Multi-Task Learning for Robust Contextualized Sentence Embedding Generation

Stanford CS224N Default Project

Santino Ramos
Department of Computer Science
Stanford University
santino@stanford.edu

Yash Dalmia
Department of Computer Science
Stanford University
ydalmia@stanford.edu

Abstract

Robust contextualized sentence embeddings are essential for natural language processing (NLP) tasks because they enable machines to analyze and process human language in a mathematical format. However, pre-trained language models that generate embeddings optimized for specific language tasks in isolation tend to yield weaker results when applied to multiple tasks simultaneously. Multi-task learning aims to improve performance on multiple tasks by learning a shared representation that can generalize to new tasks, but it is challenging because the model needs to balance the objectives of multiple tasks and avoid interference between them. In this project, we generated baseline results for a sentence classification task on the Stanford Sentiment Treebank (SST) and CFIMDB dataset using pre-trained weights loaded into a BERT architecture. We then examined different strategies for fine-tuning and adjusting contextual BERT embeddings to simultaneously perform well on multiple sentence-level tasks, including Sentiment Analysis, Paraphrase Detection, and Semantic Textual Similarity. After evaluating several techniques inspired by recent research papers for fine-tuning and extending the BERT model, we introduced cosine-similarity fine-tuning, an additive loss function (of equally weighted task-specific losses) for multi-task fine-tuning, and gradient surgery. In addition, we experimented with custom prediction heads to better capture the learnable nuances of each task. These contributions resulted in a significant improvement over baseline, yielding competitive leaderboard scores (top 30% of submissions as of March 17th) of 0.501, 0.778, and 0.662 on the SST, Paraphrase Detection, and STS tasks, respectively.

1 Key Information to include

- Mentor: Xiaoyuan Ni
- External Collaborators (if you have any): N/A
- Sharing project: N/A

2 Introduction

The pervasiveness of language models and natural language processing (NLP) across academic research and industry highlights the significance of robust and generalizable sentence embeddings. Sentence embeddings are mathematical representations of sentences that capture their semantic meaning and are usually produced by pre-trained language models. These embeddings are critical for NLP tasks as they enable machines to effectively and efficiently analyze and process human language. However, although language models have improved at generating embeddings for specific tasks, their performance deteriorates when applied to multiple tasks simultaneously.

Multi-task learning presents a solution to this challenge by training a single model to execute multiple NLP tasks simultaneously, with the objective of enhancing performance on each task by learning a shared representation that can generalize to new tasks. However, multi-task learning is complicated due to the necessity of balancing the objectives of multiple tasks and preventing interference between them. Multi-task learning diminishes the need for separate models for each task, providing a framework through which language models can more effectively capture the underlying structure of natural language, leading to improved performance on downstream NLP applications.

At present, multi-task learning methods in NLP typically entail modifying pre-trained language models such as BERT [(Devlin et al., 2018)] and fine-tuning them on multiple tasks simultaneously. Nevertheless, these methods can suffer from interference between tasks and can be computationally expensive. Furthermore, the resulting models can be challenging to interpret.

Researchers have explored various techniques to address these challenges, including custom prediction heads [(Reimers and Gurevych, 2019)], specialized layers added to the model for each task, and multi-objective optimization [Bi et al. (2022)], which involves optimizing multiple loss functions simultaneously. These approaches have shown potential in enhancing the performance of multi-task NLP models, but further research is necessary to fully comprehend their potential and limitations.

To address these challenges, our empirical studies examine the base minBERT model's performance on the SST, STS, and Paraphrase Detection datasets. We discovered that the base minBERT trained only on the SST dataset performs poorly on STS and achieves only random results on Paraphrase Detection. These baseline findings motivated us to extend the minBERT architecture to better adapt it to the multi-task objective.

To achieve this, we implemented prediction layers for all three tasks, experimented with various numbers of learnable parameters, projection sizes, dropout probabilities, and so on. We also implemented cosine similarity fine-tuning to improve the model's comprehension of semantic "closeness" for sentence pairs in the STS dataset, which had the worst baseline performance. In addition, we implemented round-robin training on all datasets and gradient surgery. We ran the modified multi-task model on the development and test datasets and thoroughly evaluated its performance and limitations.

We trained models with various hyperparameter configurations using the training data and evaluated them on the development set. Our findings revealed that the default values of dropout probability and learning rate from the original gpoesia minBERT[1] implementation were already close to optimal.

3 Related Work

Our research is motivated by the MTRec model, proposed by Bi et al. (2022), which employs a multi-task approach to learn robust sentence embeddings for Category Classification and Named Entity Recognition tasks. The authors utilize BERT to encode news titles as news embeddings and devise two auxiliary tasks on top of BERT, which are jointly trained with the primary news recommendation task. This multi-task approach exhibits an improvement in the ability of BERT to capture the semantics of news.

Moreover, we integrate certain concepts from the Gradient Surgery algorithm introduced by Yu et al. (2020) to enhance our gradient updates during round-robin training across all three datasets. The PCGrad technique addresses the issue of gradient conflicts between various tasks by utilizing projection, thereby minimizing the interference that the different tasks have on each other while learning a shared representation.

4 Approach

As previously described, our methodology entailed the integration of custom prediction heads in conjunction with the implementation of three extensions to the baseline minBERT model. Our objective was to enhance the model's ability to incorporate multi-task learning of sentence embeddings.

[1] `https://github.com/gpoesia/minbert-default-final-project`

4.1 minBERT Baseline

The initial codebase upon which we built our BERT implementation was forked from Gabriel Poesia's minbert-default-final-project[2]repository on Github. We worked to implement the following components of the minBERT architecture:

- Multi-head Self-Attention [(Vaswani et al., 2017)]: We used the following equation to guide our vectorized solution in Pytorch:

$$Attention(K, Q, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

- Transformer layer [(Vaswani et al., 2017)]: implemented the BertLayer.add_norm(), BertLayer.forward() and Bert- Model.embed() functions
- AdamW Optimizer [(Loshchilov and Hutter, 2017)] implemented the first moment and second moment updates, along with the "efficient version" of the parameter update with weight decay.

We then proceeded to load pretrained weights into the BERT model and finetune using the SST dataset in order to perform Sentiment Classification. We implemented functions in the BertSentimentClassifier.

4.2 Sentiment (SST) Prediction head

To enhance the performance of the model on the sentiment classification task, we implemented a task-specific head consisting of a linear projection layer with dropout. The input_ids and attention_mask are first passed through the model to obtain the pooled output, which is then fed through a dropout layer and a linear layer of size ($\textit{hidden\_size} \times \textit{n\_sentiment\_classes}$). This results in a vector representing the model's predicted probabilities for each of the 5 sentiment classes, from which the class with the highest probability is selected as the final prediction. Although we explored more complex configurations with additional layers and parameters, we observed significant overfitting on the training set. Therefore, we decided to keep this setup and focus on improving the training process.

4.3 Paraphrase Detection Prediction Head

To improve performance on the Quora dataset, we implemented a task-specific head consisting of a linear projection layer with dropout. We modified the input processing to account for the sentence pair input. Specifically, we performed a forward pass on the input_ids and attention_masks to obtain the pooled output representations for each sentence. We then applied dropout to each vector, concatenated the resulting outputs, and applied a linear layer with dimensionality $(2 \times \textit{hidden\_size}) \times 1$. This output layer is appropriate for the binary paraphrase detection task of the Quora dataset. Our approach is similar to the BERT paper's approach for "Sentence Pair Classification Tasks", except we omitted the "SEP" token and concatenated the embeddings back-to-back. We found that more complex configurations resulted in overfitting and therefore decided to keep this setup while focusing on improving training steps.

4.4 Semantic Text Similarity Prediction Head

For this final task, we experimented with three different strategies. The performance differences between these approaches will be compared later in the paper. The three strategies, outlined below, were implemented in order to predict the similarity scores between pairs of sentences:

- Iteration 1 - Scaled Linear Layer: This approach uses the same implementation as the paraphrase detection head. A dropout operation is applied to the concatenated tensor of dimension *2 * hidden_size*, which is then projected to a dimension of 1 using a linear projection layer. The output of this layer is multiplied by 5 to maintain consistency with the output labels being continuous similarity scores between 0 and 5.

[2]`https://github.com/gpoesia/minbert-default-final-project`

- Iteration 2 - Cosine Similarity: Instead of projecting the concatenated output vectors from the forward pass, we compute the Cosine Similarity between the two vectors and use that as the output of the prediction head.

- Iteration 3 - Scaled Cosine Similarity + ReLU: Since most of the cosine similarity scores outputted during our experiments appeared to be in the [0,1] range, we instead added a ReLU [(Agarap, 2018)] layer right after the cosine similarity computation, then scaled the output by multiplying by 5 as done in the first iteration. This resulted in the best dev accuracy scores after finetuning the minBERT embeddings.

4.5 Multi-Task Fine-Tuning

In the baseline model, fine-tuning is performed solely on the SST dataset, leading to poor accuracy scores for the remaining two tasks for which we are seeking to train embeddings. To address this limitation, we modified the training loop such that all three datasets and tasks are employed for fine-tuning concurrently. Initially, new dataloaders were created for the Quora and SemEval datasets, and subsequently, the three dataloaders were zipped together. We then implemented a Round-Robin training approach, whereby a batch was processed from each dataloader for one epoch of training. This strategy was chosen over training on one dataloader at a time to mitigate the potential risk of the model "forgetting" what it learned about the SST task when processing the Paraphrase Detection and STS tasks. By interleaving the batches, the objective was optimized for all tasks simultaneously. However, the discrepancy in dataset sizes resulted in an issue of upweighted loss for datasets with larger batch sizes. While reweighting the losses was considered, the multi-task training was considerably slower than single-task training, and testing multiple weighting schemes would have been excessively time-consuming. Thus, to address this issue, we artificially increased the sizes of the SST and STS dataloaders to match the larger paraphrase dataloader. We achieved this by creating cycle iterators out of the smaller dataloaders and using the length of the paraphrase dataloader as the stopping criterion for the training loop. The cycle iterator stores a copy of all the elements it returns, allowing for "resetting" of the dataloader and ensuring that the sizes of per-task training examples remain uniform even after exhaustion of the smaller dataloaders during training. Although the additional examples are simply repeats of the ones already encountered by the model, this approach was expected to decrease the interference observed from training on the unbalanced dataloaders, where one training objective "undoes" the learning from another after the corresponding dataloader is depleted.

4.6 Gradient Surgery

Multi-task fine-tuning presents several challenges, including potential conflicts between training objectives for each task and optimization difficulties that limit the efficiency gains compared to single-task learning [(Yu et al., 2020)]. While the exact reasons for these challenges are not fully understood, per-task loss curves can provide insight into the extent of task interference. To address this problem, we employed the Gradient Surgery extension proposed by Yu et al. (2020). This approach hypothesizes that conflicts between gradients are detrimental when they coincide with high positive curvature and a large difference in gradient magnitudes, as conflicting gradient directions from different tasks may interfere with each other. The Pytorch-PCGrad[3] algorithm corrects this by projecting the gradient of a conflicting task g onto the normal plane of the gradient of the i-th task g_i. Pseudocode for the algorithm is provided below:

[3] `https://github.com/WeiChengTseng/Pytorch-PCGrad`

Algorithm 1 PCGrad Update Rule

Require: Model parameters θ, task minibatch $\mathcal{B} = \{\mathcal{T}_k\}$

$\mathbf{g}_k \leftarrow \nabla_\theta \mathcal{L}_k(\theta) \ \forall k$

$\mathbf{g}_k^{\text{PC}} \leftarrow \mathbf{g}_k \ \forall k$

for $\mathcal{T}_i \in \mathcal{B}$ **do**

  for $\mathcal{T}_j \overset{\text{uniformly}}{\sim} \mathcal{B} \setminus \mathcal{T}_i$ in random order **do**

    if $\mathbf{g}_i^{\text{PC}} \cdot \mathbf{g}_j < 0$ **then**

      // Subtract the projection of $\mathbf{g}_i^{\text{PC}}$ *onto* $\mathbf{g}_j$

      Set $\mathbf{g}_i^{\text{PC}} = \mathbf{g}_i^{\text{PC}} - \frac{\mathbf{g}_i^{\text{PC}} \cdot \mathbf{g}_j}{\|\mathbf{g}_j\|^2} \mathbf{g}_j$

return update $\Delta\theta = \mathbf{g}^{\text{PC}} = \sum_i \mathbf{g}_i^{\text{PC}}$

4.7 Hyperparameter Search

In order to optimize performance, a hyperparameter search was conducted on the model's learning rate and dropout probability values. However, due to the significant time required for training, an extensive grid search over multiple candidate hyperparameter values was not feasible. Instead, a limited search was performed, testing only three values for each variable and selecting the best performing option. The search began by evaluating the fine-tuning option using learning rate values of 1e-4, 1e-5, and 1e-6. From there, the best performing learning rate value was selected, and a search was conducted over dropout probability values of 0.2, 0.3, and 0.4.

5 Experiments

5.1 Data

Stanford Sentiment Treebank

The Stanford Sentiment Treebank [4] consists of 11,855 single sentences extracted from movie reviews. The dataset was parsed with the Stanford parser[5] and includes a total of 215,154 unique phrases from those parse trees, each annotated by 3 human judges. Each phrase has a label of **negative**, **somewhat negative**, **neutral**, **somewhat positive**, or **positive**. For our model, we store this as a SentenceClassificationDataset before creating the SST dataloader. The input to our task-specific prediction head is a single sentence such as "Light, silly, photographed with colour and depth, and rather a good time", and the output is a class label of 0-4 corresponding to the 'sentiment' values listed earlier. Our dataset split for evaluation and testing is as follows:

- train (8,544 examples)
- dev (1,101 examples)
- test (2,210 examples)

CFIMDB Dataset

While this one was only used for the first half of the project in which we implemented and tested the minBERT baseline, we still include it because it helped us validate that our baseline model was working correctly. The dataset consists of 2,434 highly polar movie reviews. Each movie review has a binary label of negative or positive. The splits we used were as follows:

- train (1,701 examples)
- dev (245 examples)
- test (488 examples)

[4] `https://nlp.stanford.edu/sentiment/treebank.html`

[5] `https://nlp.stanford.edu/software/lex-parser.shtml`

Quora Dataset
The Quora dataset[6] consists of 400,000 question pairs with labels indicating whether particular instances are paraphrases of one another. For our model, we store this as a SentencePairDataset before creating the PD dataloader. The input to our task-specific prediction head is a pair of sentences that could be paraphrases, and the output is a binary label corresponding to the yes or no. We are using a subset of this dataset with the following splits:

- train (141,506 examples)
- dev (20,215 examples)
- test (40,431 examples)

SemEval Dataset
The SemEval STS Benchmark dataset [(Agirre et al., 2013)] consists of 8,628 different sentence pairs of varying similarity on a scale from 0 (unrelated) to 5 (equivalent meaning). For our model, we store this as a SentencePairDataset before creating the STS dataloader. The input to our task-specific prediction head is a pair of sentences that could be semantically similar, and the output is a continous float value between 0 and 5 indicating the similarity score. We use the following splits:

- train (6,041 examples)
- dev (864 examples)
- test (1,726 examples)

5.2 Evaluation method

To evaluate the effectiveness of our model, we utilized accuracy scores, which measure the percentage of correctly predicted labels for the SST and PD tasks, and Pearson Correlation, which measures the correlation coefficient between the model's predicted scores and the true scores, for the STS task. We compared our model's performance to the minBERT baseline, which was implemented in the initial phase of our project and only underwent SST fine-tuning without any custom prediction heads or extensions. Our evaluation was based on the dev set outputs, which were used to track the progress of our final architecture during each iteration. Finally, to assess the generalizability of our model, we evaluated its out-of-sample performance using the test set outputs.

5.3 Experimental details

In our experimental procedure, we pursued two main stages, namely Model Refinement and Hyperparameter Refinement. During the first stage, we conducted several modifications to the model architecture and implemented various extensions, using the dev accuracies as a measure of effectiveness and providing insights into areas that required re-examination. Overall, we produced seven iterations of the model, with the initial iteration being the baseline and each subsequent model improving on the previous by introducing significant changes aimed at resolving identified issues observed in the prediction outputs post fine-tuning. In the second stage, we performed a 3×3 hyperparameter search, as outlined in (4.7), utilizing the optimal values to fine-tune the model further, which we included in the final iteration of the model. Herein, we briefly outline the seven model iterations as follows:

1. minBERT Baseline: This iteration corresponds to our initial implementation from the milestone. We fine-tuned the model solely on SST without any additional features.
2. Increased Dropout: We introduced dropout layers to the PD and STS prediction heads, positioned between the model forward pass and the fully-connected linear projection layer.
3. Cosine Similarity Fine Tuning: This iteration involves switching from Iteration 1 to Iteration 2 for the STS prediction head architecture outlined in (4.4).
4. Unbalanced Round-Robin: We conducted round-robin training on all three datasets, interleaving batches from each dataset for a single training epoch. We ignored size differences in dataloaders and stopped processing batches from a single task once training examples ran out. However, this approach led to significant training slowdown.

[6] `https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs`

5. Gradient Surgery: We adapted the PCGrad algorithm to Pytorch, enabling the projection of conflicting SST, PD, and STS gradients during the optimizer step.
6. Balanced Round-Robin: This iteration involved reworking the round-robin training loop, changing smaller dataloaders to cycle iterators, as outlined in (4.5), to create more balanced dataloader sizes. However, this approach led to an additional training slowdown.
7. Modified Cos-Sim + Hyperparams: We switched from Iteration 2 to Iteration 3 for the STS prediction head architecture outlined in (4.4). Additionally, we included the optimal learning rate and dropout probability values obtained from the hyperparameter refinement conducted earlier.

5.4 Results

In accordance with our approach detailed in section 5.3, we utilized the dev accuracy scores as a guide for refining our model, resulting in incremental improvements in its performance. We performed a thorough examination of the model outputs after each fine-tuning iteration and identified areas of weakness where the model was misclassifying or performing poorly on certain types of sentences or tasks. We then conducted a critical analysis of the choices and assumptions that informed our model design, making isolated changes to evaluate their impact on addressing these issues, as outlined in the description of our model iterations above. The results of our dev score evaluations are presented below.

Model	SST Dev	PD Dev	STS Dev	Overall Dev
minBERT Baseline	0.391	0.380	-0.009	0.254
Increased Dropout	0.530	0.410	-0.043	0.299
Cosine Similarity Fine Tuning	0.525	0.403	0.293	0.407
Unbalanced Round-Robin	0.441	0.783	0.377	0.534
Gradient Surgery	0.455	0.779	0.421	0.552
Balanced Round-Robin	0.490	0.774	0.668	0.644
Modified Cos-Sim + Hyperparams	0.493	0.776	0.692	0.654

In light of our limited number of opportunities to make a test set leaderboard submission, we exercised discretion in deciding when to solicit out-of-sample feedback. Specifically, we pursued such feedback only when we were sufficiently convinced of the efficacy of our model. As a consequence, our two leaderboard submissions correspond to our "Gradient Surgery" and "Modified Cos-Sim + Hyperparams" iterations, each of which represented our most promising model at the time of submission. Our motivation in submitting these iterations was to gauge their performance relative to other models on the test set. The outcomes of these submissions are presented below:

Model	SST Test	PD Test	STS Test	Overall Test
Gradient Surgery	0.442	0.789	0.316	0.516
Modified Cos-Sim + Hyperparams	0.501	0.778	0.662	0.647

6 Analysis

In summary, our multi-task model's results align with our expectations, given the challenges associated with learning a shared representation for multiple tasks. Task interference and language structure can make it difficult to create embeddings suitable for all tasks. However, we are pleased with our final model, as it reflects a culmination of iterative experimentation and successive improvement, with our embeddings surpassing the baseline we aimed to exceed. We relied on qualitative analysis of model outputs to identify areas for improvement and determine the necessary edits at each iteration.

6.1 Qualitative Evaluation of Model Iterations

1. minBERT Baseline: We noted that, despite poor dev accuracy results, the train set accuracy scores were still pretty high, around 1.5-2x the score of the dev accuracy. We wanted to address the overfitting that might be occuring due to our use of fully connected linear layers

for all the prediction heads. This prompted our next iteration in which we saw much more consistent accuracy scores between train/dev.

2. Increased Dropout: Despite seeing an improvement in SST and PD task accuracy after adding dropout, we still see STS dev scores lagging behind by a lot. We concluded that the concatenated embeddings + linear layer (Iteration 1 from 4.4) approach was probably not good enough to capture semantic similarity (which is what this task is actually seeking to measure). This guided our choice for the next extension, which would replace the linear layer with a cosine similarity score and we saw siginificant improvement in the STS dev accuracy as a result.

3. Cosine Similarity Fine Tuning: Even with the improved STS scores, we see that SST has the highest accuracy of the three. We attributed this to the single-task fine-tuning that we still hadn't changed from the baseline model. At this point, we decided to leverage the extra datasets available (Quora, SemEval) in order to incorporate more information into the model to help train/learn a better shared representation.

4. Unbalanced Round-Robin: Implementing the round-robin training led to a huge slowdown in training time (roughly 50-60 minutes per epoch on AWS GPU), but we saw that it boosted the PD accuracy by a very large margin, and also improved STS. We attributed this to the fact that the Quora dataset is so much larger than the others, so the shared embeddings that were learned have been "preferentially" fine-tuned to the PD task more than the others. We wanted to address this issue of conflicting task objectives, so our next step was to adapt the PCGrad algorithm and see if we could get any improvements

5. Gradient Surgery: We see that PCGrad did indeed improve the overall balance of dev accuracies (PD decreased by a little, and SST + STS increased) which gave us evidence that the conflicting task objectives were playing a role in restricting the model's performance. Given that the PD score is still much better than the others, we decided to tackle the unbalanced dataset issue next, as we felt it would have a similar impact in improving the collective score across all the tasks.

6. Balanced Round-Robin: Our hypothesis again proved corrrect here, as our introduction of the balanced cycle-iterator dataloaders improved the accuracy of STS by a significant margin, while also slightly improving SST and negligibly decreasing PD Dev. At this point we felt like we had a solid final model, and just wanted to squeeze any remaining performance possible from the hyperparameter search and modified STS prediction head.

7. Modified Cos-Sim + Hyperparams: When inspecting our model outputs, we noted that the cosine similarity score between sentence pairs was very rarely negative. To address this we came up with the modified architecture of Iteration 3 described in 4.4, where we repurpose this anti-correlation information and do some final scaling. We also found optimal hyperparameters as described in 4.7, yielding a learning rate of 1e-5 and dropout probability 0.3.

Fine-tuning separately on the tasks probably would have yielded better scores overall (as some of the top leaderboard entries have shared on Ed), but the goal for us as a team was to learn about and experiment with multi-task techniques, which we feel like we accomplished to a reasonable extent.

7 Conclusion

In conclusion, this project explored various techniques for fine-tuning and adjusting contextual BERT embeddings to improve performance on multiple sentence-level tasks. Through the use of cosine-similarity fine-tuning, an additive loss function for multi-task fine-tuning, gradient surgery, and custom prediction heads, significant improvements were achieved over the baseline results. Our contributions yielded competitive leaderboard scores for Sentiment Analysis, Paraphrase Detection, and Semantic Textual Similarity tasks. These findings suggest that multi-task learning is a promising approach to improving the performance of pre-trained language models on multiple NLP tasks, and that fine-tuning strategies can be tailored to specific tasks to achieve better results. Future research can continue to explore the use of multi-task learning and fine-tuning techniques to further improve the performance of pre-trained language models on a wide range of NLP tasks.

References

Abien Fred Agarap. 2018. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*.

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.

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

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*, 33:5824–5836.