# BERT Mastery: Explore Multitask Learning

Stanford CS224N Default Project

**Fuhu Xiao**
Department of Computer Science
Stanford University
fuhu@stanford.edu

**Chu Lin**
Department of Computer Science
Stanford University
chul1@stanford.edu

## Abstract

Bidirectional Encoder Representations from Transformers (BERT) has revolutionized the field of natural language processing (NLP). Our project aims to comprehensively study the BERT model through hands-on implementation and delve into the multitask fine-tuning technique. We successfully implemented core components of BERT, encompassing the model architecture and optimizer, and proceeded to train classifiers for three distinct downstream tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. Through a meticulously designed series of experiments, we uncovered insights into the impact of various aspects and components of the training setup on the output performance. In particular, we verified the fundamental assumption that fine-tuning improves model performance. We compared and evaluated different batching strategies in multitask fine-tuning. While multitask fine-tuning notably enhances training efficiency, our findings indicate that it does not consistently improve results across all tasks. We quantitatively described the relationship between model performance, training duration, and the number of simultaneously fine-tuned tasks, concluding that single-task fine-tuning maximizes accuracy or scores for each task, while fine-tuning more tasks together reduces the overall training time. Additionally, we identified gradient surgery as a viable solution for mitigating gradient conflict issues in multitask learning, observing its ability to either preserve or enhance training outcomes. Our latest test leaderboard submission achieves SST accuracy 0.513, paraphrase accuracy 0.797, STS correlation score 0.332, and an overall score of 0.659.

## 1 Introduction

In traditional machine learning approaches, models are typically trained at a singular task, thereby neglecting potentially beneficial insights from the training signals of akin tasks (Ruder, 2017). Multi-task learning (MTL), a concept within the machine learning domain, seeks to harness the valuable information shared among multiple correlated tasks, thereby enhancing the generalization capabilities across all tasks (Caruana, 1997). This is often implemented in deep neural networks through the sharing of certain hidden layers across different tasks, while maintaining distinct task-specific output layers. Advantages of MTL include enhanced data efficiency, minimized overfitting, and accelerated learning, as it utilizes supplementary information from related tasks (Crawshaw, 2020).

Despite its potential, multi-task learning is fraught with its own set of challenges. The heterogeneity of task objectives, data distributions, and learning rates can lead to task interference, where the model's performance on one task can detrimentally affect its performance on another. Moreover, the absence of a standardized method for balancing the influence of each task and adjusting their relative importance complicates the training process further. These issues underscore the complexity of designing effective MTL systems that can harmonize the demands of multiple tasks without compromising their individual performances.

In recent years, BERT (Birectional Encoder Representations from Transformers)Devlin et al. (2019) has become a well-adapted selection for NLP tasks, since as a pre-trained model, it provides a rich, contextualized representation of language that serves as a powerful starting point for a variety of NLP tasks. When integrated into an MTL framework, BERT has the potential to address some of the intrinsic challenges of MTL.

To further understand the mechanism behind the BERT-based multi-task model, we designed and conducted several sets of experiments.

## 2 Related Work

For the multi-task fine-tuning project based on BERT, the following is a list of related works categorized into several sub-groups.

### 2.1 Hard Parameter Sharing vs Soft Parameter Sharing

Most multi-task learning methods fall into two main catogeries: 'hard parameter sharing' and 'soft parameter sharing'. In hard parameter sharing, all tasks utilize the same hidden layers while employing distinct output layers tailored to each task. On the other hand, soft parameter sharing allows each task to have its own separate model, while the similarity among the models' parameters is promoted through regularization, encouraging the parameters across different tasks to be alike (Duong et al. (2015) use the L2 distance, Yang and Hospedales (2017) use the trace norm). As for hard parameter sharing, one approach is Stickland and Murray (2019), which is a compact multi-head attention layer that operates alongside the conventional BERT layers and is tailored to individual tasks. For our project, we use the hard parameter sharing and we didn't save separate models for different tasks.

### 2.2 Batching & Sampling

Determining the best way to group and cycle through samples from various datasets in multi-task learning represents a important role in multi-task learning problem. Round-robin sampling is the traditional way to sample the data, through which the data will sampled in circle order. However, the small datasets will be overfitted as the samples from smaller dataset will repeat for several times. There's a noval method for sample batching from Stickland and Murray (2019), through which the samples from different tasks will be sampled proportionally to the dataset sizes and then de-emphasize training set size as training proceeds.

### 2.3 Optimization in Multi-task Learning

Other studies concentrate on tackling the challenges associated with optimization in multi-task learning ((Hessel et al., 2019), (Kendall et al., 2018)). Some studies address optimization issues by adjusting the scale of task-specific gradients (Chen et al., 2018). Multiple approaches to continual learning have studied how to prevent gradient updates from adversely affecting previously-learned tasks through various forms of gradient projection ((Chaudhry et al., 2019), (Lopez-Paz and Ranzato, 2022)). Yu et al. (2020) presents a method that is independent of the model architecture and supports positive transfer during simultaneous learning of multiple tasks. It avoids the need for solving a quadratic programming (Lopez-Paz and Ranzato, 2022) problem and applies an iterative process for projecting each task's gradients instead of merely averaging or projecting only the current task's gradient.

## 3 Approach

### 3.1 BERT Model and Classifiers

Based on the skeleton code provided in the default final project handout, we first complete the implementation of the core parts of BERT model, including the embedding layer, self-attention layer, and an efficient version of Adam optimizer (Kingma and Ba, 2017), as well as a sentiment classifier which passes the BERT output into a simple linear layer with dropout to generate sentiment scores.

After verifying the implementation, we add the classifier heads for two other downstream tasks: paraphrase detection and semantic textual similarity. Cross entropy (CE) loss is used for training the sentiment classifier since it is a classification problem, mean squared error (MSE) loss for the other two since they are regression problems. We train the classifier on SST and CFIMDB datasets, with and without fine-tuning, to verify our implementation. We train for each task separately, with and without fine-tuning, to produce the baseline result.

### 3.2 Multitask Fine-tuning

Instead of training for each task separately, we let multiple tasks share the same BERT model, combine their losses, and do backward propagation for all tasks in one step. This approach is intended to make the model more versatile while using fewer parameters (Stickland and Murray, 2019). We experiment with different configurations, including the batching strategy and the numbers of tasks fine-tuned together, to better understand how this method affect training performance.

### 3.3 Gradient Surgery

Despite the expected efficiency improvement, optimization in multitask fine-tuning is known to be challenging because different tasks can result in conflicting gradients. To overcome this issue, we apply the gradient surgery technique *projecting conflicting gradients* (PCGrad) proposed by (Yu et al., 2020): before each update step, we iterate through the gradients from each task and project it to the normal plane of any gradient from other tasks that it conflicts with, and sum up the projections to get the final value of the gradient. The core idea of projecting conflicting gradients is given by Equation 1, and the full algorithm is shown in 1. We integrate an existing implementation of PCGrad (Tseng, 2020) into our code and run experiments to compare training results with and without this remedy.

$$g_i = g_i - \frac{g_i \cdot g_j}{||g_j||^2} \cdot g_j \tag{1}$$

**Algorithm 1** PCGrad Update Rule

```
Require: Model parameters $\theta$, task minibatch $B = \{T_k\}$
1: $g_k \leftarrow \nabla_\theta L_k(\theta) \quad \forall k$
2: $g_k^{PC} \leftarrow g_k \quad \forall k$
3: for $T_i \in B$ do
4:     for $T_j$ uniformly $\sim B \setminus T_i$ in random order do
5:         if $g_i^{PC} \cdot g_j < 0$ then
6:             // Subtract the projection of $g_i^{PC}$ onto $g_j$
7:             Set $g_i^{PC} = g_i^{PC} - \frac{g_i^{PC} \cdot g_j}{||g_j||^2} g_j$
8: return update $\Delta\theta = g^{PC} = \sum_i g_i^{PC}$
```

### 3.4 Training

We use a NVIDIA RTX A6000 GPU with 48GB GPU RAM for all the training and evaluation. To limit the number of variables in experiments, for each run, we set the learning rate to 1e-3 for pretrain and 1e-5 for fine-tune, use batch size 32, and run 10 epochs.

### 3.5 Dataset Batching

While fine-tuning multiple tasks, the datasets for different tasks can vary in size, so it is usually impossible to simultaneously guarantee the same number of batches and the same number of samples from each dataset per batch. We have experimented with a few methods to handle dataset batching:

**Method 1. Fix number of batches, group data proportionally, and weigh loss.** Initially, we try to make the ratio of samples from each dataset close to the ratio of the original dataset sizes. To do so, we divide the size of the largest dataset with the `batch_size` specified to get `num_batches`, and then divide the size of each dataset with `num_batches` to get `per_task_batch_size`. All the division results are rounded up to prevent zero batch sizes or number of batches. During each step, we group the batch from each dataset and compute the loss for each task, weighed by the inverse of the ratio of the corresponding data in the batch, as shown in Equation 2.

$$loss_T = \frac{loss_T \cdot \sum_{t \neq T} \texttt{per\_task\_batch\_size}_t}{\texttt{per\_task\_batch\_size}_T} \tag{2}$$

The intention of this processing step is to eliminate the impact of difference in dataset sizes on the amount of update for each task.

The issue with this method is that when there is a significant disparity in dataset sizes, like in our case (about 20x more training examples for paraphrase detection than for other two tasks), the effective batch size of smaller datasets can be too small, resulting in noisy updates. Through some preliminary testing, we found that the result was indeed not ideal and decided not to use this method.

**Method 2. Fix batch size, finish largest dataset, and cycle others.** We apply `batch_size` to each dataset to get per-task batches and assemble them into a *super batch*, whose size is (`batch_size` $\times$ `num_tasks`). After smaller datasets are exhausted, their batches are reused and grouped with unused batches of larger datasets until the largest dataset is used up.

**Method 3. Fix batch size, finish smallest dataset, and discard others.** Like in Method 2, we put equally-sized batches of each dataset into a *super batch*. However, we end the iteration after the smallest dataset is used up.

In Method 2 and 3, each dataset is randomly shuffled in every epoch before batching to make each *super batch* unique. It also makes it possible for each example in the larger datasets to be utilized, although not all of them are guaranteed to be used.

By intuition, Method 2 makes use of all the data available, but it might cause over-fitting since smaller datasets are used repeatedly. Method 3, on the other hand, uses each example at most once in each epoch, so it might be less likely to over-fit. However, it may discard a large chunk of the larger datasets and result in insufficient training for certain tasks. We have performed an experiment (described below) to compare the two methods.

## 4 Experiments

### 4.1 Data

We used the following 4 datasets throughout the project.

- **Stanford Sentiment Treebank (SST)**: This dataset contains unique phrases labeled with 5 levels of positiveness, i.e. an integer score from 1 to 5 (Socher et al., 2013), split into train set (8,544 examples), dev set (1,101 examples), and test set (2,209 examples). We use it to train the model for the **sentiment analysis task**.
- **CFIMDB**: This dataset contains highly polar movie reviews labeled as positive (1) or negative (0), split into train set (1,707 examples), dev set (245 examples), and test set (488 examples). We use it to train the model for the **sentiment analysis task** while developing the single task classifier. Note that in later experiments, we only used the SST dataset.
- **Quora Question Pair**: This dataset contains unique question pairs, split into train set (141,506 examples), dev set (20,215 examples), and test set (40,431 examples). The questions in each example are either considered duplicates (labeled 1) or not (labeled 0). We use it to train the model for the **paraphrase detection task**.
- **SemEval Semantic Textual Similarity (STS) Benchmark**: This dataset contains unique sentence pairs, split into train set (6,041 examples), dev set (864 examples), and test set (1,726 examples). Each example comes with a similarity score from 0 (not at all related) to 5 (same meaning). We use it to train the model for the **semantic similarity scoring task**.

### 4.2 Evaluation method

For **sentiment analysis** and **paraphrase detection**, we evaluate the performance of the model on the dev dataset based on accuracy, i.e. percentage of correct predictions.

For **semantic similarity scoring**, we evaluate the model performance with the Pearson product-moment correlation coefficient (PCC) between predicted similarity scores and ground truth. Here is the formula of PCC:

$$r = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2 \sum_{i=1}^{n}(Y_i - \bar{Y})^2}} \tag{3}$$

where:

- $n$ is the number of data points,
- $X_i$ and $Y_i$ are the individual data points for variables $X$ and $Y$, respectively,
- $\bar{X}$ and $\bar{Y}$ are the means of variables $X$ and $Y$, respectively.

For single-task configurations, we run the checkpoint that achieves the best dev accuracy out of all the epochs on the evaluation dataset and report the results. For multitask configurations, we sum up the accuracy or score of each task and use the checkpoint with the highest sum.

## 4.3 Experimental details

We have designed and performed 4 experiments, each focused on the impact of a specific variable in the training setup. Note that some of the results are shared between experiments for comparison in different aspects.

**Exp. 1: Pretrain vs. Fine-tune.** We train each downstream task twice, once using pretrained weights for the BERT layer (pretrain mode), and once fine-tuning the BERT layer (fine-tune mode). For each task, we use all the training data available.

**Exp. 2: Batching Multiple Datasets.** We perform multitask fine-tuning on all three tasks twice, using Method 2 and Method 3 from Section 4.4 for dataset batching, respectively.

**Exp. 3: Number of Tasks Fine-tuned Together.** We compare fine-tuning single tasks, fine-tuning each pair of tasks, and fine-tuning all three tasks together. We always use Method 3 from Section 4.4 (i.e. exhaust smallest batch and discard the rest) for dataset batching in this experiment. Note, to keep the amount of data used for each task per epoch as a controlled variable, we use the minimum number of batches computed from the smallest dataset for all other datasets, even in single-task training. For example, with batch size 32, we get 189 batches from the smallest dataset, STS, so we use 189 batches of size 32 for paraphrase detection and SST in each run as well.

Additionally, we keep track of the average duration per epoch in each training configuration.

**Exp. 4: Gradient Surgery.** We run multitask fine-tuning with and without gradient surgery for each pair of tasks. Again, we use Method 3 from Section 4.4 for dataset batching.

## 4.4 Results

According to Table 1, each single-task model has significantly better performance with fine-tuning compared to using the pre-trained model directly.

| | SST Acc. | Paraphrase Acc. | STS Corr. |
|---|---|---|---|
| Pretrain | 0.396 | 0.561 | 0.271 |
| Fine-tune | 0.520 | 0.793 | 0.396 |

Table 1: Experiment Result, Pretrain vs. Fine-tune

According to Table 2, switching from Method 2 to Method 3 marginally improved the results of SST and STS, which have smaller datasets, but significantly reduced the accuracy of paraphrase detection, which has a much larger dataset.

According to Table 3 and Figure 1, adding the number of the tasks in multitask fine-tuning consistently reduces the overall training time, but its effect on model performance in terms of accuracy

| | SST Acc. | Paraphrase Acc. | STS Corr. |
|---|---|---|---|
| Method 2 (finish largest dataset) | 0.504 | 0.778 | 0.366 |
| Method 3 (finish smallest dataset) | 0.510 | 0.662 | 0.379 |

Table 2: Experiment Result, Batching Multiple Datasets

and score varies by task. For SST, the best accuracy is seen in single-task fine-tuning, reduced accuracy when fine-tuned with another task, and the worst accuracy when fine-tuned with both other tasks. The same tendency is repeated in paraphrase detection, which performs best when fine-tuned independently and goes worse as the number of tasks increases. Notably, adding the number of tasks has a more substantial impact on paraphrase detection than on SST. By contrast, STS follows the opposite pattern, where the best performance is achieved in multitask fine-tuning.

| Fine-tuned task(s) | SST Acc. | Paraphrase Acc. | STS Corr. |
|---|---|---|---|
| SST | 0.520 | - | - |
| Paraphrase | - | 0.749 | - |
| STS | - | - | 0.364 |
| SST + STS | 0.489 | - | 0.379 |
| SST + Paraphrase | 0.501 | 0.714 | - |
| STS + Paraphrase | - | 0.688 | 0.364 |
| SST + Paraphrase + STS | 0.510 | 0.662 | 0.379 |

Table 3: Experiment Result, Number of Tasks Fine-tuned Together. Results for task(s) not used in each run are omitted.

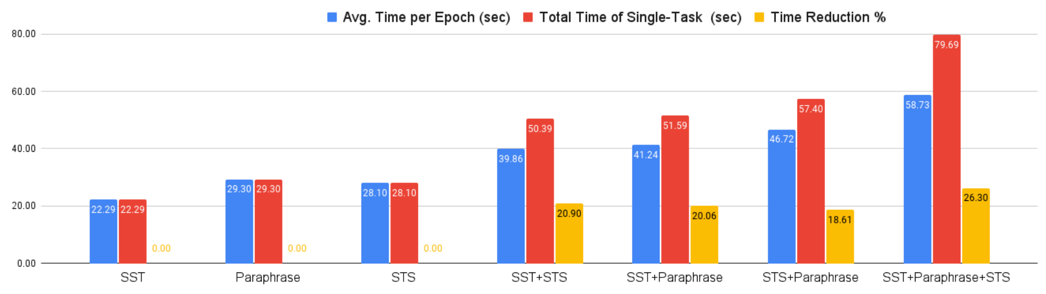

Figure 1: Experiment Result, Number of Tasks Fine-tuned Together. Training duration and improvement under each configuration.

According to Table 4, the performance of paraphrase detection is improved when gradient surgery is applied during the multitask fine-tuning with another task. There is a slight drop in SST accuracy and STS correlation score with gradient surgery, but the change is minimal. Overall, using gradient surgery results in equal or better performance in the fine-tuned models.

Based on our experimental results, among all the methods and configurations tested, **single-task fine-tuning** achieves the best accuracy or Pearson correlation scores for each individual task. Therefore, we used the individually fine-tuned model of each task to generate test set results for the learderboard. There result is shown in 5.

## 5 Analysis

The experimental results provide meaningful insights into the behavior of our BERT model and task heads under different training methods and configurations.

| | SST Acc. | Paraphrase Acc. | STS Corr. |
|---|---|---|---|
| SST + STS | 0.512 | - | 0.379 |
| SST + STS (w/ GS) | 0.510 | - | 0.374 |
| SST + Paraphrase | 0.520 | 0.714 | - |
| SST + Paraphrase (w/ GS) | 0.519 | 0.728 | - |
| STS + Paraphrase | - | 0.688 | 0.364 |
| STS + Paraphrase (w/ GS) | - | 0.707 | 0.362 |

Table 4: Experiment Result, Gradient Surgery

| SST Acc. | Paraphrase Acc. | STS Corr. | Overall Score |
|---|---|---|---|
| 0.513 | 0.797 | 0.332 | 0.659 |

Table 5: Leaderboard submission results

Exp. 1 shows that fine-tuning can largely improve model performance. This is expected because the BERT encoding from the pre-trained weights may not work ideally for all the tasks. By populating the gradient to the base model (BERT) and updating its weights, we essentially extend the learning from only the decoder to the encoder as well, making the encoding better adapted to the corresponding task. However, since fine-tuning requires more parameter updates, the training consumes more resources.

Exp. 2 illustrates the inherent trade-off between using all the available data and avoid repeating the same data. Tasks with larger datasets benefit from Method 2 because it can fully utilize all its data with minimal repetition. Meanwhile, tasks with smaller datasets are trained on the same data repeatedly, which might cause over-fitting and therefore worse performance. On the other hand, Method 3 reduces over-fitting for tasks with less data, but reduces the training data for other tasks.

Exp. 3 demonstrates that multitask fine-tuning effectively reduces training time, which matches our expectation because different tasks essentially "share" the gradient computations during the backward propagation, which is a time consuming step if executed separately. However, this technique does not uniformly improve training results. While it positively affects the STS correlation score, this improvement may be attributed to the enhanced BERT encoding facilitated by training with more diverse datasets. Conversely, the observed decreases in SST and paraphrase detection accuracy are likely due to conflicting gradients. These gradients, stemming from different task objectives, may impede learning by pulling the model parameters in opposite directions. Experiment 4 bolsters this interpretation by showing the improvements after applying gradient surgery.

Exp. 4 suggests that PCGrad makes the multitask fine-tuned model achieve the same or better performance. The difference between with and without gradient surgery is insignificant for SST and STS, which is possibly because they do not have an issue with conflicting gradients to begin with. Paraphrase accuracy has a more noticeable change likely due to its gradient conflicts with other tasks, e.g. two highly similar phrases may not be paraphrase of each other, vice versa.

After the extensive exploration, we determined that single-task fine-tuning stands out as the preferred method for maximizing accuracy or scores for individual tasks. This verdict is justifiable, given the absence of task interference when utilizing dedicated BERT layers tailored to each specific task. However, it is important to note that multitask fine-tuning offers valuable advantages, particularly in its ability to deliver comparable or superior performance for tasks such as SST and STS, all while significantly reducing training time. Furthermore, multitask fine-tuning poses a challenge in multi-dataset batching, which we have not fully addressed. The drop in paraphrase detection performance might as well to the reduced training data per epoch rather than multitask learning itself.

## 6 Conclusion

In conclusion, our project provided a comprehensive examination of the BERT model, focusing on implementing the key modules and exploring the multitask fine-tuning technique. We gained valuable insights into the nuances of training setups and their impact on performance. Our experiments confirmed the effectiveness of fine-tuning in enhancing training performance and shed light on the varying outcomes of multitask fine-tuning, highlighting its potential to improve training efficiency without consistently providing the optimal results. Our findings also demonstrated the efficacy of gradient surgery in mitigating gradient conflict issues for some, but not all tasks.

A notable limitation of our study lies in the insufficient exploration of multi-dataset batching in multitask learning, which may introduce additional variables and end up skewing our results. In order to achieve a balanced representation of each dataset within each *super batch* presents a trade-off between ensuring that each task encounters all its associated data and avoiding the repetition of data within each epoch. We acknowledge that our experiments did not adequately address this issue, which may be worth attention in future work.

Additionally, our project primarily centered on the characteristics of multitask fine-tuning rather than refining the outcomes in the final submission. Future work could explore various methods, including layer augmentation or modification, hyperparameter tuning, and ensemble learning techniques, to enhance the ultimate performance of the model.

## References

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

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. 2019. Efficient lifelong learning with a-gem.

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

Michael Crawshaw. 2020. Multi-task learning with deep neural networks: A survey.

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

Long Duong, Trevor Cohn, Steven Bird, and Paul Cook. 2015. Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 845–850, Beijing, China. Association for Computational Linguistics.

Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado van Hasselt. 2019. Multi-task deep reinforcement learning with popart. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):3796–3803.

Alex Kendall, Yarin Gal, and Roberto Cipolla. 2018. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics.

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A method for stochastic optimization.

David Lopez-Paz and Marc'Aurelio Ranzato. 2022. Gradient episodic memory for continual learning.

Sebastian Ruder. 2017. An overview of multi-task learning in deep neural networks.

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.

Asa Cooper Stickland and Iain Murray. 2019. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning.

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

Yongxin Yang and Timothy M. Hospedales. 2017. Trace norm regularised deep multi-task learning.

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