Pals for PALs: Exploring Extensions to Projected Attention Layers for Sentence-Level Tasks

Stanford CS224N Default Project

Lainey Wang
Department of Computer Science
Stanford University
lyw0015@stanford.edu

Abstract

In this project, I implement a multi-task model for the SST-5, QQP, and STS-B tasks that improves upon the performance of the BERT baseline from Devlin et al. (2018). I implement the projected attention layers (PALS) model as proposed in Stickland and Murray (2019), along with several additional variations: freezing BERT weights during training, pretraining BERT with sentence embeddings before finetuning, and adding SMART regularization Jiang et al. (2020). With a majority voting ensemble comprised of all these models, I achieve a significant performance increase over the baseline, reaching 3rd place on both the test and development set leaderboards.

1 Key Information to include

My mentor is Manasi Sharma. I have no external collaborators nor am I sharing projects.

2 Introduction

In this project, I address the NLP challenge of multi-task fine-tuning. Specifically, I focus on a model's ability to classify sentence sentiment (SST-5), predict paraphrase pairs (QQP), and detect semantic textual similarity (STS-B) while sharing BERT word embeddings.

The current state-of-the-art approach for these tasks relies on transfer learning, where a large language model is pre-trained on a language modeling objective to learn general semantic and syntactic information. That model is then fine-tuned on smaller, supervised downstream tasks like SST-5, QQP, or STS-B to produce state-of-the-art results (Liu et al., 2019; Yang et al., 2019). However, typical fine-tuning approaches require a new set of weights for each task and, if trained on multiple tasks, can suffer from catastrophic forgetting, where weights from the previous task are forgotten when training a new task, and consequently the old task's performance suffers (Mccloskey and Cohen, 1989). Thus, the main aim of this project is to implement a multi-task model fine-tuned on one set of BERT weights that can perform well across all three tasks.

My first attempt to improve upon the base BERT model is the projected attention layers (PALs) model proposed by Stickland and Murray (2019), which adds a task-specific, low-dimensional multi-head attention layer in parallel to each normal BERT layer. I also implement their proposed training scheduler, which samples tasks proportional to their training set size early in training before gradually evening out. Together, these two changes result in a huge improvement over default multi-task trained BERT.

I then experiment with the following techniques to further improve upon the PALS model:

1. Freezing most BERT weights during training so that the only layers trained are the output layer normalization layers, the pooling layer, and the task-specifc PALs functions. This is an **original** contribution, but is inspired by the performance of Adapter BERT from Houlsby et al. (2019).

2. Pretraining BERT on the MNLI and SNLI datasets with cosine similarity sentence embeddings before PALs fine-tuning. This pretraining approach is inspired by the Sentence-BERT (SBERT) models from Reimers and Gurevych (2019), but the addition of PALs layers is an **original** contribution.

3. SMART regularization from Jiang et al. (2020), which uses adversarial regularization and Bregman proximal point optimization to manage complexity and reduce overfitting.

Of these modifications, all lead to similar performance to the original PALs model when tested in isolation, with the SBERT model performing better on the SST-5 task, but worst on QQP and STS-B. Finally, we assemble an ensemble of all these models and the original PALs model to achieve our highest performing model.

3 Related Work

The primary advantage of multi-task learning is that models can take advantage of transfer learning effects where by training on one task, we can improve the model's performance on a second task and vice versa. Subsequently, many multi-task learning architectures focus on related tasks in similar domains to maximize the benefit from transfer learning. In Peng et al. (2020), the authors train a model specific to the biomedical and clinical text domains and outperform state-of-the-art transformer models by 2.0% and 1.3% respectively. Likewise, in Bi et al. (2022), to get better news recommendations, the authors train a model on the main new recommendation task and two additional auxiliary tasks, news category classification and named entity recognition, to result in improved performance.

However, in our case, the transfer learning benefits from multi-task learning may be less clear, since our tasks are in different domains. The Projected Attention Layers (PALs) model by Stickland and Murray (2019), which also operated in this setting, was very effective at using a single large base model to work with multiple less-related tasks. By adding a low-dimensional multi-head attention layer in parallel to each normal BERT layers, they were able to achieve performance on the GLUE benchmark comparable to finetuned BERT. However, Stickland and Murray found that while some tasks (like RTE) benefited from transfer effects and some tasks were agnostic to it (QQP, MNLI, QNLI), the two single sentence tasks suffered the greatest drop in accurancy (SST and CoLA), indicating that there was likely some learning interference. Since this project is not interested in the transfer effects for the RTE task and wants to maximize SST performance, I wanted to explore architectures that could reduce the effects of interference while keeping the multi-task fine tuning performance.

One approach which aimed to minimize task interference with success was the Adapter BERT model proposed by Houlsby et al. (2019). Adapter BERT adds task-specific linear adapter modules within each BERT layer and freezes the BERT weights, so that training only affects the task-specific adapter modules. The authors find that this approach is able to reach performance comparable to fully fine-tuned BERT, with a GLUE score of 80.0, compared to 80.4 for fine-tuning. An additional benefit of this architecture is that since all tasks are independent, the model can be trained on one task at a time, and tasks can be added or removed without influencing the model.

Finally, since our tasks are all sentence-based, I thought it may be worth exploring further pretraining BERT for sentence-level embeddings instead of the default word-level embeddings to improve performance. Reimers and Gurevych (2019) propose sentence-level embeddings for BERT in their model Sentence-BERT (SBERT), which is trained on the MNLI and SNLI datasets for cosine similarity loss. They find that SBERT outperforms other state-of-the-art sentence-level models on the STS and other transfer learning tasks.

4 Approach

4.1 Baselines

I use two baselines to evaluate my model. The first is fine-tuned BERT, which in the absence of transfer effects represents an upper bound on our performance since it requires tuning all BERT parameters to perform well on each task individually. To fine-tune our model, we add a classification head in the form of two feedforward layers with GELU (Hendrycks and Gimpel, 2016) as the activation function. The size of the intermediate linear layer is chosen arbitrarily to be $d_l = 256$.

The second baseline is what I refer to as frozen BERT. Frozen BERT is trained on all three tasks, but with frozen BERT weights so that the only training is on the top classifiction head which is comprised of the same two feedforward GELU layers. Frozen BERT represents a lower bound on our performance, since there is no change to the underlying model embeddings.

In both baselines, the BERT architectures are kept the same as in Devlin et al. (2018). Pre-trained BERT weights are from HuggingFace's bert-base-uncased model. For more details on the BERT architecture, we refer readers to the CS224N default project handout.

4.2 Projected Attention Layers (PALs)

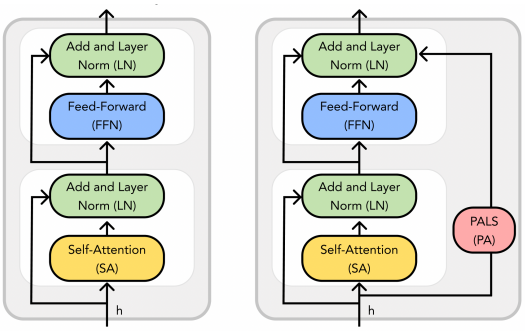
I implement the PALs model architecture from Stickland and Murray (2019) from scratch, but reference their original code at parts which are indicated in my code.

The main idea behind PALs is to add a task-specific low-dimensional multi-head attention layer in parallel to each BERT layer. In the original BERT layer from Devlin et al. (2018), which is depicted in Figure 1a, the input hidden states ($\mathbf{h}$) to each BERT layer are first transformed by a multi-headed self-attention operation (SA) which allows the model to attend to each word embedding in the sequence. The resulting attention scores are then given a residual connection and layer normalized (LN), followed by another two feed-forward GELU layers (FFN), residual connection, and a final layer normalization. We can thus represent a BERT layer (BL) with the following equations:

$$\mathbf{h}_{att} = LN(\mathbf{h} + SA(\mathbf{h})) \tag{1}$$

$$BL(\mathbf{h}) = LN(\mathbf{h}_{att} + FFN(\mathbf{h}_{att})) \tag{2}$$

To modify the original BERT layer into our PALs layer, we add a task-specific attention layer in parallel with the second residual connection and feedforward network right before the final layer normalization as in Figure 1b. The task specific function takes as input the hidden states to the BERT layer and downsamples it through a linear encoder layer (V_E) of size $d_s = 204$, which is again arbitrarily chosen (and consistent with Stickland and Murray). The encoded input is then transformed by a multi-headed self-attention operation and decoded back to the original hidden size $d_h = 768$ through a linear decoder layer (V_D). Unlike the self-attention layers, both the encoder and decoder weights are shared across PALs layers, but not by task. We can represent the new PALs layer (PL) with the following equations where the last term represents the additional task-specific PALs layer (PA):



(a) Normal BERT layer (b) Modified PALs layer

Figure 1: Side by side comparison of BERT and PALs layers.

$$PA(\mathbf{h}) = V_D(SA(V_E(\mathbf{h}))) \tag{3}$$

$$PL(\mathbf{h}) = LN(\mathbf{h}_{att} + FFN(\mathbf{h}_{att}) + PA(\mathbf{h})) \tag{4}$$

By stacking 12 of these PALs layers on top of each other like in the original BERT model, we have the completed PALs model. PALs uses the first pooled [CLS] token embedding for downstream fine-tuning tasks as Devlin et al. (2018) recommend.

4.3 PALs Extensions

I implement several PALs extensions on top of the base model. Frozen PALs is implemented from scratch, SBERT PALs references the SBERT implementation found on this website (link), along with the official repo and documentation, and SMART references this Github implementation (link), which I repurpose for our use case by referencing the MT-DNN-SMART repo implementation.

Frozen PALs After seeing some potential task interference when fine-tuning the original PALs, I wanted to experiment with freezing the BERT weights and only training task-specific layers during fine-tuning to prevent task interference.

Frozen PALs is an **original** contribution inpsired by Houlsby et al. (2019), which also freezes BERT layers during training. Frozen PALs uses the normal PALs architecture but freezes most BERT layers during training. Only the second layer normalization layers, the final pooling layer, and the task-specific PALs functions are trained on the downstream tasks. The second layer norm and final pooling layers are chosen to be trained as well because they directly affect the PALs layers, and I didn't want the layer norm and pooling to be inaccurately based on the old BERT weights. By sharing BERT weights completely and focusing training on the task-specific layers (with the exception of layer norm), I hoped to minimize interference between tasks and improve performance.

SBERT PALs Since our tasks (sentiment classification, paraphrase detection, and semantic similarity) rely more on sentence-level meanings, there may be a large gap between the learned embeddings from pretraining, which are word-embeddings from BERT, and embeddings useful for the tasks.

To pretrain my SBERT model, I use the sentence-level pretraining method from SBERT proposed by Reimers and Gurevych (2019). Instead of using only the [CLS] token embedding for our downstream tasks, I mean-pool all word embeddings in the sequence to form a sentence-level embedding. Since the model is being trained to perform classification tasks, I use multiple negatives ranking (MNR) loss to train sentence embeddings. I use MNR loss instead of the softmax loss originally proposed in the paper, since the authors find that MNR loss yields better performance (see author's note on the official Github repo (link).

MNR loss (Henderson et al., 2017) takes sets of K sentence pairs $[(a_1, b_1), \ldots, (a_n, b_n)]$ where a_i, b_i are labeled as similar sentences and all (a_i, b_j) where $i \neq j$ are not similar sentences. MNR loss simultaneously minimizes the distance between a_i, b_i and maximizes the distance (a_i, b_j) where $i \neq j$. Specifically, training minimizes the approximated mean negative log probability (cross-entropy) of the data. For a single batch, this is calculated as:

$$\begin{aligned} \mathcal{J}(x, y, \theta) &= -\frac{1}{K} \sum_{i=1}^{K} \log P_{\text{approx}}\ (y_i \mid x_i) \\ &= -\frac{1}{K} \sum_{i=1}^{K} \left[S\left(x_i, y_i\right) - \log \sum_{j=1}^{K} e^{S(x_i, y_j} \right] \end{aligned} \tag{5}$$

where θ represents the sentence embeddings and neural network parameters used to calculate S, a scoring function.

More specifically for our training, I use the positive entailment pairs from the combined SNLI and MNLI datasets as our similar sentence pairs. Each other sentence in the training batch is considered a dissimilar pair. Each sentence pair is fed into the BERT model individually and mean-pooled into sentence embeddings u, and v. We then take the cosine similarity between the embeddings, where a score of 0 indicates no similarity and 1 indicates equivalent sentences.

$$\text{Cosine similarity}\ (u, v) = \frac{u \cdot v}{\max\left(\|u\|_2 \cdot \|v\|_2\right)} \tag{6}$$

Finally, the model trains on the cross-entropy loss between the original batch indices as ground truth pairs and the cosine similarity scores. Through this method, we can pretrain the BERT model to use sentence embeddings.

When fine-tuning the SBERT model on downstream tasks, we replace the BERT layers with PALs layers and keep the SBERT weights. This combining of SBERT with PALS is an **original** contribution to my knowledge.

To train SBERT PALs, each sentence pair is passed individually to SBERT to produce two sentence embedding and we return the squared cosine similarity of these two embeddings as our prediction for the QQP and STS-B tasks. For the SST-5 task, I chose to use the [CLS] token embedding instead of the mean-pooled sentence embedding for two reasons. First, since SST-5 is a single sentence task, it is impossible to use cosine similarity to output a corresponding sentiment score with the sentence embeddings. Second, since the sentence-embeddings are trained on cosine similarity, using them purely for a sentiment classification metric creates a large training gap which may hurt performance. With better sentence embeddings, I hope to improve the performance of our model on our sentence-level tasks.

SMART Regularization Because our datasets have such widely varying sizes (140k+ examples for QQP but only around 6k for STS), aggressive fine-tuning for the larger QQP dataset could cause more overfitting for the smaller SST and STS datasets. To avoid such an effect and reduce overfitting generally, I experiment with adding SMART regularization to the PALs training.

SMART regularization adds smoothness-inducing adversarial regularization to the model by injecting the word embeddings with small amounts of noise. The model is then trained to output similar predictions even with that noise, so it is more robust to small amounts of change. Bregman proximal point optimization is used to solve the smoothness optimization equations. For more details about the SMART algorithm, review Appendix A.1.

5 Experiments

5.1 Data

We use the SST-5, QQP, and STS-B datasets provided by CS224N.

SST-5: Stanford Sentiment Treebank consists of single sentences from movie reviews each with the label of negative, somewhat negative, neutral, somewhat positive, or positive. We have 8,544 train, 1,101 development, and 2,210 test examples (Socher et al., 2013).
QQP: QQP consists of question pairs with labels indicating whether particular instances are paraphrases of one another. We have 141,506 train, 20,215 development, and 40,431 test examples.
STS-B: STS-B consists of sentence pairs of varying similarity on a scale from 0 (unrelated) to 5 (equivalent meaning). We have 6,041 train, 864 development, and 1,726 test examples (Agirre et al., 2013).
SNLI + MNLI: Together, SNLI and MNLI consist of 1 million sentence pairs annotated with the labels contradiction, eintailment, and neutral. After filtering for entailment pairs only, we have 314,315 train examples (Bowman et al., 2015; Williams et al., 2017).

5.2 Evaluation method

We use the metrics specified in the CS224N default project handout. For SST-5 and QQP, we use accuracy, a binary measure of whether the answer matches the ground truth exactly. For STS-B, we use the Pearson correlation of the true similarity values against the predicted similarity values.

5.3 Experimental details

Unless otherwise noted, all models were trained with a learning rate of $2 * 10^{-5}$ and a batch size of 16 for 10 epochs (with 6,000 training steps per epoch). All models used the AdamW optimizer with $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-7}$, and no weight decay. There was a learning rate warmup over the

first 10% of training, and linear decay of the learning rate after this, going down to zero at the end of training. Only the frozen BERT baseline used a learning rate of 10^{-3}.

BERT BERT was run with the default configuration used by Devlin et al. (2018). The size of each hidden layer was $d_h = 768$, with 12 attention heads per layer and 12 BERT layers in all.

PALs PALs was run with the default configuration used by Stickland and Murray (2019). The size of each PALs attention layer was $d_s = 204$, with 12 attention heads per layer.

SBERT Pretrained SBERT used a batch size of 32 and was trained for 1 epoch. SBERT PALs used the same settings as PALs.

SMART SMART used $T_{\tilde{x}} = 1, \sigma = 10^{-5}, \epsilon = 10^{-6}$, and $\eta = 10^{-3}$ in the notation used by Jiang et al. (2020).

5.4 Results

I present the compiled results of our models on the development set in Table 1. All the PALs models we test represent a significant improvement from the baseline frozen BERT model. However, compared to the original PALs model in Stickland and Murray (2019), all our modifications perform equally or worse on all tasks, with the exception of SBERT PALs on the SST-5 task, indicating that our changes to the architecture are not useful for improving performance. This could be because there is already enough complexity in the original PALs, and it is already sufficiently accounting for the interference, smoothing, and sentence-level embeddings that I try to encourage. Fortunately, the drops in performance are relatively minimal and all of the models perform relatively close to the individually fine-tuned BERT baseline models that represent an upper bound.

The SBERT PALs model has the most differential in performance from original PALs. SBERT PALs performs better than PALs on the SST-5 task, but worse on QQP and STS-B. The performance boost in SST-5 could be because the sentence-based weights we learn from the pretraining are actually more useful for sentiment analysis despite not using the sentence embeddings themselves. The performance decrease could be because SBERT PALs does not have the ability to attend between sentence pairs, unlike PALs.

Finally, none of the multi-task models are able to perform better than the individually fine-tuned BERT baselines on any task, indicating that there is likely very little transfer learning potential between these three tasks, which is what I expected.

Table 1: Development set results for all models. Note that the fine-tuned BERT baseline is actually three separate models which are each individually fine-tuned.

METHOD		SST-5 (dev)	QQP (dev)	STS-B (dev)	Av. (dev)
BASELINES	FINE-TUNED BERT	**0.543**	0.884	0.886	**0.771**
	FROZEN BERT	0.389	0.697	0.456	0.514
PALs	ORIGINAL PALs	0.509	0.883	0.884	0.759
	FROZEN PALs	0.5	0.883	0.878	0.754
	SBERT PALs	0.52	0.862	0.855	0.746
	SMART PALs	0.501	0.881	0.872	0.751
ENSEMBLE	PALS ENSEMBLE	**0.524**	**0.892**	**0.897**	**0.771**

I present the ensemble model's performance on the test set in Table 2. For the test set I submit two models (one submission contained an error). Expectedly, the ensemble performs the best, but overall both models perform very similarly to the test set as on the development set with two exceptions. First, both models actually improve in SST-5 performance on the test set which perhaps indicates that our model performance is more variable than shown here. Second, the original PALs model performs much more poorly on the STS-B test set (0.884 to 0.825). This is because I erroneously applied a

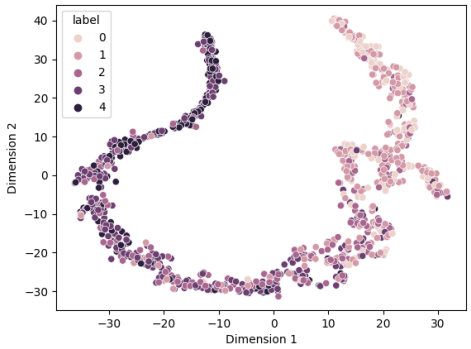
sigmoid function to the test prediction set but not while training. If the sigmoid were removed (which it was in all other tests), I hypothesize that the STS-B test score would be more similar as well.

Table 2: Test set results for the original PALs and ensembled models.

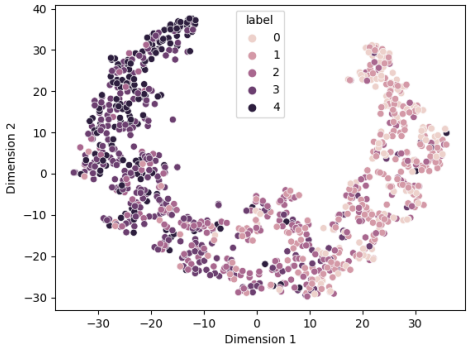
METHOD	SST-5 (test)	QQP (test)	STS-B (test)	Av. (test)
ORIGINAL PALs	0.535	0.882	0.825	0.747
PALS ENSEMBLE	**0.534**	**0.893**	**0.891**	**0.773**

6 Analysis

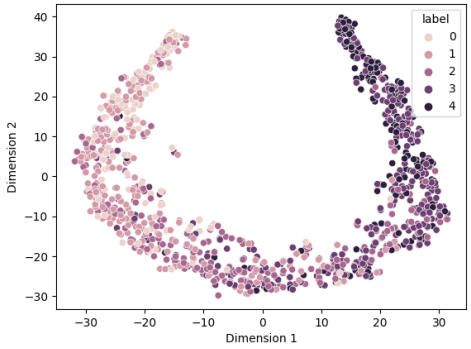
To analyze each of the PALs modifications, I use the t-SNE technique proposed by van der Maaten and Hinton (2008) to visualize the models' sentence embeddings on the SST-5 dataset. t-SNE projects high-dimensional vector embeddings into a low-dimensional space so they can be easily visualized. In Figure 2, using t-SNE, I project 1,000 development examples from the SST-5 dataset onto a 2-dimensional graph for each model, and annotate them with their sentiment labels so we can see different clustering patterns. I omit analysis of the QQP and STS-B datasets because the sentence embeddings outputs for paired-sentences are more difficult to visualize at scale with the same clustering labels (we need to group sentences by pairs as well as whether they are paraphrases or not).



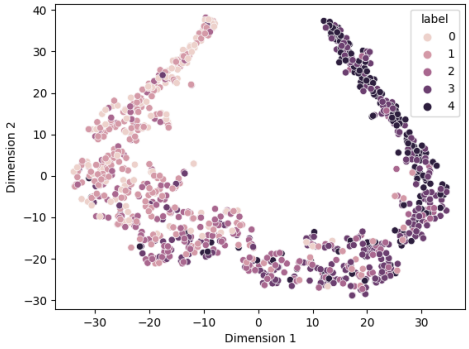
(a) PALs SST-5 dev set embeddings



(b) Frozen PALs SST-5 dev set embeddings



(c) SBERT PALs SST-5 dev set embeddings



(d) SMART PALs SST-5 dev set embeddings

Figure 2: Low-dimensional t-SNE projections of SST-5 sentence embeddings from all PALs models with their sentiment labels 0 (negative) to 4 (positive).

It's important when analyzing t-SNE graphs to not place too much attention on the relative

density/thickness of the clusters or inter-cluster distance, since these can easily change with different perplexity hyperparameters and more iterations. Likewise, t-SNE output is rotation-invariant and can be flipped across axes (van der Maaten and Hinton, 2008), so the flip from positive to negative X-dimensions we see from PALs and Frozen PALs to SBERT PALs and SMART PALs is also inconsequential. We can pay attention to the shapes of the embeddings and their clusters however.

Overall, we see that no model has very clear clustering between labels. It is very hard for the models to tell the difference between a 4 (very negative) and 3 (negative) sentence, a 3 and a 2, etc. Since accuracy measures exact matches of our model predictions to the ground truth labels, these embeddings graphs help explain why our accuracy scores hover around only 0.50, since our model is unable to consistently match the exact fine grained label. However, the clusters indicate that the models are at least getting the positive/negative sentiment of the sentences right most of the time.

Another interesting observation is the shapes of the embeddings curves. While the PALs curve has an sharp bend on the right side and a curve inwards on the left side, the SMART PALs curve has no such bends on its respective sides, indicating that even though the SMART regularization did not improve performance in this task, it still seems to have a smoothing effect on the embeddings overall.

The Frozen PALs embeddings, on the other hand, seem to be rather unsmooth, with many holes within clusters and jagged boundaries. This lack of smoothness could be a reason for why Frozen PALs performs the worst on the SST-5 task. I hypothesis that this lack of smoothness could be the result of freezing the BERT-level layers, which created word embeddings that don't quite align with the changing PALs weights for each task. As a result, the training gap between the BERT layers and the PALs layers creates worse embeddings and performance, disproving the initial hypothesis that freezing BERT layers might allow better multi-task training without interference.

Finally, it's surprising that the SBERT PALs and SMART PALs embeddings have similar beaker-like shapes, since they are pretrained on completely different objectives. This could indicate that the SBERT cosine similarity pretraining task actually encourages some sort of smoothing regularization effect in the down-stream PALs training akin to SMART. Additionally, SBERT PALs is able to handle the sharp bend on the right side of the PALs embeddings much better than SMART PALs, resulting in the smoothest and most cohesively clustered graph overall. This extra smoothing effect on the embeddings could explain why the SBERT PALs model performs the best on the SST-5 tasks.

7 Conclusion

In this project, I implement the PALs model and explore several modifications to it, like freezing only BERT layer weights during training, SBERT cosine similarity pretraining, and SMART regularization. I show that these changes in model architecture are relatively ineffectual at improving model perfomance, although they may have smoothing effects on the underlying embeddings. In addition, I create an ensemble of our models which achieves strong results on the three tasks, with an average score of 0.771 on the dev set and 0.773 on the test set. The primary limitation of this work is the inability to find any architectural change that meaningfully improves task performance. It can't say where to look for better performance; only where *not* to look. In future work, it may be interesting to explore how interference can be minimizing without creating training gaps from freezing layers. Additionally, I didn't observe any transfer learning effects from these three tasks, but there could be tasks or datasets that help encourage more transfer learning for these tasks and thus better performance. It could also be promising to explore the SBERT cosine similarity pretraining task more, since it resulted in better performance for the SST-5 task.

References

Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. 2013. *SEM 2013 shared task: Semantic textual similarity. 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, Atlanta, Georgia, USA. Association for Computational Linguistics.

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.

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. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding.

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.

Dan Hendrycks and Kevin Gimpel. 2016. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. *CoRR*, abs/1902.00751.

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.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605.

Michael Mccloskey and Neil J. Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. *The Psychology of Learning and Motivation*, 24:104–169.

Yifan Peng, Qingyu Chen, and Zhiyong Lu. 2020. An empirical study of multi-task learning on bert for biomedical text mining. *arXiv preprint arXiv:2005.02799*.

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

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Asa Cooper Stickland and Iain Murray. 2019. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2017. A broad-coverage challenge corpus for sentence understanding through inference.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR*, abs/1906.08237.

Algorithm 1 SMART: We use the smoothness-inducing adversarial regularizer with $p = \infty$ and the momentum Bregman proximal point method.

Notation: For simplicity, we denote $g_i(\widetilde{x}_i, \bar{\theta}_s) = \frac{1}{|\mathcal{B}|}\sum_{x_i \in \mathcal{B}} \nabla_{\widetilde{x}} \ell_s(f(x_i; \bar{\theta}_s), f(\widetilde{x}_i; \bar{\theta}_s))$ and $\text{AdamUpdate}_{\mathcal{B}}$ denotes the ADAM update rule for optimizing (3) using the mini-batch $\mathcal{B}$; $\Pi_{\mathcal{A}}$ denotes the projection to $\mathcal{A}$.

Input: T: the total number of iterations, $\mathcal{X}$: the dataset, θ_0: the parameter of the pre-trained model, S: the total number of iteration for solving (2), σ^2: the variance of the random initialization for $\widetilde{x}_i$'s, $T_{\widetilde{x}}$: the number of iterations for updating $\widetilde{x}_i$'s, η: the learning rate for updating $\widetilde{x}_i$'s, β: momentum parameter.

$\widetilde{\theta}_1 \leftarrow \theta_0$
for $t = 1, .., T$ **do**
  $\bar{\theta}_1 \leftarrow \theta_{t-1}$
  for $s = 1, .., S$ **do**
    Sample a mini-batch $\mathcal{B}$ from $\mathcal{X}$
    For all $x_i \in \mathcal{B}$, initialize $\widetilde{x}_i \leftarrow x_i + \nu_i$ with $\nu_i \sim \mathcal{N}(0, \sigma^2 I)$
    for $m = 1, .., T_{\widetilde{x}}$ **do**
      $\widetilde{g}_i \leftarrow \frac{g_i(\widetilde{x}_i, \bar{\theta}_s)}{\|g_i(\widetilde{x}_i, \bar{\theta}_s)\|_\infty}$
      $\widetilde{x}_i \leftarrow \Pi_{\|\widetilde{x}_i - x\|_\infty \le \epsilon}(\widetilde{x}_i + \eta \widetilde{g}_i)$
    end for
    $\bar{\theta}_{s+1} \leftarrow \text{AdamUpdate}_{\mathcal{B}}(\bar{\theta}_s)$
  end for
  $\theta_t \leftarrow \bar{\theta}_S$
  $\widetilde{\theta}_{t+1} \leftarrow (1 - \beta)\bar{\theta}_S + \beta \widetilde{\theta}_t$
end for

Output: θ_T

Figure 3: SMART algorithm from Jiang et al. (2020).

A Appendix (optional)

A.1 SMART

I use the SMART algorithm as presented in Figure 3. l_s indicates symmetrized K-divergence for classification tasks and mean squared loss for regression tasks. To do the smoothness-inducing regularization, I inject noise v_i into our embeddings to get noised logits for each task. For classification tasks, our loss is the symmetrized KL-divergence between the original logits and the noised logits; for regression tasks, our loss is the mean squared error between the original scalar output and the noised output. We then do a gradient update on the noise to move it adversarially towards increasing the difference between the two logits. To solve the smoothing function, I use the iterative Bregman proximal point optimization method the authors propose, as seen in the for-loop.