MOPS: Memory Occupancy and Performance Surveying when using Late-Stage Hard Parameter Sharing for BERT Multitask Learning

Stanford CS224N Default Project

Anthony Weng
Department of Computer Science
Stanford University
ad2weng@stanford.edu

Mark Bechthold
Department of Computer Science
Stanford University
markpb2@stanford.edu

Callum Burgess
Department of Computer Science
Stanford University
callumb@stanford.edu

Abstract

This project firstly implements the BERT (Bidirectional Encoder Representations from Transformers) architecture, which has cemented itself as a state-of-the-art model tackling a wide variety of NLP tasks since its introduction in 2018. We create hard parameter sharing regimes in a multitask BERT architecture, analyzing the efficacy of various hard parameter sharing strategies. Specifically, we look at the effectiveness of hard parameter sharing at later stages of the model to analyze whether the intuition of multitask parameter sharing providing a general understanding of language can be extended to more lower-level similarities between specific tasks. To do so, we directly compare the performance of multitask BERT architectures with varying amounts and orderings of “shared" and “unshared" layers across different subsets of tasks. We verify that hard parameter sharing at any level of a multitask model can increase accuracy. We find that there is no apparent significant increase in accuracy due to late layer parameter sharing and, at least in a simple BERT multitask finetuning architecture, it is mostly made redundant with the presence of early layer parameter sharing. We do note, however, that late stage parameter sharing can still be employed to reduce computational load and increase training efficiency.

Key information:

Mentor: Tathagat Verma — no external collaborators — no project sharing.

1 Introduction

Having a general understanding of language, sentence structure, and relationships between words and their meanings can provide an important foundation for a machine learning language model tasked with processing and answering a wide variety of natural language processing tasks. This is a very natural concept in language learning and understanding; humans don't (to great success) jump directly into a class on the deeper rhetoric and analysis of Shakespeare's work before first experiencing years of English communication and elementary reading. Similarly, it has been shown that tutoring a language model on foundational tasks, either in conjunction with or before switching to more specific language tasks, substantially increases performance (more discussion in *Related Works and Motivation*).

This machine learning concept of encouraging a higher-level, non specialized understanding of language to achieve increased accuracy in a multitask natural language processing model materializes in two central ways we will explore in this project. (1) *Language Model Pretraining*: A model is pretrained on a general task separate from the task(s) it is assigned to learn. The pretrained model is then fine-tuned on these tasks or used as a feature/input to another model. (2) *Parameter Sharing*: In a multitask model, multiple tasks have individual models which either share the same parameters between them (hard parameter sharing) or regularize their parameters amongst each other to encourage similarity (soft parameter sharing).

Along with the ability to improve accuracy, these ideas are useful for the effective allocation of limited resources and increasing training efficiency. In the case where the amount of task-specific data is limited, we can choose to pretrain our model on a general task that learns on a large and publicly available dataset, and then finally finetune our task's model on smaller datasets. The parameter weights of a pretrained model can also be made publicly available, meaning that researchers can also directly use pretrained weights to quickly see polished results. In conclusion, the process of pretraining can allow for more robust models and decreased computational need for obtaining state-of-the-art models in a wide variety of natural language processing tasks, increasing accessibility to those with limited computational resources.

In our analysis of parameter sharing multitask models within this project, we mainly focus on hard parameter sharing. Similarly to pretraining, hard parameter sharing can effectively increase the amount of training data a task model operates on, and has the potential to increase efficiency. Intuitively, by sharing weights between multiple related tasks, parameter sharing multitask models can develop a greater intuition for shared language patterns across these tasks to then achieve greater accuracy on each individual task. This is not unlike how a pretrained model can be seen as developing a higher level overall intuition for human language.

2 Related Work and Motivation

The BERT (Bidirectional Encoder Representations from Transformers) model was introduced in 2018 [1] and notably achieved high success across a wide variety of natural language processing tasks. This success is largely attributed to its pretraining across the MLM "masked language model" objective, which allowed for the model to use both left and right contexts and therefore train a deep bidirectional transformer. The paper is a successful example of pre-trained based representation models, as it achieved state of the art performance on both sentence and token level tasks; performance marks that matched or outshined specialized task-specific models. As a result, BERT serves as a great foundation for studies on the significance of certain architectural decisions in multitask models.

Experimenting with parameter sharing in a multitask BERT model has previously been done on the layers of the BERT model itself. Liu et al [2] combined pre-training and multitask learning with hard parameter sharing on a BERT model to achieve state of the art results. Their architecture shared the bottom layers of BERT amongst all tasks and then used the output of this shared BERT as a starting point for task-specific models layered on top. Similarly, in a study by Pahari and Shimida [3], the bottom layers of the BERT model were kept frozen or shared between tasks and the top layers individual. The number of shared, frozen, and individual layers was varied, and the sharing was done in a soft parameter sharing implementation. Both studies showed that parameter sharing at bottom layers of the BERT model is an effective strategy in multi-task BERT models.

While sharing layers at the bottom of a multitask model has been shown to intuitively encourage a higher level intuition of the language in general, in this project we aim to test whether placing shared layers closer to the output of the model motivates it to find more minute, lower level language similarities between tasks. In this project, we explore this conjecture by adding an identically structured 3 layer linear model for each task on top of each task's BERT output. This simplifies the structure of Liu et al's multitask BERT architecture in hopes of reducing confounding effects of the individual models of each task working respectively better than any other task. We explore the effects of shared parameters for these linear layers located relatively close to the predictions of the mode. We vary the number and location of layers with parameter sharing. One experiment keeps the BERT output constant and allows for parameter sharing on only these high level linear layers. Another adopts a fine-tuning

regiment on the shared BERT model as a whole, effectively replicating a base level of parameter sharing and its respective benefits as seen in Liu et al and Pahari et al. By also introducing lower level parameter sharing, we aim to see whether any significant amount of higher-level parameter sharing takes place, or whether higher-level parameter sharing is made redundant by lower-level parameter sharing.

We elect to adopt an established fine-tuning construction for the experiment's pretrained multitask BERT-based model (in which a smaller task specific model is added on top of the multitask BERT and fine-tuned), as opposed to one with feature extraction, due to a study by Peters et al [4] that found that finetuning in a BERT-based model generally achieved a greater accuracy than feature extraction for similar natural language tasks to our own study. More generally, this study also concluded that fine tuning is intuitively a more suited paradigm when the pretrained and fine-tuned tasks are similarly related, but generally both feature extraction and fine-tuning approaches achieve comparable accuracy.

3 Approach

Building upon the BERT architecture of the default final project with the AdamW optimizer (see appendix **A.1** for a detailed treatment), we first altered the output for the predict-similarity downstream task to compute a scaled cosine similarity metric rather than simply summing the pooled outputs ([5], [1], [6], [7]). We computed:

$$\text{output} = (S_C(\text{pooled1}, \text{pooled2}) + 1) * 2.5$$

where we add 1 and scale by 2.5 to ensure that our output is between 0 and 5 for the final computation of the correlation coefficient. Next, we designed a new loss function that is the average of the loss across all three of the downstream tasks and used this loss function for pre-training:

$$\text{LOSS} = w_{sts}\text{MSE}_{\text{sts}} + w_{sst}\text{CE}_{\text{sst}} + w_{para}\text{MSE}_{\text{para}}$$

where CE denotes the cross entropy loss function and and MSE the mean squared error loss function. Our default loss weights for each task were $w_{sts} = w_{sst} = w_{para} = \frac{1}{3}$. We did this to ensure that the learned weights in our architecture were balanced across tasks. We used this new architecture as our baseline for investigating various parameter sharing regimes. To systematically test the effects of parameter sharing we first designed a three layer network added to the output of our BERT layer. We decided upon a simple three layer linear network to standardize the finetune models of each individual task and reduce the amount of confounding variables, so that we could make a better informed decision on the effectiveness of our hard parameter sharing regimes. Each layer of this sub-network could either share parameters across all three tasks or have individual un-shared parameters. We denoted shared layers with 'S' and individual layers with 'I' in our code.

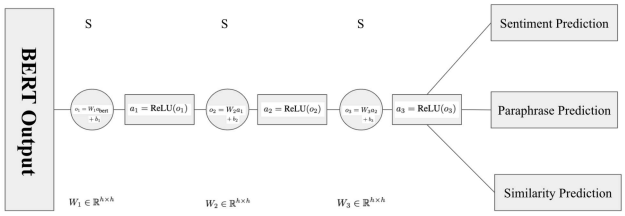

Figure 1a: Schematic of the regime where all parameters are shared across tasks: the S-S-S regime.

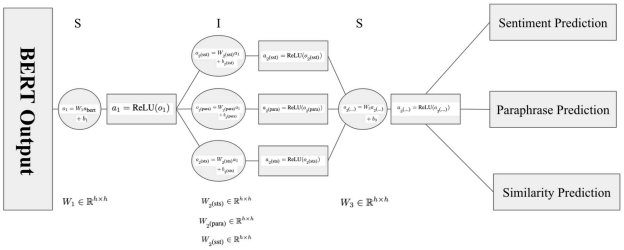

Figure 1b: Schematic of the regime where the first and last layer are shared but the intermediate layer is an individual layer: the S-I-S regime.

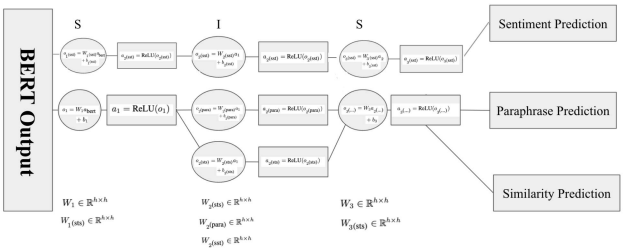

Figure 2: Schematic of the regime where the first and last layer are shared but the intermediate layer is an individual layer and the sharing is only done between the Para/STS tasks.

Figure 1a displays the basic network architecture when all three layers are shared. In this case we pass the output of the BERT through three linear layers, applying a ReLU activation to the output of each layer. Thus in the fully shared regime the output is from our shared layers is:

$$a_1 = \text{ReLU}(W_1 a_{\text{bert}} + b_1); a_2 = \text{ReLU}(W_2 a_1 + b_2); a_3 = \text{ReLU}(W_3 a_2 + b_3)$$

where $W_1, W_2, W_3 \in \mathbb{R}^{h \times h}$ and h is the number of hidden dimensions. Figure 1b. gives a representative example of a configuration of shared and individual layers. In 1b the input for a given task passes through the BERT and the first shared layer, and then depending on whether the task is sentiment, paraphrase, or similarity prediction the output from this first layer passes through one of the individual layers. Finally, this output is passed through the final shared layer before being passed to the output layers. This architecture enabled us to use the same loss function we described above to update weights since the gradient of each loss term for the various tasks will depend only on the shared layers and the individual layers for the specific task. Then, the output for each task becomes:

$$\begin{aligned}
a_{2^{(\text{sst})}} &= \text{ReLU}(W_{2^{(\text{sst})}}\text{ReLU}(W_1 a_{\text{bert}} + b_1) + b_{2^{(\text{sst})}}); a_{3^{(\text{sst})}} = \text{ReLU}(W_3 a_2 + b_3) \\
a_{2^{(\text{para})}} &= \text{ReLU}(W_{2^{(\text{para})}}\text{ReLU}(W_1 a_{\text{bert}} + b_1) + b_{2^{(\text{para})}}); a_{3^{(\text{para})}} = \text{ReLU}(W_3 a_2 + b_3) \\
a_{2^{(\text{sts})}} &= \text{ReLU}(W_{2^{(\text{sts})}}\text{ReLU}(W_1 a_{\text{bert}} + b_1) + b_{2^{(\text{sts})}}); a_{3^{(\text{sts})}} = \text{ReLU}(W_3 a_2 + b_3)
\end{aligned}$$

for sentiment prediction, paraphrase prediction, and similarity prediction, respectively, where $W_1, W_{2^{(\text{sst})}}, W_{2^{(\text{para})}}, W_{2^{(\text{sts})}}, W_3 \in \mathbb{R}^{h \times h}$.

Since each of the three layers can either be shared or individual, there were $2^3 = 8$ model configurations that were systematically tested with both finetuned and non-finetuned BERT weights.

We also decided to test the potential effectiveness of allowing only certain subsets of all tasks to share parameter weights. This would intuitively lead to higher accuracy if the subset of tasks are far more related to each other than other tasks, in which case the shared parameter scheme would hopefully encourage the model to learn lower-level intuition between the subset tasks. To test this subset architecture, we implemented an identical shared linear structure where only two of three tasks shared linear layers. As a result, we would be able to compare the results of the subset shared parameters to the generally shared parameters. The learning and output equations are symmetric to the globally shared setting for the tasks in the learned subset, and the learning and output equations for the task outside of the shared subset simply acts as an independent model. One example of such a regime, with its formalized mathematical equations, is pictured in Figure 2. The combined multitask loss function remains unchanged.

We tested our subset shared weight construction on the S-S-I, S-I-S, and I-S-S models (which exhibited some of best performances and uniformly all contained two shared layers) in all cases where possible size 2 subsets of our three total tasks shared layers. As a result, there were $3 * 3 = 9$ total model configurations trained.

4 Experiments

Five divisions of experiments are conducted:

1. Experiments 1 and 2 evaluate the absolute and time- and memory-relative performance of different hard parameter sharing regimes for multitask learning in the contexts of using pretrained and finetuned BERT weights, respectively.
2. Experiments 3 and 4 aim to further optimize the performance of a BERT model with some hard parameter sharing regime by varying the learning rate or task-specific weighting used to compute the multitask loss function (described in **Section 4**), again in the contexts of using pretrained and finetuned BERT weights, respectively.

3. Experiment 5 investigates the impact of parameter sharing across a subset of the tasks defining the multitask learning objective (described below).

For all experiments, the multitask learning objective is defined as maximizing overall performance on the following three tasks: sentiment classification, paraphrase detection, and semantic textual similarity rating. Task-specific and overall performance evaluation metrics follow in **Section 5.2**.

4.1 Data

Each sub-task defining the multitask learning objective is associated with a distinct dataset: the Stanford Sentiment Treebank (SST) dataset [8] is used for the sentiment classification task; the Quora dataset [9] is used for the paraphrase detection task; and the Semantic Textual Similarity (STS) dataset [10] is used for the semantic textual similarity rating task. Each dataset's attributes and associated task's input/output structure is described at length in the CS 224N: Default Project Handout [11].

4.2 Evaluation methods

We evaluate task-specific performance as follows: percentage accuracy of the predicted labels is used to score model performance on the sentiment classification and paraphrase detection tasks; and the Pearson correlation between true similarity values and predicted similarity values is used for the semantic textual similarity rating task. All task-specific scores are converted to a decimal basis.

Given task-specific scores, the *overall score* for a model configuration is computed as the arithmetic average of the task-specific score. Using the overall scores for model configurations employing various hard parameter sharing regimes, we can also compute the *TM-relative overall score* (i.e., time- and memory-relative overall score) of each model configuration as follows:

$$\text{TM relative score} = \text{Overall score of non-baseline model} \times \frac{\text{Training time required for baseline model}}{\text{Training time required for non-baseline model}} \times \frac{\text{Memory required for baseline model storage}}{\text{Memory required for non-baseline model storage}}$$

Figure 3: Equation for computing the TM-relative overall score

Task-specific scores, overall scores, and TM-relative overall scores are specific to a given: (1) data split (e.g., training, dev, or test data); (2) model configuration; and (3) the use of pretrained or fine-tuned BERT weights.

4.3 Experimental details

4.3.1 Experiments 1 and 2: Effect of Parameter Sharing on Multitask Performance.

Experiments 1 and 2 both extend a basic multitask learning BERT architecture (i.e., passing BERT embeddings to task-specific heads) through the use of various hard parameter sharing regimes. Parameter sharing regimes are defined by a three-layer network which transforms BERT embeddings prior to their use as inputs for the task-specific heads (see **Section 2** for details). For these experiments, we train and evaluate the performance of models corresponding to use of each of the 8 possible parameter sharing regimes, once using pretrained BERT weights (Experiment 1) and once using fine-tuned BERT weights (Experiment 2).

For both experiments, model configuration task-specific scores and overall scores on validation data are computed for each training epoch. After training finishes for a given configuration, the highest overall score obtained on the validation data during any training epoch is identified, with the associated task-specific scores and epoch of occurrence also being recorded. Model configuration size and required training time are logged and subsequently used to compute TM-relative overall scores, with the I-I-I (individual-individual-individual) parameter sharing regime being used as the baseline configuration in these computations.

As these experiments sought to isolate the effect of different parameter sharing regimes on multitask learning performance, other key factors are fixed across model configurations. All model configurations using pretrained BERT weights utilized a learning rate of $\alpha = 1\text{e}{-3}$ whereas those using fine-tuned BERT weights set $\alpha = 1\text{e}{-5}$ for the AdamW optimizer.

Across *all* experiments (including the following Experiments 3 and 4), model configurations were trained for 10 epochs, with each epoch corresponding to training on 1000 batches of data randomly sampled from the three task-specific datasets and batch size is set to 8. For every configuration, parameter values are saved for the model's state corresponding to the epoch in which the model achieves maximum overall score on the validation data. Additionally, all present dropout layers have a dropout probability of 0.3. Training a model configuration takes approximately 1 hour when using pretrained BERT weights and 1.5 hours when using fine-tuned BERT weights.

4.3.2 Experiment 3: Optimizing Learning Rates for Parameter Sharing.

Following Experiments 1 and 2, we identify the four non-baseline (i.e., non- I-I-I) parameter sharing regimes which achieve the highest overall scores (raw, not TM-relative) for each experiment. Then, for each experiment context (i.e., use of pretrained or fine-tuned BERT weights), we conduct a search over these four best regimes and variable learning rates sampled from $1e{-}1$ to $1e{-}7$ in a log 10-uniform fashion.

Conducting using the `raytune` package, the search consists of training each model configuration (associated with a candidate parameter sharing regime) with five different learning rates, totalling 20 models trained for each of the pretrained and fine-tuned BERT weights contexts. The used parameter sharing regime, any-epoch maximum overall score obtained on the validation data split, constituent task-specific scores, train time, and model size is recorded for the best-performing model (as measured by overall score) in both the pretrained and fine-tuned BERT weight contexts.

4.3.3 Experiment 4: Further Optimization through Task-Specific Loss Weighting.

From Experiment 3, we identify the single best-performing combination of parameter sharing regime and learning rate obtained for both the pretrained and fine-tuned BERT weight contexts in Experiment 3. For each of the two model configurations, we conduct a search over 6 possible task-specific loss weightings for a total of 12 model configurations trained. A task-specific loss weightings $\mathbf{w} = (w_{sts}, w_{sst}, w_{para})$ is defined as tuple of three floats, each ranging from 0 to 1 and summing to 1, where entries correspond to the weight assigned to the task-specific loss in computing the overall training loss for an epoch:

$$\text{LOSS} = w_{sts}\text{MSE}_{\text{sts}} + w_{sst}\text{CE}_{\text{sst}} + w_{para}\text{MSE}_{\text{para}}$$

Due to time constraints, we elect to manually define the search space of weightings as follows: $[0.4, 0.4, 0.2]$, $[0.4, 0.2, 0.4]$, $[0.2, 0.4, 0.4]$, $[0.5, 0.25, 0.25]$, $[0.25, 0.5, 0.25]$, $[0.25, 0.25, 0.5]$. Alongside the task-specific loss weighting, key characteristics described at the end of **Section 5.3.2** are recorded for the best-performing model for both pretrained and fine-tuned BERT weight contexts.

4.3.4 Experiment 5: Impact of Restricting Tasks Involved in Shared Layers

For Experiment 5, we select the three dual-shared layers model configurations from **only** the fine-tuned BERT weight context for experimentation. Task-specific loss weighting and learning rate is set to be uniform for all models. For each model configuration, we enforce parameter sharing across every possible subset of two tasks (with the un-shared task having an additional individual layer in place of any shared layers) and train the model. Aforementinoed key characteristics are recorded, and the best-performing model is identified.

4.4 Results

We first present a subset of the model configurations[1] which obtained the best performance (as measured by maximum any-epoch overall score achieved on the validation set) when using pretrained and finetuned BERT weights (**Table 1** and **Table 2**, respectively). Performance statistics and key model characteristics are shown. The baseline model configuration (i.e., that which uses an I-I-I parameter sharing regime - equivalently, no parameter sharing) is included and suffixed with a *. Task-specific scores are abbreviated as follows: paraphrase detection accuracy $\leftrightarrow$ PD acc.; sentiment classification accuracy $\leftrightarrow$ SC acc.; Similarity Correlation $\leftrightarrow$ STS corr.; Task-specific loss weightings $\mathbf{w}$ are presented in the order $[w_{para}, w_{sst}, w_{sts}]$. $X \notin$ S denotes that task X was not included in the subset of tasks selected for parameter sharing.

In **Table 3**, we also present the score characteristics of the model configuration used to generate predictions for and evaluate against the test data (hereafter referred to as the *tested model*[2]); i.e., the model configuration using fine-tuned BERT weights with an **I-S-S** parameter sharing regime where learning rate $\alpha = 1e{-}5$, $\mathbf{w} = [1/3, 1/3, 1/3]$, and no parameter sharing task subset.

[1] Documentation of all results can be obtained by contacting the authors.

[2] For our latest CS 224N Test Set Leaderboard submission, we use predictions generated from the same model configuration described here but with variations on the operations performed in the STS task-specific head (specifically, use of summed token-wise BERT hidden state embeddings instead of sentence-wise pooler output and alternate scaling of cosine similarity) after observing improved validation set performance with the changes. Due to time constraints, we were unable to re-run all models using these alternate computations, leading to the difference between test set leaderboard figures and the ones reported in **Table 3**.

Table 1: Summary of selected results for model configurations using *pretrained* BERT weights.

Model Configuration			Best Overall Score - Validation Set					Time and memory characteristics		
Param. Sharing Regime	Learning rate	$\mathbf{w}$	Score	Epoch Achieved (0-indexed)	PD Acc.	SC Acc.	STS Corr.	Train time (s)	Model size (kb)	TM-relative overall score
I-I-I*			0.328	5	0.609	0.397	-0.020	3188.01	497K	0.328
I-I-S			0.354	7	0.625	0.410	0.027	3251.50	483K	0.357
S-I-I	1e−3	[1/3, 1/3, 1/3]	0.342	6	0.625	0.408	-0.008	3418.73	483K	0.328
I-S-S			0.339	8	0.625	0.404	-0.013	3198.43	469K	0.358
S-S-I			0.377	9	0.587	0.397	0.147	3389.97	469K	0.375
I-S-S	8e−6	[1/3, 1/3, 1/3]	0.392	9	0.618	0.386	0.173	3210.24	469K	0.412
I-S-S	8e−6	[2/5, 1/5, 2/5]	0.313	9	0.375	0.364	0.199	3469.92	469K	0.305

Table 2: Summary of selected results for model configurations using *finetuned* BERT weights.

Model Configuration			Best Overall Score - Validation Set					Time and memory characteristics		
Param. Sharing Regime	Learning rate	$\mathbf{w}$	Score	Epoch Achieved (0-indexed)	PD Acc.	SC Acc.	STS Corr.	Train time (s)	Model size (kb)	TM-relative overall score
I-I-I*			0.492	4	0.625	0.491	0.359	6382.58	1352K	0.492
S-I-I			0.494	5	0.627	0.503	0.353	5725.13	1338K	0.556
I-S-S	1e−5	[1/3, 1/3, 1/3]	0.500	5	0.625	0.524	0.351	5656.93	1324K	0.576
S-I-S			0.469	3	0.593	0.495	0.320	5268.08	1324K	0.580
S-S-I			0.490	8	0.625	0.490	0.355	5443.27	1324K	0.587
S-I-I	8e-06	[1/3, 1/3, 1/3]	0.397	6	0.375	0.491	0.325	5434.68	1338K	0.471
I-S-S	1e-05	[2/5, 1/5, 2/5]	0.493	5	0.625	0.491	0.363	5340.03	1324K	0.602
I-S-S (SC $\notin$ S)	1e−5	[1/3, 1/3, 1/3]	0.486	7	0.625	0.470	0.365	5612.81	1338K	0.558

Table 3: Score characteristics of the model configuration used for test set evaluation.

Model Configuration			Score Characteristics					
Param. Sharing Regime	Learning rate	$\mathbf{w}$	Data split	Score	Epoch Achieved (0-indexed)	PD Acc.	SC Acc.	STS Corr.
I-S-S	1e−5	[1/3, 1/3, 1/3]	Training	0.750	5	0.639	0.823	0.789
			Validation	0.500	5	0.625	0.524	0.352
			Test	0.481	NA	0.631	0.523	0.288

4.4.1 Performance Characteristics of Tested Model

At large, the tested model performed at expectation, with test set score metrics being similar to those achieved on the validation set (see **Table 3** for exact figures). These absolute score figures represent a significant improvement from our prior implementation of a multitask learning BERT architecture (which consisted of fine-tuned BERT weights, independent single-linear layer task-specific heads, and training only on the sentiment classification task); however, they are not significantly different from the validation set score metrics achieved by baseline model (i.e., I-I-I) for the fine-tuned BERT weights context. Reasons for this seeming lack of absolute improvement are discussed in **Section 5.1.1**, but we note that the tested model presents a considerably higher TM-relative overall score as compared to the baseline model–demonstrating that use of hard parameter sharing regimes can yield comparable absolute performance and improved compute-relative performance as expected.

4.4.2 General TM Characteristics when Hard Parameter Sharing

An advantage to models with some degree of parameter sharing is the reduced training time and memory footprint demands of the trained model. Comparing the time and memory characteristics of the four model configurations which share a learning rate and $\mathbf{w}$ with the baseline models (in both the pretrained and fine-tuned BERT weights context), we find that: in the context of pretrained BERT weights, the no parameter sharing regime (e.g., I-I-I) requires the lowest training time and the highest memory footprint; and in the context of fine-tuned BERT weights, the no parameter sharing regime requires both the highest training time and highest memory footprint.

Expectations regarding model training time and memory footprint for configurations using no versus some parameter sharing are near-completely met, save the observation that the I-I-I regime required the lowest training time when using pretrained BERT weights. A likely explanation for this phenomenon is that, when using pretrained BERT weights and the I-I-I regime, propagation of each task-specific loss is conducted over independent sets of parameters allowing for under-the-hood parallelization. However, when using fine-tuned BERT weights and the I-I-I regime, this concurrency cannot be achieved since propagating each task-specific loss affects the parameters of the base BERT model. Still, the otherwise across-the-board memory footprint and training time decreases achieved by parameter sharing regimes in both the context of using pretrained or fine-tuned BERT weights suggests that

hard parameter sharing is a valid strategy to adopt when seeking time and memory gains, with the specific case of training times when using parameter sharing and pretrained BERT weights being a notable exception.

Comparing absolute and TM-relative scores on the validation data split for the same model configurations as before, we observe that: in the context of pretrained BERT weights, 4 of 7 parameter sharing regimes achieve both a higher absolute overall score and higher TM-relative overall score than the no parameter sharing regime; and in the context of fine-tuned BERT weights, 2 of 7 parameter sharing regimes achieve both a higher absolute overall score and higher TM-relative overall score and 3 of 7 parameter sharing regimes achieve only a higher TM-relative overall score compared to the no parameter sharing regime[3].

Collectively, these results suggest that parameter sharing can induce relative performance gains which can be valuable in compute-limited environments. However, if absolute performance is a priority, selection of the specific parameter sharing regime to be employed is of chief importance. Decisions regarding how to select a regime must consider the downstream learning tasks and the use pretrained or fine-tuned BERT weights. Commentary on these considerations and the gap between the number of parameter sharing regimes which induce both absolute and TM-relative overall score improvements when using pretrained versus fine-tuned BERT weights are discussed more in **Section 5.1**.

4.4.3 Value of Non-Model Optimizations for Parameter Sharing

Experiments 3 and 4 yielded inconsistent performance gains for non-baseline parameter sharing regimes across the pretrained and fine-tuned BERT weight contexts. Optimizing α for the best-performing non-baseline model in the pretrained BERT weights context lead to a significant improvement over baseline in both absolute and TM-relative scores, and optimizing $\mathbf{w}$ for the analogous model in the fine-tuned BERT weights context lead to a negligible increase in absolute score, a notable increase in TM-relative score, and a notable *decrease* in absolute score compared to the same non-baseline parameter sharing regime with a uniform task-specific loss weighting. Experiment 5 lead to a *decrease* in absolute score relative to the baseline and associated non-baseline parameter sharing regime while yielding an increase in TM-relative score relative to baseline.

For Experiments 3 and 4, lackluster results likely arise from either: (1) the limited search trials conducted over the hyperparameter space owing to time restrictions; or (2) non-separability in hyperparameter changes which lead to global (i.e., with respect to all hyperparameters) performance improvements (a hypothesis arising from the fact that our hyperparameter optimizations were conducted in a step-wise rather than concurrent fashion, again owing to time restrictions). Collectively, they suggest a more expansive and concurrent hyperparamter search may be able to improve parameter sharing regime performance, which is suggested for future works. For Experiment 5, decreases in absolute performance suggest that the task-agnostic knowledge that can be encoded in shared layers (described more in **Section 5.1.1**) may lead to performance gains which outweigh any confounding effects which occur by sharing layers with different output formats. In a multitask learning environment with more tasks than the three here, parameter sharing with subsets may be more beneficial as no/fewer singleton sharing subsets will be created.

5 Analysis

5.1 Best-Performing Parameter Sharing Regimes when using Pretrained versus Fine-tuned BERT Weights

Across both pretrained and fine-tuned weights contexts, we find the best-performing parameter sharing regimes to share identity or structural characteristics. We provide insights into this observation and for designing an optimal parameter sharing regime for arbitrary multitask learning environments.

5.1.1 Common Regimes and Knowledge Captured by Shared versus Individual Layers

The S-I-I and S-S-I regime both appear in the top four best-performing parameter sharing regimes for the pretrained and fine-tuned BERT weight contexts. These results accord with those found by Pahari et al. in their study of soft parameter sharing regimes [3], verifying their interpretation of the knowledge captured by shared versus individual layers and suggesting its generalizability from soft parameter sharing to any (i.e., soft, hard, or mixed) parameter sharing . Namely, shared layers likely capture general, domain-agnostic knowledge whereas individual layers capture domain-specific patterns in the downstream tasks. Structuring a parameter sharing regime as S*$\cdots$I* (i.e., shared layer(s) followed by individual layer(s)) thereby results in a network architecture which can first encode/extract information useful to all downstream tasks while allowing task-specific information to remain separate–limiting performance trade-offs across tasks.

This conceptual framework of shared layers capturing domain-agnostic knowledge and individual layers capturing domain-specific knowledge may also explain why less parameter sharing regimes achieve both a absolute overall score and higher TM-relative overall score in the fine-BERT weights context (4) as compared to the pretrained

[3]One such regime, I-S-I, is not included in **Table 2**.

context (2). By fine-tuning weights, the BERT architecture inputs pass through prior to the parameter sharing linear layers in effect becomes an additional shared layer. Because the BERT architecture dwarfs the parameter sharing linear layers in terms of both parameter count and sophistication, most of the domain-agnostic general knowledge is likely already encoded in the fine-tuned BERT weights–thereby reducing the potential for performance gains by using a S*$\cdots$I* parameter sharing regime.

Collectively, this analysis suggests that use of parameter sharing regimes will induce greater absolute and TM-relative performance gains when the underlying model (here, BERT) uses pretrained weight; and that, writ large, shared layers which help capture domain-agnostic knowledge should occur earlier in a given parameter sharing regime than any individual layers, which help capture domain/task-specific knowledge.

5.1.2 Common Regimes and Task-Dependency of Optimal Parameter Sharing Structures

Aside from the S-I-I and S-S-I regimes previously discussed, the best-performing parameter sharing regimes for both the pretrained and fine-tuned BERT weight context (e.g., I-I-S, I-S-S, S-I-S) all have one structural commonality: the presence of a shared layer as the last layer of the parameter sharing regime. Ostensibly, this observation conflicts with the recommendations presented at the end of **Section 5.1.1** (i.e., that shared layers should occur earlier than individual layers in a parameter sharing regime due to the type of knowledge captured by each type of layer). However, the nature of the tasks defining our multitask learning objective may explain these seemingly contradictory results.

Two of the evaluated tasks, paraphrase detection and semantic textual similarity, are closely related. Consider: two sentences which are perfect paraphrases (i.e., identical) would achieve the highest possible semantic textual similarity whereas two sentences which have very little semantic textual similarity are unlikely to be paraphrases of each other. This relationship is not a perfect bijection (one could propose a pair of sentences which are semantically similar but are not paraphrases, or a pair of syntactic paraphrases with key semantic elements changed) but still likely is governed by a positive correlation. Given this, the presence of shared layers at later stages of a parameter sharing regime may capture this inter-task output correlation in a way that early-regime shared layers cannot. Reason being, if early-stage shared layers encode domain agnostic knowledge, outputs from these layers may be too generalized as compared to the input representations provided to task-specific heads, meaning any correlation between task inputs and outputs cannot be detected.

For this reason, designing an optimal parameter sharing structure must take into consideration how correlated the outputs of the downstream tasks likely are. If outputs are highly correlated, placement of shared layers toward the end of a parameter sharing regime will likely benefit performance. Moreover, this conclusion is not mutually exclusive to that of **Section 5.1.1**–use of shared layers at the early and later stages of a parameter sharing regime may induce both types of associated benefits (as seen with S-I-S being one of the top four best-performing parameter sharing regimes in the fine-tuned BERT weights context). Extended parameter sharing regimes (i.e., those consisting of many more layers than the three layers used here) may find the most success with this early-and-late-stage shared layers architecture given the ability to interleave more individual layers between shared layers to further differentiate input representations for non-correlated tasks.

6 Conclusion

In this project we have implemented functionality for a multitask BERT model that utilizes a masked language model objective to pretrain its weights. We then designed a framework for testing hard parameter sharing regimes on the later 'finetuning' layers of the multitask model and verify that hard parameter sharing at any point in a multitask model can significantly increase performance across all tasks. We show that hard parameter sharing at later layers in our multitask model does not seem to significantly impact performance when there is hard parameter sharing at lower layers of the model, and is therefore perhaps made redundant by early sharing. Similarly, our subset-specific parameter sharing regimes also don't exhibit that much performance improvement. Our experimental results also portray the importance of having individual layers separating shared layers in a multitask model, in order to intuitively 'convert' general language knowledge into the right format of specific problems. Surprisingly, we also note that late stage hard parameter sharing regimes sometimes seem to benefit from having a shared output layer, perhaps allowing for further learning and greater task intuition between tasks with similar objectives. Overall, later stage parameter sharing regimes are more efficient due to increased effective training datasets and less overall model parameters, but do not seem to result in significantly increased performance.

Limitations and Future Work: Due to limited computation resources and in an effort to reduce confounding variables, we designed a relatively simple late stage parameter sharing regime with a smaller amount of late stage non-BERT layers. However, it is possible that the emergent benefits of late stage parameter sharing only occur in more complex and computationally demanding structures. We note that with a larger variety of tasks and a more recursively structured multitask model in which similar tasks are grouped together in specialized hard parameter sharing models at later layers, the impact of later hard parameter sharing regimes might be made more conclusive.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[2] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Multi-task deep neural networks for natural language understanding. 2019.

[3] Niraj Pahari and Kazutaka Shimada. Multi-task learning using bert with soft parameter sharing between layers. pages 1–6, 2022.

[4] Matthew E. Peters, Sebastian Ruder, and Noah A. Smith. To tune or not to tune? adapting pretrained representations to diverse tasks. *CoRR*, abs/1903.05987, 2019.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2017.

[7] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017.

[8] https://nlp.stanford.edu/sentiment/treebank.html.

[9] https://www.quora.com/q/quoradata/first-quora-dataset-release-question-pairs.

[10] https://aclanthology.org/s131004.pdf.

[11] Cs 224n: Default final project: minbert and downstream tasks.

[12] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017.

A Appendix

A.1 BERT Architecture and AdamW Optimizer

We complete the initial baseline of the default final project by implementing key parts of the minBERT architecture [1]. To implement minBERT, we first extract word, positional, and task embeddings and feed these into a multi-headed attention model. Keyly, the attention of each head is given by the following formula, where Q represents tokens in a query sequence of a given length, K the key vectors, d_k the dimension of the key vectors, and V the value vectors for the sequences [5]:

$$Attention(Q, K, V) = Softmax(\frac{QK^T}{\sqrt{d_k}})V.$$

After computing $S = \frac{QK^T}{\sqrt{d_k}}$, we apply an attention mask setting all values corresponding to a padding token in S to a large negative value. This means that when computing the softmax these values, we contribute minimally to the attention. The final multi-headed attention result is then:

$$Multihead(Q, K, V) = Concat(head_1, ..., head_h)W^O,$$

where $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$ and W_i^Q, W_i^K, W_i^V, W^O are all learned parameter projection matrices.

After passing our embeddings through the multi-headed attention, we project the output using a dense linear layer. We then add the input embeddings to the output of the dense layer with dropout and pass the sum through a layer-norm layer, computing: $y = \frac{x - \mathbb{E}[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$, where γ and β are learned parameters. Next, we pass the outputs of the layer-norm layer through a feed forward network, applying a GeLU activation function: $\text{GELU}(x) = x * \Phi(x)$ where $\Phi(x)$ is the CDF of the Gaussian distribution.

Finally, we pass the outputs through another additive and normalization layer with a residual connection with dropout to get the outputs of our minBERT. The complete architecture of our implementation includes passing the inputs through the minBERT, and passing its outputs through a linear layer with dropout to output the correct number of logits depending on the prediction task. We minimize the cross entropy loss between the predicted logits and the ground truth values:

$$J(y, \hat{y}) = -\frac{1}{m}\sum_{i=1}^{n} y * \log(softmax(\hat{y})),$$

where $\hat{y}$ are the outputted logits from our model. To perform this minimization we use our own implementation of the AdamW gradient descent algorithm [12] where we use the first and second moment and weight decay to improve convergence time. We used the computationally more efficient method [6] given by the following update rule:

$$\theta_t' = \theta_{t-1} - \alpha_t \frac{m_t}{\sqrt{v_t + \epsilon}}; \theta_t = \theta_t' - \alpha w \theta_t',$$

where $\alpha_t = \alpha\sqrt{1 - \beta_2^2}/(1 - \beta_1^2)$, $m_t = \beta_1 m_{t-1} + (1 - \beta_1)g_t$, $g_t = \nabla f_t(\theta_{t-1})$, $v_t = \beta_2 v_{t-1} + (1 - \beta_2)g_t^2$ and α, β_1, β_2 are model hyper-parameters.