Combining Contrastive Learning with Layer Utility Analysis and Experimental Multi-Task Finetuning to Improve mini-BERT Performance

Stanford CS224N Default Project

Ellie Vela
Department of Computer Science
Stanford University
elijahv@stanford.edu

Vionna Atefi
Department of Symbolic Systems
Stanford University
vionnaa@stanford.edu

Ethan Bogle
Department of Computer Science
Stanford University
ebogle@stanford.edu

Abstract

Our overall experimental approach was to run several different experiments on a base minBERT model in parallel, such as simCSE integration, layer utility analysis, hyperparameter variation, and gradient surgery, and use the best performing setups from each of our parallel experiments to create a single model that could best handle a variety of downstream tasks. Our best performance was achieved by our final minBERT model with the following extensions: layer utility analysis finetuned with multi-task learning. More narrowly our final architecture was the use of custom heads with 2 transformer layers and 1 linear layer attached to the Bert model's middle layers for SST and Paraphrase, and just a linear classifier attached to the end of the BERT model for STS.

1 Key Information to include

- Mentor: Timothy Dai
- Sharing project: Yes, E.B. is sharing project with CS 281.

2 Introduction

Natural Language Processing (NLP) is a rapidly evolving field that has experienced unprecedented growth and transformation in recent years driven by advancements in machine learning, linguistics, and an increasing availability of vast amounts of textual data. As a domain, NLP seeks to enable machines to better understand, interpret, and generate human language with the greatest degree of accuracy and sophistication. Significant innovations in this domain, such as the invention of transformer architectures or large language models, has helped NLP research in this ongoing quest.

However, the added complexity of multitask finetuning translates to unique challenges and open issues in NLP research. For example, traditional state-of-the-art sentence embedding methods have struggled to generate sentence representations that effectively capture the nuanced semantic relationship between sentences. Traditional solutions often rely heavily on supervised learning tasks or complex unsupervised methods that do not fully exploit the potential of pre-trained language models. Tianyu Gao and Chen (2021)

In this body of research, we preliminarily implemented a minimal version of Devlin's BERT architecture and used pretrained sentence embeddings for three downstream NLP tasks: sentiment analysis, paraphrase detection, and semantic textual similarity.

3 Related Work

Devlin et al.'s 2018 paper "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" introduced BERT, a groundbreaking model in the field of NLP. Key contributions of the paper included pretraining deep bidirectional representations from unlabeled text by conditioning on both left and right context in all layers Jacob Devlin and Toutanova (2018)

The paper demonstrated that BERT, and its novel bidirectional transformer-based approach, could achieve state-of-the-art results on a wide range of NLP tasks, including the General Language Understanding Evalutaion (GLUE) benchmark and the Stanford Question Answering Dataset (SQuAD) v.1.1 and v2.0. The success of BERT inspired further research into transformer-based pretrained models.

4 Approach

4.1 Bert Implementation

The BERT model architecture is composed of a stack of 12 Enoder Transformer layers. As the starting point for our research, we implemented a minimal BERT model in line with the architecture outlined in the original research paper by Devin et al. (2018). Namely we have implemented the multi-head self-attention and transformer layers of the original BERT model and completed the multi-head self-attention component followed by layer normalizations and a position-wise feed-forward layer. Dropout was applied with the setting $p_{drop} = 0.1$.

4.2 Downstream Tasks

We additionally completed an implementation of the multitask BERT model, which attempts to produce embeddings that generalize across three different tasks: Sentiment Classification, Paraphrase Detection, and Semantica Textual Simialrity.

Sentiment classification is the task of predicting whether a particular piece of example text expresses a positive or negative sentiment. Sentiment classification models can be helpful for understanding general sentiment towards a particular entity, i.e. a person, brand, organization, etc. Naturally, these models are very useful when applied in combination with Named Entity Recognition models. Our model adds a linear layer on top of the base BERT implementation that classifies inputs into one of 5 sentiments: negative, somewhat negative, neutral, somewhat positive, or positive. We trained our model on the cross-entropy loss across 5 logits, one corresponding to each sentiment category.

Paraphrase detection is the task of determining whether two piece of example text express the same meaning. Importantly, detecting paraphrases of text requires some understanding of its semantic content. Therefore, a model's performance on the paraphrase detection task captures the extent to which the model understands the semantic content of text. Like with sentiment classification, we add a linear layer on top of the base BERT model. Unlike sentiment classification, though, paraphrase detection is a binary classification task. We therefore trained our model on the binary cross entropy loss for paraphrase detection.

Semantic Textual Similarity is the task of scoring the similarity between two sentence's semantic context. As with paraphrase detection, this task captures how well a model understands the semantic context of the text it sees. It differs from paraphrase detection, though, in that it is not a binary output. Rather, it is a sliding scale from 0 to 5. A score of 0 means that the two pieces of text have entirely different meanings, and a score of 5 means that the two pieces of text are equivalent i.e. have almost the same exact meaning. Again, we add a linear layer to the base BERT model to predict Semantic Textual Similarity. We train our model on the mean-squared error for this task.

Because we take a Multi-Task Learning approach, we sum the losses to perform multitask finetuning and save model weights based on the summed loss. More details on our implementation of Multi-Task Learning are included in section 4.6.

4.3 Optimizer

We furthermore employed an Adam Optimzer as a stochastic optimization method that adjusts learning rates for various parameters, which is beneficial for dealing with sparse gradients and improving convergence. Our implementation included the step() function.

4.4 SimCSE Integration

Upon our base BERT model, we implemented a unsupervised SimCSE contrastive learning framework as proposed by Tianyu Gao (2021). The unsupervised SimCSE approach takes an input sentence and predicts itself in a contrastive objective with only standard dropout used as noise. In other words, the same sentence is passed through our pre-trained BERT model twice to obtain two different embeddings as a "positive," semantically related pair. Then all the other sentences in the mini batch are taken as "negatives." From this the contrastive framework in Chen et al. (2022) is followed and a cross-entropy objective is taken with the in-batch negatives.

More concretely, given a sentence s, we generate two embeddings, h and h' where

$$h = BERT_{dropout_1}(s)$$
$$h' = BERT_{dropout_2}(s)$$

Contrastive learning aims to improve sentence embeddings by together semantically similar neighbors and pushing apart non-neighbors by taking the following training objective with a mini batch of N pairs where τ is a temperature hyper-parameter:

$$L = -\log \frac{e^{sim(h,h')/\tau}}{\sum_{j=1}^{N} e^{sim(h,h')/\tau}}$$

4.5 Layer Utility Analysis

A difficulty with finetuning a single model on all three tasks is that the three tasks may work at odds to each other, requiring the model to represent relevant information for all three tasks that may be in conflict. The obvious solution is simply finetuning three separate copies of the model, but this is not in the spirit of the multitask learning problem at hand.

Our proposed solution is to attach the classification heads to different layers of the model. This increases the task-independence of the various model layers, but requires choosing at which layers to attach the classification heads.

It is well-known that in speech encoders, different layers of the model express different types of information accessibly. For example, Bartelds and Wieling (2022) experimented with using ASR models to distinguish regional dialects of Dutch. The middle model outputs (layers 13, 15, or 16 out of 24, depending on the model) performed this clustering best, indicating that middle layers contain the most phonological information. In audio-related tasks, Pasad et al. (2021) showed that sometimes, reinitializing the final layers of a model from scratch sometimes improves performance on downstream tasks (especially in low-resource settings), since the downstream tasks do not necessarily correlate with the pretraining task.

We test to see if these principles apply to a language encoder like BERT as well, when used on downstream tasks that do not require complete comprehension. We test which layers are best at Sentiment Classification, Paraphrase Detection, and Semantic Textual Similarity.

The model architecture for this experiment is depicted in Figure 1: We attach a classification head to each hidden layer of the model. The underlying BERT model is held fixed while these classification heads are simultaneously but independently trained. After each epoch of training, train and dev performance are computed and saved for later analysis. The structure of the classification head consists of one to three transformer layers (which can be randomly initialized or preinitialized), followed by extracting the pooler and performing linear classification (into 5, 2, and 6 classes, respectively, for SST, quora, and STS). For sentence comparison tasks, the two poolers are concatenated before the linear layer.

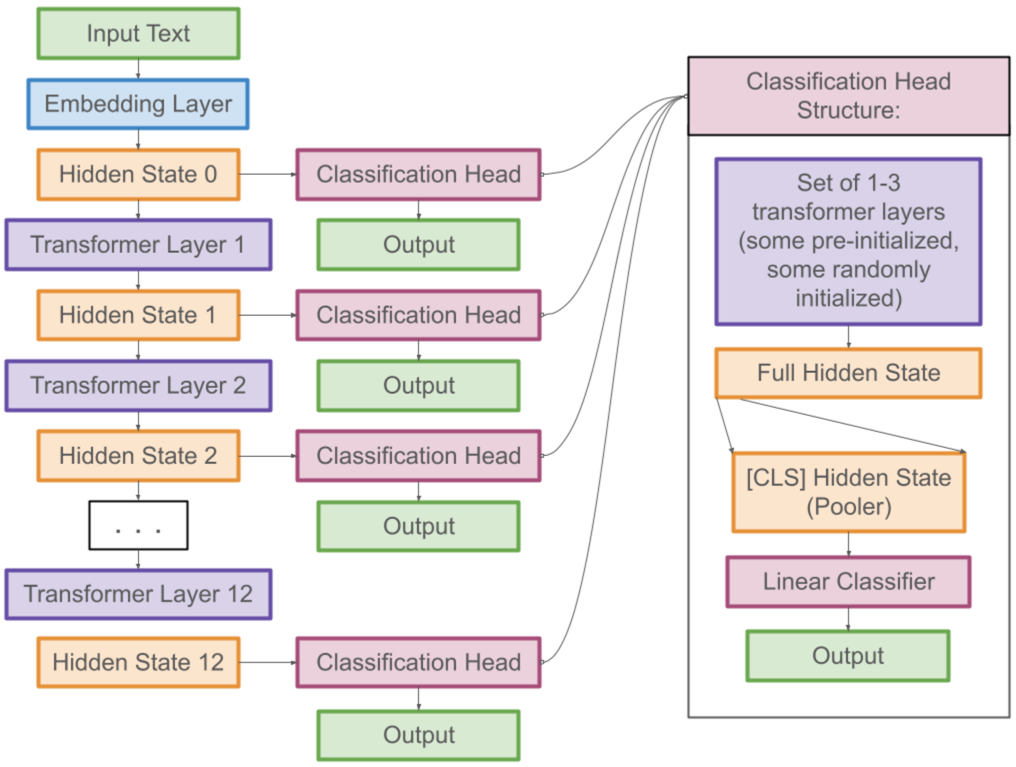

Figure 1: Model Network for BERT probing experiment. The left column is the original BERT model (weights held fixed). The middle column represents extracting the hidden states at each layer to apply an individually-trained head. The classification heads themselves consist of 1-3 transformer layers which are either randomly initialized or initialized equal to the corresponding next layer in the BERT model. For example, in the experiment condition with 1 preloaded transformer layer, the classification head attached at layer 3 would have its single transformer initialized to the layer 4 BERT transformer weights.

4.6 Multi-Task Learning

One existing method for learning generalizable embeddings targetting multiple tasks is Multi-Task Learning (MTL). Rather than training on tasks separately, MTL has the model trained on multiple tasks simultaneously. This approach leverages the use of multiple tasks and tries to learn generalizable information across the separate tasks.

To take advantage of multiple tasks at training time, a curriculum should be designed that does not simply present the model with all of task 1 data, then all of task 2 data, and so one. Instead, the curriculum should alternate between each tasks's datasets. Additionally, the training loss across each task is summed before backpropagation, again to take advantage of the mult-task setting. This all ensures that the model is able to learn generalizable information between all of its tasks.

There is a possiblity that during training the gradients of different tasks conflict. This can impede the model training. Yu et al. (2020) recommend an approach called "Gradient Surgery", which projects the gradients of conflicting tasks onto each others' normal planes:

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

. We use the PCGrad implemetation from Tseng (2020) to apply gradient surgery to our model.

5 Datasets

Our base minBERT model leveraged two datasets for the purposes of sentiment analysis: the Stanford Sentiment Treebank (SST) dataset and the CFIMDB dataset.The SST dataset consists of 11,855 single sentences from movie reviews extracted from movie reviews, extracted and parsed to include a total of 215,154 unique phrases. Each phrase has one of five sentiment labels: negative, somewhat negative, neutral, somewhat positive, or positive. Richard Socher and Potts. (2013) The CFIMDB dataset consists of 2,434 highly polar movie reviews, each labeled with a binary label of negative or positive.

For the task of paraphrase detection, we leveraged the Quora Dataset which consists of 400,000 question answer pairs, each annotated with a binary label indicating whether the pair represents a paraphrase of each other.

Lastly, to assess semantic textual similarity, we used the SemEval STS Benchmark dataset that features 8,628 sentence pairs. Each pair is scored from 0 (indicating no relation) to 5 (indicating equivalent meaning).

6 Experiments

Our overall experimental approach was to run several separate experiments in parallel. Then, the best performing setups from each of these parallel experiments were combined into a single model, and experiments ran on the combined model.

6.1 Evaluation method

For our evaluation method, we utilized the default evaluation methods outlined in the CS224N default project handout. We also utilized comparing the overall dev score across versions of our BERT model with different implemented extensions; which is calculated by normalizing the score of each downstream task between 0 and 1.

6.2 SimCSE Implementation vs. Baseline

Our initial SimCSE experiments were not particularly promising with our SimCSE-implemented model not performing much better on the STS nor CFIMDB datasets better than random chance. These initial results are summarized in Table 1. Based on these results we largely abandoned integrating SimCSE into our final model.

Table 1: Initial SimCSE-implemented model performance on the STS and CFIMDB datasets

	SST dataset	CFIMDB dataset
train loss:	58.224	57.968
train accuracy:	0.190	0.499
overall dev accuracy:	0.208	0.498

6.3 Layer Utility Analysis

The architecture depicted in Figure 1 was used to analyze how well each layer performs each task (sentiment analysis, paraphrase detection, or semantic textual similarity). Tasks were evaluated with the following configurations of the classifier head transformer block: 1 layer randomly initialized, 1 layer model-initialized, 1 model- and 1 randomly-initialized, 2 model-initialized, and 2 model- and 1 randomly-initialized.

The model was trained for 10 epochs, with learning rate 1e-5. Train and dev accuracy, correlation, and loss were recorded for all tasks. The six accuracy metrics (train/dev loss/acc/corr) were graphed against both epoch number and attachment location, with the other of the two represented by distinct lines. The most relevant graphs and a GitHub link to the repository with raw data and the remaining graphs are available in Appendix A.

Sentiment Classification (best dev accuracy: 0.51): The best dev accuracy is achieved at roughly layers 8-12 (counting the total number of pretrained layers in the pipeline, including any model-initialized layers in the classification head). Only 2-4 epochs are generally required to achieve this accuracy, and neither later layers nor more pretraining increase it. The peak dev accuracy of around 0.51 was also completely independent of the specific classification head used. The train accuracy climbs steadily, faster the more layers are in the head.

Because the model eventually reaches a uniform peak in its generalizable knowledge despite being able to overfit the training data, no matter the head, we conclude that there is a fundamental limit to the default architecture's ability to perform this task with this amount of finetuning data.

Paraphrase Detection (best dev accuracy: 0.795): Due to the size of the training set, only 1-layer random, 1-layer model-initialized, and 2-layer model-initialized heads could be run, for 5 or 6 epochs.

Dev accuracy continues to climb with each epoch. Optimal accuracy is achievable with 4 or more pretrained layers. Increasing the number of pretrained layers being finetuned improved accuracy (0.77, 0.785, 0.795 for 1-layer random, 1-layer model-initialized, and 2-layer model-initialized). The training accuracy is larger, but otherwise follows the same patterns as the dev accuracy. Clearly, we have not yet reached the model's funadmental limits for this task, and the fact that only 4 pretrained layers are required to achieve the current optima bodes well for future training efficiency improvements.

Semantic Textual Similarity (best correlation: 0.35): Results are inconsistent, though the end of the model appears to be the most reliable. Most improvement to dev correlation occurs within 4 epochs, but accuracy continues to climb beyond that. The dev correlation bounces around unpredictably.

The training correlation grows steadily, however (faster with more head layers). The combination of bouncing (but still increasing) dev correlations and steady train correlation growth suggests that the model *is* learning generalizable information, but the learning rate may be too big. This may also be attributable to a difference in model architecture, as the heads perform classification rather than returning a single decimal value.

Recommended Heads:

- Sentiment Analysis: 2-layer model-initialized head at layer 8 (total 10 pretrained layers)
- Paraphrase Detection: 2-layer model-initialized head at layer 2 (total 4 pretrained layers)
- Semantic Textual Similarity: 2-layer model-initialized head at layer 10 (12 pretrained layers; adjusted in combined experiments due to other results)

6.4 Multi-Task Learning

To get the best performance possible out of our multi-task training approach, we trained our base BERT model with MTL on several experimental setups, detailed below.

6.4.1 Hyperparameter Variation

The model seemed to be overfitting on the data. We therefore varied the model's hyperparameters to avoid this. Table 2 shows the dev set scores after trainingf or one epoch under various configurations.

Additionally, we tried varying just the learning rate with a longer training time of 3 epochs. See Table 3.

6.4.2 Gradient Surgery

We attempted to address problems with conflicting gradients by applying gradient surgery. We used MTL to finetune our model for 10 epochs, then trained with gradient surgery for an additional 3 epochs. See table 4.

6.5 Combined Model: Custom classification heads with Multi-Task Learning

Our final combined model used the custom classification heads attached at the recommended head locations from section 6.3 and training using MTL. We ran two trials: one where we finetuned the

Table 2: Dev set scores under various model configurations. Varying dropout probability had little effect, but larger learning rates correlated with improved model performance.

Dropout Probability	Learning Rate	Sent Acc	Para Acc	SemEval Corr
0.3	1e-03	0.362	0.684	0.346
0.3	1e-05	0.304	0.637	0.027
0.3	1e-07	0.142	0.632	-0.016
0.3	1e-09	0.144	0.555	-0.019
0.5	1e-03	0.359	0.643	0.342
0.5	1e-05	0.299	0.635	0.026
0.5	1e-07	0.142	0.632	-0.016
0.5	1e-09	0.144	0.553	-0.019
0.7	1e-03	0.351	0.646	0.322
0.7	1e-05	0.293	0.632	0.026
0.7	1e-07	0.142	0.632	-0.016
0.7	1e-09	0.144	0.552	-0.019

Table 3: Dev set scores with various learning rates. Dropout probability was set to 0.3. Again, we saw larger learning rates improved model performance.

Learning Rate	Sent Acc	Para Acc	SemEval Corr
0.01	0.340	0.700	0.439
0.001	0.382	0.686	0.393
1e-05	0.304	0.645	0.046

Table 4: Applying gradient surgery improved one score at the cost of the other two.

Trial	Sent Acc	Para Acc	SemEval Corr
Default Finetuning 10 epochs	0.405	0.693	0.456
Gradient Surgery 3 Additional Epochs	0.393	0.661	0.476

custom classification heads for 10 epochs, and one where we trained the full model for 1 epoch. For both trials, we used a dropout probability of 0.3 and a learning rate of 1e-05. Our results are shown in table 5.

Table 5: Test Leaderboard Results. We compare the baseline BERT embeddings against two versions of our model: one with the BERT weights frozed during training (Last-Layer) and one with updates made to the full model during training (Full-Model)

Attempt	SST acc	Para acc	STS correlation	Overall dev score
Baseline (DEV)	0.332	0.569	0.244	0.508
Full-Model (DEV)	0.496	0.785	0.331	
Full-Model (TEST)	**0.507**	**0.784**	**0.310**	**0.649**
Last-Layer (DEV)	0.480	0.802	0.146	
Last-Layer (TEST)	0.500	0.798	0.098	0.615

Our test leaderboard scores were about as expect for SST and Paraphrase detection tasks, but lower than expected for STS. Within our development pipeline we received better scores than our test leaderboard submission for our final model performance on specifically paraphrase detection and STS correlation tasks (all scores included in the table above). The version of our model submitted to the test leaderboard might be slightly overfitting.

7 Analysis

Using SimCSE didn't seem to improve scores much, even in combination with our layer utility analysis and multitask learning.

Muti-Task Learning also presented some challenges. The Mutitasking BERT model seemed to be overfitting on the data when training for multiple epochs. Train accuracy steadily increased throughout training, but dev accuracy tended to plateau after just one epoch. As such, we decreased training time to one epoch. We also tried varying learning rate and varying dropout probability to further correct for overfitting. Overall, we found that varying dropout probability had little effect. We did, however, see an improvement in model scores with larger learning rates. This led us to experiment with even larger increases to the learingrate. We observed the same result: that larger learning rates improved model performance.

We thought that these larger learning rates might be beneficial only because of the short training time of 1 epoch. We therefore tested whether large learning rates similarly improved model performance with a longer training time of 3 epochs, and we observed that they did. This potentially pointed to point to the model getting stuck in local minima.

However, this behavior could also have been due to conflicting gradients. One additional issue we observed was that the model would improve scores one one task but then get worse at another task. Gradient surgery is meant to address just this problem, so our next experiment was to try applying gradient surgery. To test our intuition, we first finetuned the model for 10 epochs without applying gradient surgery. Then, we performed additional finetuning while applying gradient surgery. Our goal was to determine whether applying gradient surgery would allow all scores to increase simultaneously by removing conflicting components of the gradients. After only 3 additional epochs, we saw no improvement when applying gradient surgery, and in fact the model performance worsened overall. The model saw slight improvements to SemEval scores, but this came at the cost of more severely lowered scores for the sentiment classification and paraphrase detection tasks.

The biggest factor in improving against the base BERT model seemed to be the layer utility analysis. This points to something deep about the BERT model – that different kinds of information are more readily available at different points in the model. The most effective way of surfacing this information is to tap into it directly rather than trying to massage the model to preserve all useful information throughout.

8 Conclusion

We found improving BERT to be an interesting challenge. While we expected that the embeddings produced by SimCSE would perform well, they did not improve above random chance. Future work might look into improving contrastive learning approaches for MTL. Additionally, the base BERT model exhibited some unexpected behaviors in the MTL training phase, like improving with fewer epochs and larger learning rates. As MTL adds additional complexity to training the model, it seems there is more work to be done in understanding how to account for the unique problems it presents, for example problems of conflicting gradients.

The most promising contribution is the layer utility analysis, which exhibited significant performance improvements over the base BERT model. Additionally, layer utility analysis presents some interesting opportunities for experimenting with the model architecture. For example, one might consider "ensembling" layers of a BERT model. Further work could also experiment with alternate architectures for the classification heads beyond just linear classifiers.

9 Ethics Statement

Ethical considerations that pertain to the BERT model broadly include issues of bias and fairness that might be embedded in the large internet-scraped datasets that are used to train models like BERT. Datasets may include biased and harmful content that exacerbate historical biases and prejudices. Mitigation efforts in this area include inspecting and curating training data and having an ongoing evaluation of the model's performance across different protected groups.

Another ethical concern might be issues around data privacy that emerge from the use of large internet-scraped datasets. A mitigation strategy in this domain includes implementing novel machine unlearning strategies for privacy issues at the intersection with legislative compliance with statutes like the European General Data Protection Regulation (GDPR) when needed (Bourtoule et al., 2019).

References

Martijn Bartelds and Martijn Wieling. 2022. Quantifying language variation acoustically with few resources. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3735–3741, Seattle, United States. Association for Computational Linguistics.

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

Ankita Pasad, Ju-Chieh Chou, and Karen Livescu. 2021. Layer-wise analysis of a self-supervised speech representation model. In *2021 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 914–921.

Jean Wu Jason Chuang Christopher D Manning Andrew Y Ng Richard Socher, Alex Perelygin 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.

Xingcheng Yao Tianyu Gao and Danqi Chen. 2021. Simcse: Simple contrastive learning of sentence embeddings. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Wei-Cheng Tseng. 2020. Weichengtseng/pytorch-pcgrad.

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

A Layer Utility Graphs

Numerous graphs were generated for the Layer Utility Experiment, and the results were inferred from these. Figures 2, 3, and 4 show the results for a single head type, 2-layer model-initialized, for SST, quora, and STS tasks, respectively. This head was either most optimal or sufficiently optimal for all three tasks, and the other head types had largely similar results. The full set of $13 \cdot 12 = 156$ (or $13 \cdot 4 = 52$ if restricting to the primary evaluation metric for each task) results graphs, as well as the raw values as `.csv` files, can be obtained from the `probing` branch of `https://github.com/elliejvela/cs224n-spr2024-dfp`.

Each figure consists of four subplots: dev/train accuracy (for SST and Quora) or correlation (for STS) vs. either layer (with each epoch being a different line) or vs. epoch (with each layer being a different line). The vs. epoch graphs show general training trends of the model over time, while the vs. layer graphs show the general model expressiveness at each layer.

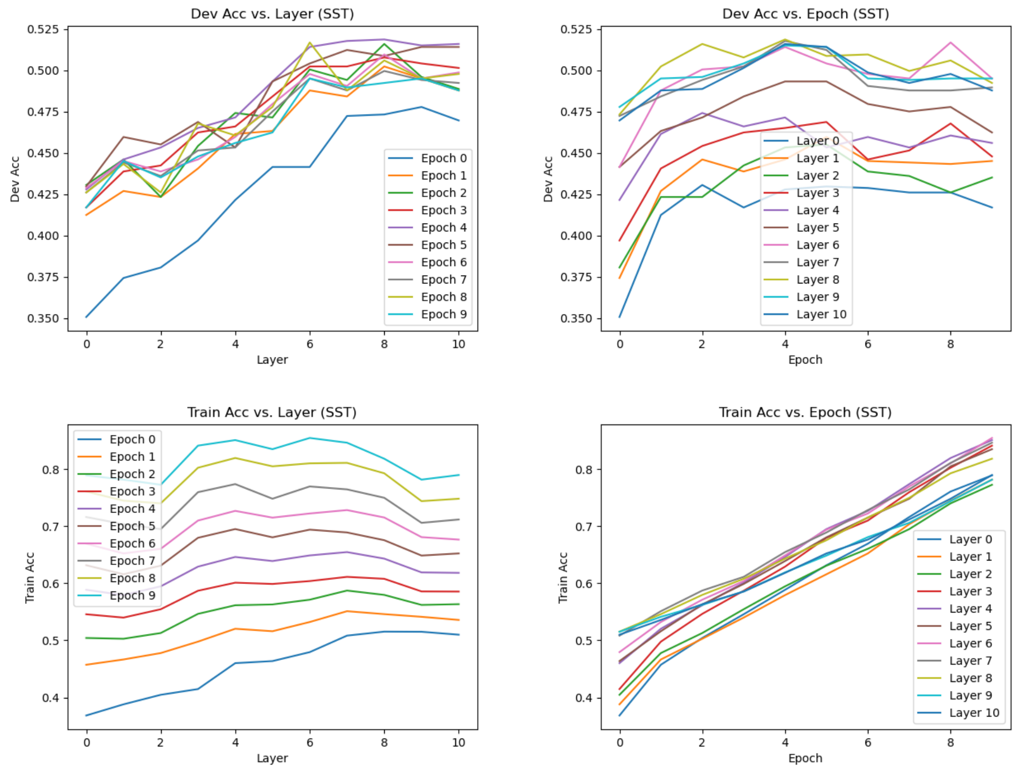

Figure 2: SST results with a 2-layer model-initialized head.

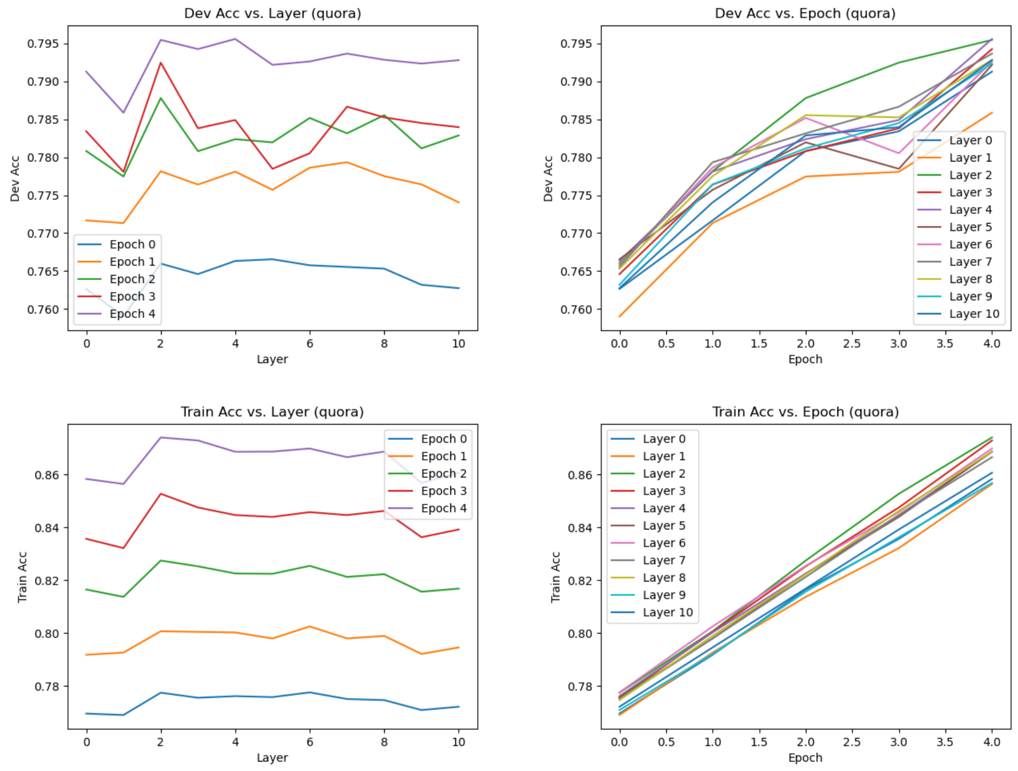

Figure 3: Quora results with a 2-layer model-initialized head.

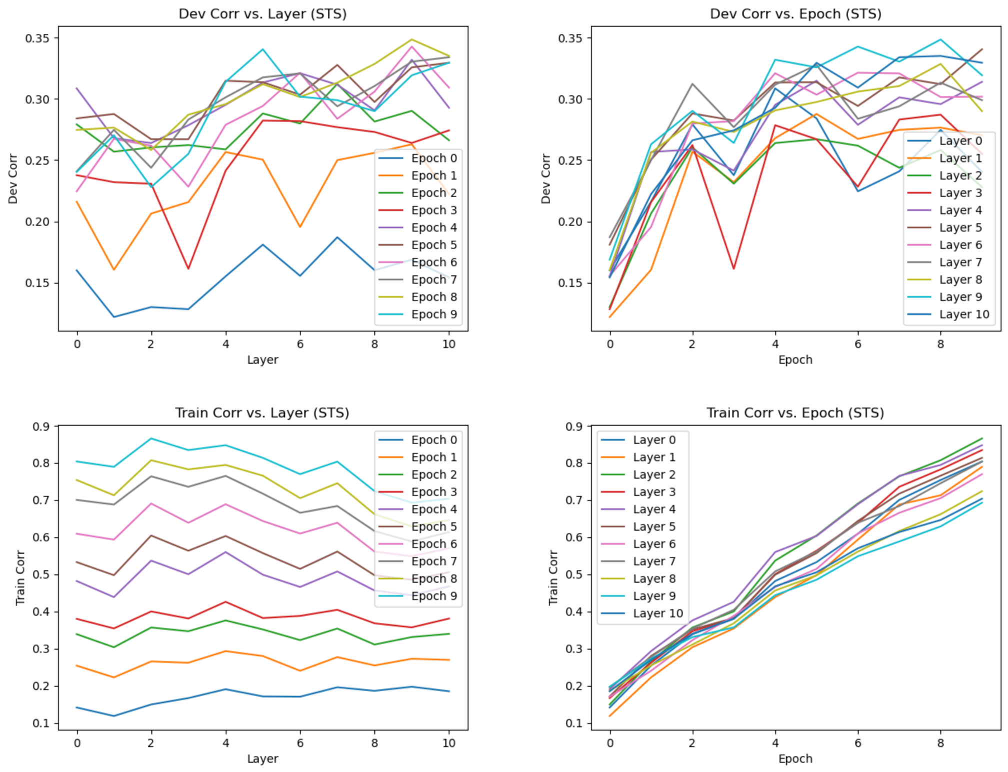

Figure 4: STS results with a 2-layer model-initialized head.