Methods to Improve Downstream Generalization of minBERT

Stanford CS224N Default Project

Ramgopal Venkateswaran
Department of Computer Science
Stanford University
ram1998@stanford.edu

Abstract

We investigate a broad suite of techniques to create robust embeddings with good down-stream performance across tasks when fine-tuning BERT. Techniques we explore include data processing and augmentation related changes, adopting the two-stage strategy of linear probing then fine-tuning, architectural improvements such as weight-sharing, loss function improvements such as contrastive loss and multiple negatives ranking loss, and regularization penalties such as smoothness-inducing adversarial regularization, among others. We also experiment with some further post-training techniques such as stochastic weight averaging and knowledge distillation. Ultimately we find that data is king - the biggest performance boosts came from better data processing, additional pre-training, re-weighting data, or simply adding more data. We also see that architectural improvements can sometimes bring about step changes (as with some changes in the paraphrase task arch) but even otherwise bring consistent incremental gain. Loss-related methods were more hit-or-miss, and so were post-training methods (aside from ensembling).

1 Key Information to include

Mentor: Cheng Chang • No External Collaborators • Project is not shared across classes

2 Introduction

BERT (Devlin et al., 2019) is a transformer-based model that significantly improved upon existing language models at its time of release, producing robust embeddings that could be fine-tuned to achieve state-of-the-art performance on a variety of downstream tasks. We investigate a suite of methods that can be applied when fine-tuning a pre-trained BERT model to improve generalization performance, and understand how well they can complement each other when to maximize performance on three tasks: sentiment analysis, paraphrase detection, and semantic textual similarity.

We broadly classify our improvements into five categories of approaches, include data-related approaches (e.g. changes in data weighting, additional data, etc.), architecture-related changes, loss function related changes (e.g. adding a contrastive learning loss, adversarial loss, etc.), hyperparameter tuning, and post-training methods (e.g. ensembling, stochastic weight averaging, distillation). We will more comprehensively list out and explain the different techniques in section 4.

3 Related Work

Our work is based on extending the pre-trained BERT embeddings first described in Devlin et al. (2019), and we use ideas from many of the subsequent works that look to fine-tune these embeddings. **Data-Related**: Sun et al. (2020) shows that a stage of additional pre-training with the masked

language modeling and next-sentence prediction objectives before fine-tuning can produce better embeddings. A two-stage strategy of just linear probing and then full fine-tuning (LP-FT) has also been shown to improve performance on out-of-distribution data on certain vision models (Kumar et al., 2022); we draw inspiration from this to try a similar two-stage approach where we do one round of fine-tuning with the BERT model's parameters frozen, and then start full fine-tuning from that initialization. Separately, many works have also shown that training on additional data and tasks can benefit prediction performance on our main tasks, such as Reimers and Gurevych (2019) and Jiang et al. (2020).
Loss-Related: Henderson et al. (2017) uses the idea of multiple negatives ranking loss in their task of training a model for response suggestion. The loss term considers the intended response as the positive and other responses as negatives and maximizes the probability of the correct response, using a simple dot product between an input embedding and candidate response embedding as the scoring function to efficiently compute the suitability of each candidate response. We apply these ideas to the tasks of paraphrase detection and semantic textual similarity. Gao et al. (2022) identifies a simple and effective contrastive learning technique of adding a loss function that maximizes the similarity between two embeddings generated from the same input, but with different dropouts applied. Jiang et al. (2020) introduces a principled way to combat overfitting by adding a regularizer that encourages smoothness by penalizing cases where we can find an adversarial input that is close (based on some distance metric, e.g. the L_∞ norm) but the output logits differ greatly.
Post-Training Methods: Stochastic weight averaging (Izmailov et al., 2019) was previously found to improve generalizability in vision; after the model converges (or is close to convergence), it keeps track of a running average of model weights and uses that for prediction instead of the final model weights (the details of this idea are further explained in the following sections). We also look into the idea of knowledge distillation from an ensemble of separately trained models to a new model, an idea applied to BERT in Liu et al. (2019). While he focus of this work was to improve the serving efficiency of knowledge distillation (since serving an ensemble is costly), recent work by Allen-Zhu and Li (2023) explores the idea that even self-distillation (distilling a previously trained model of the same size to itself, without ensembling) can result in improved performance, with some theoretical reasoning for why this might be the case. Motivated by this, we explore if either self-distillation or distillation from an ensemble of models can improve model performance.

4 Approach

4.1 Baseline

The datasets we use in our baseline are the Quora question-answering dataset for paraphrase detection, the STS dataset for similarity scoring, and the Stanford Sentiment Treeback (SST) dataset for sentiment classification. These are described in more detail in the project handout [1].

The baseline fine-tunes (all parameters of) the BERT model (Devlin et al., 2019) with the task heads described below - note that for all task heads, the input is the pooled BERT embedding corresponding to the `<CLS>` token. All linear layers described have a dropout of 0.3, the learning rate for fine-tuning is $1\text{e}{-5}$ and training is done for 10 epochs.

- SST: Linear layer with softmax activation, trained with cross-entropy loss.
- Paraphrase: Siamese network - each sentence embedding is fed into a linear layer to get an intermediate embedding (size 32) followed by a dot product with and sigmoid activation, trained with binary-cross-entropy loss.
- STS: Siamese network similar to above, but the output is the cosine similarity of the intermediate embeddings scaled by a factor of 5.0, trained with MSE loss.

We will now describe our approach to modifying the baseline.[2]

[1]Link to default project handout
[2]All approaches based on existing literature have the relevant papers accordingly cited along with explanations on the relation between the paper's work and ours. Other approaches are original. All code is self-written unless otherwise mentioned.

4.2 Data

Even Batching: We modify the shuffling logic to more evenly distribute the optimization steps across the tasks - the default baseline logic optimizes one epoch of each task at a time, in round-robin fashion. For a given epoch, this could result in the model's weights converging towards the optimum for that single task whose epoch of data is currently being consumed, with the update directions then changing sharply when the epoch of data from the next task is consumed. We instead shuffle at the batch level instead of the epoch level; do this by splitting each task's data into batches of equal size, and then shuffling the batches across all tasks. We then consider one epoch to be an epoch after training on the full shuffled dataset from all tasks' data sources.

Loss Rescaling: We observe that the training dataset sizes are very imbalanced - the SST dataset has a size of approximately 8500 and the STS dataset has a size of approximately 6000 - on the other hand, the Quora dataset has a size of nearly 210000. This suggests that if we optimize each task in the same way, we will tend to over-optimize on the Quora dataset potentially at the expense of worse performance on the STS and SST datasets. To counteract this, we down-weight the Quora data by a factor of 10 to bring it to approximately the same order of magnitude as the other two datasets - that is, the loss computed for the paraphrase task is multiplied by a factor of 0.1 (or equivalently, we can consider the learning rate for this data is a factor of 10 lower than the other two datasets). This ensures that we still see all of the data from the Quora data in an epoch (as opposed to dropping some part of the data), while keeping its effect on the model weights comparable to the other two tasks.

Additional AllNLI Data: The AllNLI dataset is a dataset that comprises pairs of sentences that entail each other and can therefore be used to model textual similarity. It is formed by a combination of the Stanford NLI dataset (Bowman et al., 2015) and the MultiNLI dataset (Williams et al., 2018). We add AllNLI data to the existing similarity scoring task - i.e. in terms of model predictions, both the data from the existing STS dataset and the added AllNLI dataset are generated in exactly the same way, with the same architecture. However, the loss for AllNLI is computed differently - we use only the multi-negatives ranking (MNR) loss function for AllNLI, while we use a combination of cosine similarity loss, MNR loss, and unsupervised SimCSE loss for the STS data (these are all described in more detail in subsection 4.4).

LP-FT: Linear probing then fine-tuning refers to the strategy of the two-step strategy of just linear-probing (i.e. just tuning the task-specific parameters) then full fine-tuning to improve generalizability, as described in Kumar et al. (2022). In this case, we first tune just the task heads, and then fine-tune the full model after initializing it from these weights. While the approach described in the paper was mostly tried for vision models and aimed to improve out-of-distribution performance, we attempt it here to see if it can also improve generalizability across tasks. We note that our task heads are not purely linear (except for the baseline), and our usage of the term "linear probing" here in the context of freezing the BERT model weights while fine-tuning the task heads does not fully match the general use of the term.

Additional Pre-training: We apply additional pre-training as described in Sun et al. (2020). We do not do this on AllNLI data, but simply on the three task datasets (SST, STS, Quora). We train on purely the masked language modeling objective with AdamW and a learning rate of $5e-5$ for 3 epochs. We first pre-process the data by combining the sentences in all three datasets and shuffling them, to feed as training data. The training code was directly adapted from the sentence-transformers library [3].

4.3 Architecture

We improve upon the baseline architecture to get the architecture shown in figure 1. The key changes are described below:
Weight-sharing: We have two fully-connected linear layers (with relu activation in between) that served as a shared architectural component for all three tasks. The goal of weight-sharing is to enable easier transfer learning across the tasks, by allowing knowledge transfer through an explicit shared representation used for all tasks (in addition to the knowledge transfer that can happen through the fine-tuning of the common BERT model parameters). These are denoted as Shared FCN layers 1 and 2 in figure 1. The output of shared FCN layer 2 is used as an input for all tasks, by concatenation with

[3] `https://github.com/UKPLab/sentence-transformers/blob/master/examples/unsupervised_learning/MLM/`

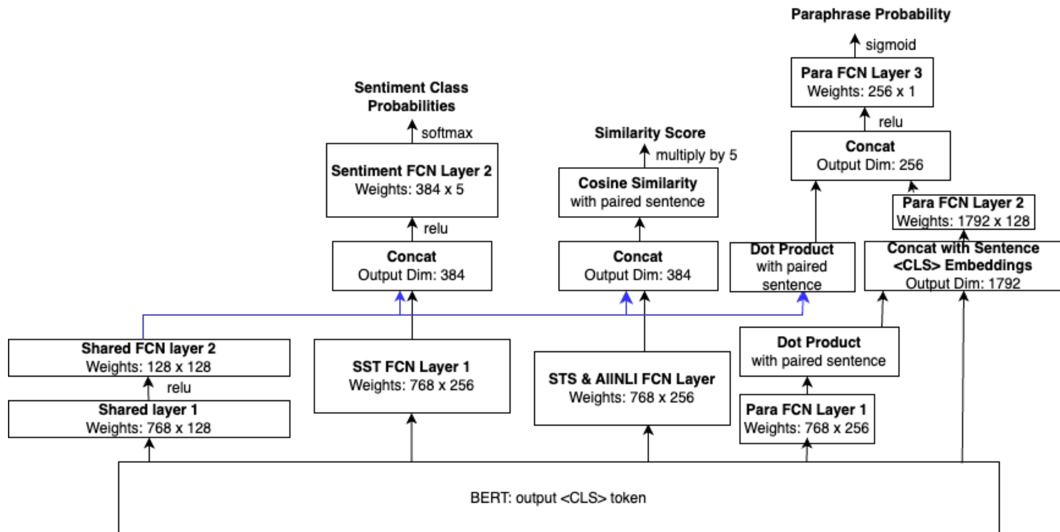

Figure 1: Diagram of the architecture yielding the best performance across tasks. The AllNLI task (discussed further below) uses the exact same archs as the similarity task.

the outputs of each task's own dedicated architectural component, followed by further transformations after the transformation.

Taskhead Non-linearities: Task heads are no longer linear transformations of the BERT model output - each task has at least one non-linear component (coming from the ReLU activation in the shared arch), with the sentiment arch and paraphrase archs having additional non-linear activations.

Structural change for paraphrase: While the sentiment task continues to predict softmax-probabilities and the similarity task continues to use a Siamese style network (outputting an embedding, with the loss computed by similarity scores) like in the baseline, the paraphrase task is no longer computed as a pure Siamese network. We first note that we still apply an element-wise dot product to the representations of the pair of sentences, at the end of shared FCN layer 2 and para FCN layer 1. If we simply summed up the element-wise dot products and applied the sigmoid transformation at the end, this would be equivalent to a Siamese network (albeit using a dot product instead of cosine similarity). However, we also concatenate these element-wise products with direct embeddings (coming from the BERT model's `<CLS>` token) and apply further non-linear transformations to them, allowing the network to learn functions that can be expressed in a more complex way than simply the dot product of the two sentences' embeddings. At the end of these transformations, we still predict a logit that undergoes a sigmoid transform and minimize the binary cross-entropy loss (as we did in the baseline).

4.4 Loss

Multiple Negatives Ranking Loss: MNR loss (Henderson et al., 2017) is useful when we are provided labeled data with several pairs of sentences that are positive labels, and we would like to predict whether a given pair will be positive or not. We use this for the semantic textual similarity task, and will explain it in this context. We first compute the likelihood that each pair of sentences in the batch is similar (here, we consider not just the given positively labeled pairs that form each sample in the batch, but also combinations of sentences that come from two different samples). For each sentence, we then look to minimize the negative log likelihood (that is, maximize the likelihood) that the original sentence it was paired with (as a positively labeled sample) is the most similar sentence among all the sentences in the batch. We can calculate this as $\mathcal{L}_{\texttt{MNR}}(x, y) = -\frac{1}{K}\sum_{i=1}^{K} \log \frac{e^{S(x_i, y_i)}}{\sum_{j=1}^{K} e^{S(x_i, y_j)}}$ where K is the batch size, and (x, y) denotes the K pairs of positively labeled sentences (x_i, y_i). S here is the function modeling the similarity between two sentences - the key is to ensure that S is efficiently computable - if each computation $S(x_i, y_j)$ required a forward pass, this would need K^2 full forward passes per batch which would be very expensive. In our case,

$S(x_i, y_j)$ is simply the cosine similarity of the vectors $m_{sim}(x_i)$ and $m_{sim}(y_j)$ where we denote the output of our similarity taskhead by m_{sim}, because of how we structured it as a Siamese network. Therefore, we would only need $2K$ forward passes (which we need regardless to compute these embeddings) and can just do K^2 dot products (after normalizing the embeddings) to compute the loss, instead of K^2 full forward passes - this is much more efficient, and can be computed easily for reasonable batch sizes (ours is just $K = 16$).

We train the AllNLI data purely with MNR loss. For STS data, we use MNR loss to augment the regular cosine similarity loss. However, note that we cannot directly use the MNR loss in the given form because not all of our samples in the STS data are positively labeled. Instead, they have different degrees of similarity, so we introduce a change to the loss function to adapt it to our use case. We can write it as

$$\mathcal{L}_{\texttt{MNR-weighted}}(x, y) = \frac{1}{K} \sum_{i=1}^{K} \frac{e^{l(x_i, y_i)}}{\sum_{j=0}^{5} e^j} \log \frac{e^{S(x_i, y_i)}}{\sum_{j=1}^{K} e^{S(x_i, y_j)}}$$

where $l(x_i, y_i)$ represents the label of the pair (x_i, y_i) (with 0 indicating no similarity and 5 indicating full similarity). This ensures that we weight pairs that are very similar much higher as opposed to pairs that are dissimilar (e.g. a pair with similarity score of 5 has e^4 times more influence on the MNR loss than a pair with similarity score of 1), since pairs that are very similar are "true positives" while it is expected that pairs with lower similarity should not have as much influence on the MNR loss. Implementation-wise, we adapt the existing implementation of MNR loss in the Transformers library [4] to our use case, making the modifications mentioned above.

Smoothness-Inducing Adversarial Regularization: We try the method proposed in Jiang et al. (2020) to combat overfitting by adding a regularization term that encourages the loss surface to be smooth. We try this for all tasks, but will illustrate it in more detail for the STS task, and describe the differences for the other two afterward. The goal of this regularizer is to ensure that all points within at most a certain small distance of a given BERT sentence embedding (call the embedding e and the distance d) have approximately the same output. d is measured in the L_∞ norm in the paper (i.e. d is the maximum absolute difference in the elements of the new embedding and the old one), and we use this as well. To do this, for each datapoint we must first compute the embedding e_{adv} that approximately maximizes the difference in outputs between itself and e. The difference in outputs is quantified through a loss function l_s - for the STS task, this is the mean-squared error. That is, we want to approximately find e_{adv} such that $e_{adv} = \texttt{argmax}_{\|e-e'\|_\infty \leq d} l_s(m_{sim}(e), m_{sim}(e'))$. This quantity is maximized via a small number of steps of projected gradient ascent - note that this is the same as gradient ascent but with a projection step at the end to project our new embedding vector into the valid region of $\|e - e'\|_\infty \leq d$. Also note that we can only do a small number of steps here because each step requires both a forward pass and a backward pass. Therefore, each step increases the computation time of the overall learning process significantly; we use just a single step, which is the default used in the paper's implementation. Once we compute e_{adv}, we then compute the regularization penalty as $l_s(e, e_{adv})$ and add it to our existing loss. The loss for the sentiment classification and paraphrase detection tasks are the symmetrized KL divergence instead of mean-squared error (since these are classification tasks and not regression tasks). The code to implement this algorithm is adapted from the code in Jiang et al. (2020) [5].

Unsupervised SimCSE This is a simple technique from Gao et al. (2022) that ensures that two forward passes generate embeddings that are close to each other (they don't always generate the same embedding because of dropout). It does so by doing two forward passes, and then applying an equivalent of the MNR loss function to the embeddings obtained from the forward passes (where the positive pairs are now the copies coming from the same input sentence, and the similarity function is the cosine similarity). We make some minor modifications to our implementation of the MNR loss function to also implement this, and modify it to use the dot product instead of the recommended cosine similarity, since it seemed to have comparable out-of-the-box performance in Gao et al. (2022) without requiring as much tuning.

[4]https://www.sbert.net/examples/training/nli/README.htmlmultiplenegativesrankingloss

[5]https://github.com/namisan/mt-dnn

4.5 Hyperparameters

The hyperparameters we tuned included the learning rate, which we tuned from $2e{-5}$ to $1e{-5}$) and to which we added a linear decay (of $4e{-7}$ per epoch). We also tuned the batch size, and experimented with both including and excluding dropout layers in the task head architectures (keeping the main BERT model's dropout rate at 0.3, and just adding or removing dropouts to the task architectures above that). Finally, we also tuned the size of the fully connected layers above the BERT model layers.

4.6 Post-training

Stochastic Weight Averaging: We tried running a few epochs of SWA at the end of our model training, to see if it improves generalizability. In vision models, SWA has previously shown improvements in generalizability with the hypothesis being that averaging weights over a few epochs results in weights that are closer to the center of the low-loss region rather than the edges (Izmailov et al., 2019) [6]. We do this for 5 epochs with a learning rate of $1e{-5}$, and implemented this with Pytorch's SWA modules.

Knowlege Distillation: We first train a model, and then attempt to use its predictions as soft labels for training another model. The main motivation for this is the work in Allen-Zhu and Li (2023) which shows that self-distillation can sometimes lead to performance improvements, with the reasoning being that the distillation process serves as a form of ensembling. We do this by adding the loss compared to the distillation soft labels as a separate term in all our tasks' loss functions, weighting it as $2/3$ of the main task's loss. We also attempt this distillation where the soft labels are not the output of a single model but rather an ensemble of models, to see if adding these soft labels can allow it to out-perform a single model. We note that there is also some prior work with knowledge distillation for BERT in Liu et al. (2019).

Ensembling: We also try three forms of ensembling - simple ensembling where we compute the prediction as the average prediction of a set of models for all tasks, selective ensembling where we select some combinations of models in the ensemble for some tasks and different combinations for others, and a simple regression-based ensembler, for which we split the existing dev set into two parts (dev-train and dev-dev), and train a logistic regression classifier to learn ensemble weights on dev-train, evaluating on dev-dev. The classifier simply takes the logits from each model and multiplies each by a learned weight before summing them and applying the appropriate activation function (sigmoid in the case of the paraphrase task, and softmax in the case of SST - we did not do this for the STS task).

5 Experiments

5.1 Data

We have included the full description of the main three data sources in subsection 4.1; we also use entailment pairs from AllNLI data as described in subsection 4.2 [7].

5.2 Evaluation method

During development, we use the average of the SST accuracy, paraphrase accuracy, and the normalized version of the Pearson correlation to determine the overall quality of the model, aligned with the project objectives and leaderboard criteria.

5.3 Experimental details

Models were trained for 20 epochs when used in the final ensemble, but for ablation studies and comparison purposes we terminate at 10 epochs - we generally find that in most cases, the best dev accuracy is achieved within 10 epochs which is why we expect that this will approximate the final values at 20 epochs well. All models were trained on a V100 GPU, which takes approximately 3 hours to train when excluding AllNLI data and 4.5 hours to train when including it.

[6]This link also has a good explanation: https://pytorch.org/blog/stochastic-weight-averaging-in-pytorch/

[7]The data is obtained from http://sbert.net/datasets/paraphrases

5.4 Results

The best single model, which we will call B1, includes the following combination of techniques described above: even batching, loss rescaling, additional AllNLI data, weight-sharing, taskhead non-linearities, structural changes for the paraphrase task, MNR loss (for AllNLI and STS), unsupervised SimCSE loss for the STS tasks, a $1e{-}5$ learning rate decayed by $4e{-}7$ per epoch, no dropout layers in the taskheads, and an FCN layer size of 256. We will denote the equivalent model without AllNLI data by B2, and the equivalent model that both excludes AllNLI and also includes taskhead dropout layers with a dropout rate of 0.3 as B3. B3 is the baseline for most of our ablation studies.

Experimental results and ablations of different combinations of techniques are given in figure 2. The best final result was obtained by applying selective ensembling - we ensemble B1 and B2 for the Paraphrase and STS tasks, while we ensemble B1, B2, B3 and B4 (which is B3 with additional distillation loss) for the SST task. This ensemble is the top row in figure 2. We obtain an **overall test set accuracy of 0.771, comprising an accuracy of 0.531 on sentiment classification, 0.854 on paraphrase detection, and 0.855 on semantic textual similarity** for this ensemble. For reference, the second row in figure 2 is a simple ensemble of B1, B2, B3, and B4 for all tasks. In 3, we also give the results of applying LP-FT on top of the baseline, in comparison with pure fine-tuning.

Row No.	Model Description	SST Dev Acc	Para Dev Acc	STS Dev Acc	Overall Score
	Best Individual Models				
1	Best Model (B1)	0.522	0.843	0.857	0.765
2	Best Model - AllNLI (B2)	0.503	0.851	0.857	0.761
3	Best Model - AllNLI + Task-Dropout = 0.3 (B3)	0.517	0.829	0.816	0.751
	Data-Related Ablations & Inclusions				
4	B3 - Even Batching	0.492	0.780	0.740	0.714
5	B3 - Additional Pre-training	0.503	0.806	0.767	0.731
6	B3 - Loss Rescaling - LR Decay	0.507	0.818	0.793	0.740
	Architectural Ablations				
7	B3 - Shared Arch - Non-linearities + FCN Size = 128	0.518	0.798	0.806	0.740
8	B3 but only one task: STS	N/A	N/A	0.806	N/A
9	B3 but only one task: Para	N/A	0.805	N/A	N/A
10	B3 but only one task: SST	0.529	N/A	N/A	N/A
	Loss Function Related Ablations & Inclusions				
11	B3 - Unsupervised SimCSE	0.500	0.821	0.815	0.743
12	B3 - MNR Loss	0.522	0.825	0.807	0.750
13	B3 + Adv Loss on STS (weight 50)	0.510	0.795	0.747	0.726
14	B3 + Adv Loss on all (weight 1)	0.520	0.766	0.797	0.728
	Post-Training Methods & Best Ensembled Models				
15	Best Selective Ensemble of Models	0.531	0.858	0.870	0.775
16	Best Simple Ensemble of Models	0.531	0.855	0.865	0.773
17	B3 + Distillation from Ensemble (B4)	0.518	0.820	0.807	0.747
18	B3 + SWA	0.518	0.829	0.821	0.752

Figure 2: Results from all model ablation studies, as well as additional experiments (except for LP-FT). The notation "BN - X" refers to a model that corresponds to model BN but with technique X ablated, while the notation "BN + X" refers to trying the technique X on top of BN.

Model Description	Dev SST	Dev Para	Dev STS
Fine-tuned Baseline (FCN Size = 32)	0.500	0.579	0.627
Fine-tuned Baseline + Para Change + FCN Size=64	0.501	0.804	0.631
LP-FT + Para Change + FCN Size=64	0.501	0.804	0.631

Figure 3: Results from milestone report, including baseline accuracies and the accuracy when trying out LP-FT and comparing it to pure fine-tuning. We did not see gain in Dev Accuracies from LP-FT.

In general, our quantitative results are along expected lines, with some exceptions. We first note that data-related methods performed very well, particularly even batching and additional pre-training;

comparing to B3, ablating each of them gives significantly worse overall performance, by 4.92% and 2.66% respectively. Loss rescaling (or, equivalently, data reweighting) and adding additional AllNLI data also boost performance, though a surprising finding is that adding ALLNLI data does not necessarily improve performance on the STS data (which flows through the exact same task-head architecture); instead, it maintains STS performance while significantly improving SST dev accuracy by $+3.77\%$ with some trade-off against paraphrase performance as well (-0.95%).

Our architectural improvements also benefit the model, as seen from the ablation in row 7 of figure 2; figure 3 also suggests that the structural change to the paraphrase arch on top of the baseline was critical in improving performance on that task. Row 8 to 10 where we only learn a single task at a time also show us that both Para and SST benefit from the joint learning of all tasks, while there is negative transfer to the SST task. Loss function related improvements were a mixed bag - unsupervised SimCSE helped all tasks, while MNR loss, which was only applied to the STS task, helped it by 1.1% but at the expense of performance on the SST and paraphrase tasks ($-1.0\%, -0.5\%$).

Ensembling was very helpful, with a 1.3% increase in overall accuracy over the best individual model and gains in all tasks; selective ensembling was slightly better than a simple ensemble. SWA yielded very marginal improvement over B3 (which may also be noise), and we could not reproduce any gain similar to ensembling by distilling ensembled predictions as soft labels, instead observing regression in the case. Finally, we note that removing dropout layers from all task-head architectures was very beneficial to both the paraphrase and STS tasks, while worsening the SST task.

6 Analysis

Overfitting: From the log loss and dev accuracy plots in figures 5 and 4 for the B1 model, we note that log loss for all tasks decreases over time, and dev accuracy generally increases or stays roughly constant for paraphrase and STS. However, SST dev accuracy gradually decreases starting around the 8th epoch, suggesting that we are overfitting to this task beyond that point, which makes sense given the low data volume (compared to paraphrase or STS augmented with AllNLI data). These curves also explain why we see B1 apparently significantly improve on SST dev accuracy compared to B2 - it is likely because B1's saved checkpoint occurred earlier in the optimization. B2 might have overall worse performance because it takes more epochs for the paraphrase and STS dev accuracies to climb to good values, by which point SST dev accuracy might have become very low.

Adversarial Regularization: This technique performed poorly compared to what we might have expected from the literature. By examining the adversarial loss for a few timesteps, the projected gradient ascent step doesn't often find a good noise vector that lies further away from the original embedding than the randomly initialized noise vector. This suggests that either further tuning of the projected gradient ascent step size and the norm used might be needed, or it could be tried at a different layer (e.g. adding noise to the `<CLS>` embedding instead of the input embedding). The performance is also not suprising in light of the fact that other common regularization techniques such as weight decay and task-head dropouts did not work well for this model.

Knowledge distillation from an ensemble of models was not able to beat single model performance. Further tuning of distillation loss weights might be needed, or it could be worth exploring whether adding the distillation loss to a separate task might instead be more useful; here, instead of directly constraining each task's head to reduce the ensembling loss, we could learn a separate task for distillation and look to improve generalization on the main tasks by indirect transfer learning through the shared architecture components.

7 Conclusion

By investigating a broad suite of techniques, we learn more about which combinations of them can work well for fine-tuning BERT on downstream tasks. We find that the biggest levers in improving performance were data-related, with some good gain from architectural and loss-related methods as well. Limitations include the fact that we did not very precisely tune hyperparameters for any one technique too much. Areas for further work include better understanding why adversarial regularization does not work well for our models, exploring using other embeddings from the BERT model besides the `<CLS>` token embedding, and existing techniques (like extending additional pre-training to AllNLI data, or unsupervised SimCSE to tasks besides STS).

References

Zeyuan Allen-Zhu and Yuanzhi Li. 2023. Towards understanding ensemble, knowledge distillation and self-distillation in deep learning.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference.

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

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2022. Simcse: Simple contrastive learning of sentence embeddings.

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.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. 2019. Averaging weights leads to wider optima and better generalization.

Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2020. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Ananya Kumar, Aditi Raghunathan, Robbie Jones, Tengyu Ma, and Percy Liang. 2022. Fine-tuning can distort pretrained features and underperform out-of-distribution.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019. Improving multi-task deep neural networks via knowledge distillation for natural language understanding.

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

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2020. How to fine-tune bert for text classification?

Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference.

A Appendix

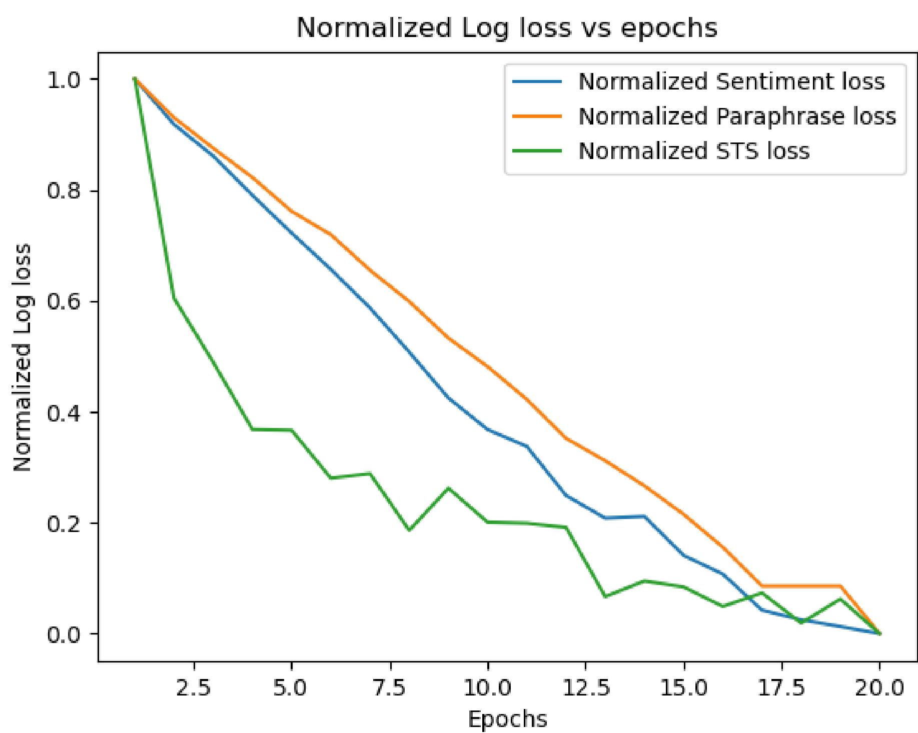

Figure 4: For our best model, we take the log-loss at each epoch for each task and normalize it (so the worst epoch would have a normalized loss of 0 and the best would have a loss of 1 - we plot this.

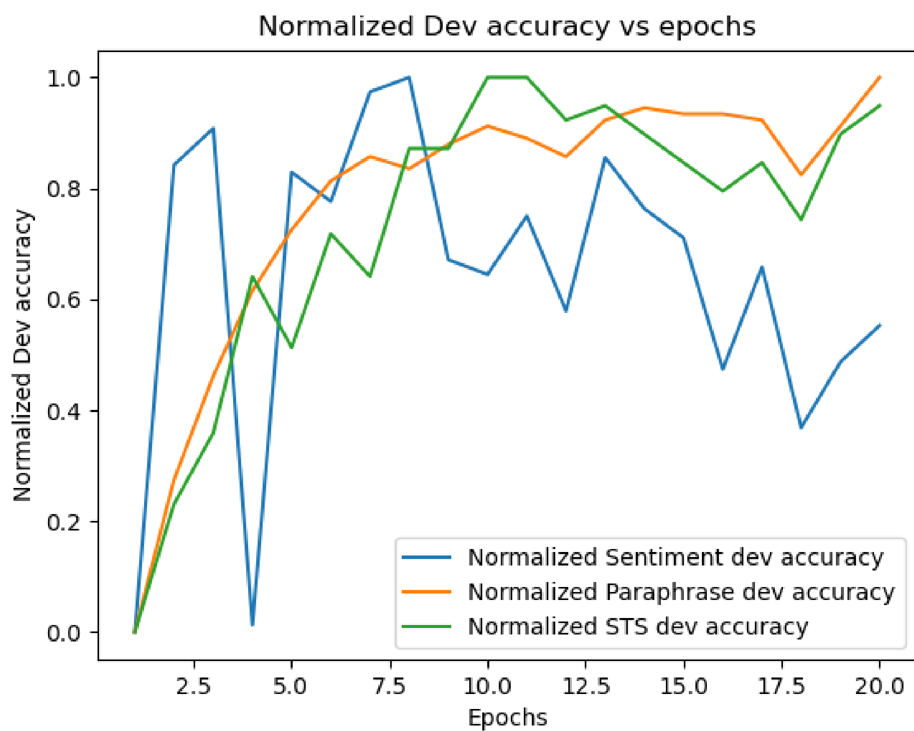

Figure 5: For our best model, we take the dev accuracy after each epoch for each task and normalize it (so the worst epoch would have a normalized accuracy of 0 and the best would have an accuracy of 1 - we plot this.