SuperBERT: Multi-task Finetuning with Domain Adaptation

Stanford CS224N Default Project

Mohamed Owda
Department of Computer Science
Stanford University
mohamed8@stanford.edu

Mohamed Osman
Department of Computer Science
Stanford University
laalays@stanford.edu

Abstract

BERT is a transformer-based model that pretrains on unsupervised tasks to learn weights that generate rich word representations from input tokens. We aimed to improve BERT's performance on paraphrase detection, semantic similarity, and semantic analysis tasks in a model called SuperBERT. To do so, we pretrained BERT using masked language modeling on datasets relevant to the three domains. We then used different multi-task learning procedures utilizing gradient surgery on top of the additional pretraining. We found that multi-task learning with gradient surgery leads to significant improvements over the baseline on all of the three tasks. We also found using mean pooling on the token embeddings with cosine similarity as similarity metric further improves the model performance on the semantic similarity task. We finally found additional pretraining on the target domain datasets improved the performance on semantic similarity but did not lead to any improvements on paraphrase detection and sentiment analysis. Thus, we conclude that multi-task finetuning on the three downstream tasks leads to performance on each task that is comparable to a model that finetunes only on the specific task, even with significantly less data.

1 Key Information to include

- No external collaborators
- No external mentors
- Not sharing project with any class

2 Introduction

In 2018, researchers from Google published the Bidirectional Encoder Representation from Transformer (BERT) [1], a model built upon 12 Encoder Transformer Layers. In BERT, sentence inputs are tokenized into word pieces and passed to an embedding layer. For each transformer layer, the input is passed through a multi-head attention sub-layer and a feed-forward neural network (with additive and normalization layers with residual connections following each). BERT then outputs [CLS] token embedding as well as the last hidden state embeddings from each word piece of the input sentence (referred to as the "LHS embeddings" for the rest of the paper). BERT is trained on two unsupervised learning tasks, masked language modeling (MLM) and next sentence prediction (NSP), from large general domain corpora. BERT represented a jump forward in the NLP space in the search for models that can generate contextual embeddings. The hope is that these learned model weights demonstrate a rich semantic understanding of language as a whole, which we can then further finetune on a specific task/tasks using a relatively smaller training set compared to the pretrained BERT tasks, to then perform at a high level on a specific downstream task.

Thus, we tackle the problem of improving BERT's performance across three downstream NLP tasks: paraphrase detection (**PD**), semantic textual similarity (**SS**), and sentiment analysis (**SA**) tasks (these three tasks are collectively referred to as the "downstream tasks") in a new model called SuperBERT. Paraphrase detection is a binary classification task of assessing if two sentences are paraphrase of

each other. Semantic textual similarity is a task where two sentences are rated for their similarity on a scale of 5 (same meaning) to 0 (does not share the same meaning at all). Sentiment analysis is the task where a sentences sentiment is classified on an integer scale of 0 (negative) to 4 (positive). This represents a challenge as a crafting robust embeddings that can perform well across the three tasks requires a pretraining and training process that is able to learn robust representations such that we do not degrade the performance of any one task to improve performance on another task.

To accomplish this goal, we first establishe a baseline BERT model. We trained three different models each finetuned on one specific task and calculated the model performance on the three tasks. As the results in Table 2 shows, the model that is finetuned on the semantic textual similarity task achieves the highest overall accuracy. Thus we use this model as our baseline. From this, we consider a new model called Vanilla Gradient Surgery (**VGS**) where we apply multi-task finetuning [2] across the three downstream tasks with the model using a linear layer to project the CLS embeddings from each task. For paraphrase detection and semantic similarity tasks, we average the two CLS embeddings from the two input sentences. We found **VGS** outperforms the baseline model. We compare the results of **VGS** to a model called **CGS** (Cosine Gradient Surgery) where we perform mean pool on the token embeddings to create LHS embeddings for the input sentences. In addition, we use cosine similarity as similirity metric for semantic textual similarity. We found **CGS** outperforms **VGS** particularly on the semantic textual similarity task. We finally pretrain the **CGS** model on target-domain data related to the three downstream tasks using MLM in a final model called **SuperBERT** which is our best performing model.

3 Related Work

The improvement in training NLP systems from embeddings created from pretrained tasks rather than using embeddings learned on a task from a randomized initial state is supported by [3] [4]. Sun et. al [5] found that using further pretraining on target domain data in conjuction with multi-task learning using BERT embeddings on text classifcation tasks leads to state-of-the-art results. The use of pretraining language models has seen growing prominence in the NLP community, with Devlin et. al [1] showing BERT's improvement through pretraining on the Masked Language Modeling (MLM) task, where random words are masked and the model attempts to predict the word, and Next Sentence Prediction (NSP). While SuperBERT will be applied to different downstream tasks compared to those in [5] , SuperBERT will also make use pretraining on additional target domain data related to the relevant downstream tasks, as well as doing multi-task learning to further finetune performance on the three downstream tasks. Similar to the original BERT model, SuperBERT will pretrain on the MLM task. However, SuperBERT does not pretrain on the next sentence prediction (NSP) task as this was found not to improve performance by a significant degree [6].

Bi et. al [7] similarly used multi-task learning over BERT embeddings on the downstream tasks of category classification and named entity recognition, and found significant improvements over baseline BERT models. Bi et. al [7] also noted the difficulties with multi-task learning, and this phenomenon has been well noted in the literature, as the model attempts to minimize loss for its parameters across the multi-task optimization landscape. Parisotto et. al [8] note how updates that lead us away from optimal areas of the landscape can result in a model performing poorly across all three tasks compared to if we simply trained on the tasks individually. In response, Yu. et. al [9] identified the co-occurrence of 3 conditions (which they call the **"tragic triad"**) that results in the multi-task model optimizer making inadequate or even harmful gradient updates across the multi-task optimization landscape. In turn, they recommended a novel gradient surgery technique called PCGrad to remedy harmful gradient updates by resolving conflicting gradients through projecting the gradients on the normal plane of one another. They found significant gains in model performance and optimization speed as a result. SuperBERT makes use of an implementation inspired from PCGrad, although our implementation is in PyTorch and is further optimized for efficiency by removing the process that differentiated between updating gradients for shared parameters, which we found empirically did not make a difference.

Reimers and Gurevych [10] showed that sentence embeddings trained on cosine-similarity loss outperformed state-of-the-art models on semantic textual similarity tasks and transfer learning tasks. SuperBERT also makes use of cosine-similarity loss for the semantic textual similarity task during multi-task learning to improve performance.

4 Approach

We experimented with a number of models to get our proposed model, SuperBERT. For all the models in the project we utilize the Adam optimizer [11]. The baseline model is BERT model finetuned on STS as described in the introduction section. We use three different loss functions when finetuning the models for the three tasks. For paraphrase detection (PD), we use binary cross entropy loss between the predictions and the labels since PD is a binary classification task. For sentiment analysis (SA), we use cross entropy loss between the predictions and the labels. For semantic similarity (SS), we use mean-squared error loss since SS dataset has a continuous label ranging from $0 - 5$.

As our models aims to improve BERT's performance on all the three downstream tasks, we use multi-task learning technique when training the BERT model. In multi-task learning, the loss across all three downstream tasks is considered during the training. The most naive approach is to add all the losses from each task during the training. Thus the total loss is calculated as:

$$L = L_{sa} + L_{para} + L_{similarity} \tag{1}$$

where L_{sa} is the cross entropy loss on the SA task, L_{para} is binary cross entropy loss on the PD task, and $L_{similarity}$ is the mean-squared error loss on the SS task. This is in contrast to the baseline model which is being trained to only minimize the loss of the task corresponding to its training set.

However, as mentioned from Yu et. al [9], multi-task learning can lead to harmful gradient updates due to certain conditions as it relates to the gradients themselves, and the optimization landscape. Thus we cannot naively sum the losses as in equation (1). To remedy this, we implement our multi-task learning with the gradient surgery from [9] called PCGrad. In PCGrad, if two gradients are conflicting, the gradient of each task is updated by projecting the gradient onto the normal plane of the gradient of the other task. This maintains much of the gradients original information while eliminating the influence of components that are conflicting, in turn making the two gradients no longer be conflicting.

if $g_i \cdot g_j < 0$ **then**
  $g_i \leftarrow g_i - \frac{g_i \cdot g_j}{||g_j||^2} g_j$
end if

Our implementation of PCGrad is inspired from Yu et. al and the PyTorch implementation [12]. However, we reimplement the algorithm in PyTorch, integrating inheritance qualities from object-oriented-programming to more seamlessly have PCGrad work on top of the Adam optimizer we implemented. Furthermore, we changed how the algorithm reacted to gradient updates across gradient parameters that were not populated across each task in a given iteration and also changed how we flattened and unflattened PyTorch tensors. This empirically resulted in a significant speed improvements while accuracy remained the same. We use PCGrad implementation in Vanilla Gradient Surgery (VGS) model to simultaneously finetune BERT weights on all the three downstream tasks. VGS model projects the CLS embeddings (average of CLS tokens from two sentences for paraphrase detection and semantic similarity) from BERT onto a linear layer. The models then uses the logits from the linear layer to get the loss for each task.

One potential issue from VGS is that the architecture projects the BERT CLS token outputs (or an average of the CLS tokens for PD and SS) through a linear layer for all the tasks. Though this architecture, and specifically the use of CLS tokens, is likely suitable for the sentiment classification, it may not work well with PD and SS as these tasks require comparisons between two inputs. For this reason, we consider a new model called Cosine Gradient Surgery (CGS). While we continue to use the same multi-task finetuning procedure from VGS, we change how we generate logits for each task. Namely, we directly get the token embeddings from BERT and perform a mean pooling operation to produce a single vector encoding that represents the sentence embedding. We then use this sentence embedding in each of the downstream tasks. For the sentence similarity, we calculate the cosine similarity between the two sentence embeddings. For paraphrase detection, we concatenate the two embeddings and project the concatenated embedding to a linear layer. For the sentiment classification, we keep the same architecture of directly projecting the embedding to a linear layer.

Finally, we observe that the original BERT model is pretrained using the unsupervised task of Masked Language Modeling (MLM) [1] on Wikipedia articles and book collections which represents a general domain data. Thus, following the findings of Sun et al. [5], we pretrain the CGS model on the

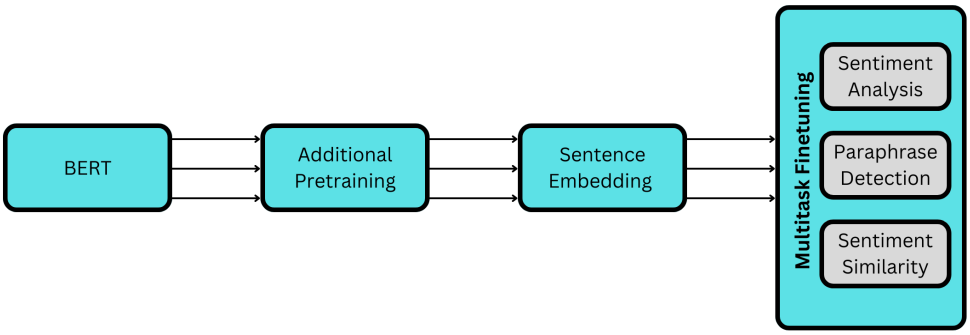

Figure 1: SuperBERT architecture

training and dev datasets for all the three downstream tasks to get further improvements from domain adaptation. As Liu et al. [6] found no significant improvements from next sentence prediction, we do not pretrain the model with next sentence prediction. When pretraining the model on next word prediction task, we follow the same algorithm of randomly masking 15% of the tokens as used in the orignal BERT paper. Finally, for the datasets that have short sentences, we concatenate two sentences to get a longer sentence as pretraining on longer sentences is shown to give better results [6]. The additionally pretrained CGS is our proposed model. We call this model SuperBERT. The architecture of this model is shown in Figure 1.

5 Experiments

5.1 Data

1. **Stanford Sentiment Treebank (SST)**: a single sentence extracted from movie reviews with 5 different labels ranging from negative to positive.

2. **CFIMDB dataset**: highly polar movie reviews with binary labels for positive or negative.

3. **Quora Question Pairs (QQP)**: question pairs with binary labels of whether they are paraphrase of each other or not.

4. **SemEval STS Benchmark (STS)**: pairs of sentences with continuous varying similarities ranging from 0 (unrelated) to 5 (equivalent meaning).

Table 1 shows the datasets used in the project. One potential issue with the original datasets breakdown is that the STS, QQP, and SST training sets are all different sizes. If one dataset is longer than the other datasets, this could lead issues when performing the multi-task finetuning. When the smaller datasets are processed during training, all the gradient updates will be coming from the longer dataset which may lead to a model that performs well on the task with the longer dataset and performs poorly on the other tasks. In order to avoid this issue, we decide to truncate the datasets to being 6,016 samples. We sample the examples from the a-prior distributions of the original datasets as we want our new training sets to be as close to the original training sets in their distribution. $6,016$ training size is the maximum number of examples we can take from STS, the smallest dataset of the three training sets, such that it is evenly divisible by a batch size of 32 which is ideal for us to better follow training behavior in order to problem shoot functionality or performance issues that arise during multi-task learning. When pretraining the BERT weights on the next word prediction, we use 6,016 examples from each dataset for similar reasons. In addition, when training the baseline model, we use the original STS dataset since the model is being finetuned for one task. We use CFIMDB data only for pretraining the model. This dataset has longer sentences, and thus is a great fit for domain adaptation. We do not use CFIMDB for finetuning SA task because it has binary label while the SA test data has 5 labels. Finally, we keep using the dev datasets from the original datasets since the dev sets do not contribute to the gradient update.

Dataset	Train Size (Baseline)	Train Size (Extensions)	Dev Size	Test Size	Associated Task	Evaluation Metric
SST	$8,544$	6,016	$1,101$	$2,210$	Sentiment Analysis	Accuracy
CFIMDB	$1,701$	1,701	2453	488	Sentiment Analysis	Accuracy
QQP	$141,506$	6,016	$20,215$	$40,431$	Paraphrase Detection	Accuracy
STS	$6,041$	6016	864	$1,726$	Semantic Textual Similarity	Pearson correlation

Table 1: Datasets breakdown and the evaluation metric for each task

5.2 Evaluation method

We use accuracy as evaluation metric for sentiment analysis and paraphrase detection as shown in Table 1. We use Pearson correlation to evaluate the semantic text similarity. We sum the three scores from each task and take the mean to get the overall performance of the model.

5.3 Experimental Details

1. **SST-Finetune**: finetunes BERT weights on SST dataset for sentiment analysis task.
2. **QQP-Finetune**: finetunes BERT weights on QQP dataset for paraphrase detection task.
3. **STS-Finetune** (**Baseline model**): finetunes BERT weights on STS dataset for semantic similarity task.
4. **VGS**: Trained on STS, SST, and QQP with multi-task finetuning with gradient surgery using CLS tokens. More information in the Approach section.
5. **CGS**: Trained on STS, SST, and QQP with multi-task finetuning with gradient surgery using LHS embeddings. More information in the Approach section.
6. **SuperBERT**: Trained on STS, SST, and QQP using multi-task finetuning that incorporates gradient surgery with LHS embeddings. Pretrained on CFIMDB, STS, SST, QQP. More information in Approach.

We used the following configurations in all our experiments: $Optimizer = Adam$, $learning\,rate = 1e-5$, $batch\,size = 32$, $number\,of\,epochs = 10$, $hidden\_layer\_dropout\_rate = 0.5$.
When pretraining the BERT weights on the target domain data, we use batch size of 8 since we are loading long sentences.

5.4 Results

Model	Sentiment Analysis	Paraphrase Detection	Semantic Similarity	Overall
SST-Finetune	0.532	0.491	-0.078	0.315
QQP-Finetune	0.248	0.791	-0.034	0.335
STS-Finetune	0.243	0.623	0.382	0.416
VGS	0.513	0.704	0.378	0.532
CGS	0.514	0.722	0.682	0.639
SuperBERT	0.513	0.723	0.790	0.675

Table 2: Results of all models on the dev set

Model	Sentiment Analysis	Paraphrase Detection	Semantic Similarity	Overall
SuperBERT	0.511	0.722	0.767	0.667

Table 3: SuperBERT performance on the official test set

Table 2 shows the performances of all the different models on the dev set. The first three models were finetuned for one specific task. As expected, each model outperforms the other two models on the

task it was finetuned for. STS-Finetune (baseline model) achieves the highest overall performance out of the three models. We discuss in the analysis section why it achieves the highest overall accuracy.

Vanilla Gradient Surgery (VGS) leads to 0.116 improvement from the baseline model. All the improvements comes from the VGS model preserving the improvements from the finetuning of the individual models. If we sum the results from the single-fined models (SST-Finetune, QQP-Finetune, STS-Finetune) on their respective tasks, we get $\frac{0.532+0.791+0.382}{3} = 0.568$ overall accuracy. This is only 0.036 higher than the overall accuracy of the VGS model. This demonstrates the multi-task finetuning with gradient surgery does better gradient updates that leads to robust embeddings which performs well for all the downstream tasks.

CGS model improves on the VGS model by performing mean pooling on the token embeddings to create sentences embeddings. It also uses cosine similarity as a similarity metric for sentence textual similarity. CGS model does not lead to any notable improvements on sentiment analysis and paraphrase detection tasks. It in fact losses 0.001 on sentiment analysis and gains only 0.018 on paraphrase detection. This is expected as the architecture did not differ much between VGS and CGS for the paraphrase detection and sentiment analysis tasks. However, CGS leads to 0.304 improvement from VGS model on semantic similarity which translates to 0.107 overall improvement. This shows cosine similarity is much better similarity metric than just averaging the CLS embeddings of the two sentences.

Finally, SuperBERT builds on the success of CGS. The only difference between CGS and SuperBERT architecture is that SuperBERT pretrains on target domain data. While the paraphrase detection and sentiment analysis scores remain roughly the same, the semantic similarity score increases by 0.108 from CGS which leads SuperBERT achieving 0.675 overall score. SuperBERT achieves 0.259 improvement from the baseline. It outperforms STS-Finetune model on semantic similarity. However, it does not outperform the other single finetuned models (SST-Finetune and QQP-Finetune) on their respective tasks. One reason why we likely don't see an improvment from QQP-Finetune to SuperBERT with respect to paraphrase detection is that SuperBERT only trains on 6,016 examples, while QQP-Finetune trains on the entire QQP training set. We finally compare SuperBERT performance on the dev set and the official test set. Its performance on the official test data set only decreases by 0.008 which gives the confidence that our models did not overfit the training and dev data sets.

6 Analysis

We observe STS-Finetune performs relatively well on sentiment classification and paraphrase detection which makes it achieve a higher overall score than the other two single finetuned models, namely SST-Finetune and QQP-Finetune. We now analyse why SST-Finetune outperformed the other two models. We believe this due to the nature of the other two tasks and our evaluation method, but not due to SST-Finetuning creating embeddings that generalize well with other two tasks. Both sentiment analysis and paraphrase detection are classification tasks with finite number of labels. Sentiment analysis task has five classes, and paraphrase detection is a binary task. Assuming we have a dataset that balances among the classes, a model that randomly guesses the output class is expected to achieve 20% and 50% accuracy on sentiment classification and paraphrase detection respectively. STS-Finetune performance (0.243 and 0.623 accuracy on SA and PD respectively) is around this range. On the other hand, sentence similarity task has continuous label that ranges from 0 to 5. It is much harder for a model to randomly guess the correct similarity score. For this reason, SST-Finetune and QPP-Finetune perform poorly on the sentiment similarity task. In addition, our evaluation methods does not take into account the nature of the tasks and equally weights the performance on each task when evaluating the overall score. For those two reasons, STS-Finetune achieves the highest overall score out of the three singlefined models.

We also notice that none of our models, including the proposed model, performs well on sentiment classification. SST-Finetune achieves the best accuracy of 0.532 which is not that much higher than the scores VGS, CGS, and SuperBERT achieve. This is expected as the architecture for predicting sentiment does not differ among those models. However, we also believe that sentiment classification is a hard task in nature. Figure 2 shows the confusion matrix of SuperBERT on sentiment analysis.

We notice that the model never confuses between the extremes. For instance, SuperBERT model never predicts positive when the ground truth is negative. However, the model confuses the close sentiment classes like positive and somewhat positive. Though a different architecture may help the model, we still believe this will be a challenge for any model as even the human evaluation differs.

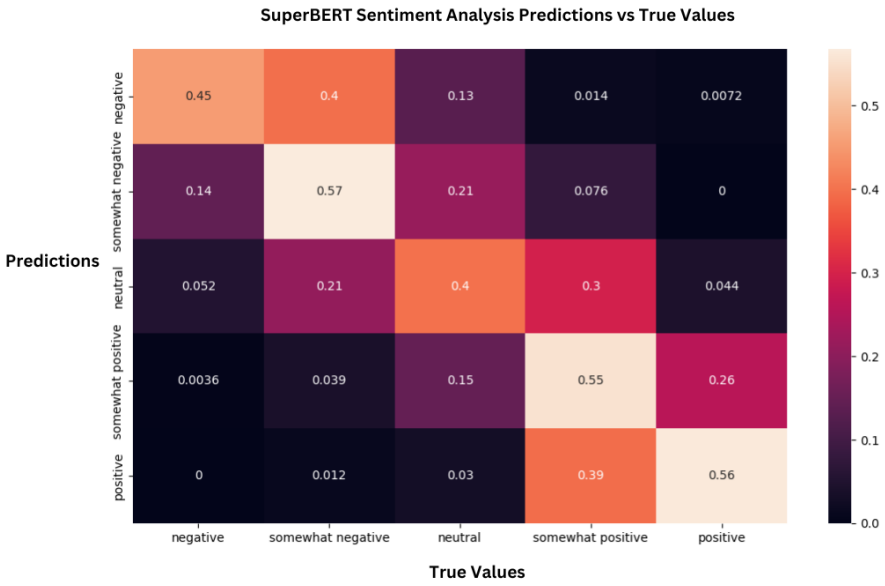

Figure 2: SuperBERT confusion matrix for sentiment analysis

We now analyses how pretraining the BERT weight on the target domain helps SuperBERT. To check if SuperBERT learned the target domain data, we compare the next word predictions of the baseline model and SuperBERT model.

1. *This is a great [MASK].* STS-Finetune: *This is a great [idea].* SuperBERT: *This is a great [film].*
2. *This movie was [MASK].* STS-Finetune top five predictions:[*cool, awesome, something, good, one*] SuperBERT top five predictions: [*great, good, amazing,* ***terrible****, terrific*]

Those examples demonstrate that SuperBERT adapted the domain data. This is also reflected in its increase in overall score. SuperBERT's overall score increases from 0.639 in CGS to 0.675. But almost all the improvement is from the semantic similarity task. One possible reason for the lack of improvement on the paraphrase detection and sentiment analysis tasks is that those tasks are classification tasks where we either get the prediction right or get it wrong. Since BERT has huge number of weights, we believe our pretraining on the relatively small data set did not perturb those weights to make a different prediction. However, for semantic similarity, even a slight perturbation that moves the BERT weights to the true value will show up in Pearson coefficient calculation, thus increasing the model's score on the task.

7 Conclusion

In our paper, we explored ways to improve BERT weights to create sentence embeddings that can perform well across a wide range of downstream tasks. We found that changing the BERT architecture to utilize the LHS embeddings and utilizing cosine simialrity to create logits for predicting the semantic similarity, along with multi-task finetuning using gradient surgery and additional pretraining all contributed to improvements in our model, allowing us to see an overall average score of 0.675 which is a 25.9 percentage point increase over baseline. We also found that pretraining did not help with paraphrase detection or sentiment analysis, but it did improve semantic similarity. While

these results are promising, there are shortcomings in our work which serve as starting points for future work. To start, we observe that sentiment analysis performs the worst across all three tasks. To remedy this, we would like to experiment with adding more layers to our model to improve performance on this task. Furthermore, while we did a significant architecture change for getting logits from the semantic similarity task that resulted in a large improvement, we did not do a similar large architecture change for paraphrase detection. Future work could go into how best to generate logits for paraphrase detection that reflect the nature of the task.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[2] Rich Caruana. Multitask learning: A knowledge-based source of inductive bias. In *Proceedings of the Tenth International Conference on International Conference on Machine Learning*, ICML'93, page 41–48, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[3] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, page 3111–3119, Red Hook, NY, USA, 2013. Curran Associates Inc.

[4] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

[5] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification?, 2020.

[6] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

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

[8] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning, 2015.

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

[10] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks, 2019.

[11] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[12] Wei-Cheng Tseng. Weichengtseng/pytorch-pcgrad, 2020.