SerBERTus: A SMART Three-Headed BERT Ensemble

Stanford CS224N Default Project

Matthew Hayes
Department of Computer Science
Stanford University
mhayes3@stanford.edu

Mentor: Gabriel Poesia No External Collaborators No shared project

Abstract

We examine different architectures, learning methods, and hyperparameter choices for fine-tuning the 110 million parameter $BERT_{BASE}$ model on three different tasks: five class sentiment analysis on Stanford Sentiment Treebank (SST) (Socher et al., 2013); binary paraphrase detection on Quora Question Pairs (QQP) [1], and regression on Semantic Text Similarity (Agirre et al., 2013). We find that a strong SMART (Jiang et al., 2020) loss combined with a novel architecture replicating only the BERT layers closest to the task-specific heads brings the greatest improvement to performance on the three tasks without too severe an increase in model size and resource consumption. Our best model is an ensemble achieving mean performance of 78.61% on the test set.

1 Introduction

We're not the strongest or the fastest, but our intelligence and our use of language to communicate intelligent ideas is perhaps the most distinguishing feature of the human species and the cause of our dominance: no other records and shares in such depth or breadth. Correspondingly, there is such a wide variety of tasks that can be expressed with our natural language and that the Natural Language Processing field of attempts to automate. While we can and have carefully architectured solutions for each task individually, general purpose alternatives in sum reduce human effort and computational resources, and can improve performance.

Scaling Vaswani et al. (2017)'s Transformer architecture, Radford et al. (2018) (GPT) and then Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2018) outperformed all prior models on respectively 9 and 11 General Language Understanding Evaluation (GLUE) tasks by extensively pre-training on simple but highly general word-prediction tasks, and then further fine-tuning the models independently on more specific tasks. The pretraining-finetuning setup, coupled with large, publicly available, general-purpose models has changed the field. But how to exploit them to their fullest potential is still largely not understood.

2 Related Work

Large pretrained models had success prior to GPT and BERT. Howard and Ruder (2018), for example, explored different methods for fine-tuning general-purpose language models consisting of stacked layers of LSTMs. They proposed and achieved state-of-the-art performance using, among other techniques: a lower learning rate for the layers close to the input, which contain more general

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

language encodings than the layers closer to the task-specific output; and keeping the parameters of these lower layers completely frozen to start and unfreezing one at a time with each successive epoch.

Instead of Radford et al. (2018)'s next token prediction task using transformer decoders, Devlin et al. (2018) found that stacking bidirectional transformer encoder layers and pretraining on the Masked Language Model (MLM) and Next Sentence Prediction (NSP) objectives produced higher quality representations for the same model size. BERT's pretraining inputs consist of two sequences separated by a special [SEP] token and preceded by a special [CLS] token. Half the time the second sequence actually follows the first in the source document, and the other half of the time it is selected at random. The NSP task is to use the representation in the [CLS] token to predict which which is the case. MLM predicts the correct original token for a randomly modified 15% of input tokens. 80% of the 15% are replaced by a special [MASK] token, 10% with a random token from the vocabulary, and 10% are left unchanged. Liu et al. (2019b) later found that dispensing with the NSP task and simply performing the MLM task on longer sequences, bigger batches, and different random token selection for each epoch, was sufficient meet and exceed BERT's performance.

Sun et al. (2019) examined decisions for further pretraining, multitask finetuning, and target-task finetuning when adapting BERT for text classification. Using IMDb (Maas et al., 2011) they found that 100K further MLM and NSP pre-training steps are optimal. For fine-tuning, they find a learning rate of 2e-5 to outperform any higher learning rate, which seem more subject to the new tasks overwriting the generality from BERT's pretraining or "catastrophic forgetting" (McCloskey and Cohen, 1989). Similar to Howard and Ruder (2018), further reducing this learning rate by a factor of 0.95 for each successive transformer layer approaching the inputoutperformed a uniform learning rate or decay factor of 0.9.

Jiang et al. (2020) took a less heuristic approach, instead introducing an additional smoothness-inducing regularization term to the loss. They randomly purturb each embedded input a small amount as measured by by ℓ_∞ norm and penalize the model for the change in output, as measured by symmetric KL-divergence for classification, and squared loss for regression. While the additional terms approximately halve the maximum batch size that can be used and increase BERT finetuning time, they find significant evaluation improvements on GLUE.

3 Approach

We first complete a minimal (BERT) (Devlin et al., 2018) implementation referencing the provided skeleton code, project handout, Vaswani et al. (2017), and Assignment 5's minGPT [2] implementation for the multiheaded self attention module. Adding skip connections, layer normalizations, a position-wise feed-forward layer and 10% dropout, we complete the BERT encoder layer. Stacking 12 such layers and applying an additional feed-forward 'pooling' layer with $\tanh$ activations to the resulting embedding in the [CLS] position, we load pre-trained $BERT_{BASE}$ weights from the web and pass the provided sanity test.

On top of the pooling output, the baseline classifier applies dropout and a dense layer with one output for each possible class. Referencing Kingma et al. Kingma and Ba (2014) and Loshchilov et al. Loshchilov and Hutter (2017), we complete the ADAMW implementation, pass the provided optimizer test, and compare to provided reference accuracies for training on Stanford Sentiment Treebank (SST) Socher et al. (2013) or CFIMDB Maas et al. (2011) using provided hyper-parameters, and either frozen pretrained $BERT_{BASE}$ weights or finetuning them.

Extending from the provided baseline for SST, we establish baselines for Quora Question Paraphrase (QQP) detection [3] and Semantic Textual Similarity Agirre et al. (2013) (STS) tasks by finetuning single-logit BERT instances with binary cross entropy and squared error losses respectively. Our multitask baseline, "triple BERT", is thus an ensemble of three $BERT_{BASE}$ models, independently fine-tuned to maximize performance on each task's dev set. While this has the disadvantages of more storage and less generalization, it permits independent design iteration for each dataset.

For STS and QQP, we run our initial experiments by simply concatenating the two passages in a random order, separated by BERT's [SEP] token. Alternatively we embed the two passages through BERT separately similar to Reimers et al. Reimers and Gurevych (2019) and then combine the

[2] `https://github.com/karpathy/minGPT`

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

outputs. This approach excels when the task is to find the closest sentence to a query from a large set since the embeddings can be cached: the query does not have to be re-embedded with each candidate. With STS and QQP, however, we are given only two sentences to compare for each example and caching their embedding is not helpful. Without the cross attention between the two inputs we see a degradation in performance in our initial experiments using either cosine similarity or regression on the concatenated embeddings (optionally also concatenating their normalized difference).

We evaluate the default hyperparameter choices for dropout, optimizer, learning rate, and number of epochs using the small SST dataset, and time-permititng validate our findings on STS before attempting to apply to the larger QQP. When training on all three datasets a single task is used for each batch and gradient step, but we randomly shuffle the batches from all three tasks. We find this performs well in practice without any weighting to account for the different dataset sizes.

Following Sun et al. (2019), we implement and evaluate a learning rate that decays with each successive BERT layer closer to the input. We extend this idea by entirely fixing the token and position embeddings at the input before the first BERT layer, or increasing the learning rate of the classification heads relative to the BERT parameters. We also implement the MLM task to explore the benefit of additional pre-training, but following Liu et al. (2019b) do not implement the NSP task.

We explore the addition of Jiang et al. (2020)'s "SMART loss" implementation from the web [4] to see if its additional regularization effect is independent of other methods, or if one is strictly better. Instead of minimizing the typical loss $\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell\left(f\left(x_i;\theta\right), y_i\right)$ for some task-specific ℓ (in our case cross entropy for SST and QQP, squared error for STS), they minimize $\mathcal{L}(\theta) + \lambda_{\mathrm{s}}\mathcal{R}_{\mathrm{s}}(\theta)$, where $\mathcal{R}_{\mathrm{s}}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \max_{\|\tilde{x}_i - x_i\|_p \leq \epsilon} \ell_{\mathrm{s}}\left(f\left(\widetilde{x}_i;\theta\right), f\left(x_i;\theta\right)\right)$ for some sampled embedded input perturbation $\tilde{x}$. We use the ADAMW optimizer to solve this minimization instead of adopting their Bregman Proximal point method.

We also consider different architectures for the three tasks. Expanding from our triple-BERT baseline, we explore the simple but novel ideas of adding a fourth or fifth BERT to a simultaneously trained ensemble. With "static BERT" we concatenate the embedding from a frozen pretrained instance, to provide generality and further mitigate "catastrophic forgetting" McCloskey and Cohen (1989) while the fine-tuned BERTs can focus on adapting to their tasks. With "mutli-BERT", we concatenate the embedding of an instance fine-tuned on all three tasks for any transferable representation. To measure alternatives between a completely shared BERT and three independent BERTs, we allow independently finetuning only the k BERT layers closest to the outputs, while sharing the $12 - k$ layers closer to the input, naming this approach CerBERTus for it's three heads, a reference to the monstrous watchdog in Greek mythology. The architectures are illustrated in Figure 6.

Finally, our experiments yield many models that may be complementary. We implement ensembling the predictions of our best models, which Liu et al. (2019b) has shown to work well for BERT. We use unweighted averaging of the ensemble's unrounded predictions, and only add models to each task's ensemble if they improve or do not change dev set performance. We explore the tradeoff between a single multi-BERT instance with approximately 110M parameters, the best model trained in a single session for each task (110M-220M parameters per task), and ensembling several training sessions for each task (up to 1B parameters per task).

4 Experiments

4.1 Data

The Stanford Sentiment Treebank Socher et al. (2013) consists of movie reviews from the website `rottentomatoes.com`. We examine a version of the fine-grained 5 class task, where sentiments range from negative (0) to positive (4). Munikar et al. (2019) report a 53.2% accuracy when fine-tuning $BERT_{BASE}$ on the public version of the dataset and we are provided a benchmark mean accuracy of 51.5% with a standard deviation of 0.4%. That dataset contains 8,544 train examples, with a majority label of "somehwat positive" (3), at 27.2%. STS's labels are averages of human judgements in the range [0, 5], thus continuous and naturally modelled with regression. The median label of the 6,041 training examples is 3 (somewhat positive). At 141,506, QQP has more than 16 times the number of training examples as SST and STS so it takes significant compute time for a

[4] `https://github.com/archinetai/smart-pytorch`

single epoch. The mode of its binary labels is 0 (not paraphrase) at 62.5%. Devlin et al. (2018) report a Pearson correlation of 86.5% when finetuning $BERT_{BASE}$ on the public version of STS, and Jiang et al. (2020) report a 90.9% accuracy on the public QQP for their $BERT_{BASE}$ reimplementation.

4.2 Evaluation method

Consistent with the provided leaderboard for results, we measure model quality by accuracy on SST and QQP, Pearson correlation on STS, and the average of these three metrics. We also consider the trade-off between these metrics and the increased number of model parameters and training time when training multiple BERT instances at once or ensembling separately trained models

4.3 Experimental details & results

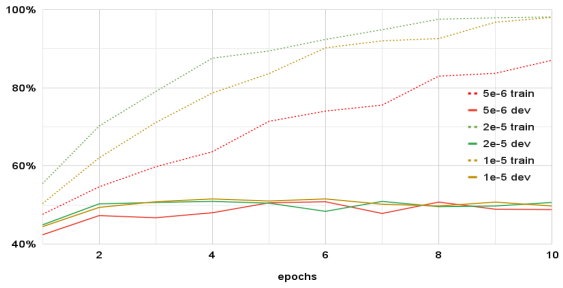

Figure 1: Effect of learning rate on SST accuracy

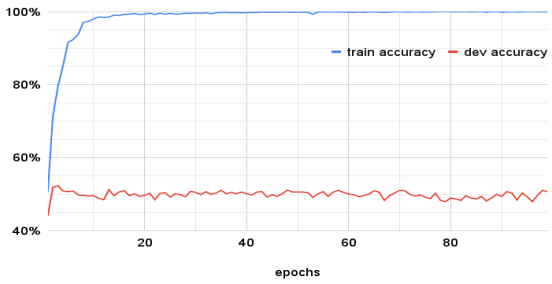

Figure 2: SST Accuracy over 100 epochs

4.3.1 Base learning rate and number of epochs

Using the default parameters of 10% dropout within the BERT layers and 30% in the classification head, and our custom ADAMW optimizer implementation with zero weight decay, we find the default learning rate of 1e-5 to be a good setting on SST. As shown in Figure 1 it converges only a little slower than the 2e-5 rate on the training set but still quickly enough during 10 epochs, unlike 5e-6, which does not quite converge. Additionally, 10 epochs seem sufficient to find the maximum dev set performance, peaking after after approximately 3-4. Consistent with Nakkiran et al. (2019)'s findings for transformer-based models, we do not observe an epoch-"double descent" phenomenon when training for 100 epochs (Figure 2)). Unless otherwise specified, for the remainder of our experiments we keep fixed the learning rate of 1e-5 and report the best dev performance of 10 epochs over the training sets.

4.3.2 Dropout

dropout prob.	mean SST accuracy	max SST accuracy	mean STS corr.	max STS corr.
0%	**51.7**	**52.6**	86.6	86.6
10%	51.6	52.0	86.4	86.5
20%	51.5	52.1	86.2	86.3
30%	51.4	52.1	86.2	86.5
50%	51.6	52.0	86.4	86.6
70%	51.4	51.9	86.4	86.4
90%	51.4	52.1	**86.8**	**87.0**

Table 1: Performance on SST and STS varying head dropout

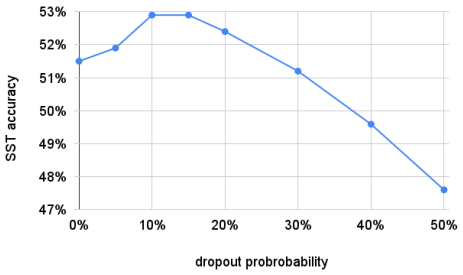

Figure 3: Effect of BERT hidden layer dropout rate on SST accuracy

The default parameters include a dropout probability of 30% between the [CLS] pooling layer and the output logits. Holding all other default hyperparameters fixed, we experiment with several other dropout values in the model heads for three different random seeds on SST. We allow up to 30 epochs for the 90% dropout, but find the best dev performance is still within the first 10. We also try two different random seeds on STS, but using PyTorch's ADAMW implementation with a weight decay

of 1e-4. We find the largest STS correlation with a dropout of 90%, but taking 19 or 22 epochs to converge. Overall for both datasets, we and find that removing the head dropout entirely improves performance within 10 epochs (Table 1).

Fixing no dropout probability in the model head, we also experiment with Bert's internal hidden layer dropout probability of 10% with layer decay 0.95 and PyTorch ADAMW weight decay 1e-5, batch size 128. We do find any other value to perform better, though 15% performs just as well. For the higher values of 40% and 50%, we allow up to 30 epochs (Figure 3). We do not vary the additional 10% dropout applied to the normalized attention scores.

4.3.3 ADAMW Optimizer

Holding dropout constant at zero and keeping the rest of the default parameters, we examine effects of the ADAMW optimizer. At the default weight-decay setting of zero, we compare the performance of our custom ADAMW implementation to the version in PyTorch. Unsurprisingly, we find that PyTorch's implementation is slightly faster and yields slightly better accuracies in Table 2, likely due to subtle optimizations.

Now using PyTorch's implementation, we consider increasing the weight decay parameter from the default value of zero. We find the common weight decay setting of 1e-2 to be too high, with a small weight decay of 1e-5 to performing the best across three random seeds on SST only, while the slightly larger 1e-4 seems to strike a balance when training on all three datasets at once (Table 3).

	custom	PyTorch
mean SST accuracy	52.6	**52.7**
max SST accuracy	52.8	**53.2**
mean 10 epoch dur. (mm:ss)	19:34	**19:09**

Table 2: SST accuracy and speed comparison of custom ADAMW implementation to PyTorch's

	1e-5	1e-4	1e-3	1e-2
multi QQP acc.	86.7	87.7	**88.9**	88.7
multi STS corr.	78.7	**87.0**	86.8	86.5
multi SST acc.	**52.7**	52.4	50.4	50.7
multi dataset mean	72.7	**75.7**	75.4	75.3
SST only mean	**52.6**	52.5	51.9	51.6
SST only max	**53.1**	**53.1**	**53.1**	52.8

Table 3: Effect of weight decay

4.3.4 Learning rate decay

We examine using layer decay with our custom ADAMW implementation without weight decay. Consistent with Sun et al. (2019), we find that a layer decay value of 0.95 outperforms a fixed learning rate or a decay rate of 0.9. Extending the idea outside the transformer layers, we find that freezing the input word and position embeddings or halving their learning rate does not improve performance on SST. Similarly, significantly increased learning rate in the classification head degrades performance, however a rate of 1.5 times the rest of the model slightly improves performance.

	embed 0	embed 5e-6	layer 0.95	layer 0.9	all 1e-5	head 1.5e-5	head 2e-5	head 1e-3	head 1e-4
mean	51.3	51.9	**52.4**	52.2	51.7	52.1	51.9	50.6	51.7
max	51.8	52.1	**53.2**	52.9	52.6	52.6	52.4	51.1	51.7

Table 4: The effect of higher learning rates closer to the task-specific heads. 'Embed' denotes changing only the rate of the initial word and position embedding matrices. 'Layer' denotes changing only the BERT layer learning rates, with a multiplicative factor for each successive BERT layer closer to the input embedding, e.g. 'layer 0.9' denotes that the last BERT layer has a learning rate of 1e-5, the second to last 9e-6, the third to last 8.1e-6 etc. 'Head' denotes only changing the learning rate in the model heads.

4.3.5 SMART Loss

We use Jiang et al. (2020)'s SMART loss with symmetric KL-divergence for the classification tasks SST and QQP, and squared error for STS regression. Using their recommended default 1 sampling step, $\epsilon = 1\text{e}{-6}$, $\sigma = 1\text{e}{-5}$, $\eta = 1\text{e}{-3}$, $p = \infty$, we vary only the weight λ_S using PyTorch's ADAMW implementation with 1e-4 weight decay. As showing in Figure 4 we find that on SST

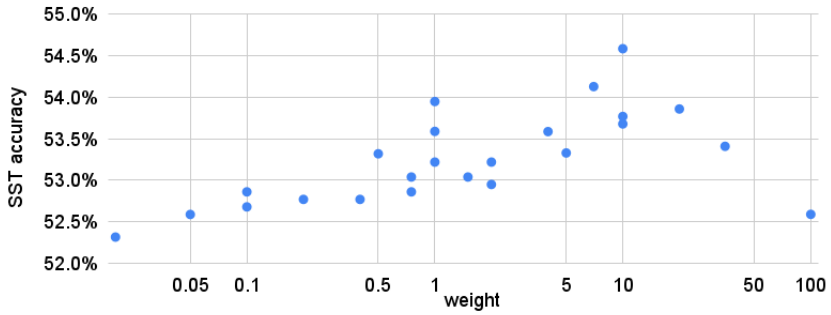

Figure 4: Effect of SMART loss (Jiang et al., 2020) weight λ_S on SST accuracy

$\lambda_S \in [1, 10]$ performs well, with $\lambda_S \geq 10$ requiring more than 10 epochs to converge. Three instances of $\lambda_S = 10$ found the best dev accuracy after 10, 11, or 12 out of 20 epochs, while for $\lambda_S = 100$ the best dev performance was after 44 of 45 epochs, and may have continued increasing.

4.3.6 Architecture

Using no weight decay, we find that providing the classifier head with both a static BERT instance and one with parameters that can be fine-tuned does not significantly improve performance, actually reducing average performance on three SST random seeds and two STS random seeds (Table 5). Given its additional memory requirement, we exclude it from the remainder of our experiments.

	SST mean	SST max	STS mean	STS max
static + finetune	51.5	52.5	86.5	**86.7**
finetune only	**51.7**	**52.6**	**86.6**	86.6

Table 5: Effect of adding a static BERT instance on SST and STS

In Table 6 we consider other architecture alternatives on all three tasks. We do not use weight decay, layer decay, or SMART loss, and use the largest batch size that fits into 16GB of GPU memory. We find that 2 task-specific layers performs better than none or more than 2. A combination of three task-specific models and a multitask model also performs well, but requires uses much more GPU RAM and thus requires a smaller batch size and more compute time.

architecture	best QQP	best STS	best SST	avg of best	best avg	batch size
multi	88.9	87.0	51.6	75.8	75.7	32
2 head layers	89.0	**87.9**	**52.4**	**76.4**	**76.1**	24
4 head layers	88.6	87.6	51.6	75.9	75.5	20
6 head layers	88.8	87.6	51.9	76.1	75.7	16
triple (baseline)	88.9	82.5	52.1	74.5	-	32
multi + triple	**89.4**	87.2	**52.4**	**76.4**	75.9	8

Table 6: Performance of different architectures on all three tasks. We report he best accuracy or Pearson correlation on the dev set achieved for each task (potentially in different epochs), as well as the best average performance on the dev set when all heads are constrained to the same epoch (except in the case of the "Multi" architecture, where the models are trained independently).

4.3.7 Further pre-training

We perform further pretraining on the MLM objective with, as in Sun et al. (2019), a batch size of 32, followed by fine-tuning without weight decay or layer decay. On SST we find that the additional pretraining generally does not improve performance. For the small SST dataset, we try three different sets of parameters during pretraining, show in in Figure 5. With a 1e-5 learning rate, we see a negative correlation between the number of additional pretraining steps and the best dev performance after fine-tuning. When decaying the 1e-5 base pretraining learning rate by a factor of 0.95 per layer

approaching the input, the degradation is not as bad, but still none of the models with additional pretraining performs as well as without. When we try a smaller learning rate of 5e-6 however, we do see some improvement in dev accuracy, peaking around 16K steps.

On QQP, we find that pretraining with the 1e-5 learning rate performs well, shown in Figure 6. On this dataset, the peak finetuned dev accuracy is achieved after approximately 66K pretraining steps, as opposed to the 100K found by Sun et al. (2019). We also see an improvement in STS correlation with a small number of pretraining steps at 1e-5 learning rate (Figure 8), but due to the small size of the dataset, diminishing returns, and time constraints, do not investigate beyond 3K additional pretraining steps.

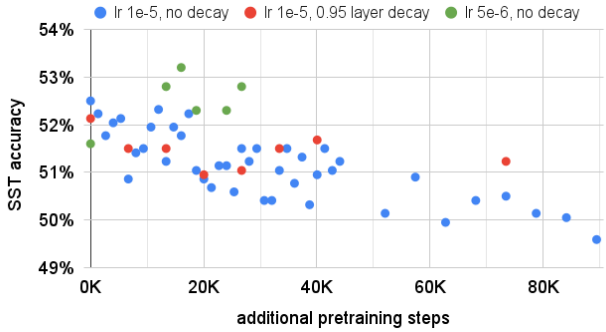

Figure 5: Additional pretraining on SST effect on accuracy after finetuning

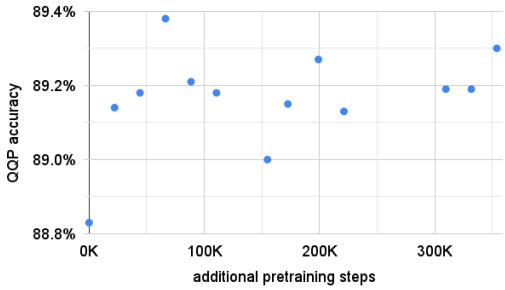

Figure 6: Additional pretraining on QQP effect on accuracy after finetuning

4.3.8 Ensembling and test results

We submit to the learderboard for test evaluation two ensembles. One ensemble consists of three models: for each task the one model that performed the best on each task's dev set. For both QQP aand STS this was a 2-head-layer CerBERTus model with SMART $\lambda_S = 12$, and layer decay factor 0.95 but from different epochs, while the SST model was independently trained on the one task without layer decay and $\lambda_S = 10$. For the second ensemble we average the predictions of between 7-9 of our best performing models per task from all experiments, only adding models to the ensmeble if they improve the dev performance. They individually achieve at least 53% dev accuracy on SST, 87% dev Pearson correlation on STS, and 88.5% dev accuracy on QQP. Table 7 shows that the larger ensemble achieves the best performance overall, but is only a minor improvement over the three-model ensemble, which itself brings only minor improvement over a single multi-BERT or CerBERTus model.

Category	Achritecture	SMART weight	Layer decay	Approx params	SST dev	STS dev	QQP dev	avg dev	SST test	STS test	QQP test	avg test
minimal size	multi BERT	12	0.95	110M	53.32	89.38	88.67	77.12				
single base same epoch	2-layer CerBERTus	15	0.95	138M	52.41	89.86	89.64	77.30				
single base same epoch	3-layer CerBERTus	12	0.95	153M	53.04	89.99	89.72	77.58				
best of each (before above STS)	QQP, STS 2-layer CerBERTus SST triple-BERT	QQP, STS 12 SST 10	QQP, STS 0.95 SST 1.0	386M	54.59	89.93	90.27	78.26	**55.48**	89.77	90.11	78.45
best ensemble	7-9 models per task	Several	Several	2.6B+	55.85	90.45	90.94	79.08	54.75	**90.25**	**90.83**	**78.61**

Table 7: Comparison of Ensemble size and performance on dev and test sets

5 Analysis

Examining the dev set predictions of each single best model, mistakes on SST are typically off by one class, and many people might agree with its predictions, e.g. the model rates "It 's a stunning lyrical work of considerable force and truth." as a positive review, but the ground truth is only 'somewhat positive'. In the cases it is off by two, the labels are sometimes even more questionable, such as the neutral label of "It 's a coming-of-age story we've all seen bits of in other films – but it 's rarely been told with such affecting grace and cultural specificity," which is predicted positive. But some

reviews are more subtle, without any words or short phrases clearly identifying the sentiment, as in the 'somehwat positive' "[Lawrence bounces] all over the stage, dancing, running, sweating, mopping his face and generally displaying the wacky talent that brought him fame in the first place," which is predicted 'somewhat negative': 'sweating' and 'wacky talent' on on their own are not clear, but when combined with the human experience, the review paints a playful scene. There are not any examples where it is wrong by three or four classes.

For QQP many mistakes are borderline, e.g. mutlti-part questions or slight variations: "How can I lose fat as a teenager?" is predicted a paraphrase of "How can I lose fat as a 15 year old?". But sometimes it seems the word embedding have overfit e.g. predicting "Is there any evolutionary advantage of baldness?" is a paraphrase of "Was there any evolutionary advantage for beards?": beards and baldness both involve head hair, but the pre-trained BERT must have learned the difference to do well on MLM. For STS we see the model may have lost meaning of expressions or be relying too heavily on word overlap, as it gives "Work into it slowly" and "It seems to work" a score of $2.7 \approx$'mostly equivalent' but they are actually different topics.

Our findings on dropout and weight decay are somewhat surprising, as we expect additional regularization to be helpful, particularly on the small datasets like SST and STS. However, BERT's hidden layer and attention probability dropouts may be sufficiently regularizing on their own. Using a different head dropout probability for each task may have performed better: our experiments showed that STS benefits from a very high dropout rate, while without the ability to combine all of the the already robust pooling dimensions the model may be forced to learn cruder approximations for fine-grained sentiment. In previous literature, we typically see the weight decay parameter fixed at the common value of 0.01, which may not be well tuned for BERT fine-tuning: potentially this higher value causes the pretrained parameters values to decay too much during fine-tuning.

In reference to Sun et al. (2019) our results on layer decay and further pretraining seem reasonable. Despite our differing data, we also found 0.95 to be the best layer decay factor. Halving or completely freezing the input embedding may not have allowed enough specialization and a smaller reduction may be effective. Similarly, when drastically increasing the head learning rate, the gradient steps may have become too out of sync with those in the BERT layers. Additional pre-training appears to be highly dataset-specific. Sun et al. (2019) also found a degradation in performance when performing additional pretraining using only the small TREC Li and Roth (2002) dataset, and a large dataset from a similar domain may have helped SST. Even for the larger QQP, Sun et al. (2019)'s 100K additional pretrianing steps was not optimal, and we did not see nearly as smooth of a relationship with finetuned dev performance. Though Liu et al. (2019b) found that the NSP task is overall unnecessary to achieve good fine-tuned performance, it may have given us more consistent results given our reliance on the [CLS] representation.

For model architecture, the static BERT dimensions may have been too highly correlated with some from fine-tuning, allowing the model to cheat regularization and overfit: perhaps a higher head dropout rate would have been helpful in this case. On the other hand, the regularizing effect of multitask training (Liu et al., 2019a) may have been sufficient for multi-BERT to effectively complement triple-BERT. However, factoring model size and resource consumption, multi-BERT alone or 2-to-3-head-layer-CerBERTus model seems to be an excellent balance, consistent with the existing understanding that the layers closer to the input are very generic and fine-tuning them to specific tasks does not bring much benefit.

6 Conclusion

We learn that the additional regularization introduced by SMART loss is highly effective. While a combination of the triple-BERT architecture and multi-BERT works well, the middle-ground CerBERTus achieves a comparable performance with significantly fewer parameters and compute time. In the future we might revisit separate embeddings with cross attention for QQP and STS, as well as further pretraining on the smaller datasets by utilizing data from the same domain or use other token embeddings in addition to the [CLS] token. Many of our choices were made with evaluation on only a single dev dataset, and using cross-validation would allow us to ensure we are not overfitting to it. For the few examples that exceed BERT's 512 token limit, we might also revisit how we handle truncation, e.g. truncating the middle instead of the end of the text as in Sun et al. (2019). We might also explore using Howard and Ruder (2018)'s slanted triangluar learning rate schedule.

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.

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

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification.

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.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization.

Xin Li and Dan Roth. 2002. Learning question classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019a. Multi-task deep neural networks for natural language understanding. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4487–4496. 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. 2019b. Roberta: A robustly optimized bert pretraining approach.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

Michael McCloskey and Neal J. Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. volume 24 of *Psychology of Learning and Motivation*, pages 109–165. Academic Press.

Manish Munikar, Sushil Shakya, and Aakash Shrestha. 2019. Fine-grained sentiment classification using bert.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. 2019. Deep double descent: Where bigger models and more data hurt.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training.

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.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune bert for text classification?

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.

A Appendix

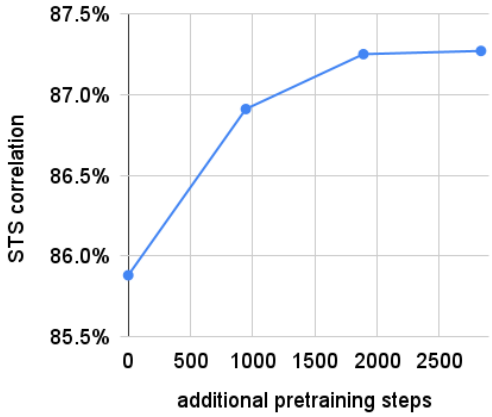

Figure 7: Effect of additional pretraining steps on STS correlation

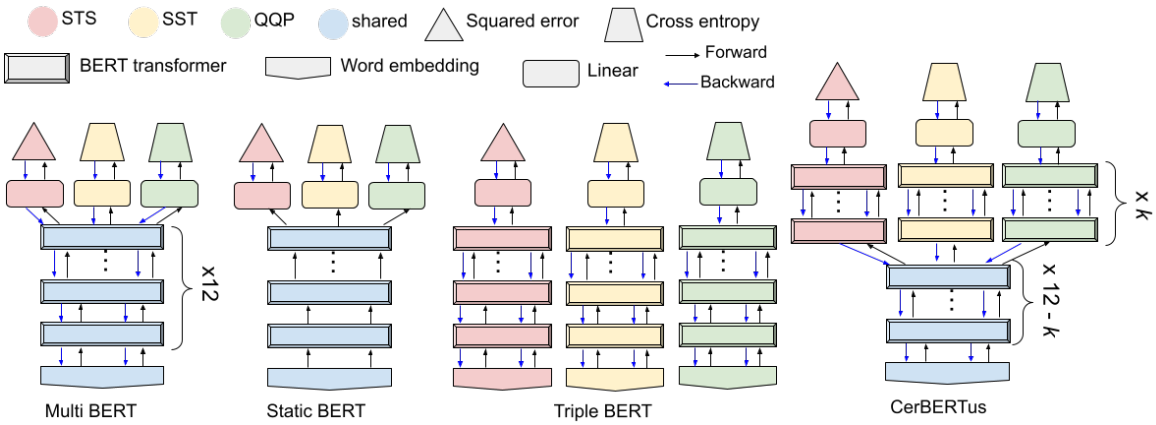

Figure 8: Illustration of architectures. Note that only the embedding in the [CLS] position is fed to the linear heads, after passing through the pooling layer.