An examination of multitask training strategies for different BERT downstream tasks

Stanford CS224N {Default} Project

Project Mentor: Olivia Y Lee

Björn Engdahl
Department of Computer Science
Stanford University
engdahl@stanford.edu

Matthias Heubi
Department of Computer Science
Stanford University
mheubi@stanford.edu

Abstract

We optimize the minBERT model for three different downstream tasks - SST-5, QQP and STS-B. We implement a multi-task training procedure and compare individual training versus multi-task training to see if we can leverage transfer learning effects across the tasks. We find that the STS-B results can indeed improve when trained with the similar QQP task, whereas we get better results with individual training for the unrelated SST-5 task. We find that mixing batches among the datasets during training is crucial for a well balanced model, sharing weights between tasks. When dataset sizes are disproportionate across the tasks, training on more samples for the smaller tasks towards the end of training may be beneficial. Finally, for QOP and STS-B involving sentence pairs, we find that concatenating the sentences before feeding them through the BERT-model is superior, as the model can leverage BERT's attention mechanism across the sentence pairs.

1 Introduction

In this project we explore different training techniques and architectures for three downstream tasks using a BERT-based language model: Sentiment classification, paraphrase detection and scoring of semantic textual similarity. We compare accuracy from fine-tuning each task separately vs. different multitask training approaches where all three tasks are trained simultaneously, to see if we can improve the model's performance by letting the network generalize across the three tasks.

We establish a baseline by adding task specific layers on top of a pretrained BERT-model - minBERT Devlin et al. (2018). Going on from there we try to improve results on two fronts:

1. Improving the *model architecture* to find the individually best performing setup
2. Improving the *training procedure* to find the approach that yields the best results

We then compare individual task training with multitask training, to see whether sharing parameters between tasks can yield better results.

The motivation for multitask training is to leverage transfer learning effects, assuming that training on one task can help learning for the other tasks, thereby improving overall results. By sharing some weights for all tasks, the model is more space-efficient compared to three individually fine-tuned models. However, multi task training can be problematic and suffer from "catastrophic forgetting", where weights from previous tasks are overwritten with the new task's weights McCloskey and Cohen (1989). We try different batch sampling techniques to alleviate effects from different data set sizes inspired by (Stickland and Murray, 2019). We also examine whether using a single combined loss function as opposed to three separate loss functions improves results. We try a combined regularized multi task loss function with learnable weights proposed by Lukas and Körner (2018).

2 Related Work

Using pretrained transformer networks for downstream NLP-related tasks was explored by Devlin et al. (2018). They used masked language modelling and next sentence prediction to train a network of stacked transformer layers.

In the context of multi task learning several approaches have been taken. Stickland and Murray (2019) added parallel task specific transformer layers and used a modified round robin approach to reduce effects of different dataset sizes when training on multiple tasks, gradually increasing samples from smaller datasets towards the end of training.

A lot of effort has been put into combining task specific loss functions into a "global" multitask loss function. However, combining individual loss functions presents a challenge: Task specific loss functions may have different scales and simply adding them together may make the gradients interfere. One way to combine the losses is to learn a task specific weight for each loss function and then take the weighted sum of the losses. Such an approach is explored by Lukas and Körner (2018) where they were able to boost performance of a main task by simultaneously learning auxiliary tasks.

3 Approach

3.1 Baseline

As a baseline, we start by implementing three simple classifier heads, one for each task, on top of a pretrained BERT model (Devlin et al., 2018), where the tasks and their heads are as follows:

- **Sentiment: Classifying a sentence into one of five sentiment categories**. The pooler output from BERT is passed through a dropout layer and then into a linear classifier with five output logits. Cross-Entropy loss is used as loss function.
- **Paraphrase: Detecting whether two sentences express the same meaning**. Both sentences are individually processed by BERT, then the two pooler outputs are concatenated, passed through a dropout layer and into a linear classifier. Binary Cross-Entropy loss is used as loss function.
- **Similarity: Determining the similarity of two sentences**. Both sentences are individually processed by BERT, then the two pooler outputs are concatenated, passed through a dropout layer and into a linear classifier. MSE loss is used as loss function.

To obtain baseline data, we train and evaluate each task individually and pick the best model from 10 epochs of fine-tuning, using an Adam-optimizer (Kingma and Ba, 2014) with weight decay, AdamW (Loshchilov and Hutter, 2017).

We notice that the network almost immediately starts to overfit with train accuracy approaching 100% after 10 epochs, whereas dev accuracy remains way below and largely stable after the first epoch.

We hypothesize that multitask training could alleviate this, because the model should generalize better when faced with three different tasks simultaneously. We thus implement a naive multitask procedure. We employ a round robin approach going through the entirety of the three datasets consecutively during one epoch. We fail to see generalization benefits from this naive multitask training procedure — in fact, accuracy drops across the board.

For comparison, we repeat the baseline runs with frozen BERT weights, so only the classifier heads get trained. This practically eliminates overfitting and confirms our suspicion that the BERT model provides too many degrees of freedom for the low number of training examples we have available.

Various experiments with regularization and higher dropout inside BERT do not improve the overfitting behavior. However, despite overfitting, we reproduce the finding from Devlin et al. (2018) that fine-tuning yields better performance. We conclude that for the given architecture, more training data or a much more refined training approach would be needed to further improve baseline results.

Based on our confirmation of the finding that fine-tuning beats frozen weights, all experiments from hereon will be conducted by fine-tuning the entire model.

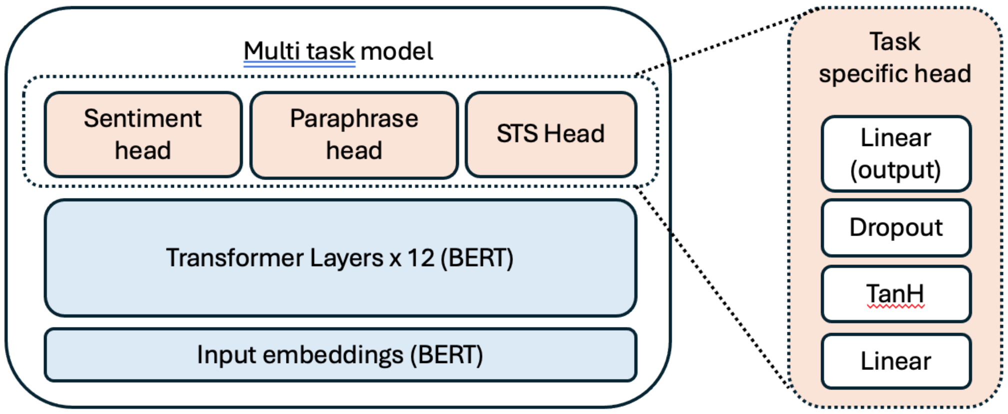

Figure 1: Model architecture used in the majority of experiments. Design of the task specific heads shown to the right.

3.2 Model architecture

Looking into architectural enhancements, we add cosine similarity (Reimers and Gurevych, 2019) between sentence encodings as an additional input feature to the classifiers for *Paraphrase* and *Similarity*, but cannot obtain a significant improvement.

We speculate that using BERT's attention mechanism to assess two input sentences at once will be more powerful than what we are likely to achieve with a classifier based on two individual BERT results. We thus feed both sentences, concatenated and separated by a [SEP] token, into BERT, yielding a massive improvement for *Similarity*, and a significant improvement for *Paraphrase*.

Regarding the surprising finding that multitask training did degrade results on the baseline, we theorize that there must be an internal model conflict when training for different tasks. We recognize that the BERT pooler output together with our linear classifiers on top forms multi-layer perceptrons. In the current architecture, the lower half of this perceptron is shared across all three tasks.

We thus move the pooler (feed-forward + tanh-activation) from BERT into each of the three heads and use the first token of the last BERT hidden state as input. This yields a substantial improvement in multitask training for *Sentiment* and *Paraphrase* and a slight improvement for *Paraphrase*, confirming our analysis.

We note however that *Sentiment* hovers substantially below the result achieved in individual training. This could be due to the fact that in naive multitask, *Sentiment* is first in round robin, with it's learning partially being wiped out by the two tasks that follow. A run in reversed order, where *Similarity* is trained first and *Sentiment* last, confirms this. We will look into resolving this problem – or even taking advantage of it – when optimizing the training procedure.

Assuming that *Paraphrase* and *Similarity* are conceptually related while *Sentiment* is different, we also run multitask training with only *Paraphrase* and *Similarity*, but fail to see a significant change.

Trying to optimize *Similarity* further we try different output functions for the classifier: Using *logits* directly, applying *clamping* [0,5] to the logits, and computing *sigmoid* of the logits multiplied by 5. All three approaches yield similar results, so we leave it at direct logit output which has the lowest compute cost.

Next, we experiment with different pooling strategies: We try basing the classifier input on: the *first token* [CLS], the *last token* [1], and the *mean* of all sentence tokens. All three strategies produce roughly the same results, so we leave it at [CLS], which is the most common approach.

[1]Note that "last token" in this context means the token at the position where the last unmasked token of each respective input sentence was, not the physically last token of the last hidden layer

3.3 Training procedure

We started the baseline by implementing a naive multitask procedure, training consecutively on the entirety of each dataset during each epoch. As expected, results are heavily influenced by the order in which the datasets are trained, with the performance of the first-in-line dataset degraded and the last-in-line boosted.

Before working to alleviate this problem, we try to take advantage of this ordering bias by producing a set of trained models where each task to be trained is placed last in the round robin queue. This *multitask with smart round robin* training approach can be viewed like a blend of balanced multitask and individual training. Indeed, we see transfer learning effects with an improvement for *Similarity* and a minuscule degradation for *Paraphrase*, but this comes at a cost, as we now have to run the entire training multiple times.

To achieve a more balanced multitask training, we need to solve the ordering bias problem, and also try to even out the differences in dataset sizes. We take inspiration from Stickland and Murray (2019) and implement an improved sampling strategy.

Using this method, during an epoch we select a batch from a dataset i of size N_i with probability p_i:

$$p_i \propto N_i \tag{1}$$

This procedure exposes the network to the full datasets during one epoch, as does our naive approach, but in a balanced way without exhausting the entire dataset before proceeding to the next one. As expected we see a more balanced learning progress, without the degradation of the first dataset of the epoch, yielding a better overall model. We also see a smoother learning curve and faster convergence compared to the naive approach.

With the QQP dataset being significantly larger than the other sets, we proceed along the path suggested by Stickland and Murray (2019) and even out dataset sizes during an epoch to see if this improves results. To do so, we introduce a hyperparameter $\alpha \in [0, 1]$:

$$p_i \propto N_i^{\alpha} \tag{2}$$

$\alpha = 1$ will let the sampling probability remain proportional to the size of the datasets, while decreasing α will increase the likelihood of sampling from smaller datasets. Stickland and Murray (2019) found that letting α decrease during training was beneficial, and we implement this technique as well:

$$\alpha = 1 - 0.8\frac{e-1}{E-1}, E > 1 \tag{3}$$

Here, e is the training epoch and E is the total number of epochs. Note that we no longer define an epoch to be the entire datasets, but rather an arbitrary number of training steps. We typically use 2500 steps per epoch. The runs with $\alpha = 0.5$ seem to produce somewhat less fluctuating results for the smaller datasets, and we see only a slight improvement from having $\alpha < 1$ or changing α during training according to 3.

Still experiencing overfitting, we also try an approach with a combined multitask loss function instead of having three separate loss functions. Believing that simply adding the losses together will be suboptimal, an alternative could be to weigh each task specific loss by a factor w_i, but instead of tuning the weights manually, we try an approach with learnable weights. We start with a combined loss function defined as:

$$\mathcal{L}_{comb}(x, y, \theta, w) = \sum_{i \in \mathcal{T}} w_i \mathcal{L}_i(x_i, y_i, \theta_i) \tag{4}$$

Here, $\mathcal{L}_i$ is the task specific loss function, w_i is a learnable weight and θ_i the model parameters. To avoid trivial solutions where $w_i \leq 0$, an additional regularization term is needed. We implement the combined loss function proposed by Lukas and Körner (2018). They define the combined loss as:

$$\mathcal{L}_{comb}(x, y, \theta) = \sum_{i \in \mathcal{T}} \frac{1}{2w_i^2} \mathcal{L}_i(x_i, y_i, \theta_i) + \ln(1 + w_i^2) \tag{5}$$

4 Experiments

4.1 Data

We use the *SST-5* (Socher et al., 2013), *QQP* (quo) and *STS-B* [2] datasets, split into train, dev and test as specified in the project handout[3].

4.2 Evaluation method

For *Sentiment* and *Paraphrase* we use dev accuracy (and test where available) , for *Similarity* we use pearson corelation between predicted value and gold on the dev/test data. For multitask-trained models we additionally provide an *Overall* score, picking the best model by computing the geometric mean over all three tasks' performance[4].

We noticed that model results could change a few percent simply due to seeding and initialization values for parameters. This brings up the question whether a change of one or two percent from one architecture to another is an actual improvement. An established approach would be to run between 5 to 50 *random restarts*, then picking the best — or recording the mean and variance, for when comparing model architectures, it may be beneficial to trade the absolute best, or the best mean for a slightly lower mean with lower variance.

Due to lack of compute, we weren't able to perform all our trials with multiple seeds. However, for *individual sentence-concat* we tried 6 different seeds for SST, finding that accuracy varied by 0.02.

We also looked into lowering training variance by implementing *weight decay* and *learning rate decay* with some promising initial results, but discovered that these hyperparameters are not generally applicable and a separate grid search would be required for each architecture and training method. So again, due to lack of compute, we had to keep the learning rate steady and weight decay at zero. This would certainly be an interesting future research topic.

We used the following three models when submitting to the test leaderboard:

- Highest score per task achieved in individual training
- Highest score per task achieved in multitask training with smart round robin
- Highest score per task achieved in multitask training with sampling (decreasing α)

Since we only had three submissions to the test leaderboard we mainly used the dev accuracy during evaluation. However, considering the comparatively small dev datasets, a chance best dev accuracy doesn't necessarily have to translate to the best test accuracy.

4.3 Experimental details

Unless otherwise stated, our experiments were run with the following hyperparameters:

- Learning rate: 10^{-3} with frozen BERT-weights and 10^{-5} for finetuning. Higher values for finetuning did not produce good results; in fact, 10^{-5} already seems to be at the aggressive upper end which is why we had looked into learning rate decay.
- Weight-decay: 0.0
- Drop-out: 0.3 for all layers. We also tried 0.1 without seeing improvements
- Batch size: 16
- For the AdamW optimizer, we used $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-6}$

4.4 Results

For all training variants the model architecture was the one shown in figure 1 with task-specific pooling layers inside the heads, using the [CLS] token of the last hidden state after feeding concatenated sentences through BERT.

[2]https://aclanthology.org/S13-1004.pdf

[3]https://web.stanford.edu/class/cs224n/project/default-final-project-handout-minbert-2024.pdf

[4]Because we submit the best model *per task* to the leaderboard, there is no *overall best* model score for test

	Sentiment SST-5 (Acc.)	Paraphrase QQP (Acc.)	Similarity STS-B (Corr.)	Overall (Score)
BASELINES	*Dev/Test*	*Dev/Test*	*Dev/Test*	*Dev*
individual, frozen-bert	0.417 / –	0.670 / –	0.271 / –	–
naive multitask, fine-tuning	0.486 / –	0.727 / –	0.389 / –	0.504
individual, fine-tuning*	**0.535** / –	0.785 / –	0.399 / –	–
MODELS				
individual, cosine similarity*	0.510 / –	0.788 / –	0.407 / –	–
individual, sentence concat*	0.530 / **0.535**	**0.888** / 0.887	0.864 / 0.869	–
naive multitask, sentence concat	0.465 / –	0.830 / –	0.871 / –	0.695
naive multitask, pooling in heads	0.503 / –	0.887 / –	**0.887** / –	0.731
TRAININGS *(pooling in heads)*				
multitask with smart round robin	0.518 / 0.527	0.884 / **0.888**	**0.887** / **0.886**	**0.732**
multitask sampling ($\alpha = 1$)	0.506 / –	0.886 / –	0.867 / –	0.728
multitask sampling (decreasing α)	0.519 / 0.533	0.884 / 0.884	0.875 / 0.872	**0.732**

Table 1: Results for dev and test (where available) datasets for the main experiments.
Note that the computation graph for SST is identical for the three models marked with * and the apparent variance (0.510 - 0.535) stems solely from different random initialization due to model changes for QQP/STS

The overall best performing model architecture employs task specific pooling layers for all three tasks and uses concatenated sentences for Paraphrase and Similarity.

For SST, no multitask-trained model performed better than the individually trained one. For STS, the multitask-trained models were slightly better. QQP held mostly steady across approaches. However, as explained above, variability from random initialization may explain away most of those differences.

Randomized sampling generally demonstrated a faster and more even convergence as can be seen in 2. Dynamically decreasing α benefits SST and STS, and slightly impacts QQP. This is expected, as a smaller α gives more emphasis to the two smaller datasets, reducing exposure to QQP.

The combined loss function was on par with the results from task specific loss. See figure 7 in the appendix for a graph of task specific loss versus combined loss.

5 Analysis

For the model architecture, we find that having a separate pooler for each task in multitask training is essential. Our theory is that different tasks pose conflicting demands on the integrated pooler. Surprisingly, what data the pooler is based on, be it the first token, the last token or the mean of all tokens is largely irrelevant. We attribute this to the fact that we do full model fine-tuning, so the BERT layers can adapt to the requirements of the poolers sitting above.

5.1 Sentiment Classification

Our best SST-5 result (0.535) is achieved when doing individual training. Getting SST correct is a difficult task even for a human and analyzing the incorrect predictions, we notice that it is not obvious which class to assign to the sentences. As a comparison, the top score as of today lies at 0.598 using a much more elaborate and powerful model [5]

Worth noting is that variability from random initialization exceeds most of the differences in obtained SST results, so without random restarts these results are not statistically significant, except for the massive drop in SST baseline performance when moving from individual to multitask training.

It can be concluded with high confidence, however, that SST has little to gain from transfer learning in our setup, which makes intuitive sense as it is conceptually different from the other two tasks which seem more related.

[5]https://paperswithcode.com/paper/an-algorithm-for-routing-vectors-in-sequences

5.2 Paraphrase Detection

For paraphrase detection, we achieve a significant improvement by passing the concatenated sentence pairs through the entire BERT model compared to passing each sentence separately. By concatenating the sentences, the model can use the attention mechanism of the BERT-layers, which seems ideally suited for such a task.

Future work could try to optimize this further by examining whether supplying different segment encodings for the two sentences have an addditional positive impact.

It seems that QQP benefits little from multitask training, which we attribute to the fact that the dataset is disproportionally large and is in fact dominating and slightly hurting its much smaller peers.

5.3 Semantic Textual Similarity

As with paraphrase detection, we find that feeding the concatenated sentence through the entire BERT model is far superior to passing individual sentences.

Similarity is the only task where we see a benefit from multitask training (see table 2). We believe the small STS-B dataset is able to leverage conceptual similarity between its task and *Paraphrase* to take advantage of the much larger QQP dataset.

5.4 Multitask training

Our work shows that transfer learning benefits from multitask training are best obtained when the tasks are related. As can be seen in table 2, only STS-B achieves an improvement over individual training.

	Sentiment (SST-5 Dev)	Paraphrase (QQP Dev)	Similarity (STS-B Dev)
Best individual	0.535	0.888	0.864
Best multitask	0.519	0.887	0.887
Difference	-0.16	-0.001	+0.023

Table 2: Comparing best results on dev datasets for individual vs multitask training

How batches are ordered during training is another important factor: Naive multitask favours learning the task ordered last, degrading performance of what was learned before. *Smart round robin training* takes advantage of this, but it comes at the expense of repeat training runs for each task. The computational cost is much higher than for individual training, as each task has to train on all datasets.

With sampling this effect is well mitigated: Figure 2 shows data obtained from a model with a single shared BERT pooler, which is prone to be pulled in different directions. As can be seen from the graphs, the first two tasks suffer heavily from their round robin position. The sampling approaches compensate very well for that degradation.

Furthermore, as can be seen from table 1, dynamically decreasing α reduces the negative impact of the disproportionally large QQP dataset on the two smaller ones, as samples become more evenly distributed across the three tasks. This effect is also visible in the STS chart of figure 2: In the later epochs, dynamic alpha sampling slightly surpasses the other two methods as alpha becomes smaller.

Our experiments with task specific losses versus a combined loss with learnable weights are shown in figure 7. Both versions produce similar results. We tried multitask training with combining loss functions for all three tasks as well as only for the related *Paraphrase* and *Similarity* tasks. Further experiments could try different types of combined loss functions such as gradient surgery (Yu et al., 2020).

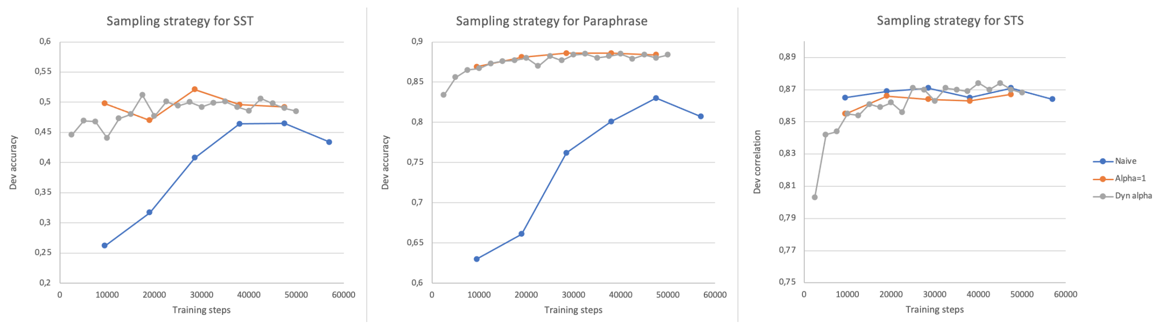

Figure 2: Effect on sampling strategy on the SST-5, QQP and STS-B datasets. Figure showing dev accuracy for naive sampling, proportional sampling $\alpha = 1$ and dynamically reducing α as in eq 3.

6 Conclusion

In this project we explored how multitask training compares to individual training for different downstream tasks, and how different multitask training approaches affect the result.

We found that multitask training is able to leverage transfer learning effects for tasks that are related. The benefit when compared to individual training is bigger for smaller datasets, whereas tasks with large training datasets may gain little or nothing but are not necessarily hurt by multitask training as long as a conflict-free model is used. Avoiding conflicts inside the model, where shared parameters are pulled in different directions by the different tasks, is absolutely essential. Otherwise multitask training will yield inferior results in comparison with individual training.

Looking at different batch sampling techniques, we found that linearly cycling through tasks will favor the last-in-line task by degrading learning results from earlier tasks. It is crucial to randomly (or at least proportionally) mix batches from different tasks during each epoch in order to obtain a balanced training result. If there is large asymmetry in the sizes of datasets, tasks with large datasets may hurt tasks with less training examples and produce an unbalanced result as well. Using a sampling technique that is not bound to the proportions in dataset sizes (with $\alpha < 1$ or dynamically decreasing α) can compensate the negative effect in these situations.

We also explored using a single combined loss function but weren't able to improve results compared to using separate task specific loss functions.

For the multi-task approaches, comparing the accuracy of each task as well as overall model performance, we find that *smart round robin* and *sampling with decreasing* α both yield almost the same results but smart round robin requires significantly more compute as well as a separate set of weights for each task.

In terms of model architecture, for the tasks involving two sentences, we found that concatenating the sentences before feeding them to the BERT-layers, thereby leveraging BERT's attention mechanism, improved performance significantly.

7 Team contribution

Both team members contributed to all tasks, but Matthias's contribution was more weighted towards developing / finding improved model architectures. Björn's contribution was more focused on developing / evaluating multitask training procedures.

References

First quora dataset release: Question pairs.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv (Cornell University)*.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv (Cornell University)*.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv (Cornell University)*.

Lukas Lukas and Marco Körner. 2018. Auxiliary tasks in multi-task learning. *arXiv (Cornell University)*.

Michael McCloskey and Neal J. Cohen. 1989. *Catastrophic interference in connectionist networks: the sequential learning problem*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *arXiv (Cornell University)*.

Richard Socher, Alex Perelygin, Jean Y. Wu, J.C.-i. 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.

Tong Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient surgery for Multi-Task learning. *arXiv (Cornell University)*.

A Appendix

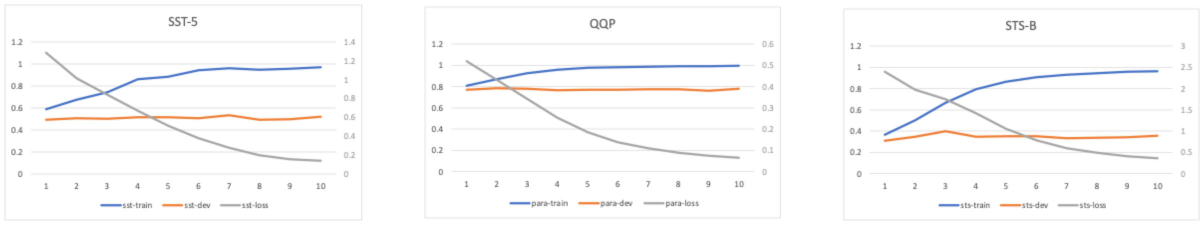

Figure 3: Training progress: Baseline - individual, fine-tuning

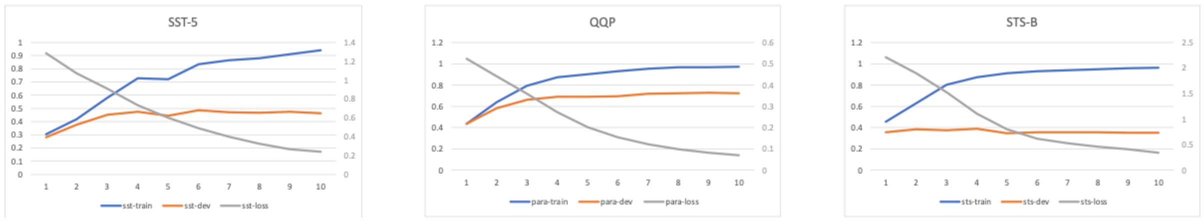

Figure 4: Training progress: Baseline - naive multitask, fine-tuning

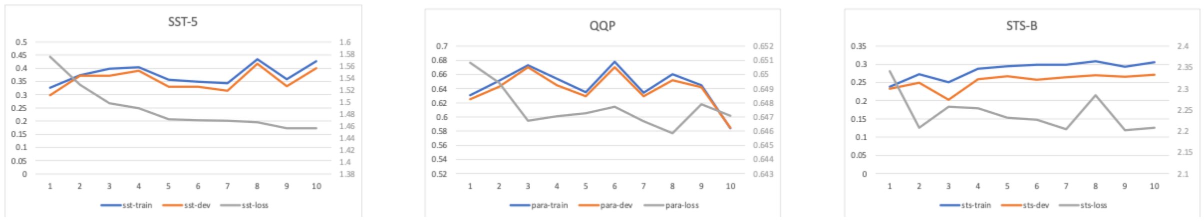

Figure 5: Training progress: Baseline - individual, frozen-bert

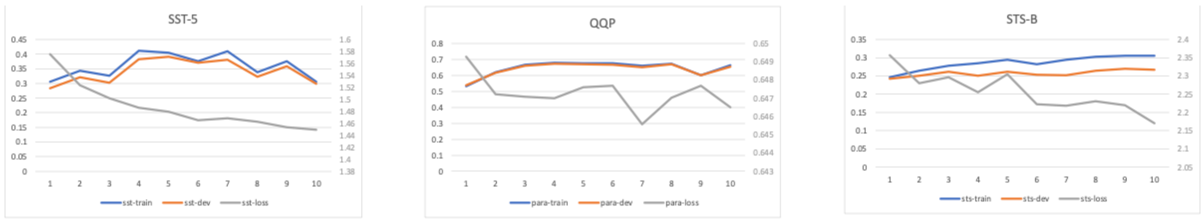

Figure 6: Training progress: Baseline - naive multitask, frozen-bert

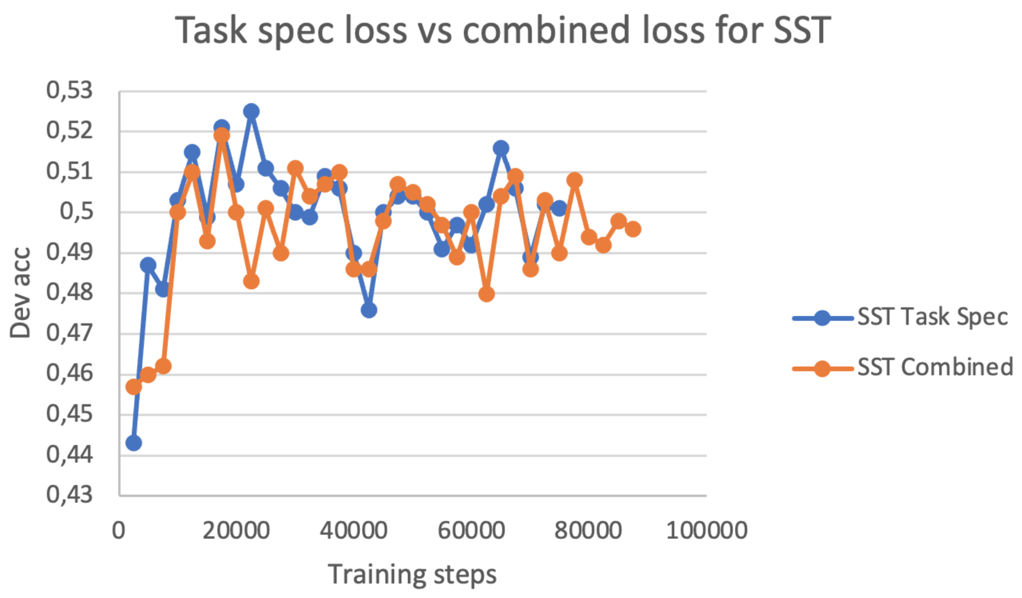

Figure 7: Task specific loss vs combined loss function (ref to equation 5. Figure showing SST dev accuracy for two training runs with the different loss functions. No significant difference.