Loss Weighting in Multi-Task Language Learning

Stanford CS224N Default Project

Julia Kwak
Department of Computer Science
Stanford University
julkwak@stanford.edu

Anna Little
Department of Computer Science
Stanford University
aslittle@stanford.edu

Abstract

Our project studies Multi-Task Learning (MTL) with a transformer-based model minBERT to perform on sentiment classification, paraphrase detection, and semantic textual similarity. Our chief approach in implementation was through Loss Weighting, a method that weighs individual task's loss in the calculation of the total loss in MTL training. We explored three loss weighting schema–Uniform Weighting, Dynamic Task Prioritization, and Homoscedastic Uncertainity Loss Weighting. We tested various training configurations, such as sequential, joint, numerous data manipulation for joint learning, as well as Cosine Similarity. We were able to achieve test accuracies of 0.513 for sentiment analysis, 0.788 for paraphrase, 0.346 for STS, and 0.658 overall.

1 Key Information to include

- Mentor: Cheng Chang
- Julia Kwak Contributions: Julia coded the implementations of the oversampling version of multitask training, and uncertainty loss weighting. She also located the papers for dynamic task prioritization and uncertainty loss weighting, and produced a majority of the figures in this report.
- Anna Little Contributions: Anna coded the implementations of dynamic task prioritization and cosine similarity, as well as the non-oversampling version of multitask training.

2 Introduction

Multi-Task Learning (MTL) is a method of machine learning where one model is trained simultaneously on multiple tasks. In principle, this approach allows the model to learn more richly by leveraging shared information, and thus improves accuracy on all tasks overall, as opposed to the tasks in isolation. Additionally, MTL significantly saves computational resources, as it takes advantage of shared architecture, reduced training time, and smaller storage demands (Crawshaw, 2020). However, in practice, MTL has not consistently met these intuitive expectations, running into challenges such as task interference or task imbalance. That being said, perhaps the most reliable application of MTL can be seen in Computer Vision, where there have been successful uses of MTL for various computer vision tasks (Liu et al., 2019). One methodology often employed is to place weight onto the loss functions of the given tasks or input samples, so that the total loss used in training is adapted to the current state of the model and task/example relationships (Gong et al., 2019). This project explores this apporach to determine the efficacy of loss weighting in an NLP setting.

With a baseline implementation minBERT, a transformer-based model, we conduct MTL to perform on sentiment classification (SST), paraphrase detection, and semantic textual similarity (STS). We study three different types of loss weighting techniques–dynamic task prioritization, homoscedastic uncertainty weighting, and equal weighting.

3 Related Work

One of the primary ways that MTL has been implemented is through loss weighting (Crawshaw, 2020). Loss weighting is the process of placing emphasis on the loss functions of the multiple tasks in MTL before they are summed to act as the total loss for training, essentially functioning as a weighted sum. This approach manipulates the loss to incentivize the model to prioritize tasks differently, and allows for more granular control of the model's learning across the different tasks. This concept of loss weighting is important for this project, given our three different tasks of varying difficulty and data sizes. A model designer could approach this by selecting weight hyperparameters to attempt to even the performance of all tasks, but this requires lengthy training time and exhausts compute resources. Other successful work let the weights of each task be trainable parameters. Guo et al. (2018) weighs hard tasks higher so that the model focuses more on improving the worst performing tasks, a method they called dynamic task prioritization. Another paper, Lin et al. (2017), instead considered focusing on the worst performing examples to manipulate the loss. Another approach to task weighting is to calculate task-dependent uncertainty. This method, detailed in Cipolla et al. (2018), uses a measure of the confidence of the model on tasks to determine how much focus should be placed on them.

Other methods that we employed in our work were drawn from NLP papers. Foundational for this project was Reimers and Gurevych (2019)'s work on Sentence-BERT, which specifically addresses the STS task. This work approaches STS as a regression task, using cosine similarity as the objective function. Also, this paper used the concatenation of sentence embeddings for classification tasks, which we found to be a helpful approach. They do make a note that concatentation is only relevant for classification, not for regression, but we were inspired by this work to consider the two together for a more comprehensive objective function.

Our team also understood the spirit of the default project to be about optimizing a multitask model, rather than training for each task separately on one model. It is a known problem that multitask models can have gradients that conflict with each other (Yu et al., 2020). Rather than performing gradient surgery as is detailed in this paper, it was our hope that by weighting certain tasks higher, the model would be nudged from this decision boundary and not stuck between two gradients that conflict.

4 Approach

4.1 Baselines

We use the performance of our sequentially trained model as a baseline to compare all of our models. Then, since we used the multitask training approach for the CV related implementations, we use the performance of our oversampling multitask implementation as the baseline for other multitask approaches. Finally, we compare multitask performance to a baseline of the performance of each individual task when trained to completion in solitude, and also compare multitask performance to each set of dual task performance.

4.2 minBERT and Predictions

We begin with a standard implementation of minBERT as defined in the starter code and project handout. This includes a multi-head attention layer, a dropout layer, the transformer layer, and AdamW Optimizer.

4.3 Multi-Task Learning

We try two different training schema for MTL. The first is sequential learning: each task is trained one after the other, only moving onto the next once the train set for the previous task is exhausted. The order was SST, paraphrase detection, and STS. In this method, the model learns the three tasks in three separate training loops. The underlying assumption is that the model will perform better as trains on increasingly difficult tasks, gaining foundational language knowledge that help with latter, more complex tasks.

The second approach is joint-training. Here, all three tasks are performed in the same training loop and contribute to the training loss. This approach is less vulnerable to catastrophic forgetting (McCloskey and Cohen, 1989), and allows for more generalization in the model's capabilities.

We test two different implementations of joint-training. These different methods are motivated by the different data sizes of the three tasks. The first implementation sets the training batch to be the size of the minimum batch across the three tasks. This method made use of some code found at (Tor). The second implementation sets the training batch to be the size of the maximum batch size of the three tasks (in this case, the Quora dataset (Quo). In the latter, we apply oversampling, where the other SST and STS datasets were cycled to match the size of paraphrase detection's.

Joint-training outperforms sequential training and is used as the baseline MTL model for all the loss weighting experiments. Results comparing the two schemas is shown in Table 1.

We use Cross-Entropy Loss, Binary-Cross Entropy, and Mean Squared Error Loss with Logits Loss, for sentiment classification, paraphrase detection, and semantic textual similarity, respectively.

4.4 Cosine Similarity

We test Cosine Similarity in our prediction for STS. For any given two word vectors A, B, the cosine of the angle between A and B measures the two words' semantic similarity. The value ranges from -1 to 1, where -1=equal, 0=independent, 1=opposite.

$$\text{Cosine Similarity} = \frac{A \cdot B}{\|A\|\|B\|} \tag{1}$$

We start out with a concatenation of the two embeddings for two input sentences via the minBERT CLS pooled output, both already separately fed through the dropout layer. We then compute the two embeddings' cosine similarity, which we append onto the initial concatenation of the two embeddings. Importantly, the two embeddings are dropped out, but not before they are passed through cosine similarity. By appending the cosine similarity score to the concatenated embedding, the model integrates both the holistic sentence-level representations and their direct semantic relationship into a dense, singular feature vector. Finally, we pass this extended embedding of size= $hiddensize * 2 + 1$ into the linear layer for STS.

4.5 Equal Loss Weighting

Our original implementation treated all tasks as equals, having the combined loss function be an average of the losses of the three tasks. No special weighting was involved. This was a naive approach to use as a baseline for the more complex methods we implemented.

4.6 Dynamic Task-Level Prioritization

In the course of training multitask models, most methods require the weighting of tasks to effectively learn from multiple tasks without one dominating another. This is often done through hyperparameter tuning, which requires extensive effort and compute to complete ablation tests. Instead, Guo et al. (2018) proposed to use the difficulty of tasks as weights, calculating the difficulty based on the model's performance on each task. To do this, they suggest creating a key performance indicator(KPI) for each task that functions as a moving average of the performance. These KPIs are used to up-weight harder tasks. The update of KPIs is accomplished according to the following, for task t:

$$\overline{\kappa}_t^{(\tau)} = \alpha \kappa_t^{(\tau)} + (1 - \alpha)\overline{\kappa}_t^{(\tau-1)}$$

Then, these KPIs are use as the input to a focal loss equation, specified as:

$$FL(\overline{\kappa}_t^{(\tau)}; \gamma_t) = -(1 - \overline{\kappa}_t^{(\tau)})^{\gamma_t} log(\overline{\kappa}_t^{(\tau)})$$

where γ_t is the task-level focusing parameter for task t. These focal losses are multiplied by the true training loss for each task, $\mathcal{L}_t$, combined together as below to create the dynamic task prioritization loss:

$$\mathcal{L}_{DTP} = \sum_{t=1}^{|T|} FL(\overline{\kappa}_t^{(\tau)}; \gamma_t)\mathcal{L}_t$$

These equations were all engineered by Guo et al. (2018), but all code to implement this was written by our team.

4.7 Homoscedastic Uncertainty Loss Weighting (HULW)

Homoscedastic uncertainty is uncertainty that is not dependent on input data, but rather changes between tasks. We can understand homscedastic uncertainty to indicate our confidence in any given task. As defined in Cipolla et al. (2018), we use Bayesian modeling principles to dynamically weigh loss functions according to Homoscedastic or Task-dependent uncertainty. Essentially, in this approach, we train our model to focus its learning on tasks with *lower uncertainty.*

We define $\mathbf{f}^{\mathbf{W}}(\mathbf{x})$ as our model output with weights $\mathbf{W}$ in input $\mathbf{x}$. We define three sets of probabilities for each of our three tasks. Given sentiment classification is a classification task, we define the likelihood of a set of predictions $\mathbf{y_{sst}}$ with model output $\mathbf{f}^{\mathbf{W}}(\mathbf{x})$,

$$p(\mathbf{y_{sst}}|\mathbf{f}^{\mathbf{W}}(\mathbf{x})) = \text{Softmax}(\frac{1}{\sigma_{sst}^2}\mathbf{f}^{\mathbf{W}}(\mathbf{x})) \tag{2}$$

For regression tasks–paraphrase detection and semantic textual similarity–the likelihood is the following Gaussian distribution, with $\mathbf{f}^{\mathbf{W}}(\mathbf{x})$ as the mean, and an observation noise scalar σ.

$$p(\mathbf{y_{para}}|\mathbf{f}^{\mathbf{W}}(\mathbf{x})) = \mathcal{N}(\mathbf{f}^{\mathbf{W}}(\mathbf{x}), \sigma_{para}^2) \tag{3}$$

$$p(\mathbf{y_{sts}}|\mathbf{f}^{\mathbf{W}}(\mathbf{x})) = \mathcal{N}(\mathbf{f}^{\mathbf{W}}(\mathbf{x}), \sigma_{sts}^2) \tag{4}$$

Let $\mathcal{L}_{sst}$, $\mathcal{L}_{para}$, and $\mathcal{L}_{sts}$ be our loss functions for SST, paraphrase detection, and STS, respectively. Upon factorizing the aforementioned probabilities, given our MTL approach, we define our loss for MTL below, for which the proof is given in Cipolla et al. (2018).

$$= -\log p(\mathbf{y_{sst}}, \mathbf{y_{para}}, \mathbf{y_{sts}}|\mathbf{f}^{\mathbf{W}}(\mathbf{x})) \tag{5}$$

$$\propto \frac{1}{\sigma_{sst}^2}\mathcal{L}_{sst} + \frac{1}{2\sigma_{para}^2}\mathcal{L}_{para} + \frac{1}{2\sigma_{sts}^2}\mathcal{L}_{sts} + \log\sigma_{sst} + \log\sigma_{para} + \log\sigma_{sts} \tag{6}$$

The noise parameter σ_i indicates the weight to apply on the corresponding loss $\mathcal{L}_i$. The higher the σ_i, the lower the $\mathcal{L}_i$'s weight, and vice-versa.The $\log\sigma_i$ functions as a regularizer that prevents σ_i from increasing too much. In application, we train the network to learn σ_i^2 as it is more numerically stable than σ_i. Here too the equations were taken from Cipolla et al. (2018) but all code was written by our team.

5 Experiments

5.1 Data

We use the Stanford Sentiment Treebank (SST) dataset (Socher et al., 2013) for sentiment analysis. This dataset includes sentences with sentiment labels 0-4, indicating negative, somewhat negative, neutral, somewhat positive, or positive, respectively. We use the Quora dataset for paraphrase detection, which consists of 400,000 sentence pairs with labels indicating whether they are paraphrases of each other (Quo). Lastly, we use the SemEval STS Benchmark dataset with 8,628 sentence pairs with labels $0-5$, indicating whether two sentences are unrelated (label=0) up to equivalent (label=5) in meaning (Agirre et al., 2013).

5.2 Evaluation method

We define our evaluation metric as in the CS224N default project expectations. For sentiment classification and paraphrase detection, we use accuracy. For semantic textual similarity, we use the Pearson correlation that captures the linear relationship between predictions and true labels.

5.3 Experimental details

Across the numerous experiments, there are components that are kept constant. The dropout probability is 0.3, the pretrain and fine-tune learning rate is $1e-5$, batch size is 8, and the number of epochs

for pretrain and finetune are each 10. The model architecture consists of one shared linear layer across all tasks, and one separate linear layer for each task. We use Cross-Entropy Loss, Binary-Cross Entropy, and Mean Squared Error Loss with Logits Loss, for sentiment classification, paraphrase detection, and semantic textual similarity, respectively. We use our implementation of minBERT, where the size of each hidden layer was 768. We ran experiments on either Google Compute Engine GPUs or Google Colab GPUs. We ran most tests on NVIDIA V100 GPUs to increase the number of operations per second.

Sequential Training No other specification.

Joint Training with Equal Weight No other specification.

Dynamic Task Prioritization Chose middle of the road $\alpha = 0.5$ and conducted ablation study to determine ideal γ value of 0.5.

Homoscedastic Uncertainty Loss Weighting Cipolla et al. (2018) found that their model was robust to initialization values for the noise parameter σ^2 between the ranges -2 to 5. And thus in our implementation, $\sigma^2 = 0$ for all three tasks.

5.4 Results

5.4.1 Undersampling vs. Oversampling Batch Size for Joint Learning

As outlined in the *Approach* section, Table 1 below shows higher overall performance of oversampling versus undersampling for joint learning. This makes intuitive sense, as more data helps the model learn a more complete representation. It is also backed by studies that compare oversampling to undersampling. Mohammed et al. (2020) compare undersampling to oversampling techniques for an imbalanced dataset, much like ours, and they find that oversampling performs better than undersampling for various classifiers and evaluation metrics.

Dataset	Smallest Set Size Adaptation	Largest Set Size Adaptation
SST	**0.503**	0.502
Quora	0.744	**0.781**
STS	0.359	**0.364**
Overall	0.642	**0.655**

Table 1: Dev Accuracy in Joint-Training

5.4.2 Full Comparison: All Loss Weighting Methods

Our results show that Homoscedastic Uncertainty Loss Weighting performs the best overall with a dev accuracy of 0.657, compared to Uniform Weighting and Dynamic Task Prioritzation with 0.655 and 0.649 respectively. Compared to Uniform Weighting, Homoscedastic Uncertainty Loss Weighting improves significantly in sentiment classification and STS, while worsening in paraphrase detection. Dynamic Task prioritization's performance decreases for all tasks against Uniform Weighting, however, performs better on Paraphrase compared to Homoscedastic Uncertainty.

Dataset	Uniform Weighting	DTP	HULW
SST	0.502	0.486	0.517
Quora	0.781	0.780	0.763
STS	0.364	0.361	0.381
Overall	0.655	0.649	0.657

Table 2: Dev Accuracy for Loss Weighting

6 Analysis

6.1 Single-Task Training vs. Multi-Task Training

As discussed in the *Introduction*, MTL has serious limitations. Our results reinforces this understanding. Below are comparisons of dev accuracy in Single-Task Learning and MTL. We can see that, for every task, the dev accuracy goes down from Single-Task Learning to MTL.

	Single-Task Learning	Multi-Task Learning
SST	0.538	0.502
Paraphrase	0.786	0.781
STS	0.377	0.364

Table 3: Highest Dev Accuracy for Single-Task Learning vs. Multi-Task Learning

The results signal that Single-Task Learning allows the model to dedicate its learning capacity to strictly understanding the given task, tailoring the model to the task's specific features and patterns. The MTL results show that not all tasks share beneficial, transferable information. The attempt to generalize across tasks leads to dilution of critical task-specific information. We presume that the model likely experiences a "negative transfer," which describes performance reduction in previous knowledge due to new learning (Zhang et al., 2023). More specifically, the model experiences "task interference," where shared parameters enable conflicting information in multiple tasks to intervene one another, resulting in a murky understanding of any one given task. Another more drastic explanation could be "catastropic forgetting," where neural networks suddenly forget previously learned information upon learning new information (McCloskey and Cohen, 1989). Moreover, competing gradient directions of the different tasks may create a tension in the parameter space. This could prevent the optimization to stay stagnant, as its being pulled in numerous directions simultaneously Yu et al. (2020).

6.1.1 Cosine Embedding: Single-Task Learning vs. Multi-Task Learning

Task/Joint Cosine Setting	Dev Accuracy
STS: No Cosine STL	0.377
STS: With Cosine STL	0.393
STS: No Cosine MTL	0.365
STS: With Cosine MTL	0.373
Overall: No Cosine MTL	0.655
OVerall: With Cosine STL	0.652

Table 4: Dev Accuracy for SST Dependent on Cosine Embedding

In efforts to improve our STS score, we implement Cosine Similarity, after which we observe a marked improvement in STS performance. This enhancement, however, comes at the cost of lowering overall performance. We posit that the model over-relies on the Cosine Similarity for its predictions. Since the information passed through the linear layer for predicting similarity consisted of both the cosine similarity and the two sentence embeddings concatenated together, we believe that the model is overrelying on the cosine similarity and focusing less on the two input sentence embeddings. We can categorize this is a mild form of reward hacking (Skalse et al., 2022), where the Cosine Similarity functions as a shortcut for the model to learn the similarity without learning the sentences. Additionally, our architecture has one shared linear layer across all three tasks and an additional linear layer for each task. It is likely that the gradient signal being passed back during backpropagation experiences task interference, where the other two tasks do not benefit from STS' increased awareness of Cosine Similarity. This results in decreased generalizability of the model across this shared architecture.

6.2 Dual-Task Training vs. Triple-Task Training

Given the limitation of Multi-Task Learning compared to Single-Task Learning, we further examine the effect of the number of tasks in MTL on performance. Our intuition was that as the number of

tasks increased, so did the overall loss. Figure 1 shows the effect of task pairs (sentiment, paraphrase), (paraphrase, STS), and (sentiment, STS) on performance, and how the tasks within these pairs perform relative to how they perform in our Uniform Weighting 3-Task model. Our initial assumption was true, all but one dev accuracy was higher in the 3-Task model compared to the first: SST with the Paraphrase and STS pair.

Pair	Task Within Pair	Dev Acc
Sentiment and Paraphrase	Sentiment	0.497
	Paraphrase	0.786
Paraphrase and STS	Paraphrase	0.777
	SST	0.366
Sentiment and STS	Sentiment	0.506
	STS	0.354

Table 5: 2-Task MTL

Task	Dev Acc
Sentiment	0.502
Paraphrase	0.781
STS	0.364

Table 6: 3-Task MTL

6.3 Comparison of Loss Weighting Methods

This sections details our various Loss Weighting methods (as displayed in Table 2) and poses explanations for their observed behavior. Homoscedastic Uncertainty achieves the highest overall performance. This is unexpected, because the principle behind HULW is that we place more weight onto tasks with less uncertainty; that is, tasks that are easier to predict. However, compared to Uniform Weighting, we observe that that paraphrase, the task that records the highest dev accuracy, performed poorer ($0.781 \rightarrow 0.763$), while sentiment ($0.502 \rightarrow 0.517$) and STS ($0.364 \rightarrow 0.381$), both tasks with lower dev accuracy, performs better. We suspect that the model overfit to the Quora training set. Since the loss weighting pushed the model to minimize the loss of paraphrase most heavily,the model may have learned too specifically to the training set and become unable to generalize to the dev set. This can be suspected in Figure 2, where the difference between the Para Dev Loss and the than training dev loss is much more significant.

Dynamic Task Prioritization performed most poorly across our methods. It also exhibited unexpected behavior. Theoretically, DTP should weigh harder tasks more heavily, however, we see that our harder tasks–sentiment and STS–both decreased in accuracy compared to Uniform Weighting. Paraphrase performed worse as well. We suspect that the model erroneously focused on the hardest task: STS. In Figure 2, we note a convergence in training loss and a stabilization in the development loss for the STS, indicating a stop of significant learning progress for this task. We hypothesize that this convergence, or stagnation rather, of STS performance is attributed to the model reaching a local minimum (given that the dev accuracy for STS is not high). Morevoer, DTP utilizes focal loss, which is a cumulative average of the performance, and thus it lacks flexibility in quickly switching its focus to other tasks. The model, due to this slowness, could not turn its focus to the other tasks quickly enough. This likely caused the model to oscillate within a narrow performance range, precluding substantive improvement. Another reason why the performance of DTP may be so low compared to the other weighting schemes is because the model overfitted to the training data. If a model is overfitting, we would expect to see the training loss go to 0 while the dev loss increases. Figure 2 shows that the DTP delta of dev loss for STS and paraphrase tasks only continues to increase. This means that the dev loss is getting further and further away from 0, indicating that the model is decreasing in performance.It is also possible that the model encountered competing gradient directions between the different tasks, pulling in opposite direction, and preventing the model from learning any of the three tasks effectively (Yu et al., 2020).

A study done by Gong et al. (2019) supports our findings. They compared numerous MTL models, including Homoscedastic Uncertainty, Uniform, and a model very similar to Dynamic Task Prioritization called Dynamic Weight Average (DWA). They observed Uniform Weighting to outperform Homoscedastic Uncertainty and DWA in many of their experiments.

This is reinforced by the fact that there is not enough robust analytic theory in the literature that understands optimal weighting for MTL, let alone optimal configuration of MTL itself (Gong et al., 2019).

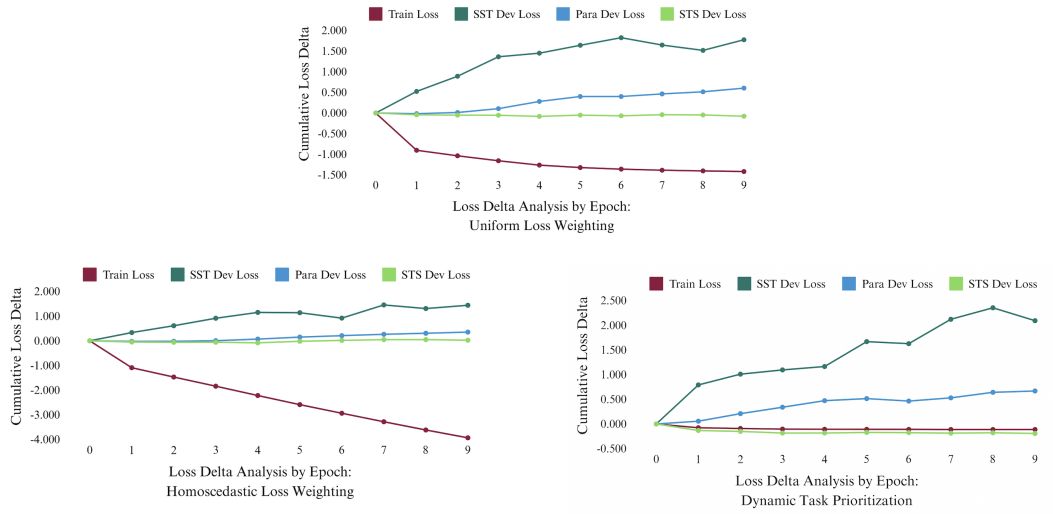

Figure 1: Cumulative Loss Delta: Train Loss vs. Dev Loss

7 Conclusion

We found that multitask learning performs better than training the model on the three tasks sequentially, but worse than training three separate models for three individual tasks. This is supported by the literature that discusses how fraught with complications MTL can be (Yu et al., 2020). The decision to train all tasks simultaneously set up the rest of the experimentation in this paper. We implemented two techniques for conducting MTL, and showed that oversampling performs better than undersampling. We also considered cosine similarity for the relative improvements it provided to the STS task, but concluded that this additional information caused the model to focus too heavily on STS at the expense of the other tasks. We then implemented three techniques for weighting losses, two of which were derived from CV techniques, and compared their performance. Ultimately, homoscedastic uncertainty loss weighting performed the best overall and dynamic task prioritization performed worse than vanilla, a result that is consistent with other studies (Gong et al., 2019). The primary limitations of this work include its potentially narrow sphere of application. This work was conducted on a minBERT implementation, using relatively small datasets, and computed on relatively small computers (single GPUs for each model). While many concepts from this work should transfer to various in-class assignments, the results may vary based on structure of model, compute, and data availability. In particular, we saw results when testing on a toy dataset that did not hold for larger scale datasets. Additionally, MTL is an underdeveloped area of work, where conceptual understanding for how it best functions is nascent. We know that many problems today are more aptly solved with single task models than with multitask models, and that we are not alone in our complications with MTL.

References

First quora dataset release: Question pairs.

Source code for torchnet.utils.multitaskdataloader. `https://tnt.readthedocs.io/en/latest/_modules/torchnet/utils/multitaskdataloader.html`. Accessed 1 March 2023.

Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. 2013. *SEM 2013 shared task: Semantic textual similarity. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, pages 32–43, Atlanta, Georgia, USA. Association for Computational Linguistics.

R. Cipolla, Y. Gal, and A. Kendall. 2018. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7482–7491, Los Alamitos, CA, USA. IEEE Computer Society.

Michael Crawshaw. 2020. Multi-task learning with deep neural networks: A survey. *ArXiv*, abs/2009.09796.

Ting Gong, Tyler Lee, Cory Stephenson, Venkata Renduchintala, Suchismita Padhy, Anthony Ndirango, Gokce Keskin, and Oguz H. Elibol. 2019. A comparison of loss weighting strategies for multi task learning in deep neural networks. *IEEE Access*, 7:141627–141632.

Michelle Guo, Albert Haque, De-An Huang, Serena Yeung, and Li Fei-Fei. 2018. Dynamic task prioritization for multitask learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar. 2017. Focal loss for dense object detection. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2999–3007, Los Alamitos, CA, USA. IEEE Computer Society.

S. Liu, E. Johns, and A. J. Davison. 2019. End-to-end multi-task learning with attention. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1871–1880, Los Alamitos, CA, USA. IEEE Computer Society.

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

Roweida Mohammed, Jumanah Rawashdeh, and Malak Abdullah. 2020. Machine learning with oversampling and undersampling techniques: Overview study and experimental results. In *2020 11th International Conference on Information and Communication Systems (ICICS)*, pages 243–248.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Conference on Empirical Methods in Natural Language Processing*.

Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. 2022. Defining and characterizing reward hacking. *ArXiv*, abs/2209.13085.

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.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient surgery for multi-task learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA. Curran Associates Inc.

Wen Zhang, Lingfei Deng, Lei Zhang, and Dongrui Wu. 2023. A survey on negative transfer. *IEEE/CAA Journal of Automatica Sinica*, 10(2):305–329.

A Appendix (optional)

A.0.1 Dynamic Task Prioritization Ablation

In an attempt to optimize DTP and get it out of this state of conflicting gradients, we performed ablation over the γ values. The original DTP paper (Guo et al., 2018) tested γ values of 1 and 2, while the paper they referenced (Lin et al., 2017) tested γ values of $0, 0.5, 1, 2, 5$. Due to our limited compute capability, we decided to test γ values of $0.5, 1$ and 2. We found the best γ value to be 0.5.

Dataset	$\gamma = 0.5$	$\gamma = 1$	$\gamma = 2$
SST	0.486	0.496	0.479
Quora	0.780	0.773	0.784
STS	0 0.361	0.345	0.353
Overall	0.649	0.647	0.646

Table 7: Dev Accuracy for Loss Weighting