Using Stochastic Layer Dropping as a Regularization Tool to Improve Downstream Prediction Accuracy

Stanford CS224N Default Project

Karthik Jetty
Department of Computer Science
Stanford University
kjetty@stanford.edu

Abstract

As Large Language Models (LLMs) become increasingly prevalent in daily applications, understanding the unique domain where BERT outshines even the most advanced LLMs is crucial. This paper delves into BERT's efficacy in three specific downstream tasks: Paraphrase Detection, Sentiment Classification, and Semantic Textual Similarity. Our objective is to enhance BERT's performance in these areas using various strategies. We concentrate on a range of regularization methods including Multi-task Learning and an innovative Stochastic Layer Drop technique. The Multi-task Learning approach utilized three distinct loss functions, each tailored to one of the tasks. We also introduced Stochastic Layer Drop, which randomly omits a transformer layer within BERT's embeddings with a probability of 0.15. This project reveals that regularization, particularly through Stochastic Layer Drop, significantly influences model training, markedly boosting the model's robustness across all evaluated tasks. These two methods, among others, created a model that is robust at handling all three downstream tasks.

1 Key Information to include

- Mentor: Hamza

2 Introduction

Language is a fundamental hallmark of our humanity, entwining complex concepts like Sentiment Classification, Paraphrase Detection, and Semantic Textual Similarity—endeavors unattainable for any other species (much less some of our own). It's our responsibility to extend our abilities to thinking machines, in the hopes that one day we can meet an equal. Mastering these tasks is essential, laying the groundwork for full-fledged conversations and the ultimate challenge of the Turing Test. The generalizability of the model is pivotal.

BERT is (suprisingly still) a state of the art model used in understanding intricate relationships within sentences and for hard NLP tasks. BERT was initially published in 2018 as a Bilateral transformer based model that was pretrained on the entire Wikipedia corpus. BERT included several groundbreaking features, including being able to reading entire groups of words at once, which allowed it to capture more information.

In humanity's quest to one day build a transformer, the size of transformer-based models have dramatically increased, with models containing billions of parameters. With the growing size of these models, combined with the increasing amount of data to train these models on, it is vital to find better ways to regularize models so that they can handle unseen data robustly. What good is a terminator if it freezes as soon as it encounters a new scenario? Therefore, for this implementation of BERT, we aim to better understand regularization techniques, such as Stochastic Layer Drop and sharing weights between tasks. Previous literature says that weight sharing can lead to more efficient training and can be particularly effective in models with recurrent structures or in multi-task learning scenarios where different tasks share underlying features (Zhang et al., 2020). Prior Research on Stochastic Layer Drop seems to indicate that carefully dropping certain layers with certain frequency could boost training speed while minimzing the impact on performance and improving robustness. However, this can be taken a step further by dropping all transformer layers with the same probability. This could better prevent overfitting by encouraging the model to learn more dispersed and robust representations. It can also lead to models that are more resilient to variations in the input data and can adapt better to new, unseen data.

3 Related Work

This section helps the reader understand the research context of your work by providing an overview of existing work in the area.

3.1 Layer Dropping

In Zhang and He (2020) and Xiong et al. (2020), we see progressive layer dropping implemented to a BERT model with Layer Normalization that happens at a different spot than usual. For this model, implementing a progressive drop probability for each layer (depending on where it is located in the model) decreased training time while reaching comparable results to the original model. Another paper studied the effects of six different types of droppings on the performance of BERT on downstream tasks. The six types of droppings include Top-Layer Dropping, Alternate Dropping, Parameter-Based Dropping, Contribution-Based Dropping, Symmetric Dropping, and Bottom-Layer Dropping. They found that Top-layer dropping performed the best, but they did not test random dropping (Sajjad et al., 2023).

4 Approach

In this project, we implemented the Default Final Project framework but introduced several modifications to enhance the model beyond its baseline capabilities. We compared two baseline models to gauge our improvements. The initial baseline model employed frozen BERT embeddings to address the three classification tasks. Meanwhile, our enhanced baseline model went a step further, fine-tuning the embeddings according to the SST loss function and the specific tasks at hand, resulting in a slightly better performance.

4.1 Multitask-Learning

Our task is to create an algorithm that performs well across a series of three tasks. Thus, it does not make sense to train on only one dataset, and furthermore, it does not make sense to only use one loss function when training. Thus, the first task was to implement training as well as loss functions for Paraphrase detection and STS.

The first method employed was Multi-task learning, which sought to train the model on all three tasks simultaneously. Since the BERT model being implemented is trying to improve performances on all three tasks, it makes sense to implement a learning algorithm that minimizes loss across all three tasks. Below are the formulas used for each of the tasks. The first formula is used for Semantic Textual Similarity, the second formula is used for identifying paraphrase, and the third formula is used for determining positivity of statements. The third formula is simply cross entropy but with only two classes.

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

$$L_i = -\sum_{c=1}^{C} y_{ic}\log(\hat{y}_{ic}) L = \frac{1}{N}\sum_{i=1}^{N} L_i$$

$$L_i = -\left[y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)\right] L = \frac{1}{N}\sum_{i=1}^{N} L_i$$

(Bi et al., 2022)

4.2 Cosine Similarity Loss

After implementing Multi-task Learning, it was observed that the accuracy for Semantic Textual Similarity (STS) was not on par with the results for SST and Paraphrase Detection tasks. To address this, we integrated the Cosine Embedding Loss, aiming to distinctly separate dissimilar embeddings. This loss calculates the cosine similarity between embeddings, scaling the weight values between 0 and 1, to assess their likeness. Ideally, this similarity ranges from 0 (completely orthogonal embeddings) to 1 (identical embeddings), as shown in the formula below.

$$Similarity = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

However, when this method was implemented the accuracy of the model significantly decreased. This is because in the implementation, the similarity score values were normalized in such a way that similarity scores less than 2.5 were given a score of 0, while similarity scores greater than 2.5 were given a score of 1. This caused a lot of granularity to be lost.

4.3 Novel Stochastic Layer Dropping

To improve upon prior work, Random Layer Dropping was implemented in an effort to see it's effects on training time, accuracy, and regularization. For the purposes of this project, the goals were to regularize the model since it was observed that the training accuracy would increase significantly while dev accuracy plateaued. When a layer was dropped, the embedding would just be passed to the next layer without any modification. This means the remaining layers must adapt to compensate, leading to a more robust model that doesn't rely too heavily on any single layer's features. The overall depth and capacity of the model are reduced, impacting its ability to capture intricate patterns in the data. As mentioned in Related Works, there has been work done looking at dropping layers. However, there is typically a strategy associated with how Layers are dropped, such as dropping every other layer or using a drop rate that is calculated for each layer-which determines how often that layer should be dropped. These models do this in an attempt to reduce training time and improve accuracy. These papers do targeted Stochastic Layer Dropping, whereas my approach is to randomly dropping the layers mainly to create robustness within my model.

4.4 Loss function tuning

After implementing Novel Stochastic Layer Dropping, it was noticed that the impact of the loss functions were heavily imbalanced as seen in the image below.

As a result, it was attempted to weight the losses so that each has a similar impact on the dataset. For paraphrase, we tried weights ranging from 0.4-0.6. Meanwhile, for STS and SST, weights ranging from 1000 to 4000 were attempted. Several other loss functions were tried, especially for STS correlation. Specifically, Mean Absolute Error and cosine embedding loss (in addition to cosine similiarity) were both tested.

4.5 Shared Weights layer

A shared weights layer was applied after the BERT layers were included. The shared weights layer is a fully connected layer that connects the output of BERT to each of the classification heads. This was done in an effort to regularize the data since it was noticed that training accuracy increased much faster than dev accuracy.

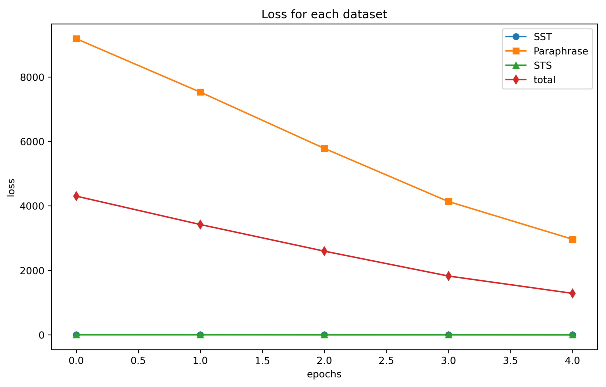


Figure 1: Loss vs. Epoch

4.5.1 Concatenation Technique

The final technique used was concatenating the input ids and attention masks before putting it through the Bert Layer. This was done for both the Paraphrase Task and the Sentiment Task. Concatenating the two sentences allows the model to learn the relationship between the sentences within the same context. This makes it particularly useful for STS, where the relationship between the sentences is pivotal. BERT also uses transformers with self-attention mechanisms that compares every word to every other word in the sequence. By concatenating the sequences together then passing it through the model, it is possible to understand relationships and dependencies between the words in the different sentences.

5 Experiments

5.1 Data

The data used was the data given to us for the default project. These datasets included the Stanford Sentiment Treebank (SST) dataset, the Quora Paraphrase Dataset, and the SemEval STS Benchmark dataset as described in the project handout. No other data was used.

5.2 Evaluation method

As with most other projects, the main method of evaluation was looking at the results of the model on the dev leaderboard. In addition to looking at the dev and test leaderboards to confirm the efficacy of the model, the predictions vs. actual values were examined to determine where the model was making mistakes. ROC curves, Confusion matrices, train accuracy over epoch, dev accuracy over epoch, loss, and scatter plots were plotted to see how each of these changed as we ran epochs as well as general accuracy.

5.3 Experimental details

The experiments were ran in several different contexts with several different configurations. The baseline model was run on a T4 GPU with a learning rate of 1e-5 for 10 epochs. This model started off with the embeddings that came with BERT and was updated as it was ran. From here, there were several modifications that were made in an attempt to increase accuracy on the given tasks. Since all three datasets were used, the models took about an hour to run per epoch–so depending on how many epochs were run, the model took anywhere from 5 to 10 hours to run. The model was trained on GCP.

5.4 Results

Table 1 shows us the accuracy of our finetuned baseline model. This baseline model only used one dataset for training purposes as well as only training on the SST dataset. As a result, only one loss

SST dev accuracy	Paraphrase dev accuracy	STS dev correlation	Overall dev score
0.514	0.485	-0.026	0.495

Table 1: Baseline Dev Accuracies

function was used–for SST. This explains the accuracy of Paraphrase hovering around 0.5 and the accuracy of STS dev correlation being 0.

The first item implemented was Multi-task Learning. In addition to adding loss functions for STS and Paraphrase, the model was also trained on the train set from STS and Paraphrase.

SST Test accuracy	Paraphrase Test accuracy	STS Test correlation	Overall Test score
0.467	0.775	0.357	0.640

Table 2: Multi-Task Learning Test Accuracies

From Table 2, there is a big jump in accuracy for STS and Paraphrase, which makes sense because these were the datasets that were added to the training. There was also a slight decrease in SST accuracy, but it was small compared to the improvements in the other tasks.

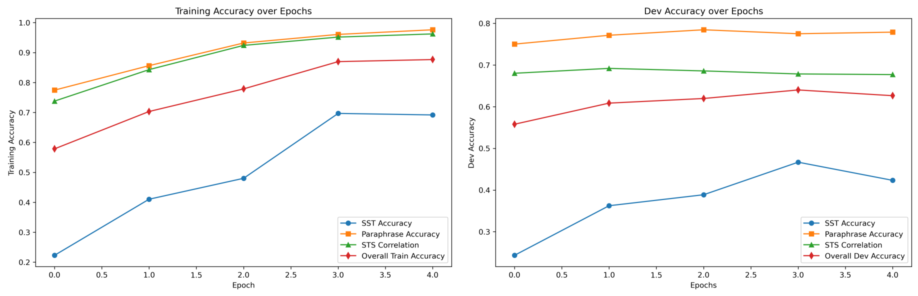


Figure 2: Train accuracy vs. Dev Accuracy

In Figures 2 and 3, the problem of overfitting becomes apparent. In these two figures, we see that train accuracy keeps steadily increasing while the accuracy of Paraphrase and STS plateau after the second epoch. To combat this, Stochastic Layer Dropping and a Shared weights layer were implemented. These two models combined helped regularize the model by necessitating that it remained robust to new data. After implementing these two additions to our model, the following results were obtained. In addition, it was observed that after these regularization techniques, the dev accuracy was much closer to the train accuracy and adding more epochs became useful in the accuracy on the dev set. After these were implemented, we got the results in Table 3.

SST Test accuracy	Paraphrase Test accuracy	STS Test correlation	Overall Test score
0.524 (+0.086)	0.0.784 (+0.004)	0.656 (+0.307)	0.712 (+0.081)

Table 3: Regularization Test Accuracies

The accuracy received by the final submission on the test leaderboard are shown in Table 4. The final improvement came from changing the order of concatenation of the embeddings.

SST test accuracy	Paraphrase test accuracy	STS test correlation	Overall test score
0.530 (+0.007)	0.877 (+0.093)	0.708 (+0.052)	0.754 (+0.042)

Table 4: Final test Accuracies after Concatenation

Cosine Similarity Loss and Loss function tuning were attempted, but they only decreased score. Specifically for loss function tuning, there was not enough training time available to adequately tune the scale of the loss function to improve accuracy.

In general, the results were in line with what I expected. I anticipated that tuning the loss function (so that each contributed the same amount at the beginning) would be an easy way to get my score lower,

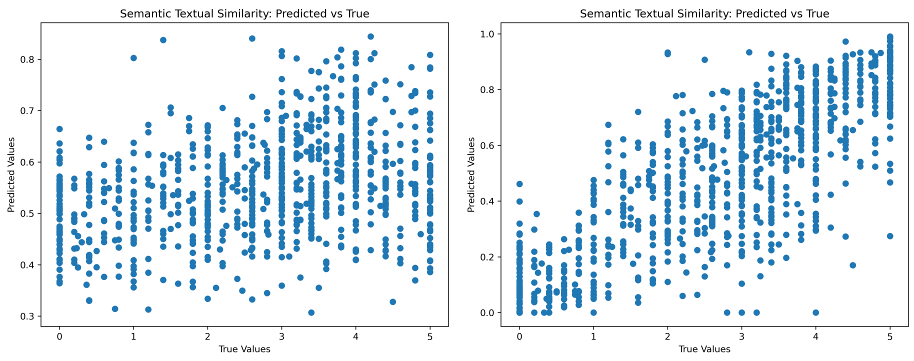
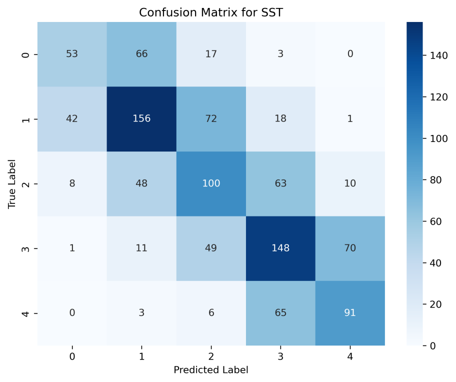


Figure 3: STS scatterplot before vs. after regularization

but it was necessary to constantly tweak the layer scale each time the model runs so all the losses are at the same place each level. In general, multi-task learning and the regularization techniques employed significantly improved the accuracy of the model. In particular, Stochastic Layer Drop combined with adding a shared weights layer between all the classification head allowed the model to not overfit to any one particular problem. Instead the model gained general knowledge about the relationships between words, which allowed it to perform better. This could inspire future research in examining Stochastic Layer Drop as a regularization technique, much like dropout.

6 Analysis

The system works with multiple regularizers in place to create robustness, allowing it to generalize to all three tasks.



	0	1	2	3	4
0	53	66	17	3	0
1	42	156	72	18	1
2	8	48	100	63	10
3	1	11	49	148	70
4	0	3	6	65	91

Figure 4: SST confusion matrix

Looking at where the model went wrong in predicting SST is a good place to understand what is wrong with the model since SST was the model with the worst accuracy. From Figure 4, we can see that when SST did incorrectly label a review, it would usually only be off by one value. For example, the most common mistake the model made was predict the sentiment of a review as a 2 when it should be 1. The next most common mistake was predicting movie reviews as 4 when it should have been 3. It is interesting to see that when the model makes mistakes, it tends to think that a movie review is more positive than it actually is. However, in general, this off-by-one error indicates that our BERT implementation has a general understanding of the sentiment classification task. In combination with our Paraphrase and STS scores, this gives us the confidence to say the model is robust at these three tasks.

7 Conclusion

From this project, I discovered that finding the optimal balance of loss functions between the tasks was a pointless task since there are so many ways to handle different scales of losses. In addition, I discovered a technique called Stochastic Layer Dropping, which randomly drops transformer layers based on a probability. This method, typically done in a calculated manner to reduce training speed while minimizing accuracy loss, actually increased the robustness of the algorithm and served in a similar manner as dropout.

The primary limitation of this finding was that Stochastic Layer Dropping was implemented in conjunction with a shared weight layer which makes it difficult to discern which of these resulted in the accuracy increasing. Future work can include isolating each of the additions made to the model to see how much of an impact each of the additions had on the baseline.

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, Dublin, Ireland. Association for Computational Linguistics.

Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. 2023. On the effect of dropping layers of pre-trained transformer models. *Comput. Speech Lang.*, 77(C).

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. 2020. On layer normalization in the transformer architecture. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 10524–10533. PMLR.

Minjia Zhang and Yuxiong He. 2020. Accelerating training of transformer-based language models with progressive layer dropping. *ArXiv*, abs/2010.13369.

Yuge Zhang, Zejun Lin, Junyan Jiang, Quanlu Zhang, Yujing Wang, Hui Xue, Chen Zhang, and Yaming Yang. 2020. Deeper insights into weight sharing in neural architecture search. *ArXiv*, abs/2001.01431.