BERT: A Master of All Trades or Jack of None?

Stanford CS224N Default Project

Tom Pritsky
Dep. Biomedical Data Science
Stanford University
tom5@stanford.edu

Josselin Somerville
Dep. Computer Science
Stanford University
josselin@stanford.edu

Marie Huynh
Dep. Biomedical Data Science
Stanford University
mahuynh@stanford.edu

Abstract

In most machine learning tasks today, we usually focus on a specific task and train and optimize our model to perform best on this one task. However, trying to learn different tasks at the same time may be useful in certain applications such as multilingual machine translation. This research field is called Multi-Task Learning. How can we make multi-task methods' results as good as current state-of-art methods (which are trained on single tasks) and thus reduce the number of parameters used by the models? As illustrated by Stickland and Murray (2019), multiple factors also motivate the principle of shared parameters across tasks. Applications with a large number of tasks–such as web-scale applications–may have constraints on the number of parameters that can be stored. Though multi-task approaches reduce the number of required parameters, optimizing the learning of multiple tasks at the same time is challenging. To this end, Stickland and Murray (2019) suggest using new adaptation modules–PALs or projected attention layers–and a novel method for scheduling training. In the literature, numerous other techniques to deal with the arising challenges of multitasking have been published. As an example, Yu et al. (2020a) proposed a method called gradient surgery to deal with competing gradients. In this project, we set up a multi-task model that can perform three separate linguistic tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. Training and evaluating on respectively the Stanford Sentiment Treebank (SST), the Quora Dataset and the SemEval Benchmark Dataset, we obtain a baseline having an arithmetic mean of 0.648 for the accuracy on the test sets. We evaluate multiple published fine-tuning approaches and study their interactions to reach a mean dev accuracy of 0.761 and a mean test accuracy of 0.767 ranking us 5^{th} on the leaderboard. Additionally, we introduce a combination of Pal scheduling (Stickland and Murray, 2019) and Gradient vaccine (Wang et al., 2020)–Gradient Compromise–that massively increases the training speed during the first epochs of finetuning.

Sharing project: *Marie Huynh is also using this final project in BIODS53 (Software Engineering for Scientists) which is a 2 credits class to learn good software engineering practices. (good use of github, organization of code, unit testing, automation, ...)*

1 Introduction

Natural Language Processing has been a fast-paced evolving field in the past years, and has achieved great performance on a variety of tasks including sentiment analysis, paraphrase detection and semantic similarity. Nevertheless, in this context, models often have a few hundred million trainable parameters, which makes "adaptations to new tasks computationally infeasible" as underlines Maziarka and Danel (2021). Furthermore, such models also face overfitting and data-scarcity problems Chen et al. (2021). Multitask learning (MTL) has emerged as a promising approach which can help us create better language representations, improve generalization of models and spare resources.

As highlighted in Crawshaw (2020), "the existing methods of MTL have often been partitioned into two groups with a familiar dichotomy: hard parameter sharing vs. soft parameter sharing". On one hand, in hard parameter sharing, you share the model weights between multiple tasks and each weight is trained to jointly minimize multiple loss functions (Crawshaw, 2020). On the other hand, in soft parameter sharing, each task has its model with its individual weights and we penalize the distance between the model parameters of the different tasks in order for the parameters to be similar.

In recent years, BERT (Bidirectional Encoder Representations from Transformers) (Devlin et al., 2018) has become a popular choice for multitasking in Natural Lan-

guage Processing (NLP) due to its strong performance on various NLP tasks. Inspired by Stickland and Murray (2019), we assume soft-parameter sharing with the whole of BERT would require too many parameters. In this context, we will consider hard parameter sharing while adding adapters to shared layers and keeping several task-specific output layers with the objective of building a multitask BERT on three separate linguistic tasks : sentiment analysis, paraphrase detection and semantic similarity. Using this approach, we hope to improve accuracy on each individual task through parameter sharing, while also reducing the total number of parameters trained (resulting in reduced model memory requirements). These optimization goals are critical for use in limited-memory consumer devices such as smartphones.

Despite the great aforementioned advantages, MTL still faces multiple challenges. As an illustration, tasks can compete with each other to achieve a better learning representation during training. Numerous techniques have been explored in literature to answer those arising challenges and we leverage and combine some approaches to improve on our BERT-based multitask model.

2 Related Work

This section will first discuss recent works of the BERT model in MTL and will then tackle novel approaches dealing with the challenges of MTL in NLP.

2.1 BERT Model and Multitasking

Introduced by Devlin et al. (2018), the BERT model has led to state-of-the-art results in many NLP tasks and has significantly reduced the need for labeled data by pretraining on unlabeled data over different pre-training tasks. BERT is first trained trained on plain text for masked word prediction and next sentence prediction tasks. We call this first step pretraining. Then, it is finetuned on a specific linguistic task with additional task-specific layers using task-specific training data. In 2019, Liu et al. (2019) proposed a multitask learning framework for BERT that simultaneously learns to classify multiple attributes of text, such as pairwise text classification and text similarity. The lower layers–the text encoding layers–are shared across all tasks, while the top layers are task-specific, combining different types of natural language understanding (NLU) tasks. The latter reached state of the art results on ten NLU tasks. Furthermore, several recent works aim to improve multitasking performance by modifying the BERT architecture itself. One such approach is the 'Projected Attention Layer' (PAL) introduced by Stickland and Murray (2019). The PAL is a low-dimensional multi-head attention layer that is added in parallel to normal BERT layers and specific to each task. This layer allows for a global attention layer that is specific to each task, without having the number of parameters multiplied by the number of tasks, as most of the parameters will be shared in the classic multi-head attention layer.

2.2 Multitask Fine-Tuning

Other recent works have also focused on optimizing the fine-tuning process for multitask learning. The latter faces numerous challenges, the first one being to make sure that training the model for a task does not ruin the accuracy of another task.

2.3 Scheduling

First, how do we iterate through the tasks? The classic approach for multi-task learning is to use round-robin sampling (cycling through the different tasks one after the other). The problem is that if the dataset contains more instances of one task than another, it will repeat several times all the examples of the task with few data points before it repeats all the examples of the other task. This will lead to overfitting for the task that repeats a lot and underfitting for the other one. To this end, Stickland and Murray (2019) propose a novel method for scheduling training. At first, the tasks are sampled proportionally to their training set size but then to avoid interferences, the weighting is reduced to have tasks sampled more uniformly.

2.4 Gradient Treatment Methods and Finetuning Regularization

Second, how do we deal with competing gradients? During finetuning, the gradients of each task are computed and used to update the shared model parameters. However, the gradients of one task can interfere with the gradients of another task, leading to suboptimal performance on both tasks. To deal with competing gradients, Yu et al. (2020a) proposed a method called gradient surgery. This approach projects competing for gradients from one task onto the normal plane of another task, preventing the competing gradient components from being applied to the network and impairing optimization. This approach therefore encourages the model to learn similar representations for the different tasks. Finally, aggressive fine-tuning often causes over-fitting and thus failure to generalize to unseen data. The SMART fine-tuning framework developed by Jiang et al. (2020) aims to improve the latter in two steps. The first step is Smoothness-Inducing Adversarial Regularization, which reduces overfitting and improves generalization by adding a smoothness inducing adversarial regularizer to the loss function. The second step is Bregman proximal point optimization, which acts as a strong regularizer to prevent aggressive parameter updates during fine-tuning. This step improves the stability and convergence of the training process by preventing the updates from deviating too heavily from the previous ones. The SMART framework has been shown to im-

prove the performance of fine-tuning on various NLP tasks with limited data.

3 Approach

In this section we will describe the approach we followed. This consists in two main steps: handling the multitasking approach of the model, i.e. how can we update BERT when we want to finetune for different tasks. The second step consists in how can we change the architecture of our Multitask classifier so that it performs better without adding too many parameters.

3.1 Baseline : BERT model

As a baseline, we implemented a simple classifier consisting of a BERT model that has on top one dropout and a linear layer trained for each task. We recall that the Bert model consists of a stack of 12 Bert layers. Each Bert-Layer can be defined as:

$$\text{BL}(h) = \text{LN}(h + \text{SA}(h))$$

with LN a normalization layer and SA a self-attention layer defined as:

$$\text{SA}(h) = \text{FFN}(\text{LN}(h + \text{MH}(h)))$$

Here, FFN is simply a feed-forward network and MH is a multi-head attention layer.

Sentiment classification and paraphrase detection are both considered as classifications, whereas semantic similarity is handled as a regression task. Our model has been first pre-trained (only the classification layers are trained) and then finetuned (all the parameters, including the BERT layers, are trained) using a random task scheduler, meaning that at each step of the training, the task used for training is sampled randomly. Except if precised otherwise, the learning rate used will be 10^{-3} for pretraining and 10^{-5} for fine-tuning.

3.2 Learning to multitask

As we have previously seen, multitasking raises two main problems: how should we schedule the learning, and how do we deal with competing gradients. In this section, we will describe approaches we used to tackle both of these issues as well as a method to make sure our finetuning does not diverge too much from our pretrained BERT model.

3.2.1 Task Scheduling

Since we have far more training examples for paraphrase detection (140,000) vs sentiment classification (8,500), we optimized task scheduling via the approach proposed in Stickland and Murray (2019).

$$\mathbb{P}(task_i) = \mathbb{N}_i^{\alpha} \tag{1}$$

$$\begin{aligned}
&\mathbb{P}(task_i) = \text{task}_i \text{ probability} \\
\text{where:}\quad &\mathbb{N}_i = \text{task i dataset size} \\
&\alpha = 1 - .8 * \frac{e-1}{E-1} \\
&e = \text{current epoch}; E = \text{\# of epochs}
\end{aligned}$$

3.2.2 Finetuning with gradient treatment methods and Regularized Optimization

To prevent opposing task gradients from impairing training, we implemented gradient surgery and tried approaches from two sources: Yu et al. (2020b) and Nzeyimana (2022) *(with Automated Mixed Precision)*.

As a reminder, gradient surgery consists of projecting a gradient g_i on g_j if they have a negative cosine similarity by doing:

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

A more sophisticated method, gradient vaccine, does not assume that all tasks must enjoy similar gradient interactions and instead uses Φ_{ij}, the cosine similarity between g_i and g_j:

$$g_i' = g_i + \frac{||g_i|| \left(\Phi_{ij}^T \sqrt{1-(\Phi_{ij})^2} + \Phi_{ij} \sqrt{1-(\Phi_{ij}^T)^2}\right)}{||g_i|| \sqrt{1-(\Phi_{ij}^T)^2}} g_j$$

On the other hand, as aggressive finetuning often leads to overfitting and failure to generalize. We incorporated the SMART regularization framework (Jiang et al., 2020) using a SMART-Pytorch library to avoid the latter.

SMART consists of two steps. The first step is Smoothness-Inducing Adversarial Regularization, which seeks to reduce overfitting and improve generalization when fine-tuning a task with limited data. This step optimizes the following equation:

$$\min_{\theta} \mathcal{F}(\theta) = \mathcal{L}(\theta) + \lambda_{\text{s}} \mathcal{R}_{\text{s}}(\theta),$$

where $\mathcal{L}(\theta)$ is the loss function, $\lambda_{\text{s}} > 0$ is a tuning parameter, and $\mathcal{R}_{\text{s}}(\theta)$ is the smoothness inducing adversarial regularizer, defined as:

$$\mathcal{R}_{\text{s}}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \max_{\|\tilde{x}_i - x_i\|_p \leq \epsilon} \ell_{\text{s}} \left(f\left(\widetilde{x}_i; \theta\right), f\left(x_i; \theta\right)\right),$$

The second step of SMART is Bregman proximal point optimization. This approach works as a strong regularizer to prevent aggressive parameter updates that occur during fine-tuning. Essentially, it prevents θ_{t+1} update from deviating too heavily from θ_t update. This approach takes the form:

$$\theta_{t+1} = \text{argmin}_{\theta} \mathcal{F}(\theta) + \mu \mathcal{D}_{\text{Breg}}\left(\theta, \theta_t\right),$$

where $\mu > 0$ is a tuning parameter, and $\mathcal{D}_{\text{Breg}}(\cdot, \cdot)$ is the Bregman divergence defined as

$$\mathcal{D}_{\text{Breg}}\left(\theta, \theta_t\right) = \frac{1}{n} \sum_{i=1}^{n} \ell_{\text{s}}\left(f\left(x_i; \theta\right), f\left(x_i; \theta_t\right)\right)$$

3.2.3 Individual Pretraining

During pretraining, the parameters of BERT are frozen, so the three classifiers are independent. This is why we implemented **individual pretraining which consists of training our multitask classifier on each task with frozen BERT layers, and then loading the 3 best sets of parameters for each fully connected layer used in the classifier**. This made a massive difference as explained in the Experiments section.

3.2.4 Our contribution: Scheduling with Gradient Treatment Methods

One problem with Gradient Surgery (or Gradient Vaccine) is that it is not compatible with any type of scheduling. Indeed, one gradient per task is needed, so the schedule must be round-robin. That is why we introduce a new method that we call **Gradient Compromise**–detailed in Algorithm 1–which combines both the idea of the PAL schedule and the treatment of the gradients from the gradient vaccine. The idea is to only update our model every batch of batches, and every time the task of a batch is chosen by a slightly modified PAL schedule.

Algorithm 1: Process several batches

input : Epoch index e, number of epochs N_e, nb. of batches $N_{batches}$, dataset size S_{data}
output : Schedule, losses

schedule $\leftarrow$ **['sst', 'para', 'sts']**;
$\alpha \leftarrow 1 - 0.8 * (e-1)/(N_e - 1)$;
$p \leftarrow (S_{data})^{\alpha}$;
$p \leftarrow \text{probs} / \sum \text{probs}$;
$p_b \leftarrow (p * n_{\text{batches}} - 1)/(n_{\text{batches}} - 3)$;
$p_b \leftarrow$ **clip**(p_b) ;
$p_b \leftarrow p_b / \sum p_b$;
+**random_choice**(names, p_b, $n_{\text{batches}} - 3$) ;
shuffle(schedule) ;
for *task in schedule* **do**
  loss[task] $\leftarrow$ loss[task] + loss(batch, task) ;
end
$\beta \leftarrow 0.2 + 0.8 * (e-1)/(N_e - 1)$
returnloss$/(S_{\text{data}})^{\beta}$

3.3 Tuning the architecture

3.3.1 Handling two inputs

Both paraphrase detection and semantic textual similarity have to deal with two input sentences and one output. How to handle these two inputs is a major decision in the model architecture. There are two main approaches: running both inputs individually in BERT and then performing some type of polling (concatenation, max pooling, cosine similarity, ...) or concatenating both sentences (separated by a [SEP] token) and feeding one long sentence to BERT. Both approaches have been tested and the results are available in the Experiment section.

3.3.2 Projected Attention Layers

As described in the Related Works section, one way to improve the accuracy of multitasking is to add some task-specific attention layers, called Projected Attention Layers. Stickland and Murray (2019) propose a low-rank approach: output $= V^D g(V^E h)$, with V^D and V^E two matrices forming a low-rank matrix.

As described in the paper, we changed the BERT layers themselves. The idea is to add a new attention term that is specific to each task and has only a few parameters. This way, most of the attention parameters are still shared in $SA(h)$, but the model can fine-tune the attention mechanism to each task by adding $PAL(h)$ in the Bert layer:

$$BL(h) = LN(h + SA(h) + PAL(h))$$

The task-specific attention is defined as $PAL(h) = V^D g(V^E h)$. The paper compares different choices for g: the identity function, a feed-forward network, or a multi-head attention. We implemented a simpler variation of the actual **Projected Attention Layer** as called in the paper. The paper proposes one multi-head attention layer, shared across all layers, while we implemented a low-rank multi-head attention layer but the attention is not shared across the layers, only the matrices V^D and V^E.

3.3.3 Classifier Architecture

In many works, researchers only use a fully connected layer that takes as an input the embedding of the [CLS] token, to build a classifier. As we share the same BERT model for several tasks, we tried many architectures or classifier heads, hoping that most of the task-specific parameters would be learned in the classifier head and not in BERT. This is why we implemented, some fully connected layers with dropout and a ReLu activation function at each layer. Then we tried to use an RNN with some fully connected layers to pull the last hidden state of the RNN. Additionally, we tried a variant that fed all the tokens except the [CLS] token to the RNN, then concatenated the last hidden state to the [CLS] embedding and fed this

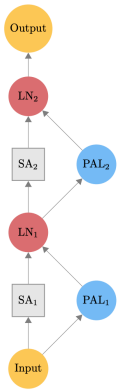

Figure 1: PAL

vector to several fully connected layers. Finally, we tried one last variation, using LSTM instead of RNN. All the architectures described were implemented individually for each task. Later on, we found out that adding an LSTM or RNN led to overfitting for some tasks so we tried using an LSTM only for the Semantic Textual Similarity task while we kept only fully connected layers for the other tasks.

3.4 Data processing and optimizations

3.4.1 Class imbalances

One major challenge in this project was to handle the repartition of the classes for the classification tasks. As illustrated in the Experiments section, the repartition of class was not uniform so we performed some data augmentation described in section 4.1 to fix that and we also filtered out inputs that were too long in order to handle larger batch sizes.

3.4.2 Considering Distance Between Classes : an intuition from the confusion matrix for SST

For the sentiment classification, we used a simple multi-task classifier with a Cross-entropy loss. However, this does not take into account the structure that follows the classes (for example, class 0 (Negative) is closer to class 1 (Somewhat Negative) than class 4 (Positive). This can be seen in the confusion matrix. One approach would be to treat this problem as a regression problem. We tried the latter, but the results were not conclusive. Instead we tried to learn a 5×5 matrix that links the predictions to the actual probabilities of each class. Let's imagine that we get the following probabilities $[0.02, 0.38, 0.24, 0.32, 0.04]$, simply taking the argmax would result in predicting class 1. Here with such predictions, we might think that instead, it is class 2, as the model is uncertain between somewhat positive, neutral and somewhat negative. This is why we multiply these probabilities by C a 5×5 matrix, that produces $\tilde{p}$ our new probabilities for each classes.

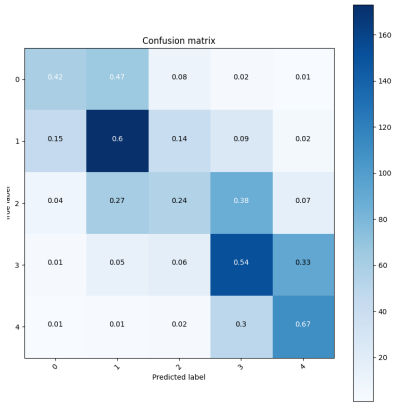

Figure 2: Confusion Matrix on the dev set of SST

3.4.3 Memory optimization

As our model (Hugging Face bert-base-uncased) is very large, our GPU could not handle a batch size of more than 16. This means that the training is slow and the updates are very irregular as the gradients are averaged over a small batch. To overcome this issue, we implemented two optimizations: Automatic Mixed Precision Micikevicius et al. (2017) and Gradient Accumulations. The former enables us to use nearly half the memory by using half-precision, while the latter only updates the model every x batches which permits us to simulate larger batches and accelerates the training process.

4 Experiments

4.1 Data and Preprocessing

We are using the provided datasets for the default projects with the following splits :

Datasets	Labels	Size:	Task
Quora Dataset	Question pairs with paraphrase labels	*Train: 141,506* *Dev: 20,215* *Test: 40,431*	Paraphrase detection
SemEval STS Benchmark Dataset	Sentence pairs labeled 0 (unrelated) to 5 (equivalent)	*Train: 6,041* *Dev: 864* *Test: 1,726*	Sentence similarity
Stanford Sentiment Treebank (SST)	Movie reviews with 5 categorical labels from neg to pos	*Train: 8,544* *Dev: 1,101* *Test: 2,210*	Sentiment analysis

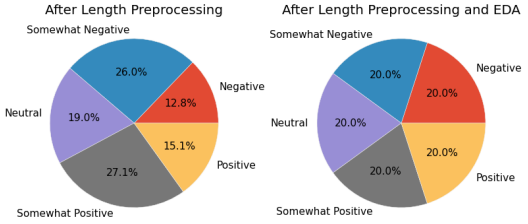

Figure 3: Sentiment Class Repartition

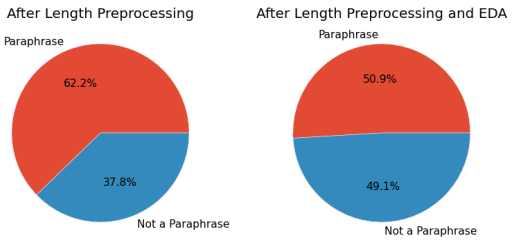

Figure 4: Paraphrase Class Repartition

We filtered out the top (2%) of the longest inputs: the longest sentences for the sentiment analysis and the longest sums of sentence pairs for the paraphrase and the similarity since we use this concatenation in our model.

Furthermore, we observed class imbalance in the training sets of the SST dataset (Sentiment) and the SemEval dataset (Similarity). Thus, we generated an augmented dataset using Easy Data Augmentation (ESA) (Wei and Zou, 2019). For each underrepresented class in the Sentiment and Paraphrase training sets, we augment them to be as present as the most represented class.

For each training sentence to augment, we perform the following 4 simple operations. During synonym replacement, we randomly choose some words–excluding stop words–from the sentence and replace each of those with one of its synonyms chosen at random. During random insertion, we find and insert a synonym of a random word–excluding stop words–into an arbitrary position in the sentence. During the random swap, we randomly choose two words in the sentence and swap their positions. During random deletion, we randomly remove words with probability p.

4.2 Evaluation method

Since sentiment analysis and paraphrase detection are classification tasks, we use a simple accuracy metric, calculated by dividing percent correctly classified examples by total examples. As for semantic textual similarity, we use Pearson correlation of the true similarity values against the predicted similarity values for the SemEval STS Benchmark Dataset.

4.3 Experimental details

For all of our experiments we first pretrained our classification head with a leaning rate of 10^{-3} ad then finetuned the model (BERT + Classifiers) with a learning rate of 10^{-5}. The batch size varied depending on the methods and the task (we can have different batch sizes for each task with gradient accumulations). Generally, the batch size was 16 for Paraphrase detection and 32 for SST and STS. For some methods such as gradient treatments, we went as low as 8 for Paraphrase and 16 for SST and STS. Generally, we always used the highest batch size possible that would fit in memory (24GB GPU), and simulated a large batch size of 128 with gradient accumulations (except for gradient treatments, for which we used 32). The hidden dimension (if relevant) was always 768 (same dimension as our BERT embeddings).

4.4 Results

Method	Dev. SST	Dev. Paraphase	Dev. STS	Dev. Mean
Random predictor	0.197	0.528	0.021	0.249
BERT + concat embed.	0.378	0.693	0.174	0.415
BERT + concat embed. + Pal schedule	0.492	0.764	0.337	0.531
BaseModel: BERT + concat sentences	0.465	0.731	0.749	0.648
BaseModel + Pal schedule	0.499	0.869	0.856	0.741
BaseModel + Pal schedule + 1 hidden	0.504	0.876	0.862	0.747
BaseModel + Pal schedule + 1 hidden + data augment.	0.513	0.879	0.868	0.753
***AdvancedModel*: BaseModel + Pal schedule + 1 hidden + indiv. pretrain + data augment.**	0.520	0.882	0.872	0.758
BaseModel + 1 hidden + PCGrad	0.484	0.871	0.832	0.729
BaseModel + 1 hidden + Vaccine	0.514	0.853	0.845	0.737
BaseModel + 1 hidden + Vaccine + SMART	0.509	0.864	0.846	0.740
BaseModel + 1 hidden + Grad. compromise (ours)	0.516	0.834	0.848	0.733
BaseModel + 1 hidden + RNN 128	0.428	0.841	0.812	0.694
BaseModel + 1 hidden + RNN 128 + [CLS] embed.	0.482	0.849	0.865	0.732
BaseModel + 1 hidden + LSTM 128 + [CLS] embed.	0.491	0.827	0.858	0.725
BaseModel + 1 hidden + LSTM 256 + [CLS] embed.	0.500	0.842	0.872	0.738
BaseModel + 1 hidden + LSTM 256 (STS only) + [CLS]	0.517	0.860	0.876	0.751
AdvancedModel + PAL	0.247	0.64	0.523	0.470
AdvancedModel + PAL (only during finetuning)	**0.524**	**0.882**	**0.876**	**0.761**
AdvancedModel + PAL + SST adjustment Matrix	0.521	0.882	0.876	0.760

Table 2: Dev accuracy results in function of the different experiments. The first section groups simple addition to handle multitasking (Fully connected classifiers). The second section dives into gradient treatment methods. The third section shows results for more advances classifiers using the BaseModel and the Pal schedule. Finally, the last section give results with some Projected Attention Layers (PAL), first added during pretraining (bad performance) and then added during a last pass of fine-tuning (our best result).

The table above attempts to summarize the most significant results we obtained. Some experiments were left out for the sake of clarity (SST as regression, cosine similarity for Paraphrase, Various hidden sizes for hidden layers, more hidden layers, ...). If a result is not present, it is because it did not have a relevant impact on the Dev accuracy.

The best model obtained consisted of: a BERT-backbone with some added Projected attention Layers (12 attention heads of dimension 11 each) with a Pal scheduling, one hidden layer of hidden size 768 for each classifier. The model was trained with the augmented dataset, first individually pretrained for 25 epochs with a learning rate of 10^{-3} with a patience of 3, then finetuned with a learning rate of 10^{-5} for 10 epochs. We then manually changed the learning rate to 5.10^{-6} and 10^{-6} to gain 0.06% on the Dev. Mean accuracy. Here are our results on the Test Set:

Model	Dev. SST	Dev. Quora	Dev. STS	Dev. Mean
Best	0.533	0.883	0.885	0.767

Let us recap some major changes. First, we introduced the pal scheduling which massively improved the accuracy for SST and STS. This makes sense as prior to scheduling we used round robin sampling, which resulted in poor performance due to the significantly larger size of the Quora dataset. Our most significant improvement was switching from embedding concatenation (for Paraphrase and Semantic Similarity) to sentence concatenation. This bumped the accuracy of STS from 0.337 to 0.749. Instead of just comparing the embeddings of both sentences, by concatenating them, BERT was able to use the context of the previous sentence to encode the next (by concatenating the sentences with a [SEP] token in between). Finally the last major improvement was to individually pretrain on each task. By realizing that each pretraining was independent, we were able to pretrain each task and keep the best verison for each instead of pretraining all of them together and keeping the overall best. This solved the problem of the model overfitting on SST and STS while it was still learning on Quora.

We also had some progress that were not as significant. The first one, was to play with the number of hidden layers for the classifiers. After some experiments we figured that one single hidden layer with a hidden size of 768 was giving us the best results. Then, we studied our datasets and after augmenting our data to balance the classes we were able to gain overal 0.6% in accuracy. Finally, the last small improvement we had was to take a model already finetuned and add Projected attention layers. After only training the projected attention layers, we were able to gain an additional 0.3%.

Eventhough many techniques improved our accuracy, most of them did not improve our results as significantly as we expected. We were first surprised by the performance of gradient treatment methods (PCGrad and Vaccine). While they did improve our ability to multitask, the accuracy was lower than the Pal scheduling which was way simpler to implement and also way faster to train (as gradient treatment methods require small batch sizes due to the projections of the gradients). This is why we came up with our new method: **Gradient compromise**. While it did not perform better than the Pal schedule of Gradient Vaccine (the two methods it tries to compromise), we will see that it was much faster at learning.
Another surprising result was that every single architecture that we tried for our classifiers performed worse than our fully connected heads. Our first approach was to feed all the embeddings (including the [CLS] token) to an RNN (one per task) followed by a fully connected layer. This performed quite poorly, especially for Sentiment prediction. We then separated the [CLS] token so that the RNN only got fed the embeddings of the tokens in the sentences and then the fully connected layers got fed the [CLS] embedding as well as the last hidden state of the RNN. This means that this classifier head had access to the same features as our best model without PAL, and had in addition the output of the RNN. So we were expecting to improve our accuracy but it was the opposite. While our accuracy after pretraining was as expected way higher, our finetuning did not reach the same results. This could be due to not enough training, but is more likely due to some overfitting in the RNN that then caused some detrimental updates in the BERT layers. In fact with a LSTM of 512, we reached a loss of near 0 for SST with an accuracy of 0.25 : the model had overfitted. We then tried to change the RNN to a LSTM, use different sizes, apply the LSTM only to Semantic Similarity, all without great results.
Finally, our last major surprise was the Projected Attention Layers. We tried to add them in many stages of the learning process (at the beginning, in between pretraining and fine-tuning, after finetuning) and various methods to train them (freeze everything except PAL, freeze only BERT, freeze nothing) with various initialization (Xavier, no-influence, ...). The only setting that worked for us was to add them after fine-tuning, by freezing everything and initializing them so that they would have no influence (this way we kept our results from our previous fine-tuning). This was done by setting V^D and V^E to 0, while having Q, K and V equal to identity matrices.

We also had some methods that did not really influence our results suchs as SMART, the SST Adjustment Matrix, Performing a second finetuning on top of the pal schedule with gradient surgery, ...

4.5 Gradient compromise results

Our method **Gradient compromise** that tries to conciliate a scheduling (here a pal scheduling) with a gradient treatment method (here gradient vaccine) did not perform better overall than other techniques as we only reached 0.733 accuracy on the dev set. However, we noticed, that even though it was not reaching a better accuracy, the training was much faster. Below, are the curves of the training accuracy for a PAL scheduler (truncated for visibility), the Gradient compromise method, and the gradient compromise method with SMART regularization.

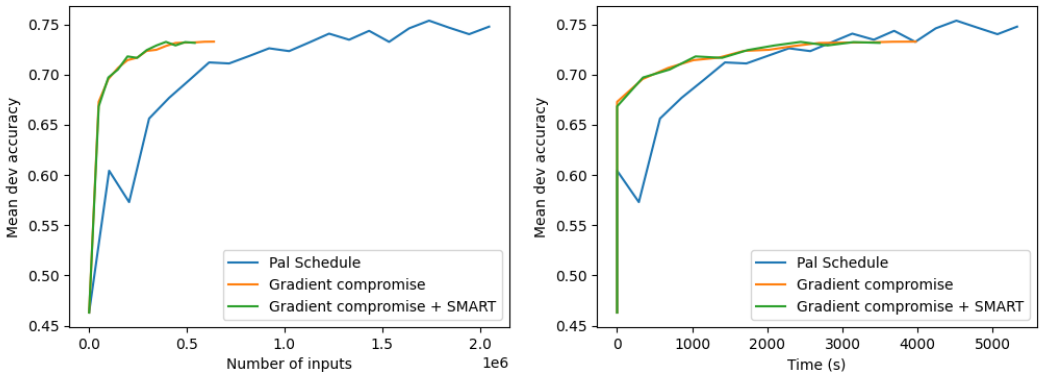

Figure 5: Gradient Compromise finetuning accuracy on the dev set versus Pal scheduling

The figure here clearly shows our improvement in training speed. While it might seem like a problem that it does not learn up to an accuracy as good as the pal schedule, it is not as we can first start the finetuning with the gradient compromise and then finish the training with a pal scheduling.

5 Analysis

Task	Example	Ground Truth	Prediction	Analysis
Sentiment analysis	If The Count of Monte Cristo doesn't transform Caviezel into a movie star , then the game is even more rigged than it was two centuries ago.	2	0	This example features the idiom 'the game is rigged' that the model might not have seen, leading to impaired prediction accuracy
Paraphrase detection	What are the best and profitable ways for saving money? What are your best ways to save money?	0	1	While similar, the two sentences differ in that only one requests the approaches be profitable. The model likely attended more heavily to the term 'best' since it often drives the sentence's meaning. Additionally, it failed to interpret the pronoun 'your' which changes the meaning (some parsing could help)
Sentence similarity	Work into it slowly It seems to work	0	2.8	The classifier failed because despite the similarity between both sentence's constituent words and their connotations, the first sentence has a hidden idiomatic meaning that is challenging for our classifier to detect, since it wasn't common in the training data.

As we can see above, our multitask model fails primarily when the text problem requires complex, high level reasoning to resolve. Idioms, which have significance beyond their literal meaning, often lead to these failures (as was the case for the sentiment analysis and similarity task examples). The model may also choose to attend to some words in the sentence at the cost of others, especially when those words are incredibly common or significantly impact the sentence meaning (meaning the model learns to attend to them more than others). 'Best' is a great example of this, having both high frequency and also serving as a strong indicator of positive connotation. Approaches like increasing the number of attention heads may improve the model's ability to attend to various parts of the sentence simultaneously, resulting in improved performance in these cases.

6 Conclusion

During this project, we tried many methods to handle multitasking. We learned how to handle several inputs (by concatenating the sentences), how important it was to adopt a proper schedule (PAL), and that adding more parameters to a classifier does not always lead to an improvement in accuracy. We also learned that gradient treatment methods do not work well with very imbalanced datasets, but were still able to come up with a new method–**Gradient Compromise**–that takes inspiration from PAL scheduling and gradient vaccine in order to more rapidly finetune a multitask classifier during the first epochs. One main limitation of our work is the variance as we did not perform any hyperparameter tuning (except by hand) and so our results depend heavily on our random seed.

Future improvements include training larger PAL layers from scratch, pre-training the LSTM and initializing its weight correctly to leverage all sentence embeddings, and finetuning the hyperparameters. Additionally, we could implement specific approaches that are known to perform well on our datasets, such as Heisen routing for SST Heinsen (2022).

References

Shijie Chen, Yu Zhang, and Qiang Yang. 2021. Multi-task learning in natural language processing: An overview. *arXiv preprint arXiv:2109.09138*.

Michael Crawshaw. 2020. Multi-task learning with deep neural networks: A survey. *arXiv preprint arXiv:2009.09796*.

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*.

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

Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2020. SMART: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2177–2190, Online. Association for Computational Linguistics.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019. Multi-task deep neural networks for natural language understanding. *arXiv preprint arXiv:1901.11504*.

Łukasz Maziarka and Tomasz Danel. 2021. Multitask learning using bert with task-embedded attention. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6. IEEE.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. 2017. Mixed precision training. *arXiv preprint arXiv:1710.03740*.

Antoine Nzeyimana. 2022. Pytorch-pcgrad-gradvac-amp-gradaccum/antoine nzeyimana.

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

Zirui Wang, Yulia Tsvetkov, Orhan Firat, and Yuan Cao. 2020. Gradient vaccine: Investigating and improving multi-task optimization in massively multilingual models. *CoRR*, abs/2010.05874.

Jason Wei and Kai Zou. 2019. EDA: Easy data augmentation techniques for boosting performance on text classification tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6383–6389, Hong Kong, China. Association for Computational Linguistics.

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

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020b. Gradient surgery for multi-task learning. *Advances in Neural Information Processing Systems*, 33:5824–5836.