Hybrid Multitask Learning with BERT

Stanford CS224N Default Project

Anna Mattinger
Department of Computer Science
Stanford University
a.mattinger@stanford.edu

Mentor: Zhoujie Ding

Abstract

This paper explores the application of Multitask Learning (MTL) for Natural Language Processing tasks, specifically focusing on sentiment analysis (SA), paraphrase detection (PD), and semantic textual similarity (STS). Despite the potential benefits of MTL, such as improved generalization and data efficiency, it introduces complexities including gradient conflicts and increased hyperparameter tuning. To address these challenges, I implement a hybrid MTL approach, combining task-specific and shared learning mechanisms using a common BERT backbone and separate top layers for each task. My method utilizes round-robin training and individual optimizers for each task to manage task-specific learning while leveraging shared knowledge. Experimental results indicate that this hybrid approach, supplemented with gradient clipping and conservative data augmentation, enhances model performance without the overhead of more complex MTL frameworks like MTRec. Our findings suggest that a nuanced, task-aware MTL framework can effectively balance the benefits of shared and task-specific learning, providing a robust model for complex NLP tasks.

1 Introduction

Multitask learning (MTL) and single-task fine-tuning are two different approaches to training machine learning models, each with its benefits and drawbacks when compared to single-task learning (STL). While MTL raises some challenges, it has been investigated with much success (Goodwin et al. (2020) looks at MTL on twenty-one tasks). Some potential benefits to an MTL approach include:

- Shared Representations, potentially leading to better generalization and improved performance on each task, particularly when there are commonalities between the tasks. For this project, all three target tasks—sentiment analysis (SA), paraphrase detection (PD), and semantic textual similarity (STS)—involve understanding sentence semantics. PD and STS, in particular, have related goals: comparing how similar two sentences are to one another in terms of meaning (the former via a binary choice, the latter via a rating from 0 to 5). The hope is that learning these tasks together might improve the model's understanding of sentence meaning.
- Improved Data Efficiency: leveraging data from multiple tasks may prove especially useful when one of the tasks has limited data, since the shared representations can help the model perform better on the low-data tasks by using information learned from high-data tasks. Here, the STS and SemEval datasets are small relative to the Quora datasets.
- Regularization Effect: by learning multiple tasks, the model is less likely to overfit to the noise in any single task's data, making the model more robust.
- Learning Task Relationships: for instance, as aforementioned, understanding semantic similarity can directly help with paraphrase detection.

These enticements aside, MTL introduces potential pitfalls not present in STL. A few of these include:

- Gradient Conflicts: different tasks with unique loss surfaces may have conflicting gradients (Fig. 1), leading to suboptimal parameter updates. Gradient surgery methods like gradient projection (i.e., if the dot product of two gradients is negative, project one onto the normal plane of the other) can help mitigate this issue, but add complexity.
- Complexity in Hyperparameter Tuning: MTL introduces more hyperparameters, such as task-specific learning rates and loss weightings, making optimization more complex and time-consuming.
- Resource Intensive: Training on multiple tasks simultaneously can require more computational resources and memory.

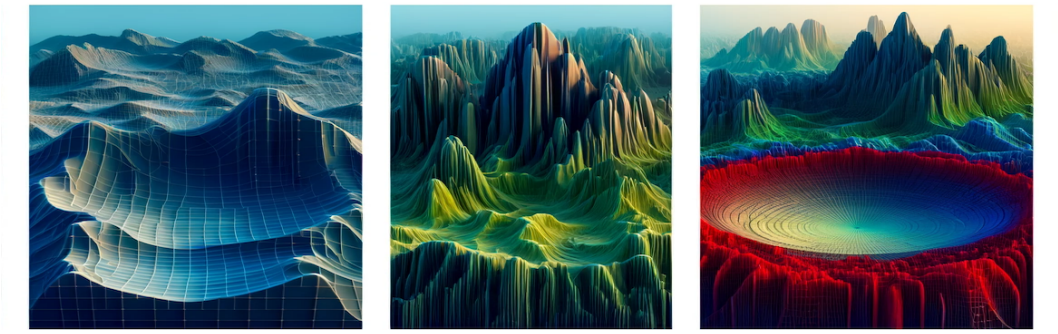

Figure 1: Conceptual visualization of gradient conflicts. Each task in an MTL training process has its own loss surface, which may conflict with finding one another's local or global optima.

2 Related Work

Gradient Surgery has been proposed by Yu et al. (2020) to manage the interference between gradients in MTL scenarios. Different tasks may have unrelated, different loss surfaces, so when training them simultaneously, this must be accounted for in order to avoid counterproductive updates. Yu's technique for reducing negative transfer between gradients is simple and elegant: take the dot product between two tasks' gradients and, if they are negative, project one onto the normal plane of the other:

$$\vec{g}_1' = \vec{g}_1 - \frac{\vec{g}_1 \cdot \vec{g}_2}{\vec{g}_2 \cdot \vec{g}_2} \vec{g}_2 \tag{1}$$

Bi et al. (2022)) explored the potential of MTL over BERT in the context of news recommendation. Their work introduces MTRec, a framework that adapts BERT for multi-task learning by optimizing shared layers across different recommendation-specific tasks. Previously, "news recommendation methods usually learn[ed] news representations solely based on news titles"; they added multi-field information through auxiliary tasks: category classification and named entity recognition (NER), and then applied a modified version of gradient surgery by creating one scaled gradient out of the auxiliary tasks, substituting into (1) accordingly (empirically, $\lambda = 0.3$):

$$\vec{g}_1 = \lambda(\vec{g}_{aux1} + \vec{g}_{aux2}) \tag{2}$$

Their approach not only enhanced the personalization of content but also addresses the challenge of sparse user interactions, a common issue in recommendation systems.

Additionally, the development of sentence embeddings using Siamese networks has been pivotal for a variety of NLP applications, particularly regarding paired sentence-level tasks (like PD and STS). Siamese networks are a type of neural network architecture that involves two or more subnetworks which share the same parameters and weights, used to compare different input samples. Reimers and Gurevych (2019) introduced Sentence-BERT, a modification of the pre-trained BERT network that employs a Siamese and triplet network structure to produce embeddings that can be effectively utilized in semantic similarity assessment.

3 Approach

3.1 Training Process

At the risk of preambling, I attempted many approaches: a fully multi-task training function inspired by MTRec (with a separate MTRec class, auxiliary tasks, etc.) and different versions of gradient surgery within my training loop; a fully single-task approach; two separate training stages and functions (one to do MTL for n epochs, and another to then do STL for m epochs). However, what I eventually landed on was a hybrid MTL version.

My `train_multitask` retains a multitask scenario for loading and handling data, as well as the MultitaskBERT model with a common BERT backbone and separate top layers for each task-specific output. This structure is typical of MTL models.

I've employed a round-robin processing of training batches among the three tasks. Alternating the tasks in this way ensures that each is trained in turn, maintaining a balance in exposure to different data types.

There are three Adam optimizers, one per task. Each of the task-specific optimizers step during their turn, whereas the shared backbone receives cumulative updates from all tasks within the same batch cycle.

Loss is calculated separately for each task depending on the active task in the current iteration, and backpropagation is performed individually. Gradient clipping has also been applied to prevent exploding gradients.

While this is fundamentally still an MTL approach, this use of multiple optimizers and task-specific batch learning adds a hybrid element to the strategy. This allows for nuanced control over how different parts of the model learn from their respective tasks, potentially optimizing the learning process for each task while still benefiting from the shared knowledge base.

3.2 Task-specific Architectures and Details

Figure 2 lays out my present neural network architectures for the three respective tasks. Previously I had incorporated many version of an MTRec class with its own layers, but this didn't improve performance.

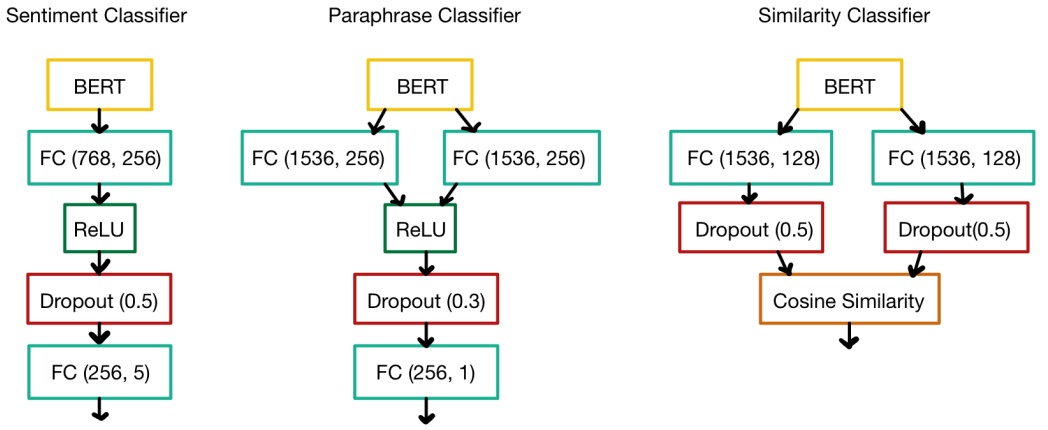

Figure 2: Current layer architecture. Previous versions included additional shared layers, task-specific additional attention heads, batch normalization, and different numbers and dimensions of fully-connected layers and dropout layers.

4 Experiments

My experimentation led to a stark pivot from my original direction; thus, my model underwent many stages.

4.1 Data

The datasets used:

1. For SC: the Stanford Sentiment Treebank (SST) dataset[1], consisting of 11,855 single sentences from movie reviews, comprising a total of 215,154 unique phrases, annotated by three human judges into five buckets (negative, somewhat negative, neutral, somewhat positive, positive) and parsed with the Stanford parser[2]
2. For PD: the Quora dataset[3] of 404,298 question pairs and a binary indicator of whether they are paraphrases of one another.
3. For STS: the SemEval STS Benchmark dataset[4], consisting of 8,628 sentence pairs and a ranking from 0 (unrelated) to 5 (functionally identical).

Additionally, preprocessing and data augmentation methods attempted (on different combinations of one, two, or all of the datasets) include:

- Parts-of-speech (POS) tagging
- Back translation
- Synonym replacement
- Random word replacement, insertion, and deletion

4.2 Evaluation method

My evaluation methods were fairly straightforward:

- **SAand PD: Accuracy**, wherein the predicted sentiment categories were compared against the true categories for SC, and the predicted binary values were compared against the true values for PD.
- **STS: Pearson Correlation Coefficient**, which involves assessing the degree of semantic similarity between two sentences, we utilize the Pearson correlation coefficient. This metric compares the cosine similarity scores predicted by our model against the human-annotated similarity scores provided in the dataset and measures the linear correlation between predicted and true scores, offering a nuanced view of how well the model's predictions align with human judgments on a continuous scale.
- **Overall performance** was computed as a modified average:

$$\frac{1}{3}\left(\{SCacc\} + \{PDacc\} + \frac{1 + \{STScorr\}}{2}\right) \tag{3}$$

All evaluation is performed under a no-gradient context (`torch.no_grad()`) to optimize computation and prevent updates to model parameters, reflecting the model's performance without interference.

The evaluation functions are designed to process data in batches, accommodating large datasets efficiently and effectively. This method also helps in handling varying sentence lengths through padding and masking, ensuring that each sentence is appropriately processed without distortion.

[1] https://nlp.stanford.edu/sentiment/treebank.html
[2] https://nlp.stanford.edu/software/lex-parser.shtml
[3] http://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs
[4] https://paperswithcode.com/dataset/sts-benchmark

4.3 Experimental details

I performed most trials on three-epoch runs, electing to run only my most promising configurations for ten epochs. My model appeared to converge slowly: often, the tenth epoch yielded my strongest results; however, running for any more than ten was invariably ineffective.

4.3.1 Hyperparameter Optimization

I experimented with a version of population-based search presented by Bai and Cheng (2024), though this proved to be intractable in my GCP virtual machine instance.

After that, I performed a grid search (see Fig. 2) on an early version of my final model (when I had fewer hyperparameters to tune), where I simply tried all combinations of three learning rates (1e-5, 2e-5, 3e-5), batch sizes (8, 16, 32), and dropout rates (0.3, 0.4, 0.5).

Grid search is inherently slow, even when each combination is only run for a few epochs, since all combinations are tried (twenty-seven, in my case).

In later versions of my model, it was no longer a viable approach—I had about eight tunable learning rates, alone—at which point I switched to a guess-and-check, leveraging what I had learned from earlier experimentation.

Part of why I conducted this grid search was to see whether my VM could tolerate larger batches without the process getting killed, in which case I could eventually train over more epochs with a larger batch. However, a batch size of 8 yielded the best results overall.

4.3.2 Architecture Variations

Initially, I was a bit aggressive in trying to incorporate several strategies: SMART (Smoothness-inducing Adversarial Regularization) with Bregman Proximal Point Optimization, per Jiang et al. (2019); MTRec/Gradient Surgery; adding three different types of learning rate schedulers (e.g., cyclic learning rates, or warmup/decay periods); separate phases for MTL and STL (complete with completely separate train/test functions and class definitions, with epochs devoted to each).

Eventually, I scrapped them all since they were not producing desirable results, or were computationally infeasible and waylaid me to time-consuming devOps troubleshooting on GCP.

Instead, I set my sights on simpler enhancements that showed immediate improvements: the single-task batched training, running different configurations (and numbers of) optimizers, hyperparameter tuning, and adding, removing, and modifying smaller and well-motivated adjustments (batch normalization, gradient clipping, weight decay).

The task-specific neural net architectures I settled on are outlined in Fig. 2.

I settled on three Adam optimizers, one per task. Previous versions involved either one optimizer (for my initial fully multi-task approach) or four (three task-specific, one shared). In the case of the four optimizers, I tried versions where all four optimizers updated the shared BERT backbone parameters, as well as versions that were fully siloed, where task-specific optimizers only updated parameters of their respective task while the shared optimizer handled the backbone.

4.4 Results

After initial attempts at MTRec-inspired MTL, pivoting to a hybrid MTL approach proved most effective.

Specifically, immediate improvements were seen with the following changes:

- Single-task batched training runs with gradient clipping.
- Four Adam optimizers: three task-specific Adam optimizers and one shared, with no overlap in the parameters associated with each.
- Using more conservative parameters on the shared optimizer: lower learning rate, higher weight decay. See Table 1 for hyperparameters.

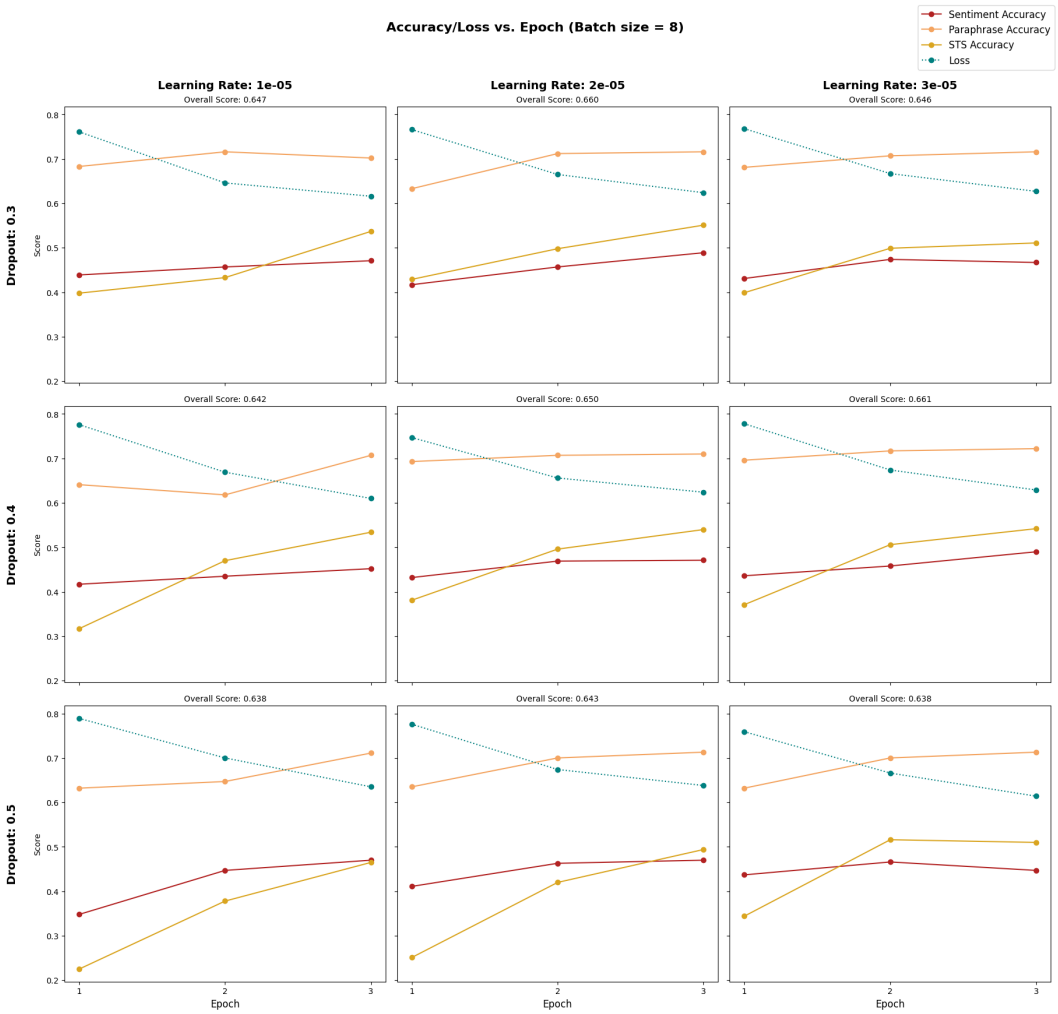

Figure 3: Hyperparameter grid search example for batch size 8.

Task	Dropout	Learning Rate	Weight Decay
Sentiment Classification	0.5	2e-5	1e-2
Paraphrase Detection	0.3	3e-5	1e-2
Sementic Textual Similarity	0.5	2e-5	2e-2
BERT backbone	0.4	1e-5	2e-2

Table 1: Best task-specific hyperparameters.

- Conservative data augmentation on the SST and SemEval datasets, using only synonym replacement via WordNet. Back translation's debatable merits did not justify its overhead runtime.
- A Siamese network for the STS task, and directly computing cosine similarity as a logit (as opposed to rescaling SemEval labels for CosineEmbeddingLoss).

However, even during the last several days of this project, I had a persistent issue I was unable to get around: the tendency to overfit to the PD task, and the tendency to underfit to the STS task. I elaborate upon this in the next section.

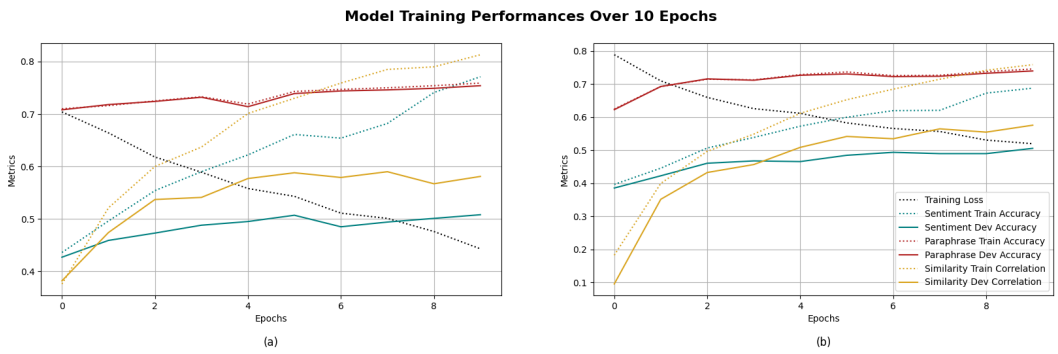

Figure 4: 10-epoch runs, with (b) being my best at the time of writing. Note the probable overfitting for SA and STS, and relative underfitting for PD (given the top leaderboard metrics for this task).

5 Analysis

After many initial attempts at MTRec-inspired multitask learning, pivoting to a hybrid MTL approach proved most effective.

Specifically, creating single-task batched training runs, instead of training all three tasks simultaneously, showed an immediate improvement. Furthermore, siloing the optimizers into three task-specific optimizers *and* a shared optimizer—and only updating shared parameters with the shared optimizer—resulted in further improvement.

Using more conservative parameters on the shared optimizer (lower learning rate, higher weight decay) worked best. This makes intuitive sense: in the event of task conflicts, the shared optimizer shouldn't be overriding the task-specific optimizers through aggressive updates. My overall takeaway was that, in my hybrid approach, to defer to STL and leverage MTL as a supplementary enhancement.

Regarding my underfitting and overfitting issues, my thought is that this is partly due to the discrepancy in dataset sizes. This is part of why I elected to augment the SemEval (and SST) sets, but not the Quora set—though it was also important to be conservative with my data augmentation. I believe the reason for this is that employing too many augmentation techniques (especially ones that don't guarantee the preservation of sentence meaning, and may not fit the labels of their originating sentence) adds unhelpful noise.

Other mitigation strategies largely revolved around changing the dimensionality of fully-connected (FC) layers, the number of FC layers, introducing nonlinearity through ReLU, adding (and removing) batch normalization, reducing or increasing dropout, learning rates, and weight decay.

Interestingly, the changes I made sometimes caused very unexpected effects; e.g., increasing the dropout rate or number of dropout layers to combat overfitting sometimes appeared to make the discrepancy between my train and dev STS correlation worse, as did reducing the dimensionality of my STS linear layers to hedge against the model memorizing the training data. Likewise, adding deeper layers (FC and Dropout) and batch normalization to PD did not yield notable improvement. Further exploration is needed to determine some of the causality behind this behavior.

6 Conclusion

While not my original intended direction, I found that a hybrid learning approach leveraged some of the strengths of both MTL and STL.

Additionally, my best performance consistently arose from my simpler enhancements: e.g., direct cosine similarity computation instead of the use of CosineEmbeddingLoss (which required rescaling the SemEval scores). Overcomplicating my task architectures with too many individually-tuned linear, dropout, or other layers, or with multiple granular training and testing functions, proffered little to no improvement.

My takeaway from the above is that, though there are powerful tools out there, if they aren't already tailored to a given application, trying to shoehorn them in is often a wasted effort compared to taking a more straightforward approach and incremental changes.

Revisiting my original idea of leveraging an MTRec-like approach (auxiliary tasks and gradient surgery) may be worthwhile, with better customization to my specific data and target tasks.

Unfortunately, my hybrid learning approach was only developed late into the project. It begs further exploration and optimization (for instance, more exhaustive hyperparameter tuning on my latest model).

Another possible avenue of exploration: fully separated two-phase learning using MTL/hybrid, and then STL, complete with separate class definitions and train/test functions.

7 Ethics Statement

We must acknowledge the potential ethical implications of deploying NLP models, particularly concerning fairness, accountability, and transparency. While our models aim to improve NLP system accuracy and efficiency, they may inadvertently perpetuate biases present in the training data.

- Fairness: NLP models can unintentionally propagate biases found in training datasets, leading to unfair treatment of certain groups. For example, SA models might misinterpret language from specific dialects or sociolects, while PD and STS models could exhibit biases in understanding diverse cultural contexts.
- Accountability: The deployment of NLP models carries a responsibility to ensure their outputs are reliable and ethical. If something goes wrong with an LLM, ascribing fault is both ethically and legally complex.
- Transparency: Openness about the development and functioning of our NLP models is crucial. The design, training, and evaluation processes should be well-documented and accessible to both the research community and the public. This transparency helps build trust and allows others to replicate or improve upon our work. Moreover, clear communication about the intended use cases and potential risks associated with our models is essential.

Mitigation Strategies: To actively address these ethical concerns, we implement several strategies:

- Bias Audits: Examining false positive and false negative rates across various subgroups. If biased outputs are detected, we can adjust our training processes and data (e.g., through augmentation with contrasting data) to minimize them.
- Transparency Norms: There should be transparent documentation of our models' limitations and potential biases, allowing users and stakeholders to make informed decisions. This includes detailing the datasets used, the training processes, and any detected biases or errors in model predictions.
- Cautious Deployment: The deployment of these models in real-world applications should not be taken lightly—once Pandora's box is opened, it can't be closed. Extensive beta testing is crucial to observe the models' behavior in diverse, real-world scenarios *before* full deployment. This phase allows for the collection of user feedback, identification of potential issues, and further refinement of the models when its reach is limited. Only after demonstrating consistent and equitable performance should the models be widely deployed.
- User Feedback: Incorporate mechanisms for users to provide feedback on biased or incorrect outputs, which can be used to further refine and improve the model.
- Continuous Monitoring: Establish a system for continuous monitoring of the models' performance and biases in real-world applications, ensuring that they remain fair and effective over time. Third-party audits should also be encouraged.

References

Hui Bai and Ran Cheng. 2024. Generalized population-based training for hyperparameter optimization in reinforcement learning.

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

Travis Goodwin, Max Savery, and Dina Demner-Fushman. 2020. Towards zero-shot conditional summarization with adaptive multi-task fine-tuning. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. Association for Computational Linguistics.

Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2019. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. *arXiv preprint arXiv*.

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

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient surgery for multi-task learning. *Advances in Neural Information Processing Systems*, 3:5824–5836.