Around the BERT model: from a basic implementation to advanced optimizations and Multi-Task Learning

Stanford CS224N

Ines Dormoy
ICME
Stanford University
idormoy@stanford.edu

Yoni Gozlan
ICME
Stanford University
yonigoz@stanford.edu

Joachim Studnia
ICME
Stanford University
jstudnia@stanford.edu

Abstract

This report introduces a simple and efficient architecture for Multitask learning in natural language processing, based on a pretrained BERT model. Using BERT contextualized embeddings and making them go through a single additional layer per task, the Multitask BERT consistently achieves decent scores on three target language problems: sentiment analysis, paraphrase detection and semantic textual similarity (less than 20% below state-of-the-art performances). During training, one could choose to freeze BERT weights and only update additional parameters, but we found that fine-tuning the BERT block to fit the underlying distribution yielded better results. The use of PCGrad to get rid of conflicting gradients, as well as gradient accumulation to boost training and additional datasets to better understand diverse language styles are a few examples of ideas leveraged by our model. Our main contribution is the simplicity of the model, which requires less that a single specific dense layer per task, contrary to many state-of-the-art papers in existing literature. This architecture is thus particularly adapted to end users with limited resources willing to simultaneously achieve acceptable baseline results on several downstream language tasks.

1 Introduction

Sentiment Analysis, Paraphrase Detection, and Semantic Textual Similarity are three standard natural language processing tasks which have significantly benefited from the advances of the past decade, namely recurrent neural networks and large language models. As for many language tasks, linguistic or statistics based methods (*eg* counting words having positive connotation for the first task, and identifying synonyms amongst words or n-grams for the two others) were the only known baselines up until the late 2000s, but they performed very poorly. With the GPU revolution of the 2010s, traditional methods to solve these tasks shifted towards neural networks, in particular LSTM, which were capable of understanding sequential characteristics and context (see [1] in 2015 for an LSTM-based model on the Stanford Sentiment Treebank dataset). In the late 2010s, the rise of attention, Transformers ([2]), and large language models (GPT and BERT in 2018 for instance) was a major breakthrough that allowed previously unseen improvements of machine performance (see [3] in 2022 for the same task, with an improvement of more than 15% in accuracy).

This report presents an efficient architecture for a Multitask BERT, built on top of a simplified version of BERT ([4]). Contrary to the common approach of fine-tuning BERT weights on each specific problem, as suggested in the original BERT paper, this model is designed to be trained on a variety of tasks at once, and is split between a common block for all tasks, and a specific layer for each task with very few parameters.

One of the key ideas of our approach has been the use of PCGrad, a gradient surgery framework designed to tackle the update of model parameters that are common to several subtasks, developed in [5]. Other contributions include the use of a learnable parameter to scale cosine similarities between

average last hidden layers of BERT embeddings (inspired from [6]), the additional training on other datasets to avoid overfitting, as presented in [7], or gradient accumulation to speed up training.

It is to be noted that we developed this model having in mind to reach the best compromise between complexity (number of additional layers after BERT, training time...) and performance metrics on each task. As an example, the use of additional LSTM on BERT contextualized embeddings has been abandoned as it hardly brought any measurable improvement while significantly increasing complexity.

This report is structured in the following way. Section 2 presents the BERT model, and how it has historically been fine-tuned to solve multiple NLP tasks, with a particular emphasis on multitask and transfer learning. Section 3 describes the architecture of Multitask BERT, as well as the construction of our own minBERT model, the training modes and optimization methods, and our main contributions. In section 4, we enumerate all experiments that we conducted on our Multitask BERT, with a focus on customization ideas that produced a significant increase in performance. Finally, we analyze in section 5 some of the results of our model on the dev set of each task, both quantitatively and qualitatively.

2 Related Work

BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained natural language processing model developed by Google in 2018 [4]. BERT is based on the transformer architecture, which was introduced in the paper *Attention is All You Need* in 2017 ([2]). The BERT architecture is pre-trained on a large corpus of text. The authors of *Bert: Pre-training of deep bidirectional transformers for language understanding* [4] show that the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of NLP tasks, such as question answering and language inference, without substantial task-specific architecture modifications. After the publication of BERT in 2018, many others have worked on how to fine-tune it for specific tasks. The authors of *How to Fine-Tune BERT for Text Classification?* present some ways of fine-tuning BERT for text classification. In this paper, further pre-training of BERT with target domain data is suggested to better adapt to the specific data distribution of the target domain. Plus, multi-task fine-tuning is explored as a means to exploit shared knowledge among multiple tasks in a target. [8] In the specific case of classification tasks for additional pre-training, several experiments have been made to find working architectures for each specific task. Some authors suggest adding a MultiLayer Perceptron before the final classification layer to boost accuracy [9]. Other researchers have boosted their accuracy by adding LSTM layers after the pre-trained BERT model [10]. For some specific tasks like similarity computations, other authors tested cosine similarity layers [6]. Even if the number of additional fine-tuning layers added to BERT varies, the authors of *Bert: Pre-training of deep bidirectional transformers for language understanding* claim that little layers should be needed to fine-tune BERT for a specific task. [4] Finally, the literature also discusses if it is better to use the last hidden state of the output of BERT, or if it is sufficient only to pool the last CLS token. Some authors found better results for the similarity task by using the whole hidden state layer and not only pooling the CLS token. [11]

When confronted to multiple tasks like semantic similarity, sentiment analysis and paraphrase detection, one possibility is to use only one base BERT model for the three tasks and to fine-tune it simultaneously on the three tasks. This approach is called multi-task learning. Without pre-trained language models, multi-task learning has shown its effectiveness in exploiting the shared knowledge among multiple tasks. [8] When there are several available tasks in a target domain, an interesting question is whether it still brings benefits to fine-tune BERT on all the tasks simultaneously. [12][13] In practice, it is sometimes hard to implement a multi-task approach as learning one task can interfere with the learning of the other tasks. The authors of *Gradient Surgery for Multi-Task Learning* suggest a gradient correction approach to correct interfering gradients from multiple tasks. [5] Other authors have worked on how to combine the losses between different tasks. [14]

One drawback of multi-task learning and its evolutions is its high computational cost. [14] Therefore, deep learning researchers have developed strategies for faster implementations, like using half-precision floating point numbers, without losing model accuracy or having to modify hyper-parameters. [15] Other works use gradient accumulation, which implies using a gradient delta that is smaller compared to an accumulated gradient and provides a better direction towards a minimum

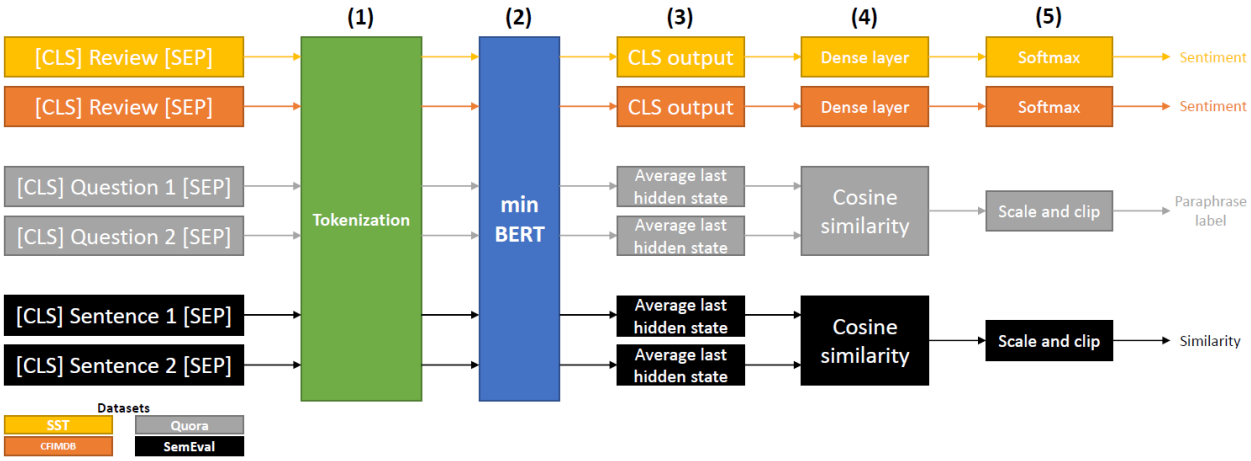

Figure 1: Architecture of the Multitask BERT model

compared to first-order gradients. The advantage of this implementation is that it requires fewer parameter updates. [16].

Finally, an approach relevant to our work is additional pretraining. According to the authors of *SMART: Robust and Efficient Fine-Tuning for Pre-trained Natural Language Models through Principled Regularized Optimization*, due to limited data resources from downstream tasks and the extremely high complexity of pre-trained models, aggressive fine-tuning often causes the fine-tuned model to overfit the training data of downstream tasks and fail to generalize to unseen data. Therefore, the authors suggest training the model on more tasks to improve its overall performance on several tasks. [7] An example of an additional training dataset is the Yelp reviews polarity dataset constructed by Xiang Zhang. It is first used as a text classification benchmark in the following paper: *Character-level Convolutional Networks for Text Classification*. [17]

3 Approach

This section presents the architecture of the Multitask BERT that we developed in an iterative way throughout this project. The training and optimization processes are also explained in this section.

3.1 Description of the architecture

Figure 1 shows the global architecture of the Multitask BERT model adapted to the three NLP tasks. This model is organized around 5 successive steps.

Step (1). A batch of text data from 4 datasets is loaded and preprocessed to be given as input to BERT. This step includes splitting sentences into words and then into word pieces, tokenizing those word pieces (30K known word pieces, others are marked as [UNK]), padding short texts to 512 tokens with [PAD], adding an initial [CLS] token and [SEP] to separate sentences, and converting all those tokens to ids. It is to be noted that we used an additional dataset (namely CFIMDB) to further pretrain and achieve better performance on the Sentiment Classification task (see the additional pretraining section).

Step (2). Each sequence of tokens in the batch passes through our minBERT model, thoroughly described in the subsequent subsection. For each input i, this step outputs a hidden representation $H_i^{(1)} \in \mathbb{R}^h$ for the [CLS] token where h is BERT hidden size), and a last hidden state $H_i^{(2)} \in \mathbb{R}^{l \times h}$ representing all tokens in the input where l is the maximum length of inputs.

Step (3). We are retrieving the [CLS] output for the Sentiment Classification task, giving $o_i^{(3)} = H_i^{(1)} \in \mathbb{R}^h$ for each input i. For the 2 other tasks, inputs goes by pair (i, j), and we are averaging all the last hidden states, giving $o_i^{(3)} = \frac{1}{l_i} \sum_{k=1}^{l_i} (H_i^{(2)})_{k,\cdot} \in \mathbb{R}^h$ and $o_j^{(3)} = \frac{1}{l_j} \sum_{k=1}^{l_j} (H_j^{(2)})_{k,\cdot} \in \mathbb{R}^h$, where l_i, l_j is the actual lengths of the tokenized inputs i, j.

Step (4). Elements o_i of the previous step go through a dense layer with 5 outputs for the Sentiment Classification task, yielding $o_i^{(4)} \in \mathbb{R}^5$ (the layer is common for the 2 additional datasets and distinct for the SST dataset). For the 2 other tasks, given a pair (i, j), we compute the cosine similarity between $o_i^{(3)}$ and $o_j^{(3)}$ as $o_{i,j}^{(4)} = \frac{o_i^{(3)} \cdot o_j^{(3)}}{\|o_i^{(3)}\|_2 \|o_j^{(3)}\|_2} \in [-1, 1]$.

Step (5). We compute $o_i^{(5)} = \text{Softmax}(o_i^{(4)}) = \frac{1}{\sum_{k=1}^{5} e^{(o_i^{(4)})_k}} (e^{(o_i^{(4)})_k})_{1 \leq k \leq 5} \in \mathbb{R}^5$ for any input i of the sentiment task. For the 2 other tasks, given a pair of inputs (i, j), we compute $o_{i,j}^{(5)} = \max(0, \min(1, a_{\text{task}}(o_{i,j}^{(4)} - 1) + 1))$ (scaling and clipping), where a_{task} here represents 2 additional learnable parameters a_{para} and a_{sim}.

3.2 Construction of the minBERT block (2)

This subsection describes the structure of the minBERT model, reproducing a simplified version of the original BERT [4].

Embedding layer. Our minBERT implementation computes the sum of learnable token embeddings mapping each of the 30K word pieces to vectors in $\mathbb{R}^d$ (with $d = 768$ here), and positional embeddings mapping each of the $l = 512$ possible token positions to vectors in $\mathbb{R}^d$. Since we only consider individual sentences, no segmentation embedding as in [4] has been implemented.

BERT Transformer layer. As described in [4], embeddings $v \in \mathbb{R}^{d \times l}$ pass through 12 successive Encoder Transformer layers (represented in 3). Each of those 12 layers is made up of multi-head attention, an additive and normalization layer with residual connection, a feed-forward layer, and another additive and normalization layer with a residual connection. More information on the Transformer layer can be found in the appendix A.1. Note that dropout is applied at each sublayer level before Add & Norm, and at the entrance of BERT Transformer layer as well.

BERT finally outputs final hidden states for all tokens in the input (contextualized embeddings denoted as $H^{(2)} \in \mathbb{R}^{l \times h}$ above) as well as the CLS token embedding $H^{(1)} \in \mathbb{R}^h$. In order to avoid having to fully retrain our minBERT, we retrieved the base BERT weights after being trained on the masked token prediction and the next sentence prediction tasks.

3.3 Loss and optimizer

We used cross-entropy loss for the sentiment analysis and the paraphrase detection tasks, and mean squared error for the semantic textual similarity task (thus viewed as a regression problem). As for the optimizer, we made use of the efficient version of the Adam optimizer, presented in [18]. Compared to the basic stochastic gradient descent, the originality of this first-order optimization algorithm is that it estimates exponential moving averages of the gradient and the squared gradient of the loss, allowing a better control of the extent of parameter updates.

3.4 Min-BERT training and fine-tuning methods

Figure 2 shows the numerous customization arguments that our Multitask BERT supports.

Freezing or unfreezing BERT weights. In both cases initial BERT weights are loaded from a pretrained model, but we still have to choose whether to fine-tune minBERT on the three downstream tasks (as suggested in [4]) or to freeze its weights. Recall that while all tasks share the same

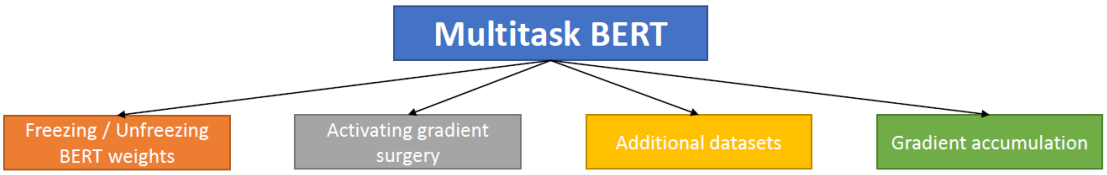

Figure 2: Customizations options of Multitask BERT

embedding and BERT layers, they all have distinct parameters to train to produce the final prediction (dense layer for some tasks, scaling parameters for the others).

Activating gradient surgery. If we choose to fine-tune the common BERT weights (here represented by a vector $\theta \in \mathbb{R}^D$ with D large) on the three tasks, we might end up having three distinct directions and magnitudes by which we would like to update θ, given by the gradients $g_k = \nabla_\theta \mathcal{L}_k(\theta) \in \mathbb{R}^D$ for $k \in \{1, 2, 3\}$. This issue is raised by [5] and named *conflicting gradients*, and the default way Multitask BERT tackles solves this is by using as resulting gradient the average $g = \frac{1}{3}(g_1 + g_2 + g_3)$. However, as one of the gradients may dominate the others, or simply because the average direction may be completely irrelevant for all tasks, Multitask BERT can also be trained using PCGrad ([5], code in [19]). In short, $g_1^{\text{PC}}, g_2^{\text{PC}}, g_3^{\text{PC}}$ are initialized as g_1, g_2, g_3, as long as there exists $k_1 \neq k_2$ with $g_{k_1}^{\text{PC}} \cdot g_{k_2} < 0$, $g_{k_1}^{\text{PC}}$ is being replaced by its projection on the orthogonal subspace to g_{k_2}, namely $g_{k_1}^{\text{PC}} - \frac{g_{k_2} \cdot g_{k_1}^{\text{PC}}}{\|g_{k_2}\|_2^2} g_{k_2}$. Up to a multiplicative constant, we eventually set $g^{\text{PC}} = \frac{1}{3}(g_1^{\text{PC}} + g_2^{\text{PC}} + g_3^{\text{PC}})$ as our final parameter update.

Additional datasets. [7] shows that fine-tuning BERT models on additional datasets may help build more robust embeddings. Our current pipeline allows Multitask BERT to be simultaneously trained on many datasets, while its performance is still mainly evaluated on the three initial datasets. Insofar as datasets have very different sizes in general, we had to manually choose how often a given input for some task would be seen during an epoch. In other words, instead of weighing tasks through objective functions, we transferred this to the dataset level, which drastically reduces the computational cost.

Gradient accumulation. Multitask BERT supports a custom configuration of gradient accumulation. This is used to speed up training by only updating parameters once every few batches, and aggregating gradients between two successive updates.

3.5 Main contributions

The main contribution of this work is to demonstrate that a Multitask BERT architecture with very few additional layers can still be very efficient at three NLP tasks simultaneously. As as comparison, [7] achieves state of the art scores on many problems, but uses much bigger base models such as RoBERTa or T5. As an illustration, the performance of our model on the sentiment analysis task is very close to the performance of a single BERT model fine-tuned on this specific task. With limited resources and little data, our Multitask BERT thus has a decent training time and could certainly be adapted in the future to tackle many other simultanous NLP problems.

Adding a learnable parameter to properly scale cosine similarities (step (5) in 1 before clipping them is also new, and led to a significant enhancement in paraphrase and similarity results. In particular, this idea does not appear in [6], which precisely focuses on building sentence embeddings fit for semantic similarity comparisons.

Finally, as specified in [11], taking the average last hidden state works best for sentence similarity prediction, and our experiments showed that this statement also held for paraphrase prediction when using a Siamese network architecture. We thus used the average last hidden state instead of the CLS output to compute the cosine similarity between two inputs for similarity and paraphrase prediction.

4 Experiments

This section describes the experiments that we launched to test the performance of Multitask BERT, and the associated results.

4.1 Exp-I: Additional layers for each specific task: testing MLP and LSTM

We first experimented with adding task specific layers to the models. Following the approach of the authors of [9], we tried adding a Multilayer perceptron specific to each tasks. Then, inspired by the architecture proposed in [10], we replaced the MLP with an LSTM network in each task specific heads.

4.2 Exp-II: Cosine Embedding loss

Another way of maximizing cosine similarity is to use cosine embedding loss instead of using its similarity layer. One drawback of Pytorch's implementation of cosine embedding loss is that the targets must have values of -1 or 1, when the STS Benchmark has continuous labels varying from 0 to 5. In our implementation, we labeled targets from 2.5 to 5 as 1, and the rest as -1. Mapping the labels in that way induced a loss of information and hindered the model's learning. We achieved a 24% accuracy increase on the STS dataset by changing the Cosine Embedding loss setting to a similarity layer setting as presented in Section 3.

4.3 Exp-III: Conflicting gradients

An important problematic in multi-task learning is to train a model to perform well on one task without adversely affecting the others. One way multiple tasks can adversely affect each others is when the gradient learned from one task opposes one from another task. To combat this effect, we implemented Gradient Surgery [5]. Gradient Surgery mitigates gradient interference by altering the gradients directly. If two gradients are conflicting, that is if they have a negative cosine similarity, the gradients are altered by projecting each onto the normal plane of the other, preventing the interfering components of the gradient from being applied to the network.

4.4 Exp-IV: Imbalanced datasets

With the datasets being of very different sizes, we had to define how training on one epoch would look like. Using a simple round robin approach would result in the model being trained on the entirety of the small datasets much more than the bigger ones (by a factor of more than 20 in our case), thus with an important risk of overfitting on the small datasets, and not enough training on the bigger ones. We thus limited the amount of times the model could be trained on a smaller datasets before it reached the end of the bigger datasets. This allowed us to have more control over overfitting on small datasets, and allowed for faster training times on the bigger datasets. In our experiments, we found out that a factor of five worked the best.

4.5 Exp-V: Additional datasets

As suggested in the SMART paper ([7]), we tried additional pre-training of the BERT model. [7] We decided to focus first on the sentiment analysis dataset, as we achieved a good accuracy on the STS dataset, and the Quora dataset is bigger than the STS and SST datasets. We first started by implementing the CFIMDB dataset as it was already provided. We also decided to work on the Yelp Polarity dataset because of its large size (560,000 train examples) and its broad range of reviews (hotels, restaurants, doctors...) in comparison to the CFIMDB and SST datasets which only focused on movie reviews. [17] However, the experiments on the 5 datasets had high computational costs (13 hours for 1 epoch) so we did not have the time and computational resources to run experiments with more than two epochs. Overall, the model achieved very good accuracies on the CFIMDB and the Yelp datasets (0.759 on CFIMDB and 0.948 on Yelp), but did not outperform the initial model without the additional datasets. One possible explanation is that the fine-tuning layers between CFIMDB/Yelp and SST were not common, so the model did not learn simultaneously on those tasks. Also, additional epochs would have probably helped enhance the accuracy as our best accuracies

are reached on 10 epochs training. Finally, we were not able to use PCGrad in this setting as it significantly slows down training.

4.6 Exp-VII: Adding gradients accumulation

We added accumulated gradients to our pipeline to compensate for our limited resources and use a larger effective batch size during our training, so as to provide better direction towards a minimum of the objective function compared to updating gradients every batch. Effectively, we update the gradients every four batches. Using this technique gave us better overall results, especially towards the end of the training.

4.7 Exp-VI: Hyperparameter tweaking

As running even one epoch of Multitask BERT on all tasks is already computationally expensive, we had no way to use grid-search based methods to optimize hyperparameters, let alone cross-validation. We thus manually tweaked their values in a set of 4 or 5 possibilities and observed which one gave the best dev scores. Among those hyperparameters were the data imbalance ratio, the learning rate, the optimizer, the initial scaling parameter for cosine similarity, the gradient accumulation steps.

4.8 Data

Table 1 describes the datasets that were used throughout this project.

Datasets	Classes	Type	Train samples	Dev samples	Test samples
Quora		Para	141.506	20.215	40.431
SemEval STS		Similarity	6.041	864	1.726
SST	5	Sntimnt	8.544	1.101	2.210
CFIMDB	2	Sntimnt	1.701	245	488
Yelp polarity reviews	2	Sntimnt	650.000	50.000	

Table 1: Datasets

4.9 Evaluation method

Accuracy was used as main evaluation metric for the Sentiment Analysis and the Paraphrase Detection tasks. We also used additional ways of measuring performance such as F1-score for Paraphrase Detection directly, and for binary classification subproblems of Sentiment Analysis as well. Pearson correlation was the main metric to evaluate performance for Semantic Textual Similarity.

4.10 Experimental details

All experiments were conducted on AWS GPUs, and the average training time for one epoch has more of less always been close to one hour. In our latest experiments, we used Automatic Mixed Precision (ie carrying the most expensive computations in 16-bit precision instead of 32-bit floats), which improves our training time by 20%. Gradient accumulation was performed every 4 steps, and we set the learning rate to $1 \cdot 10^{-5}$ when fine-tuning BERT weights, and $1 \cdot 10^{-3}$ when freezing them (using the pretrained version).

Finally, we ran an ablation study to study the efficiency of the use of PCGrad, of additional layers (MLP, LSTM), and of additional pretraining on other datasets. The results are shown in the following subsection.

4.11 Results

Table 2 reports the results achieved on each specific task by different variants of our Multitask BERT model.

We were surprised to get better results when only using a common BERT backbone for all three tasks and no additional layer (apart from a dense and softmax for the first task) compared to adding MLP

	CFIMDB		Paraphrase (Quora)		SemEval STS		SST	
	Dev	Test	Dev	Test	Dev	Test	Dev	Test
Random baseline (theoretical)	0.5	0.5	0.5	0.5	0	0	0.2	0.2
State-of-the-art	/	**0.962**	/	**0.975**	/	**0.924**	/	**0.598**
No additional layers (finetune)	/	/	0.74	/	0.845	/	0.506	/
No additional layers + pcgrad (finetune)	/	/	0.782	/	0.844	/	0.491	/
MLP (finetune)	/	/	0.758	/	0.778	/	0.485	/
MPL (pretrain $\rightarrow$ finetune)	/	/	0.75	/	0.804	/	0.492	/
LSTM (pretrain $\rightarrow$ finetune)	/	/	0.757	/	0.772	/	0.43	/
Pcgrad + CFIMDB (finetune)	/	/	0.779	**0.775**	0.845	**0.838**	0.512	**0.507**
Yelp + CFIMDB (finetune)	0.759	/	0.723	/	0.826	/	0.486	/

Table 2: Comparison of the performances of the tested models for the different NLP tasks. All tasks use accuracy as their main metric except STS which uses Pearson correlation coefficient

or LSTM layers specific to each task. We tried to mitigate the difference of initialization of the two parts of the models by first pretraining the task specific heads and keeping the BERT layers frozen, and then finetuning the whole model, but with little success. This observation would need further investigation to try and find a better approach to training task specific layers.

The low score for SST was to be expected considering the state of the art only manages to get an accuracy of 0.598. This is partly due to the fact that this is a 5-class classification task and not a simple binary problem (a random model would have a 0.2 accuracy and not 0.5).

However, we did not expect similarity detection to work considerably better than paraphrase detection, when the architecture we used for these two tasks are very similar, and we have more training data for paraphrase detection. This may signify that either our task specific heads are not tuned properly to suit each tasks, or the losses we have used are not the most appropriate. It could also be a result of the difference in metrics (Pearson correlation for STS versus accuracy for paraphrase detection).

Finally, as described in the previous section, we decided to keep the PCGrad optimization framework, as well as the additional dataset CFIMDB, which gave the best dev scores. The fact that the test scores are only 1% lower than the corresponding dev scores shows that our model did not overfit on any of the train or dev sets.

5 Analysis

This section further analyzes the results of the model, exposed in the previous subsection.

5.1 Additional metrics on the dev set

Sentiment analysis. For this 5 task classification problem, the prediction accuracy on the dev set is 0.512 (way over the random baseline of 0.2). The mean squared error (when identifying each class to the corresponding integer between 0 and 4) is 0.708, which shows that predictions are on average very close to true values, up to at most one unit). More precisely, 6.4% of examples have a prediction and a true value that differ by 2 units or more. Those observations can be visualized in the confusion matrix, shown in table 4 is appendix. Finally, the two series are reasonably correlated (coefficient 0.776).

Paraphrase detection. Some metrics for binary classification are presented in table 5. Figure 5 in appendix also shows the ROC curve. We observe that our model is very efficient at detecting paraphrases (high recall) but seem to have a lot of false positives (giving low precision). In any case the model is behaving much better than the random baseline.

Semantic textual similarity. With a correlation coefficient of 0.845 and a mean squared error of 0.028 when scaling all similarity values between 0 and 1, this is the task on which our model performs the best. Figure 6 shows that although the model guesses the right score up to ± 0.2 more than 75% of the time, the actual distribution of true similarity scores and predicted values are quite

Task	Input	Real	Predicted	Explanation
Sentiment	‘ It ’s a lovely film with lovely performances by Buy and Accorsi .’	3	4	This example would even be confusing for human judgment, since there is absolutely no indication of anything negative...
Paraphrase	‘I have 95% in 10th, 95% in 12th and CGPA 7.8 in B.Tech @ IIT KGP ECE, what should be my CAT score to make it to an IIM?’ / ‘I have X% in 10th, Y% in 12th and Z% in Undergrad, what should be my CAT score to make it to an IIM?’	0	1	The two sentences are grammatically and semantically extremely similar, which lures the model into thinking that they mean the same thing...
Similarity	‘A girl is eating a cupcake’ / ‘A woman is eating a cupcake’	0.52	0.94	The real similarity score scaled between 0 and 1 seems quite arbitrary, since those sentences are semantically very close... No surprise the model gets confused

Table 3: Examples of inputs for which Multitask BERT fails to classify correctly or predict a close similarity score

different. In particular our predicted scores seem biased towards 1, which might be a consequence of the customized scaling procedure. This would need further investigation to be corrected.

5.2 Qualitative analysis

Table 3 presents a list of examples where Multitask BERT fails to predict the right sentiment class, paraphrase label, or similarity score, and a guess for a qualitative explanation. The main take-away is that our model seems to get confused to determine the proximity between two sentences when they present similar grammatical, semantic or linguistic structure. For some of those examples though, the choice of ground truth score seems somewhat arbitrary (the distinction between classes for sentiment analysis is particularly blurry...).

6 Conclusion

In this report, we showed that a simple Multitask BERT architecture was able to reproduce (and even sometimes outperform) single-task baselines while using common weights for the minBERT block. Making use of PCGrad to avoid conflicting gradients when training on multiple tasks, of cosine similarity layers and learnable scaling parameters for paraphrase detection and semantic textual similarity, and of additional pretraining to reduce overfitting, this Multitask BERT is therefore a reproducible and computationally efficient way to achieve reasonable scores on the three tasks of interest.

The primary limitation of this work is that the 3 test scores remain very far from the state-of-the-art. It might be due to the simplicity of our final model (a single dense layer is being learned for the sentiment analysis task, while simply one scaling parameter is being learned for each of the two other tasks). In that respect, the use of MLP or LSTM on top of a pretrained BERT deserves to be further investigated. Other ideas for future work include ensembling distinct models (high computational cost), using smoothness-inducing adversarial regularization (incentivizing the model to output close results for close inputs, as in [7]), taking advantage of additional input features (named-entity, part-of-speech) or using attention layers in task-specific heads.

References

[1] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. *CoRR*, abs/1503.00075, 2015.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017.

[3] Franz A. Heinsen. An algorithm for routing vectors in sequences. 2022.

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

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

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

[7] Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, Tuo Zhao. SMART: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. 2020.

[8] Chi Sun, Xipeng Qiu, Yige Xu, Xuanjing Huang. How to fine-tune bert for text classification? 2018.

[9] Qinjin Jia, Jialin Cui, Yunkai Xiao, Chengyuan Liu, Parvez Rashid, and Edward F. Gehringer. All-in-one: Multi-task learning bert models for evaluating peer assessments, 2021.

[10] Nishant Rai, Deepika Kumar, Naman Kaushik, Chandan Raj, and Ahad Ali. Fake news classification using transformer based enhanced lstm and bert. *International Journal of Cognitive Computing in Engineering*, 3:98–105, 2022.

[11] Maike Behrendt and Stefan Harmeling. Arguebert: How to improve bert embeddings for measuring the similarity of arguments. In *Conference on Natural Language Processing*, 2021.

[12] Simon Vandenhende, Stamatios Georgoulis, Wouter Van Gansbeke, Marc Proesmans, Dengxin Dai, and Luc Van Gool. Multi-task learning for dense prediction tasks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3614–3633, 2022.

[13] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Improving multi-task deep neural networks via knowledge distillation for natural language understanding. *CoRR*, abs/1904.09482, 2019.

[14] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. *CoRR*, abs/1711.02257, 2017.

[15] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training, 2018.

[16] Joeri Hermans, Gerasimos Spanakis, and Rico Möckel. Accumulated gradient normalization, 2017.

[17] Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *CoRR*, abs/1509.01626, 2015.

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

[19] Wei Cheng Tseng. Reimplementation of gradient surgery for multi-task learning: github.com/weichengtseng/pytorch-pcgrad.

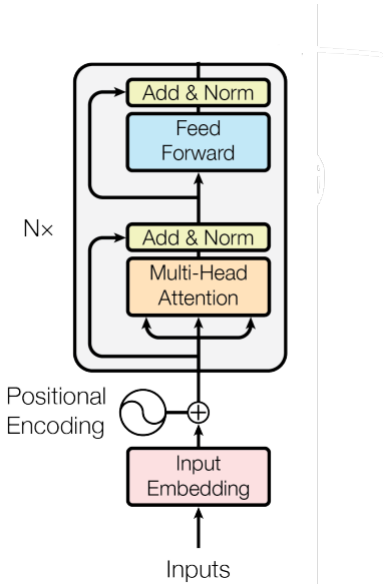

Figure 3: Encoder Layer of Transformer used in BERT ([2])

Predicted / Actual	0	1	2	3	4
0	58	65	15	1	0
1	48	170	54	17	0
2	8	57	86	71	7
3	3	7	50	164	55
4	1	0	11	67	86

Table 4: Confusion matrix for sentiment analysis on the dev set

A Appendix

A.1 Transformer layer

In this appendix section, we describe the Transformer layer, initially presented in [2] (see figure 3). The element of figure 3 that deserves further explanation is undoubtedly the multi-head self-attention. This attention procedure is built upon the simpler scaled dot-product attention process, consisting in linearly combining value vectors according to the interaction between a set of queries and keys. Mathematically, given $Q, K \in \mathbb{R}^{n \times d_k}$ and $V \in \mathbb{R}^{n \times d_v}$, scaled dot-product attention is computed as

$$\text{Attention}(Q, K, V) = \text{Softmax}(\frac{QK^T}{\sqrt{d_k}})V \in \mathbb{R}^{n \times d_v}$$

Multi-head attention is then designed to simultaneously pay attention to information from different representation subspaces. More precisely, given queries $Q \in \mathbb{R}^{l \times d_{model}}$, keys $K \in \mathbb{R}^{l \times d_{model}}$ and values $V \in \mathbb{R}^{l \times d_{model}}$ (directly computed as linear transformation of the input $X \in \mathbb{R}^{d_{model} \times l}$), we fix a number of heads h, learn parameters $W_i^Q, W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ for $1 \leq i \leq h$, and compute $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \in \mathbb{R}^{n \times d_v}$. We finally learn a matrix $W^O \in \mathbb{R}^{hd_v \times d_{model}}$ and compute $\text{MultiHead}(Q, K, V) = [\text{head}_1 | \dots | \text{head}_h]W^O \in \mathbb{R}^{l \times d_{model}}$. We obtain the same shape as X up to a transposition, which allows the process to be repeated. Figure 4 taken from [2] summarizes this multi-head self-attention layer.

A.2 Quantitative analysis of the results of Multitask BERT on the dev sets

This appendix subsection presents a few tables and figures of interest to understand the behavior of the model.

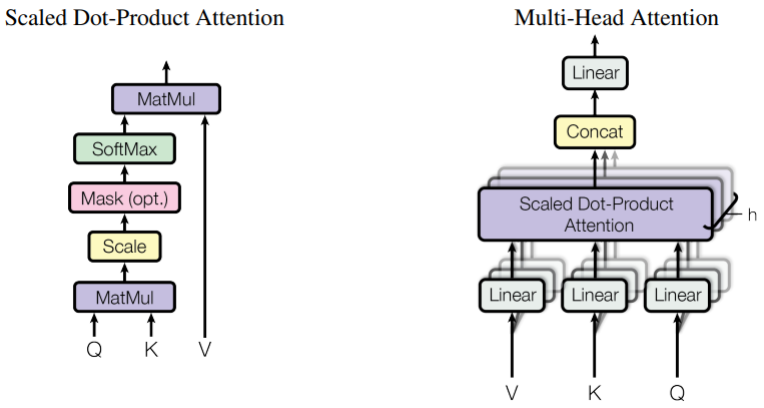

Figure 4: Scaled dot-product attention and multi-head self-attention

Accuracy	0.779
Precision	0.644
Recall	0.913
F1-score	0.755
AUC	0.875

Table 5: Binary classification metrics for paraphrase detection

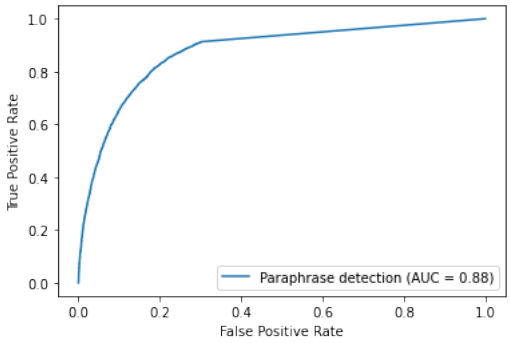

Figure 5: Receiver Operating Characteristic (ROC) curve for paraphrase detection on the dev set

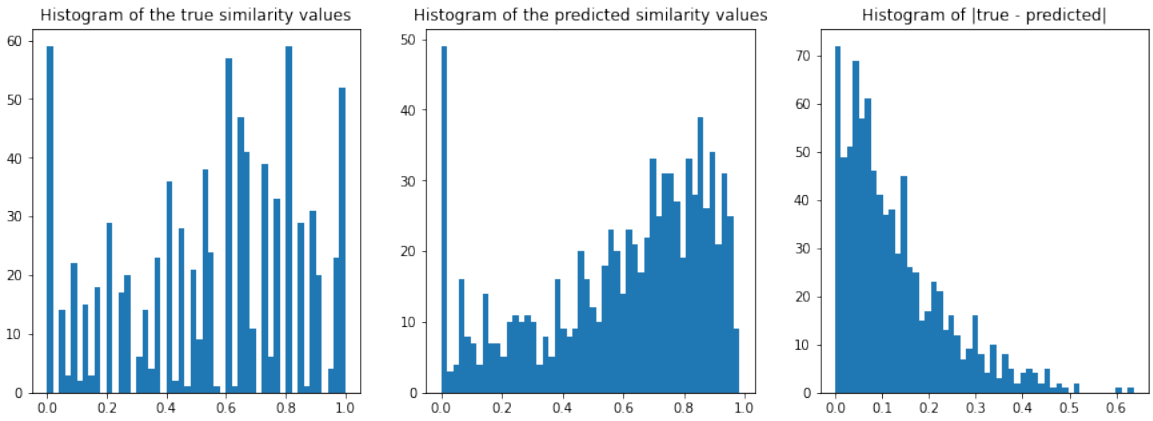

Figure 6: Distribution of similarity values, of predicted similarity values and of the absolute differences between true scores and predictions