Exploring Multi-Task Learning with Unbalanced Datasets and Gradient Surgery

Stanford CS224N Default Project

Julien Darve
Department of Computer Science
Stanford University
jdarve@stanford.edu
TA Mentor: Arvind Venkat Mahankali

Abstract

Multi-task learning allows a single model with a shared set of parameters to compute several different tasks. A model trained on multiple tasks is inherently more efficient than training several different models on each task, as we only need to run the model once and store only one set of parameters. This paper explores multi-class learning in the context of fine-tuning a BERT model on the sentiment classification, paraphrase detection, and semantic similarity tasks. However, there are several added difficulties that come with multi-task learning, especially if training on datasets of vastly unequal size per task. The central goal of this paper is to find mitigation strategies against unweighted datasets to improve multi-task learning performance. One issue with multi-task learning is that gradients from different tasks can destructively interfere. The major contribution of this paper is a custom implementation of Gradient Surgery as described in Yu et al. (2020) which addresses the problem of interfering gradients. We also experiment with dataset reweighting and loss function modifications to address dataset imbalance. Our findings are threefold. We find that our base model overfits badly on the sentiment classification task as it is our smallest dataset. We can mitigate this overfit by adding a coefficient of 0.5 on its loss function, leading to our best results over all three tasks. Finally, we found that gradient surgery leads to the most balanced results between all three classes and effectively mitigated overfit without the need for hyper-parameter tuning.

1 Introduction

The BERT architecture, based on the self-attention mechanism, has become the core of modern NLP foundational models, as originally proposed in Devlin et al. (2019). Modern BERT models are be trained on massive datasets including but not limited to most of the World Wide Web, social media posts, scientific papers, code on github, and movie subtitles, and can have up to hundreds of billions of parameters. A common practice in NLP is to take a pre-trained BERT model trained on these massive datasets, add a single linear layer on top, and finetune the model for a specific sub-task involving word processing, making use of its rich pre-trained embeddings.

In multi-task learning, several different linear layers are added on top of BERT leading to several different outputs, and the model is trained on all tasks at once. In practice, this allows us to re-use the entire BERT model and its output embeddings in order to compute all of our tasks, increasing space and time complexity. However, by the nature of making one model for multiple tasks, the performance on each task individually might be worse than for a model trained solely and fully specialized for each task. Thus, our task is to create a multi-task fine-tuning pipeline that makes as little tradeoffs on performance for multi-task learning as possible, to enable both high accuracy and high computational efficiency.

We explore this topic through the specific multi-task learning problem of fine-tuning BERT on the sentiment classification, paraphrase detection, and semantic similarity tasks. We will investigate two specific problems with multi-task learning. First, our datasets for these three tasks are massively unbalanced, with our dataset for paraphrase detection being more than 40x the size of the semantic similarity dataset. Second, it is theorized that when a model is trained according to multiple gradients, if these gradients point in different directions, they can destructively interference, preventing optimal convergence in the gradient descent landscape.

In order to address these issues, we explore three solutions. First, we try re-weighting the datasets by duplicating the smaller datasets to match the size of the larger dataset. Secondly, we try adding a coefficient to the loss function of one of the classes with a smaller dataset to reduce overfit and allow training on an increased number of duplicates. Thirdly, the major contribution of this paper, is a custom implemented of PCGrad, i.e. Gradient Surgery, which projects the gradients for each loss function against each other to reduce destructive interference and provide more optimal descent through the gradient landscape. We find that reweighting the datasets causes greater overfitting on the tasks with smaller datasets, which we can mitigate using smaller coefficients on these tasks. Furthermore, we find that Gradient Surgery balances the results between each task the best. We conclude that loss function re-weighting and Gradient Surgery are effective measures to optimize the training process for Multi-task learning models. These results, while specific to our tasks and datasets, can be re-purposed and applied to any other model and training task that involves multiple datasets or loss functions.

2 Related Work

There are many different approaches that have been taken to improving the use of datasets and gradients for multi-task machine learning problems. A common method is to increase the number of task-specific parameters in a model, such as by adding a larger number of linear layers per task. Another is to vary the learning rate per task.

A more architecture-focused way of employing more parameters per task is explained in Stickland and Murray (2019), a paper which proposes Projected Attention Layers (PALs) for BERT. These are modules added to the attention parts of the model that improve the way BERT encodes information from input sentences to improve performance, at the cost of added parameters. However, we are more interested in sharing as many parameters as possible between models, only keeping the basic linear projection layer at the end for classification.

A popular method for multi-task learning for unbalanced datasets is annealed sampling, employed by the same paper. Instead of taking a batch from each dataset and training all at once, this method will randomly sample batchs one at a time from each single dataset and train on that batch. The sampling is done with a probability proportional to the square root of the epoch number, and size of the dataset relative to the others, so that larger datasets are more likely to be sampled at the start and training equalizes near the end.

The Gradient Surgery approach we take was proposed by Yu et al. (2020). The advantages of PCGrad is that it direct targets gradient optimization in the model, rather than changing the model architecture. In addition, it maintains the aspect of multi-task learning of updating parameters for every task all at once, rather than doing them one at a time as in annealed sampling. We believe that this will help the model learn the tasks equally, rather than constantly switching back and forth between tasks. In addition, it is a universal technique that can be applied anywhere there is gradient descent for several loss functions, a significant advantage.

3 Approach

3.1 The BERT architecture

The architecture used in this paper is an extension on top of the classic BERT model, as proposed in Devlin et al. (2019). It employs the self-attention mechanism, invented by Vaswani et al. (2017). Given input token embeddings, the model linear transforms them into queries Q, keys K, and values

V. They are combined by the following equation:

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

with multi-headed self-attention being a concatenation of several such attention calculations. The encoder part of this architecture starts with a multi-headed self-attention layer, which is fed into a two layer feed-forward neural network, then layer norm is applied. The model includes residual layers between the different layers, and positional embeddings for spatial awareness. The BERT model used in this paper uses 6 such encoder layers, with 8 self-attention heads, with a hidden size of 768.

The BERT model learns by predicting words masked out of input sentences in regular text. It includes a CLS token at the top of each sentence, the output of which represents a classification of the overall sentence. Our model takes only this CLS token, and attaches a single linear layer on top of it, with dropout and a relu non-linearity.

3.2 Task baselines and datasets

We have three different linear layers, one for each task. The baselines for these tasks are provided in the default final project handout, referenced in the following references: Stanford Sentiment Treebank (SST) Socher et al. (2013), Quora Paraphrase Dataset (Para) Fernando and Stevenson (2008), SemEval (STS) Agirre et al. (2013). We use cross entropy loss for the SST task and mean squared error loss for the STS task. I experiment with two different loss functions for the paraphrase task, mean squared error and binary cross entropy loss, as explained in the experiments section.

3.3 Gradient update methods

As will be explained in the experiments section, we evaluate each task with its own loss function, and we have different ways of combining these loss functions. Our base model naively computes

$$l_{\text{total}} = l_{\text{sst}} + l_{\text{para}} + l_{\text{sts}}.$$

Then, we have a second model that weights the base loss function, introducing coefficients λ_{sst}, λ_{para}, λ_{sts}:

$$l_{\text{total}} = \lambda_{\text{sst}} l_{\text{sst}} + \lambda_{\text{para}} l_{\text{para}} + \lambda_{\text{sts}} l_{\text{sts}}.$$

Finally, I tried calculating the gradients not through linear combination, but using Gradient Surgery. Gradient Surgery is a method proposed in Yu et al. (2020). A sample implementation in PyTorch is provided in 1. I implemented PCGrad myself for this code (I did not use any outside code other than that provided in class; the paper does not provide a pytorch implementation). Essentially, for each task separately, I perform a loss.backward(), record the gradients, then zero the gradients and loop on the next task. This gives me three separate gradients for each task alone. If the cosine similarity of the gradients is negative, I subtract the projection of one from the other, to reduce destructive interference. As shown in 1, I use this to get a single gradient from the three, then update the model's parameters manually using the newly calculated gradients.

Gradient Surgery is meant to address convergence when the gradients from different tasks 1) are pointing in opposite directions 2) are vastly different in magnitude and 3) exhibit high curvature. It is theoretical shown in Yu et al. (2020) that Gradient Surgery more effectively descents the gradient landscape in these conditions.

4 Experiments

4.1 Data

For first task, sentiment classification, I use the Stanford Sentiment Treebank (SST) from Socher et al. (2013) for the sentiment dataset. For SST, the sentences are input to the BERT model by itself. The output will be one of five classes numbering 0 through 4, where 0 is negative, 2 is neutral, and 4 is positive. These are represented in the model as 5 class probabilities, normalized through cross entropy loss.

```
def PCGrad_Update_Rule(tasks, grads):
    for task,grad in zip(tasks, grads):
        task.grad = grad
        task.newgrads = grad
    for task_i in tasks:
        for task_j in tasks.shuffle():
            dot = torch.linalg.vecdot(task_i.newgrad, task_j.grad)
            if dot < 0:
                frac = (dot / task_j.norm)
                newgrad -= torch.mul(frac, task_j.grad)
    return [t.newgrad for t in tasks]
```

Figure 1: A custom implementation of PCGrad, for reference. While not my real implementation as that would not fit in a single figure, it exemplifies the pseudocode behind the algorithm. The tasks variable stores the gradients and projected gradients per task, the grads variable represents the gradients for each parameter in the model. Code is based off pseudocode provided in Yu et al. (2020)
.

For the second and third tasks, paraphrase detection and semantic textual similarity (STS), they involve two sentences. Paraphrase detection is trained on the Quora dataset Fernando and Stevenson (2008), and semantic textual similarity on the SemEval dataset Agirre et al. (2013). I determined that the best method for inputting the data from these tasks into the BERT model was to separate the two sentences by a sep-token, which the BERT model is trained to differentiate sentences with, as explained in Devlin et al. (2019). Paraphrase detection is measured from 1 (is a paraphrase) to 0 (not a paraphrase). Similarity is a float measured from 0 to 5, with 0 representing no similarity and 5 representing the same meaning. The output of the model for either task is a single float representing the model's estimation of the correct answer.

The SST dataset uses 11,855 examples, the Quota dataset 404,298, and the SemEval dataset 8,628. As we can see, the paraphrase task is greatly overrepresented in our training data.

4.2 Evaluation method

The evaluation metric for the sentiment classification and paraphrase detection tasks will be a simple accuracy of correct to total predictions. Paraphrase detection logits will first be put under a sigmoid function and rounded before calculation of accuracy. Semantic textual similarity is evaluated by pearson correlation between the model output and the correct labels, as proposed in its original paper.

4.3 Experimental details

While I ran and trained almost 20 models, the highlights of my results lie in 7 models. First, I tried to use mean squared error loss on the paraphrase task, which resulted in low accuracy. The remaining 6 models are divided in groups of three based on the weighting placed on their datasets, which I will call the small and large split, respectively. In the first group I use the weighting "3-1-4", which corresponds to 3 times the SST dataset, 1 times the Quora dataset, and 4 times the SemEval dataset. The model will move on to the next epoch once the first dataset is emptied, essentially taking the minimum across each of the task's data points, which for these numbers evaluates to 24,160 sentences per task, limited by 4*SemEval. The second group is the split "8-1-10", representing 60,400 sentences per task, limited by 10*SemEval.

For each of these groups, I have 3 models. One is a base model (named base model), which naively adds all three losses and back-propagates on the sum. The second model uses PCGrad to update the gradients (named PcGrad). The last one does the same as the first but with a linear combination of the losses, with a coefficient of 0.5 on the SST task (named 0.5 on SST), hypothesized to reduce overfitting. This is meant to be an alternative to the PCGrad as a different way of augmenting the base model.

4.4 Accuracy results

Our top three models have their results on the test split displayed in fig 2. 2. Overall, we got that the model with the modified loss function on the larger split performed the best overall, and the best on the STS task. The PCGrad model was about tied overall, and it was the best on the paraphrase task.

Model	Overall test score	SST test acc	Paraphrase test acc	STS test corr
3-1-4 Base Model	0.784	**0.541**	0.875	0.871
8-1-10 PcGrad	0.785	0.528	**0.890**	0.874
8-1-10 0.5 on SST	**0.785**	0.531	0.875	**0.880**

Figure 2: Test set results on the top three models, with the best result per category bolded. Results largely follow dev results, although SST scores were higher. The 0.5 on SST model on the larger split remains the best model.

Model	Overall dev score	SST dev acc	Paraphrase dev acc	STS dev corr
3-1-4 MSE Loss	0.695	0.521	0.631	0.866
3-1-4 Base Model	0.780	**0.529**	0.874	0.876
3-1-4 PcGrad	0.776	0.513	0.876	0.876
3-1-4 0.5 on SST	0.776	0.508	**0.881**	**0.880**
8-1-10 Base Model	0.780	0.509	**0.891**	0.878
8-1-10 PcGrad	0.780	0.513	0.888	0.877
8-1-10 0.5 on SST	**0.782**	**0.518**	0.887	**0.880**

Figure 3: Dev set results per model on each task. The top 4 rows of the table represents the smaller split (3-1-4), while the bottom 3 represent the larger split (8-1-10). Between these categories, the best result per column is bolded

The base model on the smaller split was chosen due to its dev performance on the SST task, and it remains the best at this task.

The more interesting results lie in the dev split, as we have access to all model experiments. The results obtained for each model on each of the tasks and a measure of their overall success on the dev set is shown in fig. 3 3.

What we see is that the base model has the best accuracy on the SST task but the worst on the paraphrase task in the smaller split. But, in the larger split, it instead scores the worst on SST and the best on the paraphrase task. The effect was reversed for the 0.5 on SST model; its SST accuracy was the worst on the smaller split but the best on the larger split, and it is overtaken on the paraphrase task by the base model for the larger split. While I expected the 0.5 on SST model to perform the worst on the SST task for both splits, I was surprised when it performed the best on the larger split. We will further discuss these results in the analysis section.

Another interesting result we notice is that the PCGrad model scored at the median of the other models for both split sizes. In addition, it tied with the best on the test set. These results are in line with the theoretical results of Gradient Surgery, and stands as empirical evidence for its effectiveness on unbalanced tasks.

Using mean squared error on the paraphrase task made the model perform the worst on that task, and seemed to drag down the results from the other task as well.

4.5 Loss curve results

Furthermore, I also recorded loss curves for the training and dev splits per epoch, as I was interesting in seeing how the different models converge, shown for the base model in fig. 4 4. What the plots show is extreme overfit on the SST dataset, where the train loss goes down the farthest, but the dev loss never converges and instead steadily increases. These results are suprising, especially because the dev accuracy on the SST task was increasing even as its loss was getting worse.

Fig. 5 5 shows the loss curves from the 0.5 on SST model on the two different dataset splits, and the base model on the smaller size. What these curves show is that, for the larger split, the base model learns the SST task earlier than the 0.5 on SST model. Moreover, for the smallest dataset, we see that for the 0.5 on SST model the train loss of SST never overtakes STS. Both of these results are expected from the fact that the SST task has a coefficient slowing down its training. Finally, for the base model on the smallest data, we can see that the SST loss overtakes the STS loss at the final epoch, which surprisingly caused the best results on this task.

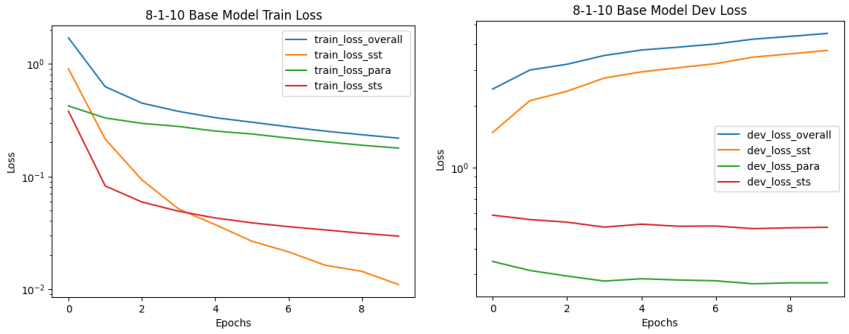

Figure 4: Loss curves on the train and dev datasets for the base model, per epoch. SST is measured in cross entropy, Para in binary cross entropy, and STS in mean squared error, normalized to log scale. The overall value is a simple sum between the three values. As we can see, there is massive overfit on the SST task. Not shown are the loss graphs on the PCGrad model, which were visually identical. Plots were made with matplotlib.

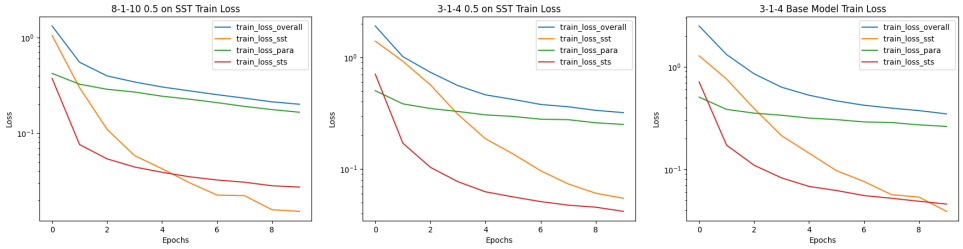

Figure 5: Loss curves on the train datasets for the large and small dataset splits for the 0.5 on SST model, and the small split on the base model, same format as fig. 3

5 Analysis

To start, binary cross entropy loss worked miles better than mean squared error on the paraphrase detection task. The reason for this is that we apply a sigmoid before rounding the logits for evaluation. The sigmoid function puts all negative numbers under 0.5 and all positive numbers above 0.5, allowing the easiest way for the model to differentiate between the two categories. The mean squared error function, in addition, does not work well for values between 0 and 1, as the squaring will decrease most of the distances instead of increasing them as it is supposed to.

Consider the fact that the base model goes from the best to worse on the SST dev accuracy when it goes to a larger dataset split, and that its loss curves have the dev loss get progressively worse even as the train loss gets to the best. This is clear evidence of overfitting on this task. This is most likely because the split weight on the dataset is too high. In addition, the base model on the smaller split has an SST curve that just barely intersects the STS curve at its end, so that it does not have the time to overfit, which could be the reason why it has the best results on this task. It also performs the worse on the paraphrase task, possibly because it is not able to take as big an advantage of the larger dataset, and the model is focusing on learning SST.

Thus, it is logical to add a coefficient of 0.5 on its loss function, to force the model to learn this task slower, yet learn the other tasks at the same rate. This worked for the larger split, as the model had more time to learn this task, so its results improved without overfitting too much. Our results show that this coefficient of 0.5 can mitigate the effects of overfitting. In addition, the model benefited from training on additional data from the paraphrase task, increasing its accuracy there.

However, on the smaller split, the 0.5 on SST model performed the worst on this task but the best on the others. We hypothesize that this was because the gradient updates from the SST task were muted, which allowed the updates from the other tasks to dominate. This is further evidenced by our loss curves. This model configuration did not have the time to fully converge on the SST task compared to the others. From the fact that the SST task being worse was correlated with the others being the best,

we can deduce that the gradient on the SST task is at least partially pointing in a direction opposite to the two other tasks.

We see that our model's performance might have plateaued on the STS task, as moving from 4 times the dataset to 10 times the dataset largely makes no difference in the results. In addition, its loss curves seem to be completely flat in the last few epochs. I hypothesize that no more amount of training would significantly increase results, and maybe this is due to mean square loss not being very effective with small changes. Moreover, I believe that it is not overfitting like the SST task, even though the data is being similarly over-trained, because the paraphrase task is very similar in nature to similarity. Training on that dataset prevents this task from straying towards an overfitting state, and increases its own accuracy. Furthermore, more data increases results on the paraphrase task, and its loss curves are still improving on the dev set per epoch, pointing to the fact that it could be underfitting, although not significantly.

Finally, our results show that Gradient Surgery provides the most balanced results. We can imagine that PCGrad is able to reduce destructive interference on the gradients so that the model does not optimize one task over the others. In addition, between splits, on the two datasets that were duplicated excessively, the model makes no reduction in accuracy, yet improves in the larger class, hinting that this method is resistant to overfitting. These results are a strong endorsement of this method to optimize gradient descent.

6 Conclusion

Through this analysis, we have come to a better understanding of multi-task learning when faced with unbalanced datasets. On our smallest dataset, SST, we experienced significant overfit. We found that experimenting with weighted loss functions is an effective way of mitigating this overfit, in our case by slowing down training on smaller classes. We also showed that Gradient Surgery results in a more balanced distribution of class results, without any sort of hyperparameter optimization.

For this project, I experimented with many different types of loss functions, loss function coefficients, data set splits, and I wrote a custom implementation of PCGrad in PyTorch. In addition, behind the scenes, I implemented several utility functions to efficiently debug and analyze my code to make my results reliable. For example, I made a testing mode in my code that trained the model using only a small sample of the data to run the code all the way through and make sure that there were no compilation errors. I trained over 20 models and analyzed their loss curves, convergence patterns, and dev results to determine which extensions to the base model provides the best results, and record enough data to understand why. I now understand machine learning at a more fundamental level, and I feel more confident in reasoning about model training and analysis.

Future work can experiment using a coefficient even lower than 0.5 on the SST loss, maybe 0.25, and training on even more data to increase the performance on the other datasets. Another direction could be to use cosine similarity loss on the STS task to possibly help it learn beyond the threshold I reached, which seems like a maximum with these configurations. A limitation of this work is only having using two sizes of splits, and only one additional set of coefficients on the loss function. Given more time and compute I could have experimented with even more models and make stronger claims about the data, in addition to having better results. Multi-task learning has a lot of challenges, and this paper only address a handful. There are many further directions to take this research, and I am proud to have made a contribution myself.

7 Ethics Statement

An important ethical consideration with multi-task learning is that the model might be better at some tasks than the others due to lack of training data, the issue at the center of this paper. For instance, imagine a machine learning model that is trained on translating between multiple languages. Each language represents a different task, and will have its own set of translations between every other language. A common problem in machine learning is that we have a lot of data about English, as well as many other European languages because all of the data is online, but not as much from other languages from developing countries. A translation bot trained on a mix of languages where some languages have vastly more translation data than others might lead it to expect a given input is from a popular language rather than a less popular one because it had so much extra data from the popular

language. This can lead to discrimination against these languages as the model will have a lower accuracy and users will more likely misunderstand it, rather than a model trained direction on just that particular language.

For this application, is crucial to learn all tasks equally, and get the most balanced accuracy, so that the model does not reinforce existing discrimination and inequalities. Thus, research into accurate training of multiple datasets using strategies like loss function reweighting and Gradient Surgery is essential. A possible mitigation strategy for this problem is increased research and dataset collection for each task. In our example, researchers could investigate more deeply into the language of developing countries and obtain more accurate data. For example, the researchers could find written books or other forms of text, scan them and convert them to digital copies to increase the amount of reliable training data.

Another ethical consideration is that multi-task learning inherently has difficulties with interpretability. A model trained on a single task is already hard to interpret on its own. Introducing several tasks only reduces interpretability, as the model embeddings are now optimized for multiple different purposes, making them that much more complex. For an extreme example, imagine a multi-task model trained on generating happy, helpful messages for therapy on one task, and generating sad, angry messages for a movie script on another task. Our model embeddings may mix in ways we do not understand, and these tasks might start to blend in implicit ways. Our happy messages might start sounding more negative and our movie scripts unrealistic, in ways that are subtle and not noticeable at first glance. We can have dangerous effects if applied in real-life therapy or other settings.

In this paper, I have spend a lot of time gathering and analyzing results, in the form of loss curves and accuracy data. This helped me understand my problem better, and hypothesize about strategies like loss re-weighting and understanding the benefits of Gradient Surgery, which ultimately improved my performance. However, if the issue is an implicit bias in the outputs of my models for certain tasks, interpretability data like the ones I obtained are rather rudimentary. A mitigation strategy would be to avoid applying multi-task learning to problems like these. If we want to make a machine learning bot to help us provide therapy to people who need help, we should not try to cut costs by using a model trained for other tasks as well, especially ones opposite to the one I want. We should train solely on messaging that is possible and helpful for a therapy patients, not necessarily a one-size-fits-all chat-bot.

References

Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. 2013. sem 2013 shared task: Semantic textual similarity. *Second joint conference on lexical and computational semantics (* SEM), volume 1: proceedings of the Main conference and the shared task: semantic textual similarity, pages 32–43*.

Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics*.

Samuel Fernando and Mark Stevenson. 2008. A semantic similarity approach to paraphrase detection. *11th annual research colloquium of the UK special interest group for computational linguistics, pages 45–52*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. *Proceedings of the 2013 conference on empirical methods in natural language processing, pages 1631–1642*.

Asa Cooper Stickland and Iain Murray. 2019. BERT and pals: Projected attention layers for efficient adaptation in multi-task learning. *CoRR*, abs/1902.02671.

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

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient surgery for multi-task learning. *CoRR*, abs/2001.06782.