Is training all you need? Exploring further pretraining and multi-task finetuning on BERT

Stanford CS224N Default Project

Richard Liu
Department of Statistics, Stanford University
`haoyuliu@stanford.edu`

Umar Maniku
Department of Statistics, Stanford University
`manikui@stanford.edu`

Abstract

While large language models have achieved state of the art results on many natural language tasks, their size makes them computationally infeasible for many users. Instead, we explore to what extent a model's performance can be improved through additional training, without changing its architecture. We investigate the effect further pretraining and multi-task finetuning have on the model's ability to generate robust sentence embeddings, measured by its performance on sentiment analysis, paraphrase detection and semantic textual similarity. We find that multi-task finetuning with tweaks to the model's prediction heads is the most performant training approach, achieving an overall test score of 0.724, with 84.3% accuracy on paraphrase detection - a significant improvement over our baseline model.

1 Key Information to include

External collaborators: None; Mentor: N/A; Sharing project: N/A

2 Introduction

BERT revolutionized the natural language modeling space when it was introduced in 2018, spawning a series of ever-larger language models that have continuously improved upon existing results on natural language tasks. At the core of this innovation lies the use of pre-trained contextual word embeddings. Sun et al. (2019) noted that pre-training models on large corpora yield substantial improvements in predictive performance for various NLP tasks, especially over embeddings learned from scratch. This hints at the importance of generating robust sentence embeddings not only for better downstream predictions, but improving how well a natural language system captures the semantic meaning of text. Such a system is task-agnostic and ultimately more valuable as it provides greater opportunity for analyzing large corpora without intensive fine-tuning.

As training large language models like GPT-4 is infeasible for many users, we were motivated to explore to what extent a relatively smaller model's performance can be improved through different training approaches, without significant changes to its architecture.

This paper examines how to generate robust sentence embeddings from BERT that simultaneously perform well on sentiment analysis (SST), paraphrase detection (Para) and semantic textual similarity (STS). We investigate the effect the following extensions to BERT's training process have on its performance:

- **Additional pretraining:** We implement masked language modeling (MLM) to pretrain the BERT model on in-domain and cross-domain data.
- **Multi-task finetuning:** We simultaneously finetune the model on our tasks by updating both BERT and prediction head weights based on all tasks' objectives simultaneously.

We additionally explore different prediction head configurations to increase their ability to decode word embeddings generated by BERT to make predictions.

Of these extensions, we find that multi-task finetuning is the key to greatly improving performance over the baseline implementation. Models trained just with multi-task finetuning far outperform those pretrained using the MLM objective or jointly pretrained then multi-task finetuned. With further tweaks to our multi-task finetuned model, including adding more layers and defining a correlation loss function for the STS task, we are able to achieve our highest-performing model.

3 Related Work

Devlin et al. (2018) introduced BERT, a bidirectional transformer-based language model pretrained using MLM and next sentence prediction. This approach was significant in moving away from unidirectional language models, which only restricts attention to previous tokens. This is particularly harmful for sentence-level tasks where bidirectional context is necessary. Crucially, the MLM objective works well within the BERT architecture as it allows the model to learn bidirectional representations by combining left and right contexts to predict the masked word's token. The BERT architecture has minimal differences between finetuning and pretraining for downstream tasks, alleviating the need for language models with complicated task-specific architectures. We leverage this in our report by finetuning BERT model weights on our predictive tasks, then freezing those weights to just train the task-specific prediction heads.

Sun et al. (2019) expands on Devlin et al.'s work by presenting a task-agnostic framework for finetuning BERT for text classification. They first propose to pretrain using the MLM objective on in-domain and cross-domain data. This differs from Devlin et al., who pretrains on a general domain. Sun et al. then multi-task finetunes simultaneously on all tasks before optionally performing a final finetuning round on each target task.

Sun et al. follow the multi-task learning procedure proposed by Liu et al. (2019) that allows a language model to learn representations across multiple natural language tasks. They use minibatch stochastic gradient descent that updates the model based on the minibatch's task-specific objective. This allows the model to optimize for the sum of all task objectives. Liu et al. note that this approach is particularly useful for related tasks as the knowledge the model learns from one task can jointly benefit the other tasks, which matches our use case.

Ultimately, Sun et al.'s work shows that a combination of finetuning, pre-training and multi-task finetuning can further improve upon BERT's state-of-the-art performance on natural language tasks. Notably, in-domain and cross-domain further pretraining result in greater performance improvements than completing multi-task finetuning prior to single-task finetuning. We aim to assess the validity of their training framework on our prediction tasks by assessing each approach's impact on performance in isolation, then combining them in different configurations.

4 Approach

4.1 Baseline minBERT model

Our baseline model consists of the provided minBERT model and ADAM optimizer (Staff, 2023), with pretrained BERT weights from Huggingface's `bert-base-uncased` model [1].

We implement prediction heads that use the generated BERT word embeddings to output the relevant logits for their respective prediction task. For our BERT model, each embedding layer has dimensionality $k = 768$. For SST, we implement a linear layer with dropout that takes in $u \in \mathbb{R}^k$, the final layer [CLS] token embedding, and outputs a length-5 vector of logits corresponding to the sentiment labels being predicted. We use cross-entropy loss and pass the logits through softmax to get the predicted sentiment class. For para, we take in u, v, the [CLS] embeddings of the candidate sentences, and concatenate them: $[u, v] \in \mathbb{R}^{2\times k}$. This is fed into a linear layer with dropout and a sigmoid function to output a single logit denoting whether the sentences are paraphrases. We use binary cross-entropy loss to train this layer. Similarly, STS has its own linear layer with dropout but instead outputs a logit denoting how similar they are, and uses MSE loss.

4.2 Further pretraining

We pretrain our BERT model using the masked language modeling (MLM) objective. We implement Devlin et al. (2018)'s procedure of random masking where 15% of the wordpiece tokens are replaced with the [MASK] token 80% of the time, a random token 10% of the time or the same token [2]. This prevents mismatches during pretraining and fine-tuning, where the [MASK] token does not appear.

[1]For more details, refer to http://web.stanford.edu/class/cs224n/project/default-final-project-bert-handout.pdf

[2]Implementation adapted from https://huggingface.co/docs/transformers/main/tasks/masked_language_modeling

We add a MLM prediction head to our BERT model that takes in each token's final hidden state and outputs $o \in \mathbb{R}^{|V|}$. This vector is passed through a softmax over the vocabulary size $|V|$ to predict the original token, using cross-entropy loss evaluated only on masked tokens.

Sun et al. (2019) note of the potential performance BERT can gain from cross-domain pretraining, as Devlin et al. pretrained on a large general corpus. It promises to expose our baseline model to a larger set of data, giving it more context to the types of language it will be expected to generate predictions from. We thus run the MLM objective on our domain-specific data using two approaches:

- **In-domain pretraining:** The pretraining set is obtained from combining data from related tasks, as text from one set may give additional context that helps the model predict on related tasks. For example, combining Para and STS data may help as they both focus on predicting pairwise sentence similarity. We experiment with combining SST and Para, SST and STS and Para and STS datasets.
- **Cross-domain pretraining:** The pretraining set is a combination of all tasks' training sets.

When combining datasets, we downsample each to the minimum size across the 3 training datasets, so as to avoid one being over-represented. We observe the model's performance on the three predictive tasks following each of the aforementioned pretraining strategies. While BERT pretraining includes the next sentence prediction task, we did not implement this as our datasets are generally disjoint: adjacent sentences are not necessarily part of the same text.

4.3 Multi-task finetuning

We implement multi-task finetuning, which updates both BERT and prediction head weights based on all tasks' objectives simultaneously. Following Liu et al. (2019), we experiment with two different round-robin approaches for iterating through tasks and their respective datasets. The first is **Sequential-Round-Robin**, in which we first get 3 separate data loaders from our datasets, each downsampled to have the same batch size. We then zip the 3 data loaders and at each iteration, concatenate the batches to train the model on the combined batched dataset sequentially [$\text{batch}_{\text{SST}}$, $\text{batch}_{\text{Para}}$, $\text{batch}_{\text{STS}}$] [3]. The second is **Drop-Round-Robin**, where at each iteration, we randomly select one batch and its corresponding task among the 3 batches, and just train the model on said task [4]. Both approaches approximately optimizes the sum of all task objectives, $\ell_{Multi} = \ell_{SST} + \ell_{Para} + \ell_{STS}$ (Bi et al., 2022), allowing the model to update its embeddings and learn for all tasks concurrently.

4.4 Prediction head architectures

We experiment with different configurations for the prediction heads as well as shared layers, as displayed in figure 1:

- Within the classification layers for para and STS, we investigate Reimers and Gurevych (2019)'s method of concatenating the sentence embeddings u, v with their absolute value element-wise difference $|u - v|$, before passing this through a linear layer with dropout.
- We add a linear layer with dropout and leaky ReLU activation that is shared across tasks. It takes in the final layer [CLS] token embedding $u \in \mathbb{R}^k$ and outputs a vector representation $s \in \mathbb{R}^{\frac{k}{2}}$ which is passed into each task-specific head.
- Within the prediction heads for Para and STS, we calculate the pairwise [CLS] embedding cosine similarity: cosine-sim$(u, v) \in \mathbb{R}^k$ and add this to the output from the linear layer: $o = W(u, v, |u - v|) +$ cosine-sim(u, v). For Para, we pass this summed output of logits through a sigmoid and calculate the binary cross entropy loss. For STS, we define a correlation loss function between the predicted text similarity scores and the labels.

[3]Details in Appendix A.1 Algorithm 1

[4]Details in Appendix A.1 Algorithm 2

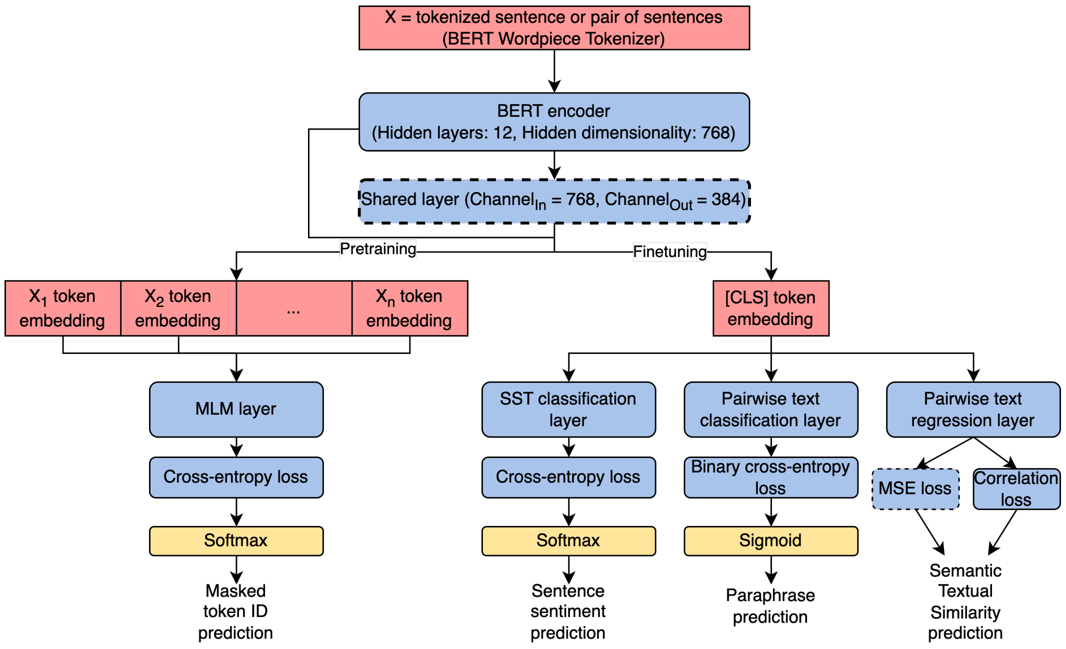
X = tokenized sentence or pair of sentences (BERT Wordpiece Tokenizer)
BERT encoder (Hidden layers: 12, Hidden dimensionality: 768)
Shared layer (Channel$_{In}$ = 768, Channel$_{Out}$ = 384)
Pretraining
Finetuning
X$_1$ token embedding
X$_2$ token embedding
...
X$_n$ token embedding
[CLS] token embedding
MLM layer
Cross-entropy loss
Softmax
Masked token ID prediction
SST classification layer
Cross-entropy loss
Softmax
Sentence sentiment prediction
Pairwise text classification layer
Binary cross-entropy loss
Sigmoid
Paraphrase prediction
Pairwise text regression layer
MSE loss
Correlation loss
Semantic Textual Similarity prediction

Figure 1: Final model architecture

4.5 Training approaches

Figure 2 shows the training approaches we explore:

1. **Baseline:** We freeze the BERT weights and just finetune the prediction heads. Using this, we can evaluate how robust the baseline embeddings are when predicting across our tasks.
2. **Sequential Finetuning (SFT):** We train both BERT weights and prediction heads on each task consecutively, passing the finetuned model with the best dev performance from each task onto the next one. Then, we freeze the BERT weights and re-train just the prediction heads to account for the new embeddings resulting from the previous finetuning.
3. **Further pretraining (MLM):** We further pretrain just the BERT weights using the MLM objective. Then, we freeze them and only train the prediction heads.
4. **Multi-task finetuning:** We train both BERT weights and prediction layers using multi-task finetuning. Then, we freeze the BERT weights and only re-train the prediction heads.
5. **Combination:** Last, we follow Sun et al.'s framework where we further pretrain, then multi-task finetune the BERT weights from the pretrained model and the prediction heads. Following this, we freeze the BERT weights and re-train the prediction heads.

We investigated the model's performance following each of the aforementioned training approaches.

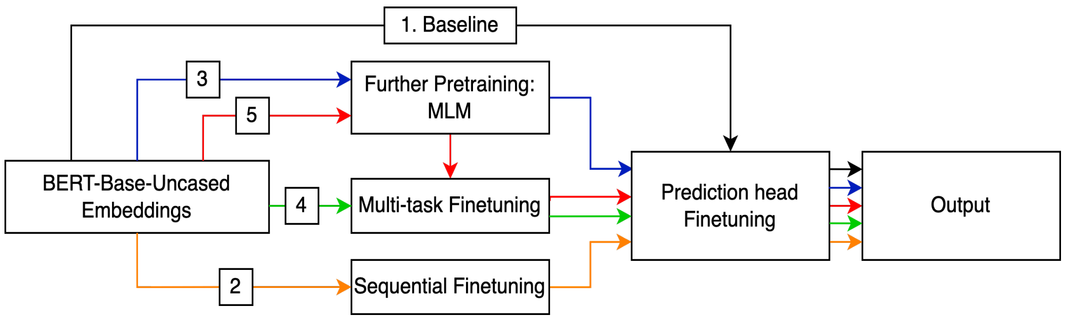

Figure 2: Different training approaches

5 Experiments

5.1 Data

Each of our datasets were split into train/dev/test sets. **SST:** Stanford Sentiment Treebank dataset of 11,855 sentences with 5 class labels; CFIMDB dataset of 2,434 sentences with binary labels. **Para:** Quora Question Pairs dataset of 400,000 questions pairs with binary labels. **STS:** SemEval Benchmark dataset of 8,628 sentence pairs with continuous score labels in the range [0, 5].

5.2 Evaluation method

For MLM, we evaluate using the model's cross-entropy loss on a held-out dev set. For our main prediction tasks, we evaluate the model's accuracy on the dev set for SST and Para. For STS, we calculate the Pearson correlation of the true similarity values against the predicted similarity values across the dev dataset. For a holistic evaluation across model configurations, we use an overall score that is the average of the prediction task metrics.

5.3 Experimental details

We maintain the same set of hyperparameters across our experiments and training approaches, for consistency. We set batch size to 8, dropout probability $p = 0.1$ and learning rate to 1e-5. We find that a lower learning rate tends to avoid the catastrophic forgetting problem where the model's performance on a task it was trained on drops after training on a different task. This is because the knowledge it extracted from the previous task is erased when it learns the new task. Sun et al. found that BERT training fails to converge with a learning rate greater than 1e-5, and our experiments validate this. Further we save the model with the best dev set performance for each experiment.

As our datasets have different sizes, we downsample the training and dev sets for MLM and Multi-task finetuning. This is to avoid the Para dataset exerting large influence on the model training process due to its relatively large size. We use the full dataset for each task during prediction head finetuning. We run all baseline, SFT and Multi-task finetuning experiments for 10 epochs while MLM runs for 20 epochs. Model training takes about 1.5 hours for sequential finetuning with 10 epochs for all tasks, 2 hours for MLM on 20 epochs and 1 hour for Multi-task finetuning on 10 epochs on an AWS g5.xlarge instance with a NVIDIA A10G GPU.

5.4 Results and discussion

Before discussing the results from our different training approach experiments, we first investigate the best prediction head configurations among those discussed in section 4.4. In table 1, we start with our baseline model from section 4.1 and iteratively amend the prediction head configuration. We maintain the amendment for subsequent experiments if it improves the overall score. We use our baseline training approach for these experiments.

Prediction head configuration	Score	SST	Para	STS
Baseline	0.388	0.326	0.630	0.206
Para and STS: SBERT modification	0.405	**0.326**	**0.655**	0.234
All: Shared layer	0.363	0.308	0.569	0.212
STS: Cosine similarity with MSE loss	0.397	**0.326**	**0.655**	0.209
STS: Cosine similarity with correlation loss	**0.410**	**0.326**	**0.655**	**0.248**

Table 1: Results on dev set for different prediction head configurations.

Table 1 shows that the SBERT modification of concatenating the sentence embeddings u, v with their absolute value element-wise difference $|u - v|$ increases performance relative to the baseline on both the Para and STS tasks. Further, implementing cosine similarity with correlation loss also improves STS performance, beating MSE loss which only slightly outperforms the baseline. This is perhaps because adding cosine similarity between u, v gives the model another notion of semantic closeness aside from what it learns within the linear layer. As cosine similarity measures the angle between two vectors, the training process will make two similar sentences have similar directions in its high dimensional embedding space. Combining cosine-similarity with linear layers allows our prediction head to measure both the distance and angle between two sentence embedding vectors. We are surprised to find that adding a shared layer was ineffective, as it reduces the model's overall performance. We had believed that the layer could encode common knowledge across the tasks, allowing the prediction heads to focus on the nuances of their respective tasks. However, it looks as if the shared layer just adds more confusion, perhaps providing conflicting signal with the prediction head layers. Overall, we find that SBERT + Cosine similarity with correlation loss on STS gives the best overall dev performance, so we will keep these two updates in later experiments.

Training Approach		Score	SST	Para	STS
MLM	SST and Para	0.399	0.311	**0.646**	0.240
	SST and STS	0.375	0.306	0.620	0.198
	Para and STS	**0.405**	**0.325**	0.645	0.**244**
	All	0.380	0.309	0.637	0.193
Multi-task Round Robin	Sequential	**0.723**	0.512	**0.844**	**0.814**
	Drop	0.657	**0.513**	0.788	0.669

Table 2: Results on dev set of models either pretrained on MLM or Multi-task finetuned.

We see in Table 2 that downstream performance following further pretraining greatly depends on the dataset used. Pretraining on SST and STS and, surprisingly, cross-domain pretraining on all datasets, yields lower overall performance than the baseline. This may be because the texts are conflicting in grammatical structure or context in ways that hurt the model's performance. However, pretraining on Para and STS datasets improved performance over the baseline, but was still lower than using the baseline embeddings with our prediction head amendments.

Models that are multi-task finetuned clearly outperform those that are further pretrained. This performance difference was unexpected but is likely because multi-task finetuning adjusts both BERT and prediction head weights according to all tasks in unison. With further pretraining, the model may have just learned the datasets instead of useful context for the tasks. Sequential multi-task finetuning performs best as it uses the maximum amount of data (concatenation of 3 tasks' batches) for each epoch. With this approach, the model is exposed to a broader range of text and is always evenly exposed to the 3 tasks. As Drop-Round-Robin randomly selects a task for each batch and drops the rest, it loses potentially helpful information encoded in the other batches.

		Multi-task Sequential Round Robin			
		Score	SST	Para	STS
MLM	SST + Para	**0.652**	**0.518**	0.792	0.647
	SST + STS	0.650	0.516	0.791	0.642
	Para + STS	0.644	0.506	**0.797**	0.630
	All	0.647	0.492	0.785	**0.663**

Table 3: Matrix of results on dev set of models pretrained on MLM then Multi-task finetuned.

Combining a further pretrained model with multi-task finetuning improves performance over further pretraining, but still lags behind multi-task finetuning. This highlights how multi-task finetuning is the key to better downstream performance, and that the embeddings pretrained with MLM are not as robust as those from the baseline. We experimented with pretraining for more epochs and lower learning rates to avoid the catastrophic forgetting problem, but multi-task finetuning still outperformed pretraining. This was surprising as Sun et al. found that multi-task finetuning improved performance less than further pre-training, but it may be because they used different but related datasets to pretrain while we used the same ones for both pretraining and finetuning.

Training Approach	Dev Set				Test Set			
	Score	SST	Para	STS	Score	SST	Para	STS
1. Baseline	0.388	0.326	0.630	0.206				
2. Sequential Finetuning	0.669	0.356	0.809	**0.843**				
3. MLM	0.405	0.325	0.645	0.244				
4. Multi-task	**0.723**	0.512	**0.844**	0.814	0.724	0.537	0.843	0.793
5. MLM + Multi-task	0.652	**0.518**	0.792	0.647				

Table 4: Results from best models for each training approach.

Table 4 demonstrates the power of careful model finetuning. Training models with multi-task sequential round robin finetuning yielded the best overall score on the dev set, and equally well on the test set, although the STS task slightly overfit. Sequentially finetuning the model on SST, Para then STS also performed well. However, its performance was likely hindered by catastrophic forgetting as parameter updates that benefitted the SST task were forgotten during Para and STS updates. This is likely why Multi-task finetuning outperformed it as the gradient updates were made to minimize the summed loss across the 3 tasks. Its good performance perhaps follows analogously to when people apply knowledge from related tasks to help learn a new task (Liu et al., 2019). The multi-task finetuning process allows our model to learn to generalize its embeddings to all 3 tasks. Sun et al.'s training framework of combining pretraining and multi-task finetuning did improve performance from the baseline. However, further pretraining seems to have hindered more than it helped.

6 Analysis

6.1 Error analysis on best model

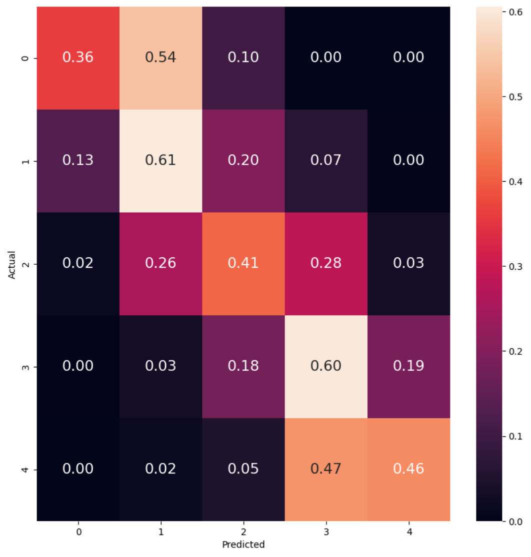

Figure 3: Confusion Matrix of SST predictions from our best model

SST
Text: "the film is quiet, threatening and unforgettable."
Label: 4; **Predicted:** 1

The confusion matrix on the left shows that the model performs relatively well when predicting 1 or 3, and worse on the extremes. This is surprising as extremely negative or positive text should be easier for the model to identify due to, for example, the strength of the language used. However, we see in the example above that this is not necessarily true. The model predicted the text as slightly negative when it was actually a very positive statement. It probably associated "quiet" and "threatening" with having negative connotations when they were the features that made the movie "unforgettable".

Nevertheless, the model seems to favor predicting sentiments in the middle, between 1 and 3. We see this as most very negative text are actually predicted as relatively negative by the model. Similarly, most very positive text is classified as relatively positive. The model is biased towards predicting a middle ground.

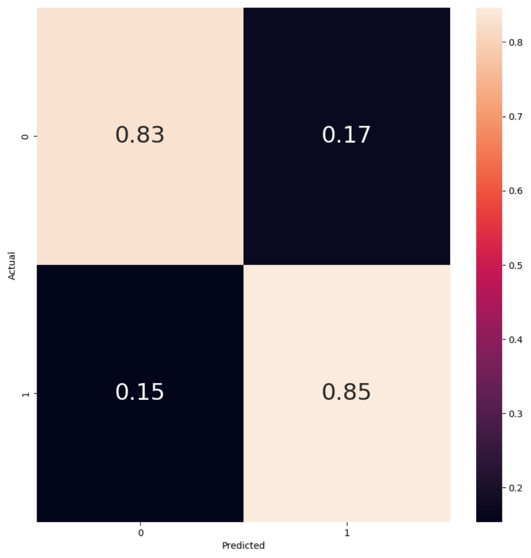

Figure 4: Confusion Matrix of Para predictions from our best model

Para
Text A: "how did you learn java?"
Text B: "how do i learn java internals?"
Label: 0; **Predicted:** 1

Overall, the model performs well on paraphrase prediction. From the confusion matrix, recall is very similar to accuracy. A lot of the sentence pairs the model got wrong were very subtle. For example, at first glance, texts A and B are similar as they both talk about learning Java. However, A asks about one's experience while B asks about how to learn one aspect of Java. This difference would require the model to encode more of the sentence's grammatical structure through parts-of-speech tagging, for example. The pairs of text the model is currently predicting incorrectly may require more explanatory power through additional layers or task-specific feature engineering in its prediction head.

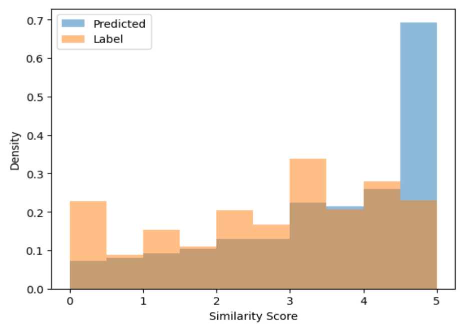

Figure 5: Histogram of the distribution of predicted STS Dev Similarity Scores from our best model

STS
Text A: "on monday, as first reported by cnet-news.com, the riaa withdrew a dmca notice to penn state university's astronomy and astrophysics department."
Text B: "last thursday, the riaa sent a stiff copyright warning to penn state's department of astronomy and astrophysics."
Label: 1.75
Predicted: 5

The above histogram shows the model is heavily biased to predicting scores close to 5 (similar sentences), and neglects predicting that sentences are not similar. In fact, the distribution of predictions is generally upward biased. We see an example above where both texts do talk about the same event - a DMCA notice. However, text A reports on the notice's withdrawal while text B talks about it being sent. The model placed too much focus on both sentences mentioning this notice/warning but failed to identify key words like "withdrew" and "sent" that indicate the texts are not similar. This may be because of our choice of loss function. Although correlation loss performs well, it may cause the predicted similarities to not necessarily be close to true labels, but parallel them. For example, labels [1,2,3] and labels [3,4,5] may have a high correlation but poor MSE loss. Thus, it is worth further exploring this tradeoff when using correlation loss. Also, it is important to determine whether correlation is a good evaluation metric on the semantic similarity analysis task that may cause the same problem as us.

7 Conclusion

We find that multi-task finetuning is the key to extracting better performance from a model. However, carefully engineering the task-specific prediction heads is also essential as simple modifications can yield big improvements in downstream performance. For example, we showed that as Para and STS both measure some notion of pairwise sentence closeness, incorporating embedding distance measures in different ways lifted performance beyond our baseline. However, we note that our approach did not exhaustively try every combination of prediction head configurations. We did not pursue this as there were at least 85 different combinations of prediction heads and training approaches to explore. With more time, the exact optimal configuration for our downstream tasks could be found following an exhaustive search or an ablation study.

As to the training approaches we investigated, multi-task round robin finetuning performed better than sequential finetuning, as the latter falls victim to the catastrophic forgetting problem, where the model forgets previously trained tasks. Pretraining using the MLM objective did not yield significant performance improvements, even when paired with multi-task finetuning. While this does not align with Sun et al. (2019)'s findings, furter pretraining might still be useful if the model is trained on multiple similar datasets as that of the downstream tasks.

Further, as this paper focuses on examining the impact of different training approaches, we maintained the same set of hyperparameters for consistency. There is tremendous scope for finding hyperparameters that work well for a given training approach. Learning rates played a big role and we saw sizeable performance differences with small learning rate changes in our experiments. Another example is for adding shared layers or additional prediction head layers. We found that adding a second layer, that halved the input dimensionality, to each prediction head helped for some tasks. There may have been some configuration of number of layers and their dimensionalities that would have improved results. With more time, we would explore this further.

In general, our current best model almost doubles its overall dev score compared to the baseline model. We have shown that with careful selection of prediction heads and using multi-task finetuning, a simple model can be trained to generate robust sentence embeddings for a variety of tasks.

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*.

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

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

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

CS224N Course Staff. 2023. CS224N Default Final Project: minBERT and Downstream Tasks.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to Fine-Tune BERT for Text Classification?

A Appendix

A.1 Multi-task finetuning round-robin algorithms

Algorithm 1: Sequential Round Robin, adapted from Liu et al. (2019)

```
Set the number of epochs to train: $epochs$ ;
We have total 3 tasks, and $t$ is the task number;
for $t$ in $1, 2, 3$ do
    Pack the dataset of task $t$ into mini-batch as a Data-Loader: $D_t$
end
for epoch in 1,2,... $epochs$ do
    for batch in $Zip(D_1, D_2, D_3)$ do
        $batch_{SSTS}, batch_{Para}, batch_{STS} \leftarrow batch$;
        1. Compute cross-entropy loss for SST using $batch_{SST}$, compute gradient then update
           the model;
        2. Compute binary cross-entropy loss for Para using $batch_{Para}$, compute gradient then
           update the model;
        3. Compute correlation loss for STS using $batch_{STS}$, compute gradient then update the
           model;
    end
end
```

Algorithm 2: Drop Round Robin, adapted from Liu et al. (2019)

```
Set the number of epoch to train: $epochs$;
for $t$ in $SST, Para, STS$ do
    Pack the dataset of task $t$ into mini-batch as a Data-Loader: $D_t$
end
for epoch in 1,2,... $epochs$ do
    for batch in $Zip(D_{SST}, D_{Para}, D_{STS})$ do
        $batch_{SST}, batch_{Para}, batch_{STS} \leftarrow batch$;
        randomly choose $a$ from {1, 2, 3};
        if $a = 1$ then
            Compute cross-entropy loss for SST using $batch_{SST}$, compute gradient then update
              the model;
        else if $a = 2$ then
            Compute binary cross-entropy loss for Para using $batch_{Para}$, compute gradient then
              update the model;
        else if $a = 3$ then
            Compute correlation loss for STS using $batch_{STS}$, compute gradient then update the
              model;
    end
end
```