Learning Better Together: Exploring Multi-Task Learning for Natural Language

Stanford CS224N Default Project

Kapil Iyer
Department of Computer Science
Stanford University
`kapil.iyer@stanford.edu`

Abstract

In this paper, we compare three approaches to multi-task learning, where a single model is able to learn multiple related tasks simultaneously. These are namely the classic approach where we aim to optimize the sum of the tasks' losses all together, a round robin approach where we cycle optimizing each task's loss, and a gradient surgery approach which aims to capture the shared structure of these multiple tasks while not letting their conflicting components interfere with training. We find that gradient surgery performs the best in a multi-task learning model, and we were able to achieve a an overall test score of 0.637 across three NLP tasks.

1 Key Information

- Mentor: Xiaoyuan Ni

2 Introduction

Multi-task learning (MTL) is a subfield of machine learning in which a single model with a set of task-specific heads is able to tackle multiple downstream tasks. While traditionally a separate model would be trained for each objective, MTL has gathered increased interest in recent years due to it theoretically being able to provide stronger performance on the set of downstream tasks, relative to individual training, while also being space-efficient and data-efficient (Fifty et al., 2021). By sharing parameters across tasks which are trained over data from across tasks, MTL is able to learn an expressive representation of the space of all downstream tasks with fewer parameters and less data in total. Additionally, MTL is generally applied when dealing with similar classification tasks with a shared underlying structure, such as multiple language tasks which can share the same underlying word embeddings, meaning that the commonalities across tasks can be exploited for increased signal (Ito et al., 2022). As described in the widely cited Caruana (1997), MTL "improves generalization by using the domain information contained in the training signals of related tasks as an inductive bias...while using a shared representation; what is learned for each task can help other tasks be learned better."

That being said, minimizing multiple distinct objective functions with a set of largely shared parameters poses a challenging optimization problem in practice. In a MTL model, it is not uncommon to observe that each individual task gradient is nonzero, yet the optimizer stalls and makes little progress despite being nowhere near a critical point in the optimization space. It is also not uncommon to see that the model ends up performing very well for one task, or a group of especially related tasks, but very poorly for all other tasks. A known, but not fully understood, cause of these phenomena is the "conflicting gradients" problem. This problem arises from gradients varying in magnitude and pointing in different directions such that they cancel each other's components out or dominate one another (Liu et al., 2021). However, there has been a wide variety of proposals to decrease the impact of the conflicting gradients problem on MTL models. These range from specific changes to certain model's architectures (Devin et al., 2016) to changes in the way loss is computed and gradients are backpropogated (Chen et al., 2018). The latter set of proposals which are agnostic to model architecture are of special note due to their generalizability; MTL has been applied across a wide variety of subject areas and fields, from natural language processing (NLP) for social media moderation (Zhang et al., 2023) to computer vision (CV) for identifying celebrities (Vandenhende, 2022), across which many different types of model architectures are used.

In this paper, we specifically analyze the use of MTL as it applies to building upon a pre-trained BERT model and perform well on three different NLP tasks: Sentiment Analysis (SST), Paraphrase Detection (PARA), and Semantic Textual Similarity (STS). We do this by finetuning BERT's sentence embeddings while training task-specific heads to convert each task's input to logits. We test and compare different model-agnostic solutions for the conflicting gradients problem as it applies to these tasks. Namely we will try a "classic approach," where we process one batch from each task simultaneously by computing our loss function as the sum of the individual loss functions; and a "gradient surgery" modification to the classic approach (PCGrad) to directly improve upon the classic approach with respect to the conflicting gradients problem; and a "round-robin approach," where the optimizer steps once per task and rotates between tasks. We find that perturbations in where gradients are evaluated allows the round robin approach to perform stronger on average than the traditional approach which is more susceptible to conflicting gradients, but gradient surgery builds upon the traditional approach by directly reducing the conflicts in gradients while maintaining their shared components and structure.

3 Related Work

Devlin et al. (2019) describes the BERT model which serves as the backbone of our model. It describes self-attention mechanism the Transformer which makes fine-tuning this model straightforward for many downstream tasks. The model accepts task-specific inputs and labels and allows for finetuning all of its embedding representations for sentences end-to-end. At the output layer, the `[CLS]` representation just needs to be taken and fed into a task-specific head, which itself contains trainable parameters, for classification on any given task. The paper goes on to fine-tune BERT individually for eleven different tasks. Given how its `[CLS]` representations can be easily fed into multiple task specific heads, it is a natural step to attempt to finetune BERT with MTL.

Yu et al. (2020) attempts to formally define and theoretically understand the conflicting gradients problem in for the classic approach of MTL, where the sum of individual loss functions are minimized, and proposes the PCGrad gradient surgery solution to simultaenously reduce conflicting gradients and preserve their shared signal. PCGrad is notable because it fits right in between the backpropogation mechanism and optimizer to directly modify gradients and remove their conflicting components. Specifically, it projects each gradient onto the planes normal to the other gradients. While this paper does an excellent job defining the problem and its proposed solution and demonstrates that it achieves great performance on average, one area where it was limited was in its practical analysis of the conflicting gradients problem. We naturally hope to add to this practical analysis in this paper by examining conflicting gradients arising from specific examples in our training process.

Our work is also inspired by McCann et al. (2018), which introduces to us a round-robin batch-level sampling approach for MTL as applied in the realm of NLP to achieve stronger performance on 10 tasks simultaneously than individual training with their same model.

4 Approach

To begin, we built on top of a template provided by the default final project instructions and implemented key aspects of the original BERT model, including multi-head self-attention as well as a Transformer layer. We also implemented our own AdamW optimizer for training purposes. From here, we were able to feed inputs for the SST, PARA, and STS tasks into a tokenizer for a token representation, which when run through the pretrained BERT model we could extract their `[CLS]` representation, which we call the "BERT embeddings," to feed into task specific classification heads which we defined for each of our three tasks. We were then be able to train several different models based on each of our MTL approaches in order to train these task-specific heads, as well as finetune the BERT embeddings.

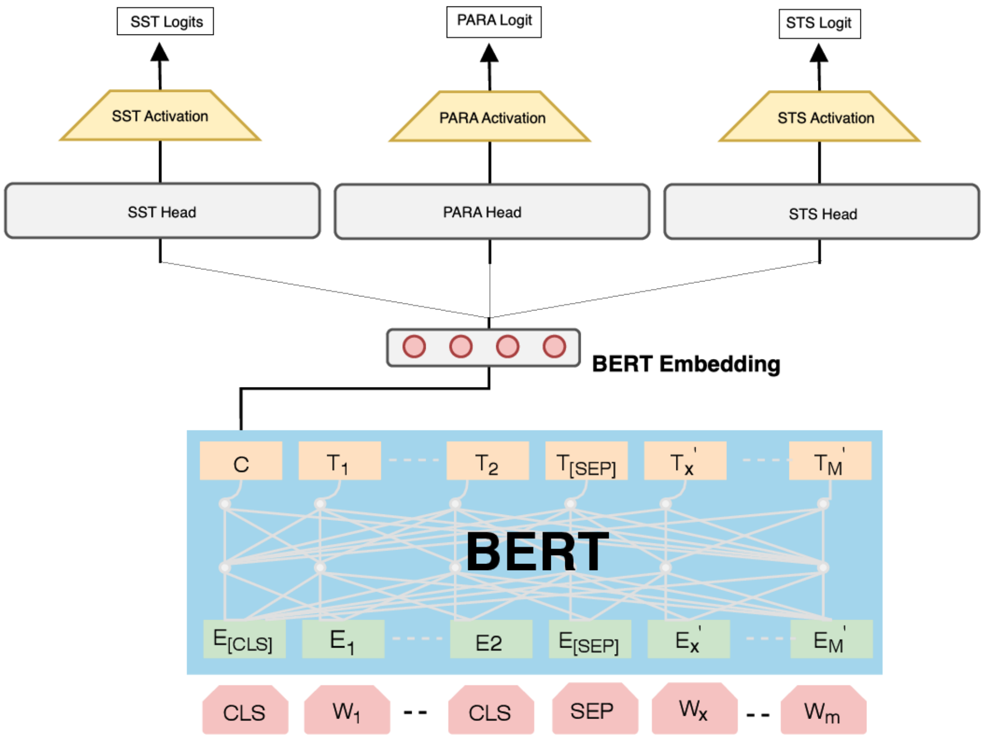


Figure 1: Our end-to-end model. Inspired by Figure 1 in Zahera and Sherif (2020).

4.1 MTL Approaches

We explored 3 main approaches to MTL, and we ensured that the model at least performed better than a minimum baseline of random guessing for each approach, as well as the base BERT model, as described in Results. We also ensured that finetuning the BERT embeddings with any of these approaches performed better than just using the pretrained weights.

In the case of the classic approach, we simultaneously processed one batch from each task, calculating loss as

$$\mathcal{L} = \mathcal{L}_{SST} + \mathcal{L}_{PARA} + \mathcal{L}_{STS}, \tag{1}$$

and stepping the optimizer to update all shared parameters, as well as our task-specific heads.

In the case of our gradient surgery approach using PCGrad, we continue to use the same loss function as described by Equation 1, but we directly modify the gradients associated with each task in hope of reducing conflicting gradients. We implement this ourselves by modifying the provided code in Yu et al. (2020) to form a wrapper around our AdamW class which allows it to intercept gradients and modify them before passing them to the step function. Specifically, it iterates over all pairs of gradients $g_i, g_j \in \{\nabla\mathcal{L}_{SST}, \nabla\mathcal{L}_{PARA}, \nabla\mathcal{L}_{STS}\}$ and if $g_i^* \cdot g_j < 0$, where g_i^* is the updated gradient so far for g_i, then the key modification,

$$g_i^* = g_i^* - \frac{g_i^* \cdot g_j}{||g_j||_2^2} g_j, \tag{2}$$

is made. That is, whenever two gradients conflict, we project them onto each other's normal planes.

In the case of the round-robin approach, we cyclically repeated a three-step update process on one batch from each task, where the loss on optimizer step $i \in \{1, 2, 3\}$ was computed as

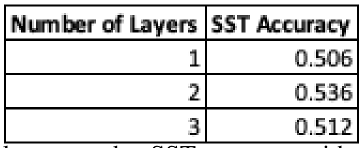
$$\mathcal{L}_i = \begin{cases} \mathcal{L}_{SST} & \text{if } i = 1 \\ \mathcal{L}_{PARA} & \text{if } i = 2 \\ \mathcal{L}_{STS} & \text{if } i = 3, \end{cases} \tag{3}$$

updating our shared parameters at each step and the task-specific heads at the respective task's step.

4.2 Task-Specific Heads

In designing our task-specific heads, we iterated on many different designs and adapted based on their relative performance on the dev split when using the classic MTL approach.

For the SST head, given a single input sentence, we wanted to output five logits representing a multinomial probability distribution for classifying the sentence into five classes of negative to positive sentiment. We used dropout, followed by a linear layer cutting dimensionality in half, followed by ReLU activation, followed by dropout, followed by a linear layer boiling everything down to five non-normalized logits, which we applied softmax activation to normalize. We then used multi-class cross-entropy against one-hot labels to compute loss, since we are solving a basic multi-class classification problem. Through experimentation, we observed that this two-layer model outperformed a similar one-layer or three-layer model across all tasks on the dev set by 3% accuracy, as desired (see Evaluation Method).



Number of Layers	SST Accuracy
1	0.506
2	0.536
3	0.512

Table 1: Impact of number of layers on dev SST accuracy with classic approach, for example.

For the PARA head, given two input sentences, we wanted to output a single logit representing the Bernouli probability that the two sentences are paraphrases of one another. We used a dropout on both inputs, followed by the same linear layer on both inputs, followed by ReLU activation, followed by concatenating these results, followed by dropout, followed by a linear layer boiling everything down to a single non-normalized logits, which we applied sigmoid activation to normalize. We then used binary cross-entropy against binary labels to compute loss, since we are solving a basic binary classification problem. Through experimentation, we observed that this model outperformed models where we added more layers to either the individual-sentence stage or to the fused-sentence stage of the model in each approach by up to 3% accuracy, as desired (see Evaluation Method).

For the STS head, given two input sentences, we wanted to output a single logit rating that two sentences are similar on a scale of one to five. We used a dropout on both inputs, followed by computing their cosine similarity, which we applied ReLU activation to normalize into the zero to one range and then multiply by five. We then used MSE against 5-ary labels to compute loss, since this objective by definition aims to maximize correlation between our predictions and the labels, as desired (see Evaluation Method).

4.2.1 Fusing Sentences and Cosine Similarity

For the PARA and STS tasks, we had to determine how to best fuse the two input representations. We considered the best options for both tasks among summing their hidden representations and applying a linear layer, concatenating their hidden representations and applying a linear layer, applying cosine similarity directly on their embeddings, or applying cosine similarity on their hidden representations. That is, for two (unbatched) inputs with embeddings $c_1, c_2 \in \mathbb{R}^n$ and respective hidden representations $h_1, h_2 \in \mathbb{R}^n$ and some arbitrary weight matrices $W^* \in \mathbb{R}^{mxn}$, $W = [W_1 W_2] \in \mathbb{R}^{mx2n}$, we want to decide between the following options, respectively:

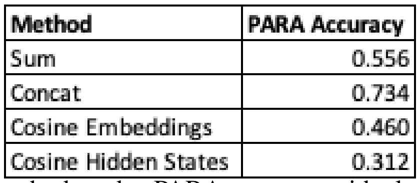
$$W^*(h_1 + h_2) = W^* h_1 + W^* h_1, \tag{4}$$

$$W \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = [W_1 W_2] \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = W_1 h_1 + W_2 h_2, \tag{5}$$

$$\cos(\theta_{e_1,e_2}) = \frac{e_1 \cdot e_2}{||e_1||_2 ||e_2||_2}, \tag{6}$$

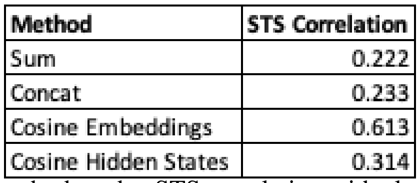
$$\cos(\theta_{h_1,h_2}) = \frac{h_1 \cdot h_2}{||h_1||_2 ||h_2||_2}. \tag{7}$$

We ran experiments trying each combination of measurements for the PARA and STS tasks and ultimately decided to go with the second option above (Equation 5) for PARA and the third option above (Equation 6) for STS. This makes some intuitive sense, as STS is a task which takes into account total sentence similarity, which one can get an understanding of by looking at the similarity of the sentence embeddings directly. On the other hand, just one word can transform two sentences from paraphrases to complete opposites, so cosine distance makes less sense for the PARA task, and instead we use an interaction layer with concatenation for greater expressivity associated with W_1 not necessarily equalling W_2 unlike the sum method.



Method	PARA Accuracy
Sum	0.556
Concat	0.734
Cosine Embeddings	0.460
Cosine Hidden States	0.312

Table 2: Impact of fusing method on dev PARA accuracy with classic approach, for example.



Method	STS Correlation
Sum	0.222
Concat	0.233
Cosine Embeddings	0.613
Cosine Hidden States	0.314

Table 3: Impact of fusing method on dev STS correlation with classic approach, for example.

4.3 Hyperparameter Optimization

When configuring our models, we had a range of hyperparameters to optimize. We identified the most important relevant hyperparameters to be our batch size, which affects both training time and generalization (Keskar et al., 2017); our dropout layer probabilities and AdamW weight decays, which can balance out overfitting (Lengerich et al., 2021); and our learning rate, which affects both training time and the optimality of our final set of weights (Wu et al., 2019). In order to optimize their hyperparameters, we wrote an original script from scratch that was able to perform a grid search for the best hyperparameters for each model we developed. Each combination of plausible hyperparameter values were tested and the combination which completed with the highest dev set performance was chosen.

Specifically, we performed a grid search across the following arrays of values recommended by Turc et al. (2019) for finetuning BERT: batch sizes in $B = [8, 16, 32, 64, 128]$, dropout probabilities in $D = [0.1, 0.3, 0.5, 0.7]$, learning rates in $R = [3e-4, 1e-4, 5e-5, 3e-5]$, and weight decays in $W = [0, 1e-5, 1e-4, 1e-3, 1e-2]$. In total, our grid search covered up to $|B \times D \times R \times W| = 5 * 4 * 4 * 5 = 400$ combinations to find the optimal hyperparameters

$$\lambda^* = argmax_{\lambda \in (B \times D \times R \times W)} perf(\lambda) \tag{8}$$

for each approach we tested, where *perf*(λ) is the averaged performance metric (described in Evaluation Method) achieved on the dev set with model hyperparameters λ.

5 Experiments

5.1 Data

For the SST task, we use the provided Stanford Sentiment Treebank Dataset with the following splits: train (8,544 examples), dev (1,101 examples), and test (2,210 examples). Each datapoint consists of a sentence and an associated integer label from one to five rating how negative or positive the sentence's associated sentiment is.

For the PARA task, we use the provided Quora Dataset with the following splits: train

(141,506 examples), (20,215 examples), and test (40,431 examples). Each datapoint consists of a pair of questions and an associated binary label stating whether the two questions are paraphrases of one another.

For the STS task, we use the provided SemEval STS Benchmark Dataset with the following splits: train (6,041 examples), dev (864 examples), and test (1,726 examples). Each datapoint consists of a pair of sentences and an associated label on a scale of one to five rating how semantically similar the two sentences are.

5.2 Evaluation Method

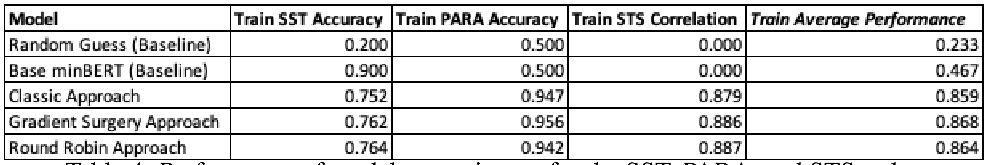
As described in the default final project instructions, we keep track of the accuracy of predictions for the SST task, the accuracy of predictions for the PARA task, the correlation of predictions to labels for the STS task, and most importantly the average of these metrics. We also record the individual training curves for each task over time for, as well as sample a few task gradients, for analysis purposes.

5.3 Experimental Details

Using our hyperparameter grid search script, we determined that the optimal hyperparameters for each model were the same: a batch size of 32, a dropout probability of 0.5, a learning rate of 3e-5, and an AdamW weight decay of 1e-2. However, we found that we had to limit the batch size to 16 for the model using the gradient surgery approach due to computational limitations, since gradient surgery required storing two times the gradients at any step (the set of modified gradients and the set of original gradients), as described in Equation 2. The paper referenced with regards to our chosen hyperparameter search space also recommended a maximum of 10 epochs when finetuning BERT-like models for experimentation (Turc et al., 2019). Also note that in practice, we only trained on a randomly sampled 6,041 examples per task train set within each epoch, in order to account for differences in dataset size without favoring one task's progression over that of another task.

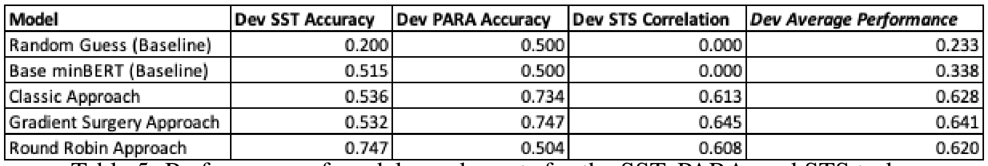
5.4 Results

Final test set score (gradient surgery approach chosen): SST Accuracy: 0.533 Paraphrase Accuracy: 0.751 STS Correlation: 0.626 Overall score: 0.637.



Model	**Train SST Accuracy**	**Train PARA Accuracy**	**Train STS Correlation**	***Train Average Performance***
Random Guess (Baseline)	0.200	0.500	0.000	0.233
Base minBERT (Baseline)	0.900	0.500	0.000	0.467
Classic Approach	0.752	0.947	0.879	0.859
Gradient Surgery Approach	0.762	0.956	0.886	0.868
Round Robin Approach	0.764	0.942	0.887	0.864

Table 4: Performance of models on train sets for the SST, PARA, and STS tasks.



Model	**Dev SST Accuracy**	**Dev PARA Accuracy**	**Dev STS Correlation**	***Dev Average Performance***
Random Guess (Baseline)	0.200	0.500	0.000	0.233
Base minBERT (Baseline)	0.515	0.500	0.000	0.338
Classic Approach	0.536	0.734	0.613	0.628
Gradient Surgery Approach	0.532	0.747	0.645	0.641
Round Robin Approach	0.747	0.504	0.608	0.620

Table 5: Performance of models on dev sets for the SST, PARA, and STS tasks.

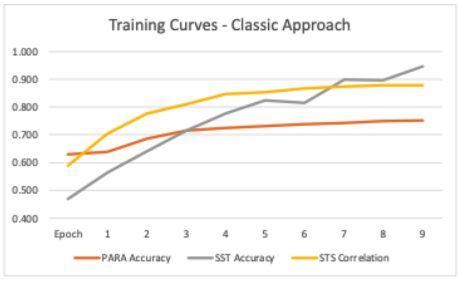
The above two tables show that on the dev set, the three MTL approaches were able to achieve stronger accuracy on the SST task than the Base minBERT model that was finetuned on just the SST task. This is an example of how MTL allows different NLP tasks to learn better from one another through a shared representation of language, specifically the BERT embeddings in this case. Furthermore, all approaches fared much better than random guessing, as desired.

The results above also align with the view that the classic approach to MTL may be especially prone to conflicting gradients, given that it had the lowest performance on the dev set out of the three approaches examined. On the other hand, the round robin is able to perform marginally better due to evaluating different task gradients at slightly different locations in the parameter space, having an optimizer step in between each gradient computation, which may lead to less conflicting gradients. However, the gradient surgery task performed by far the best overall, perhaps because it was able to negate some of the conflicting gradients that the classic approach suffered while maintaining a better shared structure than the round robin approach by evaluating gradients as a sum all together.

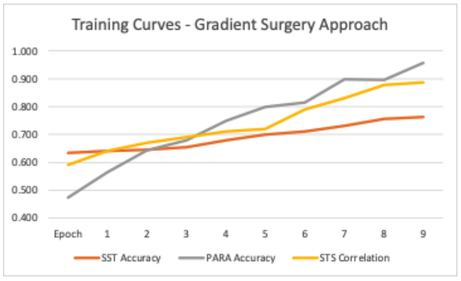
Interestingly, the round robin approach yielded notably superior performance on the SST task relative to the other approaches. The reason for this may be that the round robin approach is biased in the direction of the first (which also happens to be the largest) SST gradient, since a step for the SST task is made before any other tasks are considered, whereas the other two approaches always step in a direction taking all gradients evaluated at one point in the parameter space at the same time.

6 Analysis

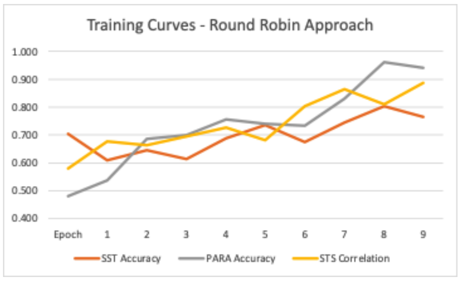
6.1 Conflicting Gradients: Individual Task Training



(a) Classic Approach



(b) Gradient Surgery Approach



(c) Round Robin Approach

Figure 2: Task training curves for classic, gradient surgery, and round robin approaches.

The character of the above individual task curves tend to reflect the MTL approach used, as expected. The classic approach at first improves its performance on all three tasks simultaneously, but at one point it clearly succumbs to the conflicting gradient problem, and both the PARA and STS training curves seem to completely level off while the SST curve dominates. The round robin approach has very chaotic training curves, as the optimizer takes turns stepping in each task's gradient direction, and in some cases, as we see at the end with SST, taking a step in one particular direction in the parameter space will inflict poor performance for the other curves, rather than focusing on maintaining a shared structure among tasks. The gradient surgery approach not only has the highest average performance, but it also has the smoothest training curves which all appear to continue to increase up through the last epoch of the experiment, being able to simultaneously capture shared structure and discard conflicting components of gradients, though it does tend to train slower after a while.

6.2 Conflicting Gradients: An Example

At the beginning of the training cycle for each model, we considered its response to the following inputs for the three tasks:

1. SST - Input: "As an actress, Madonna is one helluva singer." Label: "4."

2. PARA - Input one: "Who is the most famous contemporary singer/band in German-speaking world?" Input two: "Who is your favorite singer/band from your country?" Label: "0."
3. STS - Input one: "Red-shirt leader calls not to oppose army." Input two: "Beijing not to send army." Label: "0.6."

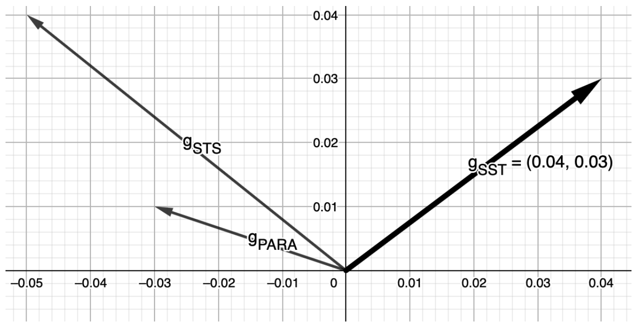


Figure 3: Projected gradients for SST, PARA, and STS tasks given the above.

Above, we have plotted the respective task-specific gradients given the above, projected onto the subspace spanned by two random parameters for visual purposes. We have the cosine distance of full PARA and STS gradients as 0.7, while their cosine distances with the full SST gradient is negative, similar to what the figure above depicts. Clearly, the PARA and STS gradients here have shared structure, and this is something we notice quite a bit given the relative similarity of semantics and paraphrasing, whereas the SST gradient is strongly conflicting. The frequency with which this happens may explain why our models in general performed more poorly on the SST task than the other two tasks, both in absolute terms and relative to improvement over our baselines. In this specific case, the conflict is perhaps caused by the PARA task wanting to push two positive terms apart (also having to do with singing like the SST task), as well as the STS task wanting to push two negative terms apart. And in line with our results, the gradient surgery approach makes the greatest improvement in average performance after its optimization step for this set of examples, removing the conflicting components of the gradients and moving in a direction straight up with respect to the subspace depicted above, whereas both the classic and round robin approaches move a lot more sharply left and drive the SST accuracy down.

7 Conclusion

In summary, we have shown that MTL has the potential to achieve greater performance on a set of similar tasks than individual training does, while also needing less data and taking up less space, due to its ability to share common goals of the tasks between shared parameters. Using different approaches to MTL to finetune a BERT model to perform well on three downstream tasks, we showed that gradient surgery can perform better than a round robin approach or the classic approach to MTL due to its ability to abate the conflicting gradients problem, though it can train slower. We achieved an overall score of 0.637 across the SST, PARA, and STS tasks using a basic model with the help of MTL with gradient surgery.

Due to time and compute constraints, the amount of epochs the models were trained for were smaller than desired, and we only saw part of the whole picture. We also did not get the chance to explore other techniques which tackle conflicting gradients with GradNorm which learns meta-parameters to weight the task gradients differently over time (Chen et al., 2018); this would be especially useful to investigate in the future in the context of this paper, given that the SST task here seemed to be dominated by the other two tasks.

References

Rich Caruana. 1997. Multitask learning. *Machine Learning*, 28(1):41–75.

Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. 2018. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks.

Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. 2016. Learning modular neural network policies for multi-task and multi-robot transfer.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Christopher Fifty, Ehsan Amid, Zhe Zhao, Tianhe Yu, Rohan Anil, and Chelsea Finn. 2021. Efficiently identifying task groupings for multi-task learning.

Hiroshi Ito, Kenjiro Yamamoto, Hiroki Mori, and Tetsuya Ogata. 2022. Efficient multitask learning with an embodied predictive model for door opening and entry with whole-body control. *Science Robotics*, 7(65).

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2017. On large-batch training for deep learning: Generalization gap and sharp minima.

Benjamin Lengerich, Eric P. Xing, and Rich Caruana. 2021. Dropout as a regularizer of interaction effects.

Bo Liu, Xingchao Liu, Xiaojie Jin, Peter Stone, and Qiang Liu. 2021. Conflict-averse gradient descent for multi-task learning.

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

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: On the importance of pre-training compact models.

Simon Vandenhende. 2022. Multi-task learning for visual scene understanding.

Yanzhao Wu, Ling Liu, Juhyun Bae, Ka-Ho Chow, Arun Iyengar, Calton Pu, Wenqi Wei, Lei Yu, and Qi Zhang. 2019. Demystifying learning rate policies for high accuracy training of deep neural networks.

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

Hamada M. Zahera and Mohamed Ahmed Sherif. 2020. Probert: Product data classification with fine-tuning bert model. In *MWPD@ISWC*.

Zhihan Zhang, Wenhao Yu, Mengxia Yu, Zhichun Guo, and Meng Jiang. 2023. A survey of multi-task learning in natural language processing: Regarding task relatedness and training methods.