Pals and Gradient Vaccine for NLP Multitask Learning

Stanford CS224N Default Project

Michela Marchini
Department of Computer Science
Stanford University
marchini@stanford.edu

Kate Callon
Department of Computer Science
Stanford University
kcallon@stanford.edu

Hannah Cussen
Department of Computer Science
Stanford University
hcussen@stanford.edu

Abstract

As opposed to the paradigm of finetuning models for individual tasks, multitask learning trains many tasks at once, learning features general enough to allow them to perform well across many tasks. This paper explores multi-task learning in the natural language processing (NLP) tasks of sentiment analysis, paraphrase detection, and semantic textual similarity (STS). This work mainly explores two methods for improving multi-task learning on top of the BERT model: an architecture-based approach and an optimizer-based approach. The architecture-based approach adds task-specific projected attention layers (PALs) into the BERT model itself. The optimizer-based approach, called gradient vaccine, addressed gradient conflicts between tasks. The combination of these two multi-task approaches is able to make gains in classification accuracy across tasks. Notably, however, ultimately the largest impact found on the BERT model comes in the form of the task-specific classification update of cosine similarity for the STS task, highlighting that optimized task-specific procedures and prediction measures are key as first steps in developing models for multitask classification.

1 Key Information to include

- Mentor: Shai Limonchik
- External Collaborators (if you have any): N/A
- Sharing project: No

2 Introduction

Multitask learning is a setting within machine learning in which the goal is to simultaneously perform a number of tasks using a single model. As opposed to the paradigm of finetuning models for individual tasks, multitask learning trains many tasks at once, learning features general enough to allow them to perform well across many tasks. This paper specifically explores multitask learning using the BERT model with the goal of improving performances across sentiment analysis, paraphrase detection, and semantic textual analysis tasks.

Multitask learning has been utilized in various subfields of machine learning for decades. Caruana (1997) originally motivated multitask learning with the idea that training over related tasks can give the model better understanding and performance than if it was trained on a single task. As computational power has grown and machine learning models reach unprecedented sizes, however, multitask learning has found a new motivation in that it can present a more efficient approach than deep learning and deep reinforcement learning models on a single task whose data requirements can make them difficult to generalize and incredibly inefficient to train many times over many individual tasks (Yu et al., 2020).

There are a variety of current methods for multitask learning which sit at various points in the model training process. Of the two models explored in this work, BERT and PALs (Cooper Stickland and Murray, 2019) suggests an architecture-based approach that adds task-specific layers into the BERT model itself, while gradient vaccine (Wang et al., 2020) works as a part of the optimization step to update losses. These two approaches have both proven successful in improving multitask-learning accuracy; however, they have not been combined. By focusing only on a single element of the multitask learning problem (either the model itself or the optimization method), these approaches fail to consider the entire scope of the classification task. This project seeks to combine the two methods to improve both the BERT model and the training optimization process in an attempt to make gains in multitask classification through utilizing multitask learning at each step of the model creation and training process.

3 Related Work

Learning several tasks simultaneously creates a complex optimization problem, which may result in poorer overall performance than learning tasks independently. Multiple multi-task learning approaches have addressed this by using independent training as a subroutine before consolidating the independent models (Rusu et al. (2016), Levine et al. (2015)). This approach produces a bloated multi-task model which relies on separate training for all tasks and therefore does not fully utilize the efficiency benefits of multitask training (Rusu et al., 2016). Moreover, independently trained and then composed models cannot benefit from sharing training among similar tasks. The Levine et al. (2015) paper found that end-to-end multitask learning produced better results and a different set of feature identifiers than individual task learning, suggesting that unified models detect more relevant features. This paper was on robotics, but it stands to reason that the same would hold in the NLP domain.

Previous work on multitask learning in the NLP domain includes Collobert and Weston (2008), which used a single convolutional neural network to perform part-of-speech labeling, chunking, named entity recognition, semantic role recognition, word-level semantic similarity, and sentence likelihood (both grammatical and semantic) evaluation. This work was expanded by pretraining with the language modeling objective by the same team in 2011 (Collobert et al., 2011). This is only one route of improvement in multitask learning, however. Different ways to improve multitask learning include alterations to architecture, training curriculum, and optimizer.

Alterations to the architecture include hard parameter sharing, soft parameter sharing, and adding additional layers in series or in parallel with the BERT blocks. Hard parameter sharing means that all tasks use the same embedding parameters, while soft parameter sharing means that all tasks get their own copies of the parameters that are then finetuned, but the parameters are constrained so that they do not stray too far between tasks. Cooper Stickland and Murray (2019) utilizes hard parameter sharing, since soft parameter sharing vastly increases computation and storage space requirements. Additional layers may also be chained to the BERT model on either end (adding them in series) or inserted within the BERT blocks (in parallel).

Training curriculum refers to the order in which tasks are served to the model for training. McCann et al. (2018) defines round-robin training as batches being selected from all tasks in a fixed order. Alternates to round-robin training include interleaved training, where one full epoch of training occurs of each task before proceeding to the next task, or sequential training, in which one task is fully trained before proceeding to the next task. However, sequential training can lead to catastrophic forgetting (McCloskey and Cohen, 1989). Modified round-robin training, also known as annealed sampling, is also sometimes used (Cooper Stickland and Murray, 2019).

Lastly, the optimizer may be tweaked. A more stable version of standard stochastic gradient descent, the most commonly used optimizer is Adam (Kingma and Ba, 2014). However, as a descendent of gradient descent, Adam is still prone to thrashing and conflicting gradients. Yu et al. (2020) posit that conflicting gradients between tasks is a major source of inefficiencies in multitask learning. They introduce PCGrad, which performs "gradient surgery" to eliminate gradient conflicts. Though the paper was eventually accepted to NeurIPS 2020, an ICLR reviewer rejected the paper out of concern with the underlying assumption that gradient conflicts are prevalent and problematic for multitask learning. The PCGrad work builds off of Chen et al. (2017) and Sener and Koltun (2018), which present different mathematical approaches to de-conflict gradients. The gradient surgery approach, in particular PCGrad, has seen some success in applications such as in Bi et al. (2022). Wang et al. (2020) presents gradient vaccine, a technique that claims to improve upon gradient surgery by proactively boosting gradients.

4 Approach

This work implements and combines two methods of multi-task learning. On the model level, we implement PALs layers as introduced in Cooper Stickland and Murray (2019) and on the optimization level, we implement Gradient Vaccine as proposed in Wang et al. (2020). Our PAL implementation references and is based upon the code presented in Cooper Stickland and Murray (2019), but ultimately the implementation is our own.

PALs are Projected Attention Layers which are added in parallel to normal BERT layers inside the BERT model. PALs are task-specific and take the form

$$PAL(h) = V^D MH(V^E h)$$

where MH(·) is a low-dimensional multi-headed attention and V^D and V^E are encoder and decoder matrices shared across layers. As such, the added complexity is limited as the number of encoder and decoder matrices added is equal to the number of tasks. The implementation presented here creates in total 3 encoder and 3 decoder matrices which are stored in PALs layers associated with their respective tasks and shared across all 12 BERT layers. Each BERT layer now has a list containing one PAL for each task, allowing it to choose the specific PAL to use depending on the task at hand. A comparison between the BERT and BERT and PALs block architecture can be seen in Figure 1. With the PAL added, each BERT and PAL block now has the following components: a multi-head self attention layer, an add-norm that takes the input and output of the multi-head attention layer, a feed forward layer, a PAL, and an add-norm that takes the output of the feed forward layer and the sum of the output of the PAL and the input to the feed forward layer.

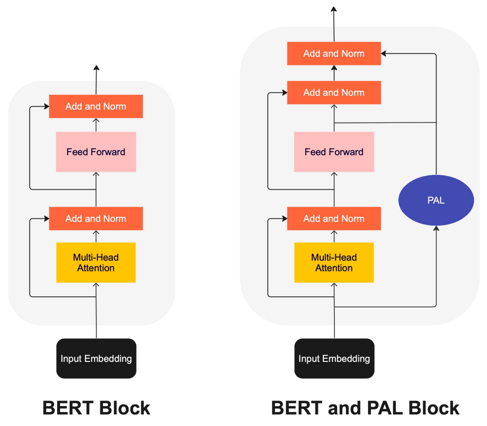
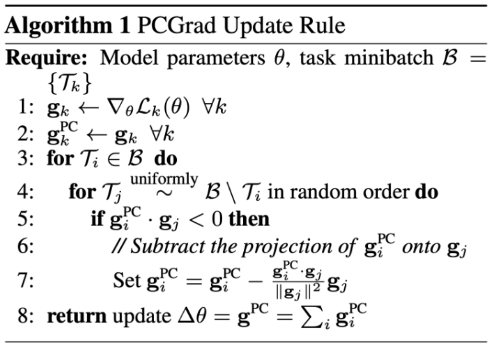


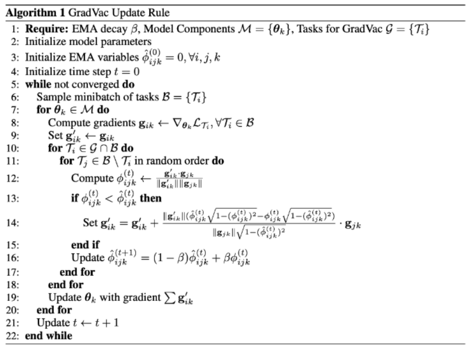
Figure 1: The PALs architecture adds a PAL and an Add-Norm to the base BERT block which allowing each layer to utilize task-specific parameters in its calculation of hidden states.

Gradient vaccine is a modification to an optimization technique known as gradient surgery as proposed in Yu et al. (2020). The general intuition behind gradient surgery is to resolve a phenomenon known as "conflicting gradients". Conflicting gradients occur when gradients from different tasks have negative cosine similarity during training. Especially in areas of high curvature in the optimization landscape and when the magnitude of the gradients differ significantly, these conflicting gradients prevent optimal multitask learning by heavily biasing the multitask gradient towards one task and degrading the performance of another task. In order to prevent conflicting gradients, the gradient surgery paper proposes a new algorithm known as PCGrad, which projects one of the conflicting task's gradient onto the normal plane of another task's gradient. This method has led to observed improvement is multitask learning objectives.

However, the PCGrad algorithm makes a very large assumption: any two tasks should have a gradient cosine similarity of zero after projection. Furthermore, it only modifies gradients with negative cosine similarity, and does not modify task gradients with detrimentally low positive cosine similarities. Thus, a new algorithm known as GradVac is proposed in Wang et al. (2020) to address this flaw. Instead of updating the gradient by projecting one task's gradient to the normal plane of another, gradient vaccine alters the gradient such that it matches a certain gradient cosine similarity that seems "reasonable" for two tasks. Therefore GradVec now allows for alterations of gradients for both negative cosine similarity and detrimentally low positive cosine similarity values between two tasks, ensuring that optimization progress is being made. Our implementation was rewritten based off of the Pytorch PCGrad respository Tseng (2020) and a forked repository which added gradient vaccine to the implementation Nzeyimana (2022). However, key differences in our implementation include removing the "retain graph" command for the backward call for all losses, setting zero gradients to "None" for any modified gradients, defining gradient vaccine as a function within the optimizer instead of a separate class, and rewriting the implementation to handle three tasks. A comparison between the PCGrad algorithm and the GradVac algorithm are pictured in Figure 2



```
Algorithm 1 PCGrad Update Rule
Require: Model parameters $\theta$, task minibatch $\mathcal{B} = \{\mathcal{T}_k\}$
1: $\mathbf{g}_k \leftarrow \nabla_\theta \mathcal{L}_k(\theta)\ \forall k$
2: $\mathbf{g}_k^{\text{PC}} \leftarrow \mathbf{g}_k\ \forall k$
3: for $\mathcal{T}_i \in \mathcal{B}$ do
4:     for $\mathcal{T}_j \overset{\text{uniformly}}{\sim} \mathcal{B} \setminus \mathcal{T}_i$ in random order do
5:         if $\mathbf{g}_i^{\text{PC}} \cdot \mathbf{g}_j < 0$ then
6:             // Subtract the projection of $\mathbf{g}_i^{\text{PC}}$ onto $\mathbf{g}_j$
7:             Set $\mathbf{g}_i^{\text{PC}} = \mathbf{g}_i^{\text{PC}} - \frac{\mathbf{g}_i^{\text{PC}} \cdot \mathbf{g}_j}{\|\mathbf{g}_j\|^2} \mathbf{g}_j$
8: return update $\Delta\theta = \mathbf{g}^{\text{PC}} = \sum_i \mathbf{g}_i^{\text{PC}}$
```



```
Algorithm 1 GradVac Update Rule
1: Require: EMA decay $\beta$, Model Components $\mathcal{M} = \{\boldsymbol{\theta}_k\}$, Tasks for GradVac $\mathcal{G} = \{\mathcal{T}_i\}$
2: Initialize model parameters
3: Initialize EMA variables $\hat{\phi}_{ijk}^{(0)} = 0, \forall i, j, k$
4: Initialize time step $t = 0$
5: while not converged do
6:     Sample minibatch of tasks $\mathcal{B} = \{\mathcal{T}_i\}$
7:     for $\boldsymbol{\theta}_k \in \mathcal{M}$ do
8:         Compute gradients $\mathbf{g}_{ik} \leftarrow \nabla_{\boldsymbol{\theta}_k} \mathcal{L}_{\mathcal{T}_i}, \forall \mathcal{T}_i \in \mathcal{B}$
9:         Set $\mathbf{g}'_{ik} \leftarrow \mathbf{g}_{ik}$
10:        for $\mathcal{T}_i \in \mathcal{G} \cap \mathcal{B}$ do
11:            for $\mathcal{T}_j \in \mathcal{B} \setminus \mathcal{T}_i$ in random order do
12:                Compute $\phi_{ijk}^{(t)} \leftarrow \frac{\mathbf{g}'_{ik} \cdot \mathbf{g}_{jk}}{\|\mathbf{g}'_{ik}\| \|\mathbf{g}_{jk}\|}$
13:                if $\phi_{ijk}^{(t)} < \hat{\phi}_{ijk}^{(t)}$ then
14:                    Set $\mathbf{g}'_{ik} = \mathbf{g}'_{ik} + \frac{\|\mathbf{g}'_{ik}\|(\hat{\phi}_{ijk}^{(t)}\sqrt{1-(\phi_{ijk}^{(t)})^2} - \phi_{ijk}^{(t)}\sqrt{1-(\hat{\phi}_{ijk}^{(t)})^2})}{\|\mathbf{g}_{jk}\|\sqrt{1-(\hat{\phi}_{ijk}^{(t)})^2}} \cdot \mathbf{g}_{jk}$
15:                end if
16:                Update $\hat{\phi}_{ijk}^{(t+1)} = (1-\beta)\hat{\phi}_{ijk}^{(t)} + \beta\phi_{ijk}^{(t)}$
17:            end for
18:        end for
19:        Update $\boldsymbol{\theta}_k$ with gradient $\sum \mathbf{g}'_{ik}$
20:    end for
21:    Update $t \leftarrow t + 1$
22: end while
```

Figure 2: The PCGrad Algorithm in Comparison to the GradVac Algorithm.

Though the model and training architecture is consistent across the three tasks, there are necessarily slight differences in the classification functionality as a whole as the tasks differ in input and output. For sentiment analysis, a dropout layer (p=0.3) is applied to the pooler output of the BERT model which is followed by a linear layer to output 5 logits, one for each sentiment class. For paraphrase detection, a single combined output is generated by concatenating the two pooler outputs of both input sentences. A dropout layer is then applied to the concatenated output and it is passed through a linear layer to output 2 logits. For both of the above tasks, the largest logit corresponds to the prediction label for that input, which is compared with the true label of the input to determine the cross entropy loss. The classification functionality for these two tasks is consistent across all experiments. For semantic similarity, however, the classification methodology used in the baseline and naive multitask yielded results far below the baselines of the other two tasks. As such, this functionality was updated using semantic similarity-specific updates. In the initial methodology the two pooler outputs were again concatenated and a dropout layer was applied. It was then passed through a linear layer to output 1 logit which was passed through a sigmoid layer and multiplied by 5 to normalize the it and keep it within the range 0-5. MSE loss was then used to calculate the loss between this output and the true ranking. The improved methodology, however, makes use of cosine similarity to vastly improve accuracy on the semantic textual similarity task. The cosine similarity implementation of the classifier calculates the cosine similarity between the two pooler output embeddings resulting in an output in the range [-1, 1]. As the desired output is in the range [0, 5], the output of the cosine similarity is run through a RELU layer and multiplied by 5 to scale it appropriately. MSE loss is again used to calculate the loss between this output and the true semantic textual similarity score. In addition to cosine similarity, multiple negatives ranking loss and triplet loss were also implemented in an attempt to improve semantic similarity. Multiple negatives ranking loss attempts to minimize the distance between positive sentence pairs while maximizing the distance between dissimilar sentence pairs. For this extension, a specialized dataloader was written to retrieve batches containing one data point with a sentence pair with a similarity label greater than 3 and the rest containing pairs with similarity labels less than 3. The prediction of the positive sentence pair and negative sentence pair logits were then calculated using the predict similarity function, and these predictions were used to calculate multiple negatives ranking loss with the following formula: $p - log(e^n)$, where p represents the positive sentence pair prediction and n represents the sum of the negative sentence pair predictions.

Triplet loss is similar to multiple negatives ranking loss, but is specific to sentence embeddings. It utilizes three sentence embeddings known as anchor, positive and negative. The anchor and positive embeddings are similar to each other, whereas the negative embedding is dissimilar to the anchor embedding. Similar to multiple negatives ranking loss, it attempts to minimize the distance between the anchor and positive embedding and maximize distance between the anchor and negative embedding. This implementation retrieves a sentence pair and uses one sentence embedding as the anchor and the other as the positive sample. It then retrieves another random sentence and uses the first sentence as the negative example. The loss is calculated by subtracting the negative embedding by the positive, and weighting the loss by the anchor and positive label, as the less similar they are the less the loss obtained should be weighted.

5 Experiments

5.1 Data

To perform multitask learning, we trained on three NLP tasks: sentiment analysis, paraphrase detection, and semantic textual similarity (STS). The datasets used for sentiment analysis are the SST dataset (train: 8,544 examples, dev: 1,101 examples, and test: 2,210 examples)Socher et al. (2013) and the CFIMDB dataset (train: 1701 examples, dev: 245 examples, test: 488 examples) Chen and Manning (2014). For paraphrase detection, the Quora Question Pairs dataset is used to identify duplicate questions (train: 141,506 examples, dev: 20,215 examples, and test: 40,431 examples) DataCanary (2017). Finally, the SemEval dataset is used for STS (train: 6,041 examples, dev: 864 examples, and test: 1,726 examples). May (2021).

5.2 Evaluation method

We evaluate the sentiment analysis and paraphrase detection tasks based on their accuracy. Since sentiment analysis is a multiclass classification problem, subset accuracy is used. Since STS is a regression task, the pearson correlation metric serves as the accuracy metric. All 3 datasets compare the model's predicted labels to their ground truth labels from the dataset.

5.3 Experimental details

All experiments were run on a Deep Learning AMI GPU PyTorch 1.12.0 image, running Ubuntu 20.04 on an AWS EC2 instance. With the exception of the baseline, all experiments were run with the finetune option, meaning that the BERT embedding weights are not frozen. All experiments other than the baseline used fixed-order round robin sampling. Our learning rate is $1e^{-5}$, batch size is 8, number of epochs is 10 and in each epoch we run 18000 batches, as this means the model will see each data point in each of the tasks at least once for each epoch. The PALs configuration added a PAL into every BERT layer, so there were 12 total PALs. The PAL size was 204 as suggested in Cooper Stickland and Murray (2019), so PALs are about 3.7 times smaller than the hidden size of 768. Finally, our training time for the naive multitask model was about 47 minutes per epoch, but including all of our computations brought the combined model run time up to around 2 hours.

To compare the various improvements made, we created a naive baseline for each task. For each task, a single trainable layer that post-processed BERT sentence embeddings was added. These layers were trained using the parameters from above, with the exception that the BERT parameters were frozen. These layers were trained using sequential sampling (since the shared parameters were frozen, the level of interleaving of the training curriculum did not matter).

Then, the following experiments were run:

1. Replace sequential training with round robin training (we call this naive-multitask)
2. Replace concatenated linear layer baseline with cosine similarity (on STS only)
3. Negative Sampling (STS only)
4. Multiple Negatives Ranking Loss (stopped early)
5. Triplet Loss (stopped early)
6. Replace standard BERT block with PALs
7. Replace Adam optimizer with Gradient Vaccine Optimizer
8. Combine all promising improvements (PALs, Gradient Vaccine, cosine similarity)

5.4 Results

Dev Results

	SST	Paraphrase	STS	Overall
Baseline (separate)	0.516	0.783	0.361	0.553
Naive Multitask Model	0.498	0.765	0.364	0.542
Gradient Vaccine Multitask Model	0.494	0.771	0.367	0.544
Pals Multitask Model	*0.501*	0.773	0.363	0.546
Cosine Similarity Model	0.492	0.769	*0.757*	*0.673*
Combined Final Model	0.499	*0.787*	0.727	0.671

Table 1: Results for each experiment as evaluated on the dev set. Maximum performance is italicized.

Test Results

	SST	Paraphrase	STS	Overall
Cosine Similarity Model	0.505	0.501	0.773	0.579
Combined Final Model	0.512	0.500	0.703	0.572

Table 2: Results of selected models on the test set.

Since they were significant implementations, it was disappointing to discover that both pals and gradient vaccine had little improvement on our accuracies and correlation. However, the improvement of cosine similarity for the STS dataset led to a more significant improvement than expected. Since the cosine similarity model and the combined model showed the greatest overall accuracies, we chose to obtain test results for both. Given the results of the models on the dev leaderboard, the accuracy for the SST dataset and the correlation for the STS dataset was not unexpected, but the accuracy for paraphrase detection was significantly worse for both models. Further analysis of these results in context of our approach can be found in the analysis section.

6 Analysis

As can be seen above, the multitask model that achieved the highest accuracy overall was the combined model which made use of PALs added inside the BERT model, gradient vaccine for loss optimization, and cosine similarity as a metric for semantic textual similarity. That this model performed best out of all the multi-task approaches is unsurprising as it combined multitask learning improvements on multiple levels, specifically in the model architecture and the optimization task. It also used a task-cognizant approach which aimed to find the best classification metric for each task and motivated the use of cosine similarity for the STS task.

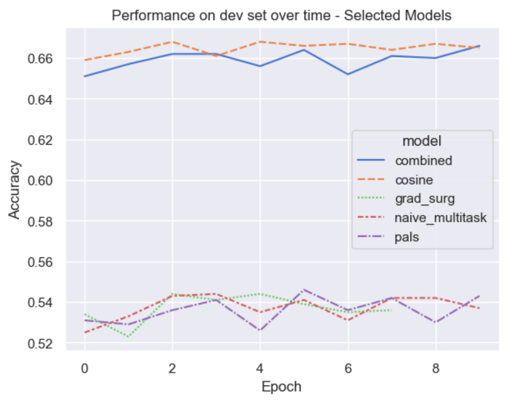


Figure 3: The combined and cosine similarity models vastly outperform all others.

Another clear pattern within the results is that on the dev set the paraphrase task consistently outperforms the other two tasks. A reason for this likely has to do with the fact that the paraphrase dataset is significantly larger than the dataset for the other two tasks. For the baseline models which were trained separately, the fact that paraphrase had more data to train on makes a lot of sense for why it would perform better, however it is also likely relevant in the disparity of performance in our multitask models. Though our multitask model employed round robin sampling in order to ensure it saw an equal amount of data from each task, the Quora dataset for paraphrase detection was over double in size compared to the other two tasks. When the STS and SST datasets ran out of data, we started the round robin sampling again over the dataset meaning that though the model saw an equal amount of data from each task, for STS and SST that data was not always unique. This fact provides a potential explanation as to why paraphrase detection consistently achieved the highest accuracy. This also explains why there is such a stark drop in accuracy for the paraphrapse detection task from the dev results to the test results.

6.1 PALS

That PALs did not vastly improve the accuracy is not entirely surprising. In the original Cooper Stickland and Murray (2019) paper, adding PALs represented a gain of only 1.7 points in accuracy. The promising, interesting feature of PALs, however, is that they can maintain multitask performance with lower training time and fewer parameters. This is borne out: the training time for PALS was almost 50% shorter per epoch than the combined model. A reason why PALs may not improve much over the naive multitask model is that while the BERT layers are pretrained, the PALs are not, which may make it difficult for the approach to make significant gains over the fully pretrained and fine-tuned model. Regardless, the fact that improvements, however small, were made in this parameter constrained environment is notable.

6.2 Loss

As seen in Figure 4, loss usually decreases until Epoch 4 or 5, then plateaus. The model only saves when accuracy on the dev set improves, and these saves usually occurred at epoch 4 or 5 and sometimes would not occur again even after the full 10 epochs of training had passed. For this reason, stopping training prematurely was determined to be acceptable, so long as epoch 4 had already completed. We believe that overfitting may happen between epochs 5 and 9. Notably, this is not the case for the combined model, which improved at epoch 9 as can be seen in 3. This implies that the combination of the added extensions may make the model less likely to overfit to the data.

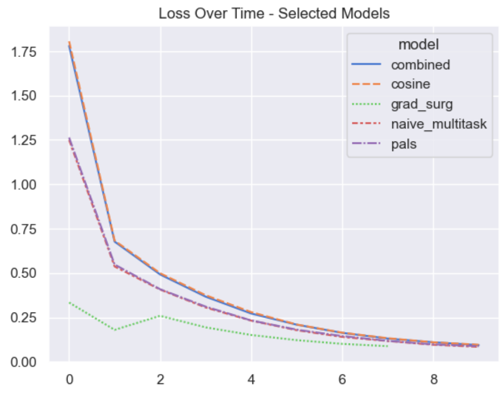
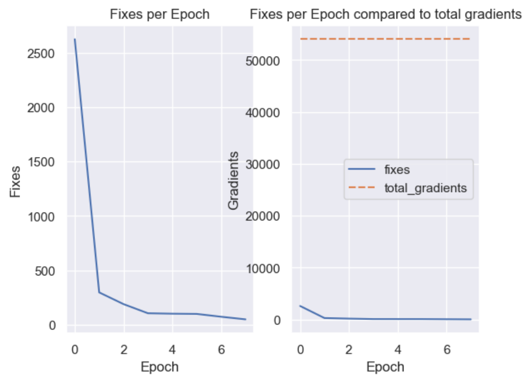
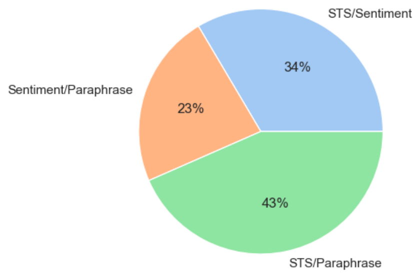


Figure 4: Loss through training epochs of selected models.

6.3 Gradient Vaccine



(a) The number of times a conflicting gradient is sensed and fixed by gradient vaccine goes down every epoch. It is insignificant when compared to the total number of gradient computations done.



(b) The majority of task conflicts in the first epoch occurred between the STS and Paraphrase Detection tasks.

Figure 5: Gradient Vaccine statistics

Gradient vaccine did not have a large impact on the accuracy. This points to an concern with the assumption that underpins these gradient-altering techniques for multitask learning: the assumption that gradients between tasks do conflict, and that this causes detrimentally low cosine similarity objectives which degrade overall performance. Our data does not support this assumption: after adding a metric to keep track of the number of gradient fixes, gradient vaccine only altered 2000 out of a possible 54,000 gradient update steps in the first epoch, and altered less than 0.5 percent of the gradients for the remaining epochs. (Figure 5a). From this, we can conclude that the tasks overwhelmingly do not cause conflicting updates, so gradient-editing techniques are not very useful. As stated in Wang et al. (2020), not only do gradient similarities reflect language proximity, but there is also a correlation between gradient similarities and model quality. This causes us to draw the tentative conclusion that these tasks may be too similar to have gradient conflicts. Logically, this makes sense. If sentences are semantically similar, they are also likely paraphrases of one another. Since these two tasks are similar, their gradients likely rarely conflict, as they will likely concur on shared parameter updates. However, in analysis of the gradient changes that were made, the paraphrase and similarity tasks had the greatest gradient updates between them, as pictured in Figure 5b. Since gradient vaccine targets task gradients which have less than their "similarity goal", it makes sense that gradient vaccine would largely target the paraphrase and similarity task gradients since these tasks should have the greatest "similarity goal". Ultimately, however, this concentration on certain tasks is not important given the sparse amount of gradient updates total. In terms of future work, the hyperparamter β used to support calculations for an exponential moving average of the cosine similarity goal between tasks could be manipulated to see if this would result in greater gradient updates.

6.4 STS Improvement

The modification with the greatest gain in accuracy with comparison to both the separate and multitask baselines, was replacing the baseline linear layer for STS prediction with a cosine similarity metric. The baseline concatenated the two BERT embeddings of the input sentences, then did a single feed-forward layer to output a single regression prediction. Concatenating the pooler outputs of the two input sentences introduces no inherent metric of comparison between the two. If we conceive of the pooler outputs in semantic embedding space, which is a reasonable conception given the embedding task, it stands to reason that semantically similar sentence vectors should exist close to one another directionally, which is the metric that cosine similarity measures. As such, calculating and optimizing the loss over the cosine similarity of the inputs will allow the model to better learn semantic textual similarity than optimizing over a low-rank representation of the concatenation of the two inputs. That this extension only significantly impacted STS scores makes sense as it only touches the prediction step for the STS task, so we would not expect it to have huge cross-task impacts.

Additionally, as stated in the approach, while implementations for multiple negative ranking loss and triplet loss were written, ultimately they were not included in our final model due to poor results. After conducting short experiments with our STS baseline, multiple negatives ranking loss achieved a dev accuracy of 0.008 and triplet loss achieved a dev accuracy of 0.116 compared to the baseline of 0.341 after 1 epoch. Further runs of both losses only resulted in degradation to this accuracy. Since STS has a continuous range of labels instead of a two (a positive and a negative), both losses arbitrarily defined the threshold for a positive label to be three or greater. However, this lack of distinct positive and negative labels could be a large reason for the lack of effectiveness for both losses in improving STS. Future work may include altering the positive threshold or applying these losses to the paraphrase detection task instead, since it only has two potential labels.

7 Conclusion

This work explored the effects of alterations to architecture, loss, training curriculum, and optimizer for multitask learning. Ultimately, multiple negatives ranking loss and triplet loss were not fruitful avenues to pursue improvements on the tasks of sentiment analysis, paraphrase detection, and STS. The most significant explorations were into adding cosine similarity, gradient vaccine, and PALs.

Cosine similarity was the single most impactful addition made, and it only effected the STS task. From this, the conclusion can be drawn that making sure the architecture reflects the structure of the task is a more significant contributor than adding additional layers, tweaking the loss, or tweaking the optimizer. The other significant conclusion of this work is that, for at least the three tasks we explored, gradient conflicts are not a large source of inefficiencies in multi-task learning. Therefore, the additional computational complexity added by techniques like gradient surgery and gradient vaccine are likely not worthwhile for these tasks. Further work in Gradient Vaccine would be to compare metrics of task similarity with the observed number of task-task conflicts.

Though this work does not show PALs as being particularly instrumental in improving performance it was shown to maintain accuracy, which was notable. Further work in PALs could involve a more complete experimental exploration of the tradeoff between number of parameters and performance.

References

Qiwei Bi, Jian Li, Lifeng Shang, Xin Jiang, Qun Liu, and Hanfang Yang. 2022. MTRec: Multi-task learning over BERT for news recommendation. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2663–2669, Dublin, Ireland. Association for Computational Linguistics.

Rich Caruana. 1997. Multitask learning. *Machine Learning*, 28:41 – 75.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Qatar. Association for Computational Linguistics.

Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. 2017. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. *CoRR*, abs/1711.02257.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 160–167, New York, NY, USA. Association for Computing Machinery.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. 2011. Natural language processing (almost) from scratch. *CoRR*, abs/1103.0398.

Asa Cooper Stickland and Iain Murray. 2019. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning. In *International Conference on Machine Learning*.

DataCanary. 2017. Quora question pairs.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. 2015. End-to-end training of deep visuomotor policies. *CoRR*, abs/1504.00702.

Philip May. 2021. Machine translated multilingual sts benchmark dataset.

Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. 2018. The natural language decathlon: Multitask learning as question answering. *CoRR*, abs/1806.08730.

Michael McCloskey and Neal J. Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. volume 24 of *Psychology of Learning and Motivation*, pages 109–165. Academic Press.

Antoine Nzeyimana. 2022. Pytorch-pcgrad-gradvac-amp-gradaccum/antoine nzeyimana.

Andrei A. Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. 2016. Policy distillation.

Ozan Sener and Vladlen Koltun. 2018. Multi-task learning as multi-objective optimization. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

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

Zirui Wang, Yulia Tsvetkov, Orhan Firat, and Yuan Cao. 2020. Gradient vaccine: Investigating and improving multi-task optimization in massively multilingual models. *CoRR*, abs/2010.05874.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient Surgery for Multi-Task Learning.