarity returns values between -1 and 1 but our labels are between 0 and 5, we then normalized the ground truth STS labels to be at the same scale. Finally, we used mean squared error (MSE) loss for evaluation of the predicted logit.

$$similarity(x_1, x_2) = \frac{x_1 \cdot x_2}{max\{\|x_1\|_2 \cdot \|x_2\|_2, 1e{-}8\}}$$

Multitask Fine-tuning with GradNorm The GradNorm algorithm Zhao Chen (2018) allows us to learn the weight functions $w_i(t)$ for the multitask loss function $\sum w_i(t)L(t)$. We define the notations:

- W: subset of full network weights where we apply GradNorm. This is the last shared layer of weights i.e. the pooler dense layer.
- $G^{(i)}_W(t) = \|\nabla_W w_i(t)L(t)\|_2$: L2-norm of the gradient of the individual weighted task loss with respect to W.
- $\bar{G}_W(t) = E_{task}[G^{(i)}_W(t)]$: average at training time *t*.
- $\tilde{L}_i(t) = L_i(t) / L_i(0)$: measure of the inverse training rate of task *i*.

- $r_i(t) = \tilde{L}_i(t) / E_{task}[\tilde{L}_i(t)]$: relative inverse training rate of *i*.

The algorithm (1) places gradient norms for different tasks on a common scale through which we can reason about their relative magnitudes and (2) dynamically adjusts gradient norms so different tasks train at similar rates. To accomplish (1), we use the average gradient norm $\bar{G}_W(t)$ as the common scale, which establishes a baseline at each time-step t by which we can determine relative gradient sizes. Meanwhile, the relative inverse training rate of task *i*, $r_i(t)$, can be used for (2) to rate balance our gradients. The higher the value of $r_i(t)$, the higher the gradient magnitudes should be for task *i* in order to encourage the task to train more quickly.

Thus, our desired gradient norm for each task *i* is then: $G_W^{(i)}(t) \rightarrow \bar{G}_W(t) \times [r_i(t)]^\alpha$ (1) where α is an additional hyperparameter. α sets the strength of the restoring force which pulls tasks back to a common training rate and can be thought of as an asymmetry parameter.

We then update our loss weights $w_i(t)$ to move gradient norms towards Equation (1) as our target for each task. GradNorm is implemented as an L1 loss function L_{grad} between the actual and target gradient norms at each timestep for each task, summed over all tasks which gives us Equation (2): $L_{grad}(t; w_i(t)) = \sum_i \| G_W^{(i)}(t) - \bar{G}_W(t)) - [r_i(t)]^\alpha \|_1$. L_{grad} is then only differentiated with respect to w_i and the gradients are updated. The pseudo-code is attached in Appendix A, and we adapted the code from Sharifi-Noghabi (Nov 2021).

Multitask Fine-tuning with PCGrad PCGrad is an implementation of gradient surgery, which removes conflicting gradients among tasks that interfere with smooth optimization in multitask learning Tianhe Yu (2020). We apply the PyTorch implementation Tseng (2020). PCGrad compares pairwise gradients in a round-robin manner and projects a conflicting gradient onto the normal plane of the other. A visualization is attached in Appendix A. When two gradients are conflicting, i.e. their dot product is negative, one gradient g_i is updated as follows:

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

Model Size: Number of Layers and Nonlinearity To combat underfitting, we increased the model complexity by adding more linear layers with ReLU activation in the prediction heads for the downstream tasks. Specifically, for SST, the linear layers accepted inputs of size 768, 256, and 64; they output 5 logits for the number of sentiment classes. For Para and STS, the layers accepted inputs of size 768×2, 256×2, and 64; they output 1 scalar, which was a logit for Paraphrase.

Data Engineering We experimented with two sizes of training data. First was taking a random subset of the SST and Quora datasets to match the size of the STS dataset, which has the smallest number of samples. To make more use of the available data, we decided to increase our training dataset by using the full SST dataset, while repeating samples in the STS dataset and taking a random subset of the Quora dataset, such that all datasets match the length of the SST dataset.

4 Experiments

4.1 Data

For SST, we used the **Stanford Sentiment Treebank** dataset[1]. This dataset consists of 11,855 single sentences extracted from movie reviews, which are split into 8,544/1,101/2,210 examples for train/dev/test. Each sentence is parsed with the Stanford parser; overall, there are 215,154 unique phrases, and each is labeled negative, somewhat negative, neutral, somewhat positive, or positive. For Para, we used the **Quora** dataset[2], which contains 400,000 question pairs with binary labels, indicating if a question pair is a paraphrase. The dataset is split into 141,506/20,215/40,431 examples for train/dev/test. For STS, we used the **SemEval STS Benchmark** datasetEneko Agirre and Guo. (2013). This dataset contains 8,628 sentence pairs with labels from 0 to 5, indicating sentence similarity from unrelated to equivalent meaning. The dataset is split into 6,041/864/ 1,726 examples for train/dev/test. Additionally, we conducted initial sentiment analysis using the SST and **CFIMDB**

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

[2] `https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs`

dataset[3]; CFIMDB contains 2,434 sentences of movie reviews, each labeled negative or positive. The dataset is split into 1,701/245/488 examples for train/dev/test.

The input into our model is either single or paired sentences with their labels, which varies depending on the downstream task. The backbone BERT model converts these sentences into token sequences using the Word Piece tokenizer. The input into our multitask model is the output of the backbone BERT model, which is the last hidden state of the [CLS] token of length 768. We experimented with two dataset sizes, as described for the Data Engineering extension.

The output of our model also varies among tasks; for sentiment analysis, the model outputs logits for the five classes that are passed into a softmax function to obtain a single predicted sentiment. For paraphrase detection, the model outputs a scalar that is passed into a softmax function and compared against the true binary label. For textual similarity, the model also outputs a scalar, which is compared against the true label on a scale of 6.

4.2 Evaluation method

For the sentiment analysis and paraphrase detection tasks, we evaluate our model with the cross entropy loss and accuracy scores. For the textual similarity task, we evaluate our model using MSE and Pearson correlation, which measures the linear relationship between two variables on a scale from -1 to 1.

4.3 Experimental details

We use the following abbreviations to denote the extensions:

- ReLU: Increased model size i.e. number of layers and added nonlinearity
- Cos: Cosine-similarity fine-tuning
- GN(α): Multitask fine-tuning with GradNorm (asymmetry parameter α)
- PCG: Multitask fine-tuning with PCGrad
- MLM: Additional pretraining with masked Language Modeling on the SST dataset
- Loop: using repeated samples from STS dataset to match the size of the SST dataset.

Model configurations Our multi-task baseline model was the baseline BERT with 3 linear prediction heads for each of the 3 tasks. This baseline BERT was trained with batch size = 8, epochs = 25, learning rate = *1e-5*, hidden dropout probability = 0.3, optimizer=*AdamW*, and using the option *finetune*. We saw in our initial sentiment analysis that *finetune* increased scores significantly over *pretrain*, by more than 0.12 1. *finetune* requires the parameters in the backbone BERT model to be trained, whereas *pretrain* does not. This increases the model expressivity but also the training time. To maximize the model performance, we used the *finetune* option for all of our experiments.

Our initial model configurations consisted of this baseline with a few extensions added one at a time. This helped us investigate the effectiveness of individual extensions for ablation studies. We first experimented with ReLU, as well as Cos as the prediction head for both paraphrase detection and semantic textual similarity tasks. Since we saw Cos was only effective for STS, we applied the extension to the STS prediction head going forward. We also experimented with Loop to improve the accuracy on sentiment classification by using the maximum number of samples available for that task. We also applied GN and PCG separately. PCGrad did not have conflicting gradients when used with baseline, so we added this extension with the other extensions. Lastly, we applied MLM for additional training on SST. Our nine ensembled models are listed in 2. All experiments were run on GPU on an AWS instance.

Hyperparameter Tuning We experimented with four values of the asymmetry parameter for GN: α=[0.5,1.0,1.5,3.0]. Additionally, we experimented GN with batch size = [16, 32] and *lr*= [1e-5, 1e-4, 1e-3]. However, the default values of batch size 8 and *lr*=1e-3 performed the best. Additionally, we increased the dropout probability from 0.3 to 0.5 since we saw signs of overfitting for the ensembles model with Cos, ReLU, and PCG extensions. However, this decreased the model performance, so we used to the default 0.3. Therefore, all of our experiments were run with the baseline hyperparameters.

[3] `https://nlp.stanford.edu/software/lex-parser.shtml`

4.4 Results

Dev. Scores	SST Acc.	CFIDB Acc.
Pretrain	0.399	0.792
FineTune	0.528	0.963

Table 1: Dev. Scores For Initial Sentiment Analysis on SST and CFIDB

Dev. Scores	Avg.	SST Acc.	Para Acc.	STS Corr.
Baseline	**0.452**	**0.495**	**0.5**	**0.360**
Baseline + ReLU	0.516	0.493	0.688	0.367
Baseline + Cos	0.508	0.496	0.526	0.502
Baseline + GN	0.485	**0.523**	0.614	0.319
Baseline + Cos+ ReLU + Loop	**0.581**	0.497	**0.717**	**0.528**
Baseline + Cos + ReLU + PCG	0.553	0.477	0.678	0.503
Baseline + Cos + ReLU + PCG + Loop	0.557	0.502	0.695	0.475
Baseline + Cos + ReLU + PCG + MLM	0.562	0.504	0.700	0.482
Baseline + Cos + ReLU + PCG + MLM + Loop	0.551	0.502	0.697	0.454
Baseline + Cos + ReLU+ GN + MLM + Loop	0.553	0.515	0.699	0.446

Table 2: Dev Scores. GN with $\alpha = 1$.

Test Scores	Avg.	SST Acc.	Para Acc.	STS Corr.
Baseline + Cos + ReLU + Loop	0.576	0.528	0.718	0.482

Table 3: Best model performance on Test Set: Overall and Task-wise Scores

We summarize the overall and task-wise scores on the Dev set in 2 and on the Test set with the best overall model in 3. Our best overall model was the baseline model with the Cos, ReLU, and Loop extensions; it improved the baseline average score by 0.13 on the Dev set. This model also performed the best in the Paraphrase and STS tasks individually, significantly improving the baseline task-wise scores by 0.22 and 0.17 respectively. For the SST task alone, the baseline model with GN performed the best, improving the baseline by 0.03.

We also experimented with different values of α for GradNorm 4. The highest ensemble model performance with GN is when $\alpha = 1$ with the second-highest SST accuracy on the Dev set (0.515). Overall scores are similar between $\alpha = 0.5$ and $\alpha = 1.5$, but we can see that the individual task scores differ. As α increases, paraphrase detection accuracy improves from 0.650 to 0.697, while the SST accuracy decreases from 0.483 to 0.474. For $\alpha = 3.0$, overall performance as well as individual task performance is the lowest.

5 Analysis

We compare the Baseline and Ensemble models to evaluate the effectiveness of the extensions 2.

ReLU, Cos Adding nonlinear layers in the prediction heads for all three tasks significantly improved the Paraphrase accuracy by 0.19 and slightly improved the STS correlation by 0.08, but not the SST task. This suggests the effectiveness of added complexity for the prediction of STS; it also suggests the need to explore ways to improve the SST score other than model architecture, such as engineering the input embeddings. Comparing the Baseline and Baseline + Cos models, we observe a significant increase in STS correlation alone. Cosine similarity might be a better indicator of how similar two embeddings are than a linear mapping to one scalar since it is a scaled dot product of the two embedding vectors.

GradNorm As seen in Table 4, GN on our multi-task architecture has the highest performance when $\alpha = 1.0$. An $\alpha = 1.0$ suggests that there is high asymmetry between the tasks as low asymmetry values range between $\alpha = 0.0$ and $\alpha = 0.5$. Higher asymmetry suggests that these three tasks are different in complexity and learning dynamics as a larger α is needed to enforce stronger training rate

Dev. Scores	Avg.	SST Acc.	Para Acc.	STS Corr.
GN(α = 0.5)	0.541	0.483	0.650	0.489
GN(α = 1.0)	**0.553**	**0.515**	0.699	0.448
GN(α = 1.5)	0.539	0.474	0.697	0.445
GN(α = 3.0)	0.497	0.477	0.625	0.389

Table 4: Dev Scores for Ensembled Model with Cos, ReLU, MLM, Loop, and GN extensions with Four Values of α.

balancing. Higher α also pushes the weights $w_i(t)$ further apart which we observed as our best model weights for α = 1.0 were [SST loss = 0.95, Paraphrase loss = 1.07, STS Loss = 0.989]. This pushing of weights reduces the influence of tasks that overfit or learn too quickly.

In our configuration with GN alone, the $\alpha = 1.0$ seems to improve the baseline performance of SST by 0.028 and Paraphrase 0.11 while decreasing the performance of STS by 0.5 on the Dev set. This suggests that the algorithm found that the scale of the STS loss was larger than those of the other tasks - this is reasonable as STS loss is MSE while the others are cross-entropy loss. It also suggests that STS had a faster training rate than the other two, which is why a higher α forces the STS to slow down while allowing the other two tasks to learn in the meantime. As a result, it seems that either SST and Paraphrase performance or STS performance alone is maximized. Additionally, this suggests that perhaps multi-task finetuning is not appropriate for all three tasks together and that STS may benefit from single-task finetuning.

Ensemble 1 (Best Model): Cos, ReLU, and Loop Combining Cos and ReLU, and using more data with Loop improved the baseline by 0.13, resulting in the best overall ensemble model. The extensions seemed to enhance the performance on the Paraphrase and STS tasks, possibly because the two tasks are similar in the sense that they both consider pairs of sentences. It was unexpected that the SST accuracy didn't improve, since we used the full SST dataset with the Loop extension. We hypothesized that this may have to do with the optimization for SST conflicting with the optimization for Para and STS and decided to explore multitask fine-tuning methods i.e. PCGrad and GradNorm.

Ensembles 2 and 3: Cos, ReLU, PCG, Loop Comparing Ensembles 2 (without Loop) and 3 (with Loop), we saw an increase in the SST and Para tasks, while a significant decrease in the STS correlation by 0.25. This result might suggest that there is a trade-off between the performances on the SST and STS tasks, i.e. an improvement in the SST accuracy comes at the Cost of decreasing performance on the STS task, perhaps due to significant differences between these tasks. The Paraphrase task, on the other hand, is easier to optimize for because there is some overlap with the two other tasks. For example, the SST and Para share the same loss function, and the STS and Para share the same input type of paired sentences.

Ensemble 4: Cos, ReLU, PCG, MLM We were able to confirm that MLM was helpful in improving the SST accuracy score, by 0.27 compared to the model without MLM. However, it came at the cost of a decreasing STS correlation by 0.2. This result reiterates the difficulty in learning both the SST and STS tasks.

Ensemble 5: Cos, ReLU, PCG, MLM, and Loop From adding Loop to Ensemble 4, it was unexpected to observe a nonchanging performance in the SST and Para scores. We also observed a significant decrease in the STS correlation, indicating possible overfitting of the model due to the repeated samples in the STS dataset.

Ensemble 6: Cos, ReLU, GN, MLM, and Loop Compared to Ensemble 5 with PCG, we see an improvement in the SST score but a decrease in the STS score, both by around 0.1. The earlier hypothesis about STS requiring single-task finetuning seems to be supported as the GN with α = 1 combined with Cos for STS improves performance for STS from the baseline. However, Ensemble 6 still outperforms all other ensemble models at SST accuracy while having a slightly decreased STS correlation. This shows that GN optimizes learning for SST and Para, suggesting that SST has a slower learning rate than STS. This lends further support to the idea that all three tasks' performance cannot be optimized simultaneously.

To summarize, the most effective extensions for our model were cosine similarity for the STS task and the increased model size for the Paraphrase task. Through GradNorm, we found the tasks to be highly asymmetric, meaning the tasks have different complexities and learning dynamics, suggesting further fine-tuning of the individual tasks might be helpful in improving the overall model performance. PCGrad was helpful when multiple tasks were combined, in which case the gradients relative to the task-wise losses conflicted frequently. Additional pre-training was somewhat effective in improving the SST accuracy, which was the most difficult to achieve out of the three tasks. Lastly, we found optimizing for all three tasks very challenging. Different ensembles yielded either an improvement in the Paraphrase and STS scores or an improvement in the Paraphrase and SST scores, but not all three simultaneously. The Paraphrase task was the easiest to optimize for, perhaps because it shared features with the two other tasks, such as the loss function or input type of paired sentences.

6 Future Work

While we greatly improved baseline performance with the extensions implemented, there is scope for improvement in designing the model architecture and fine-tuning the model. For pre-training, we can explore different masking techniques with masked language modeling, such as masking out specific parts of speech or increasing the distance between masked elements. In terms of data engineering, we can delete or add random words in sentences and increase the number of sentences by using synonym replacements. For fine-tuning, to further improve the effectiveness of the similarity metric, we can also explore using euclidian or manhattan distance to improve the overall performance on the STS task and overall. We can also tune the hyperparameter α in GradNorm in a finer range between (0.5, 1.5) and adding a correction for conflicting gradients as in PCGrad. Finally, our experiments showed that learning the tasks simultaneously with a shared backbone encoding greatly improved the performance on each task. Training on additional tasks in a similar multitask manner such as question-answering, named entity recognition, and classifying the category of sentence subject matter could further help us achieve optimal performance over all tasks.

7 Conclusion

In this project, we proposed a BERT-based multi-task learning architecture for three NLP tasks, namely sentiment analysis, paraphrase detection, and semantic textual similarity. To improve the performance of our baseline model, we experimented with various model architectures, pre-training, and fine-tuning techniques.

Our findings showed that cosine similarity and increased model size through ReLU were particularly effective for the STS and Paraphrase tasks, respectively. Both GradNorm and PCGrad helped us understand the relationships between the individual tasks and how they could be fine-tuned to prevent conflicting gradients and further improve performance. Additional pre-training (MLM) also showed potential for improving accuracy for the SST task. Although ensembling was challenging as optimizing for one task often decreased scores in others, we found that ensembling different approaches led to significant improvements, especially in the Paraphrase and STS tasks due to their inherent semantic overlap. Therefore, our best overall model incorporated the Cos, ReLU, and Loop extensions, improving the baseline average score by 0.13 on the Dev set and significantly improving the task-wise scores for Paraphrase and STS by 0.22 and 0.17, respectively.

While we were able to significantly improve from our baseline model, there is still scope for improvement in designing the model architecture and fine-tuning the model. One limitation of our work was the challenge of ensembling approaches, where optimizing for one task often decreased scores in others. To address this, we can explore using different masking techniques, data engineering methods, various similarity metrics, and utilizing additional NLP tasks to further improve our model's performance across all tasks.

Overall, our experiments demonstrated that a BERT-based multitask learning architecture, combined with various pre-training, fine-tuning, and ensembling techniques, can significantly improve the performance of NLP tasks such as sentiment analysis, paraphrase detection, and semantic textual similarity.

References

Qiwei Bi, Jian Li, Lifeng Shang, Xin Jiang, Qun Liu, and Hanfang Yang. 2022. Mtrec: Multi-task learning over bert for news recommendation. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2663–2669.

Teaching Team CS224N. Cs 224n: Default final project: Minbert and downstream tasks.

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

Mona Diab Aitor Gonzalez-Agirre Eneko Agirre, Daniel Cer and Weiwei Guo. 2013. * sem 2013 shared task: Semantic textual similarity. In *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*.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019. Multi-task deep neural networks for natural language understanding. *arXiv preprint arXiv:1901.11504*.

Iryna Gurevych. Nils Reimers. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *EMNLP*.

Hossein Sharifi-Noghabi. Nov 2021. hosseinshn/gradnorm.

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

Wei-Cheng Tseng. 2020. Weichengtseng/pytorch-pcgrad.

Chen-Yu Lee Andrew Rabinovich Zhao Chen, Vijay Badrinarayanan. 2018. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *ICML*.

A Appendix

Algorithm 1 Training with GradNorm

Initialize $w_i(0) = 1\ \forall i$
Initialize network weights $\mathcal{W}$
Pick value for $\alpha > 0$ and pick the weights W (usually the final layer of weights which are shared between tasks)
for $t = 0$ **to** max_train_steps **do**
  Input batch x_i to compute $L_i(t)\ \forall i$ and $L(t) = \sum_i w_i(t) L_i(t)$ [standard forward pass]
  Compute $G^{(i)}_W(t)$ and $r_i(t)\ \forall i$
  Compute $\overline{G}_W(t)$ by averaging the $G^{(i)}_W(t)$
  Compute $L_{\text{grad}} = \sum_i |G^{(i)}_W(t) - \overline{G}_W(t) \times [r_i(t)]^\alpha|_1$
  Compute GradNorm gradients $\nabla_{w_i} L_{\text{grad}}$, keeping targets $\overline{G}_W(t) \times [r_i(t)]^\alpha$ constant
  Compute standard gradients $\nabla_{\mathcal{W}} L(t)$
  Update $w_i(t) \mapsto w_i(t+1)$ using $\nabla_{w_i} L_{\text{grad}}$
  Update $\mathcal{W}(t) \mapsto \mathcal{W}(t+1)$ using $\nabla_{\mathcal{W}} L(t)$ [standard backward pass]
  Renormalize $w_i(t+1)$ so that $\sum_i w_i(t+1) = T$
end for

Figure 2: GradNorm: Training Algorithm

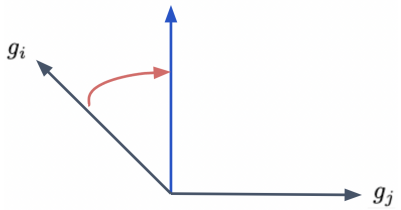

Figure 3: PCGrad: Gradient Update Formula and Visualization