# BERT and Learnie:
# Multi-task Classification Across Semantics Street

Stanford CS224N Default Project

**Sonia Chu**
Department of Computer Science
Stanford University
chush@stanford.edu

**Flora Huang**
Department of Computer Science
Stanford University
flora221@stanford.edu

**Kachachan Chotitamnavee**
Department of Computer Science
Stanford University
kchotita@stanford.edu

## Abstract

This paper explores various approaches to improving the performance of the BERT model across the three tasks of sentiment analysis, paraphrase detection, and semantic textual similarity evaluation. We focused particularly on exploring methods to train all three tasks simultaneously on a singular BERT backbone that can perform well across all three downstream tasks. We did this by exploring multiple model architectures, supplemented by our implementation of a wide range of hyperparameter search algorithms, scheduling techniques, regularization methods, and optimization techniques such as gradient surgery. Our results showed that using a singular BERT backbone but separate linear layers for all three tasks led to significantly better performance than the baseline model. Furthermore, we also found that implementing gradient surgery, data augmentation, variable learning rate with linear warmup and linear decay, early stopping, and performing aggressive hyperparameter search using a population-based tuning algorithm yielded our best-performing model, which was 102% better than our baseline implementation. We hope that our study will provide a well-performing and easily extensible baseline on which others can quickly train a small BERT model that is efficient across multiple downstream tasks.

## 1 Introduction

BERT (Bidirectional Encoder Representations from Transformers) is a transformer-based model that generates contextual word representations [1]. Since its release, it has become a popular tool in NLP, and has been applied in tools such as web searching [2]. Today, BERT is used in nearly every single English web search query in Google [3]. Our group was motivated to enhance BERT's performance on downstream tasks, particularly sentiment analysis, paraphrase detection, and semantic textual similarity evaluation because of their role as common benchmarks for the development of model-based natural language learning. Improving performance for BERT over these downstream tasks will have critical implications for understanding how AI models can develop natural language understanding, and overall enhance our usage of BERT models on downstream applications.

The main goal of this project was to develop a multi-task classfication model based on a single BERT backbone. In particular, the BERT sentence embeddings would remain the same across all tasks – the only thing changing is the downstream prediction function dependent on the particular task we wish to accomplish. Multi-task learning is a subfield of machine learning with countless applications, as

it offers the opportunity to share structure across multiple (often related) tasks, exploit similarities and differences across tasks, make up for undersampled tasks, and generally allow for more efficient learning. However, it also presents a variety of new challenges, which we explore throughout our own project. In particular, we grappled with overfitting, some tasks being more overrepresented than others, and just how much of the same architecture any two tasks should share.

## 2 Related Work

Prior to BERT's development in 2018 [1], common pre-trained general language representations included unsupervised feature-based such as ELMo [4], fine-tuning approaches, and transfer learning from supervised data. BERT's distinction lies within its bidirectionality, giving it the ability to consider the full context of a given word, dependent on its surrounding words. This means that BERT has a deeper sense of language context than single-direction language models that came before it. Additionally, BERT has a unified architecture across different tasks, with minimal differences between its pretrain and downstream architecture.

BERT was regarded as transformative for NLP research, representing a jack of all trades within the NLP machine learning domain in comparison to existing models due to its ability to solve for many of the most common language tasks [5]. Since its release, it has gone through copious modifications, and its flexibility has been exploited for different applications, from creating NLP models in different languages [6] and specific uses such as tailoring BERT to different contexts, for example medical diagnosis [7]. Enhancing BERT's performance overall and on specific NLP tasks are also being explored, through methods such as casual attention masking [8] and linguistic information injection [9], respectively. This has generated many variants of BERT, such as optimizing for training speed [10] and performance gains [11]. We intend to continue this search for higher BERT performance.

Our work in this paper was informed by prior studies, such as the analysis done by Yu et. al. on using gradient surgery to limit the effects of cancelling gradients [12], Marius et. al. on using variable learning rate and early stopping [13], and the baseline BERT model [1] on informing architecture changes. Through the application of their work on minBERT, our paper offers insight into improving minBERT's overall performance on our three downstream tasks.

## 3 Approach

We built up to our final model by developing via an iterative procedure, taking note of what did and did not work for each new iteration of our model. In the following subsections, we detail each approach we took. Highlights from the experimental results for each approach are further expanded upon and analyzed in future sections.

### 3.1 Baselines

Our original baseline model was an elementary extension of the single-task minBERT model. In particular, we only trained on the sentiment classification task using the SST and CFIMDB datasets. Our first `predict_sentiment` function is the exact same as the one used in the single-task minBERT model. We implemented the `predict_paraphrase` and `predict_similarity` functions based on the cosine similarity between the two resulting sentence embeddings. In particular, we ran the original `forward` function, consisting of the BERT encoder, a dropout layer, and a linear layer, on both `(input_ids_1, attention_mask_1)` and `(input_ids_2, attention_mask_2)`. This yielded the two embeddings $E_1$ and $E_2$. We then calculated

$$Sim(E_1, E_2) = \frac{E_1 \cdot E_2}{\|E_1\|\|E_2\|},$$

before scaling the output value to match the labels for each respective task. Henceforth, when we refer to our "baseline model", this is the model that we are referencing.

We implemented our baseline multi-task model following the project milestone, training and evaluating on all three data-sets, but keeping the prediction functions we implemented earlier. We used naive round-robin training across all three classification tasks, interleaving batches from each dataset during each training epoch. In particular, for initial ease of implementation, the three datasets and

classification tasks were trained on a shared training loss and optimizer. We also noticed that the Quora dataset used for the paraphrase task was significantly larger than any of the other datasets, which meant a longer training time. To ensure that our training procedure was efficient while remaining effective, we introduced a `RandomSampler` to the paraphrase training `DataLoader`, sampling $6,000$ random datapoints each time (number chosen to be comparable to the other two datasets sizes).

### 3.2 Learning Rate Scheduling and Early Stopping

The first main extension to the baseline model we worked on revolved around learning rate scheduling. In particular, we referenced the paper "On the Stability of Fine-tuning BERT: Misconceptions, Explanations, and Strong Baselines" by Marius et. al. [13] Marius et. al. found that standard finetuning parameters often have "optimization difficulties that lead to vanishing gradients", and empirically showed that using a scheduler with variable learning rates dramatically improved the stability of their results. The paper explored a wide series of scheduling schemes, and we chose to experiment with linear decay with linear warm-up, step decay, and cyclic varying of learning rates. Linear decay with linear warmup starts with a learning rate of 0, increases the learning rate linearly to its pre-set value over the first 10% of steps, and then decreases the learning rate linearly to 0 during the remaining steps [14]; step decay would drop the learning rate by a certain epoch every few epochs [14]; and cyclic learning rates would vary them between two boundary values [15]. We found that linear warmup with linear decay performed the best, and we kept the scheduler active for all of our remaining experiments, but the gains were quite minimal (please see table 1 below).

Upon implementing this extension, we found that overfitting was an immediate issue. Marius et. al. addressed this problem by introducing early stopping as a form of regularization. We successfully implemented early stopping, which reduced computational time and also led to a tiny increase in score. However, the increases were minimal enough that we weren't sure if the increase in score can be directly attributed to the implementation of early stopping or if it was just randomness.

Though we experimented with a variety of different learning rate schedulers, we failed to see the improvement that Marius et. al. described in their paper. We suspected that we may be initializing the model with the wrong hyperparameters. In an attempt to further explore these methods, we implemented a rigorous hyperparameter tuning algorithm that would allow us to perform an expansive search through different hyperparameter combinations.

### 3.3 Hyperparameter Tuning

We implemented hyperparameter search, leveraging Ray Python library's existing modules for hyperparameter search. Some initial ideas included implementing grid search, random search, an ASHA (Asynchronous Successive Halving Algorithm) scheduler, and a PBT (Population-Based Training) scheduler. Since Ray Tune was defined to work with the transformers library in particular, which we did not use, implementing these schedulers required significant modifications to our model's source code and interface. We first attempted to implement an ASHA scheduler [16], which is designed to effectively search the parameter space through exploiting parallelism. This is done by ending the training of tasks with suboptimal performance early, saving both computational energy and training time. We decided on trying ASHA first instead of other popular searching schedulers due to its efficacy when the hyperparameter search space is not very large, but the training process is long [17], particularly when massive parallelism is not available. Although the more common PBT and BOHB (Bayesian Optimization Hyperband) [18] schedulers are methods that also exploit this partial training, ASHA's aggressive early stopping has been shown to outperform both of those schedulers [17], which made it attractive for our own hyperparameter tuning. However, after implementing ASHA, we found that it came with a significant flaw in that its computational runtime was unreasonably long without parallel computing. In practice, after running the tuner for 24 hours, we finished less than 5 total trials. As a result, under the computing constraints we had, our implementation of ASHA on our model did not run effectively enough.

As such, we ultimately decided to use a PBT scheduler instead. While not as aggressive with its early stopping as ASHA, it required fewer computational resources memory wise, and was able to complete hyperparameter search with significantly less computing time. PBT is an asynchronous optimization algorithm which begins like a grid search, randomly sampling hyperparameters and weight initializations, but reassessing performance periodically [19] to exploit and explore the

parameter space. If the current running model is underperforming, the model will replace its own hyperparameters with those of a better performing model, representing exploition by hijacking its own underperforming parameters. The better model will then continue exploring the parameter search space by modifying its own hyperparameters to new hyperparameters.

In particular, we focused on the hyperparameters of learning rate, batch size, hidden dropout, and weight decay, looking for the best trade-ofs between training speed and performance while avoiding over-fitting. Through the use of hyperparameter search, we found some increase in accuracy across the three tasks, however, they were still limited to below average scores as compared to other students.

### 3.4 Gradient Surgery

Despite an extensive use of hyperparameter tuning, we saw little improvement in our results and our accuracy scores remained quite low. We hypothesized that this might be due to the shared training loss and optimizer we used across all three classification tasks. In multi-task classification, it's possible that the loss gradients for each task point in different directions. Then, updating the same optimizer according to these contradictory gradients would possible obfuscate a meaningful update step in any of the desired directions. In an effort to alleviate this conflict, we turned to gradient surgery.

Gradient surgery is a method to mitigate potential gradient interference through modification of the gradients directly. The method we chose to use is a form of gradient surgery known as projecting conflicting gradients ("PCGrad") [20], due to it being model agnostic, allowing it to be implemented to many different tasks. PCGrad projects any given task's gradient onto the normal plane for the gradient of any other task with a conflicting gradient. Specifically, denote $g_i$ to be the gradient for task $\mathcal{T}_i$. (In our case, $i \in \{1, 2, 3\}$.) For a given task $\mathcal{T}_i$ and any other task $\mathcal{T}_j$, we first determine whether or not the gradients $g_i$ and $g_j$ conflict with each other. To do so, we calculate the cosine similarity between the two vectors, where a negative value indicates a conflict. If a conflict exists, we update $g_i$ as follows:

$$g_i := g_i - \frac{g_i \cdot g_i}{\|g_j\|^2} \, g_j.$$

This represents the projection of $g_i$ onto the normal plane of $g_j$. PCGrad repeats this same process for all tasks in the batch, in random order. To implement this extension in our model, we used the tools provided by the `PCGrad` library [12] [21], with some significant modifications to make it work nicely with our hyperparameter search harness.

We found that our model appeared to converge faster after the implementation of gradient surgery, but still lacked any significant increase in accuracy scores. In particular, before we implemented gradient surgery, our experiments typically trigger early stopping after 15+ epochs, whereas after gradient surgery they typically early-stop within 10 epochs.

### 3.5 Combatting STS Overfitting

Out of our three tasks, we noticed that the task with consistently the lowest devset accuracy was semantic textual similarity (STS) evaluation. On the other hand, the training accuracy was always very high. In another effort to combat this overfitting, we attempted data augmentation on the STS training set. Alongisde the SemEval STS Benchmark dataset that was provided, we decided to incorporate the SICK dataset [22], which contains sentence value pairs along with similarity values and has been used in many other NLP papers as a benchmark for semantic textual similarity. However, augmenting our training set with the SICK dataset did not raise our STS accuracy noticably.

For this iteration of our model, we also pivoted away from using cosine similarity to calculate `predict_paraphrase` and `predict_similarity`, as it seemed like a crude metric for measuring how close two embedding vectors were. Instead, we concatenated the two embeddings (defined as $E_1$ and $E_2$ in Section 3.1, as the output after passing the tokenized sentences into the `forward` function) into a single sentence embedding. This single vector was then passed through a new linear layer with the appropriate dimensions. This would allow the model to train for the optimal weights to produce the best predictions.

Given that our paraphrase detection task was performing quite well with random sampling, we experimented with implementing a sampler for the STS dataset as well. However, we still saw negligeable improvement.

We further experimented with different hyperparameters, chosen using the PBT scheduler in hopes that reevaluating parameters would help us improve accuracy. We saw some mild improvements, however, despite hyperparameter tuning, our accuracy still remained under a threshold of 0.5.

### 3.6 Final Model

After experimenting with a variety of extensions and running an extensive hyperparameter search multiple times without seeing any major improvements, we came to the conclusion that there must be a limitation in our model architecture itself. We reanalyzed the ways we designed our sentiment and paraphrase prediction algorithm and found that making some changes to the way we dealt with the inputs and the resulting embeddings led to significantly higher performance. In particular, we had been running each of our two input sentences separately through the `forward` function and then concatenating them together and passing them through an individual predictive linear layer to generate our final predictions (as described in Section 3.4). As such, this means that the three classification tasks shared more than just the BERT backbone – they also shared the same initial linear layer. We hoped to create a multi-task classifier for which a single BERT encoder could produce sentence embeddings suitable for three separate downstream tasks. But by further including a shared linear layer, we were unnecessarily entangling the three tasks.

Two modifications were made to this architecture that dramatically increased our results. Our first modification was to preprocess the data for both `SentencePairDataset` and `SentencePairTestDataset`. Rather than keeping the two sentences separate, we concatenated the tokens for both input sentences together. Sentences came with a '[CLS]' token appended to the beginning and a '[SEP]' token appended at the end. To match the tokenized input used by the original BERT paper for the STS task, we stripped out the '[CLS]' token at the beginning of the second sentence and the '[SEP]' token at the end of the second sentence. This single tokenized representation for both sentences was then passed to `predict_paraphrase` and `predict_similarity`. Our second modification was that we also implemented a separate `forward` function for each task, meaning that we kept the shared BERT backbone but used separate linear layers for each. As such, different tasks were no longer sharing weights while training, and each linear layer could be dedicated to being optimized for a single task. Running our new model architecture showed that we had significantly less overfitting across all three tasks, and our text similarity task in particular improved the most.

After that modification to our model, we reran PBT scheduler-based hyperparameter tuning, in order to develop our final model and parameters. We found a significant increase in our scores across all three tasks compared to our baseline.

## 4 Experiments

### 4.1 Data

For our project, we used the Stanford Sentiment Treebank (SST), CFIMDB, Quora, and SemEval datasets to finetune our model. The SST dataset consists of 11,855 sentences from movie reviews [23], while the CFIMDB dataset [24] contains 2,434 polarized, multi-sentence movie reviews. The Quora dataset is [25] composed of 400,000 question pairs with labels indicating whether particular instances are paraphrases of one another, and the SemEval STS Benchmark dataset [26] consists of 8,628 different sentence pairs of varying similarity on a scale from 0, symbolizing unrelated similarities, to 5, where the pairs have equivalent meaning. We used all of these datasets to train our minBERT model, and then evaluated the trained model on the following tasks: sentiment analysis, paraphrase detection, and semantic textual evaluation.
Additionally, we used the SICK dataset [22] in order to augment the semantic textual similarity evaluation task. This dataset is composed of 4,439 sentence pairs generated from image and video captions, with each pair given a relatedness score from 1 to 5.

### 4.2 Evaluation method

For the evaluation metric, we compared the score of our model across the three tasks: SST, Paraphrase, and STS SemEval, as well as the aggregate score. The model variant scores were compared to the scores attained by our baseline model, which was a modified single-task sentiment analysis minBERT model with AdamW Optimization. This allowed us to gauge the overall improvement of our model.

In particular, the baseline aggregate score was 0.361, breaking down into accuracies of 0.52, 0.37, and 0.18 for sentiment analysis (using the SST dataset), paraphrase detection (using the Quora dataset), and semantic textual similarity (using the STS SemEval dataset) respectively.

### 4.3 Experimental details

For each of the non-starred elements in the table below, our configurations were as follows: we trained on a BERT model (for which each embedding layer has dimensionality 768, and the overall BERT encoder consists of 12 Transformer layers) with updatable parameters, using appropriately sized linear layers, a dropout layer with probability 0.1, a learning rate of 1e-5, a weight decay of 0.0, a batch size of 16, trained over 20 epochs (with early stopping), saving best performing model.

For the starred elements, we ran the same model configuration through hyperparameter search, using our PBT scheduler with 20 trials. We allowed the learning rate to pertube between 1e-6 and 5e-4, the weight decay to pertube between 0.01 and 0.2, and the hidden dropout to pertube between 0.01 and 0.5. We ran the hyperparameter search over just 5 epochs for each trial to save time, and re-ran the best performing hyperparameters over 20 epochs (with early stopping) to get our results.

### 4.4 Results

Our initial experimentation began by evaluating our baseline model over our three tasks, with selected experiments shown in Table 1. Importantly, these initial experiments were run when our implementation of the Paraphrase and STS prediction functions were incorrect. Compared to the baseline, which is defined on line 1, adding in multitask did increase our overall score, by 0.048.

(The * symbol in the tables represents the best hyperparameter search score, detailed in section 4.3.)

Table 1: Improper Prediction Metrics for Paraphrase and STS

| Experiment | SST Dev Acc | Paraphrase Dev Acc | STS Dev Corr | Overall Dev Score |
|---|---|---|---|---|
| `Baseline` | 0.524 | 0.374 | 0.184 | 0.361 |
| `Baseline` + `Multitask` | 0.526 | 0.625 | 0.077 | 0.409 |
| `+ Multitask` + `Variable Learning` + `Early Stopping`* | 0.530 | 0.375 | 0.205 | 0.370 |

After accounting for proper prediction metrics, we conducted gradient surgery, data augmentation, and random sampling, running PBT hyperparameter search over the model that included all three augmentations. Gradient surgery increased our overall dev score to 0.500, a significant boost from our earlier scores. However, applying data augmentation and random sampling had the unintended effect of decreasing the overall dev score.

Table 2: Proper Prediction Metric and Inferior Architecture

| Experiment | SST Dev Acc | Paraphrase Dev Acc | STS Dev Corr | Overall Dev Score |
|---|---|---|---|---|
| `+ Gradient Surgery` | 0.499 | 0.726 | 0.274 | 0.500 |
| `+ Gradient Surgery` + `Data Augmentation` | 0.480 | 0.741 | 0.268 | 0.496 |
| `+ Gradient Surgery` + `Data Augmentation` + `Random Sampling`* | 0.482 | 0.722 | 0.208 | 0.471 |

Our modification of the model architecture (as described in section 3.6) led to significantly improved results. Running hyperparameter search lead to a final improvement over our baseline by 0.368. In particular, the best hyperparameters for our final model architecture was a learning rate of 4.187e-5, a weight decay of 0.1829, a hidden dropout rate of 0.403, and a batch size of 8. From our original baseline score, it represents an increase in our overall dev score by nearly 102%, and an increase over the non-tuned model by 0.29.

Table 3: Optimal Architecture

| Experiment | SST Dev Acc | Paraphrase Dev Acc | STS Dev Corr | Overall Dev Score |
|---|---|---|---|---|
| `Final Model Architecture` | 0.499 | 0.799 | 0.803 | 0.700 |
| `Final Model Architecture`* | 0.484 | 0.851 | 0.853 | 0.729 |

Finally, our submissions to the test sets are displayed below, in Table 4. Within our final model architectures, hyperparameter tuning demonstrated an increase in score by 0.048.

Table 4: Test Set Scores

| Experiment | SST Test Acc | Paraphrase Test Acc | STS Test Corr | Overall Test Score |
|---|---|---|---|---|
| `Final Model Architecture` | 0.529 | 0.796 | 0.812 | 0.712 |
| `Final Model Architecture`* | 0.532 | 0.857 | 0.854 | 0.748 |

## 5 Analysis

Overall, our model was able to significantly improve upon the baseline single-task minBERT model with AdamW optimization. Modification of our model architecture had the largest influence towards increasing accuracy, and when used in tandem with hyperparameter search and other scheduling/regularization techniques, allowed us to significantly improve on our three downstream tasks.

Perhaps most notably, our final model performs relatively well on the paraphrase detection and semantic textual similarity tasks. On the other hand, the sentiment analysis task performs quite well during training but plateaus around $0.5$ during validation and testing. We hypothesize that this is because sentiment analysis is a more nuanced task. Though semantic textual similarity also deals with the meaning encoded in a sentence, it also relies on the comparison of two sentence samples, which is likely easier to encode within a BERT model. On the other hand, sentiment analysis task revolves around a single sentence, and relies on significant knowledge beyond the context of a BERT model. It's common that the same word used in different contexts conveys an entirely different sentiment, and so on. Further, one goal of multi-task learning is for the different tasks to inform each other and provide some extra context. Intuitively, it makes more sense for the sentiment analysis task to provide insight into the other two tasks than for the opposite, particularly because the other two tasks rely on comparisons of two sentence samples. Also, it is likely that the two tasks of paraphrase detection and semantic textual similarity is able to inform each other more, owing to the fact that both deals with comparisons between two sentences and as such can co-opt each others weights to an extent. As such, it's possible that the semantic analysis task is at a disadvantage.

Finally, note that our final model architecture indeed performs better after having been trained with the best hyperparameters as opposed to the default parameters given in the handout. Plots for the train and dev accuracy during these two experiments is in the Appendix.

## 6 Conclusion

Though our modification of BERT, we found that our modified architecture with the addition of all our extensions (variable learning, early stopping, gradient surgery, data augmentation, random sampling, and PBT hyperparameter search) was ultimately to achieve a final improvement over the baseline minBERT model with AdamW optimization by 0.368, or by nearly 102%.

In the future, we will tweak our current implementation and consider further regularization methods, including AUBER [27] and data augmentation techniques such as synonym replacement, random insertion, random swap, and random deletion [28]. We may also look into re-implementing the ASHA scheduler for hyperparameter tuning with greater computational resources, which will allow for greater finetuning.

It must be noted that our best results came at the very end, when we implemented changes to our overall model architecture. Running an ablation study with all the other extensions we included on this final model architecture would allow us to see which ones are truly successful and which ones are less important or even detrimental. An additional experiment we might hope to explore is different ways to preprocess the joint tokenized sentence representation. In our final model, we removed the '[CLS]' and '[SEP]' tokens from the second tokenized sentence to match the input format from the BERT paper, and we're curious to see how it would affect our performance if we didn't do so.

## References


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

[2] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. A primer in BERTology: What we know about how BERT works. *Transactions of the Association for Computational Linguistics*, 8:842–866, 2020.

[3] Barry Schwartz. Google: BERT now used on almost every English query — searchengineland.com. `https://searchengineland.com/google-bert-used-on-almost-every-english-query-342193`. [Accessed 19-Mar-2023].

[4] Matthew E. Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. Semi-supervised sequence tagging with bidirectional language models. *CoRR*, abs/1705.00108, 2017.

[5] BERT 101 - State Of The Art NLP Model Explained — huggingface.co. `https://huggingface.co/blog/bert-101`. [Accessed 19-Mar-2023].

[6] Zihan Zhang, Jinfeng Li, Ning Shi, Bo Yuan, Xiangyu Liu, Rong Zhang, Hui Xue, Donghong Sun, and Chao Zhang. Rochbert: Towards robust bert fine-tuning for chinese, 2022.

[7] Pavel Blinov, Manvel Avetisian, Vladimir Kokh, Dmitry Umerenkov, and Alexander Tuzhilin. Predicting clinical diagnosis from patients electronic health records using BERT-based neural networks. In *Artificial Intelligence in Medicine*, pages 111–121. Springer International Publishing, 2020.

[8] Ziyang Luo, Yadong Xi, Jing Ma, Zhiwei Yang, Xiaoxi Mao, Changjie Fan, and Rongsheng Zhang. Decbert: Enhancing the language understanding of bert with causal attention masks, 2022.

[9] Nicole Peinelt, Marek Rei, and Maria Liakata. GiBERT: Enhancing BERT with linguistic information using a lightweight gated injection method. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2322–2336, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[10] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2020.

[11] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[12] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *arXiv preprint arXiv:2001.06782*, 2020.

[13] Marius Mosbach, Maksym Andriushchenko, and Dietrich Klakow. On the stability of fine-tuning bert: Misconceptions, explanations, and strong baselines. *Saarland Informatics Campus*.

[14] Suki Lau. Learning Rate Schedules and Adaptive Learning Rate Methods for Deep Learning — towardsdatascience.com. `https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1`. [Accessed 20-Mar-2023].

[15] Leslie N. Smith. Cyclical learning rates for training neural networks, 2017.

[16] Liam Li, Kevin G. Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. Massively parallel hyperparameter tuning. *CoRR*, abs/1810.05934, 2018.

[17] Jack Parker-Holder, Vu Nguyen, and Stephen Roberts. Provably efficient online hyperparameter optimization with population-based bandits, 2021.

[18] Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale, 2018.

[19] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population based training of neural networks, 2017.

[20] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning, 2020.

[21] Wei-Cheng Tseng. Weichengtseng/pytorch-pcgrad, 2020.

[22] Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, and Roberto Zamparelli. The SICK (Sentences Involving Compositional Knowledge) dataset for relatedness and entailment, May 2014.

[23] Jean Wu Jason Chuang Christopher D Manning Andrew Y Ng Richard Socher, Alex Perelygin and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, 2013.

[24] Cs 224n: Default final project: Minbert and downstream tasks. *CS 224N: Default Final Project: minBERT and Downstream Tasks*.

[25] Shankar Iyer, Nikhil Dandekar, and Kornél Csernai. First quora dataset release: Question pairs. *Quora*, 2017.

[26] Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity multilingual and cross-lingual focused evaluation. Aug 2017.

[27] Hyun Dong Lee, Seongmin Lee, and U. Kang. Auber: Automated bert regularization. *PLOS ONE*, 16(6):1–16, 06 2021.

[28] Jason W. Wei and Kai Zou. EDA: easy data augmentation techniques for boosting performance on text classification tasks. *CoRR*, abs/1901.11196, 2019.

# A Appendix

Figure 1: A representation of the train accuracy as it changes per epoch, for the default parameters and the tuned hyperparameters for the final model.

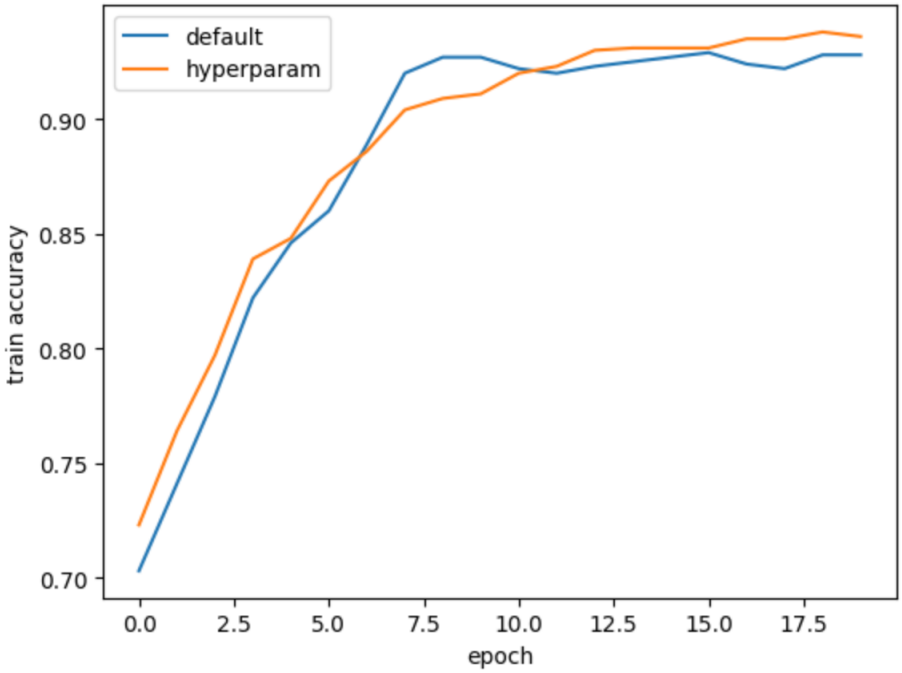


Figure 2: A representation of the dev accuracy as it changes per epoch, for the default parameters and the tuned hyperparameters for the final model.