Transfer-learning through Fine-tuning: What Parameters to Freeze?

Axel Dinh Van Chi
Supervisors: Martin Jaggi, Matteo Pagliardini
Machine Learning and Optimization Laboratory, EPFL, Switzerland

***Abstract*—In this project, we investigate and try to get insights on the effects of fine-tuning a pre-trained BERT model by freezing most of its weights. We first verify that doing so can be competitive with a full fine-tuning, and we show that, even with very low amount of parameters to train, performances drop in low data regimes.**
In a second time, we show that choosing randomly the freezed parameters can already show results close to the traditional fine-tuning, and, surprisingly, quite consistenly. We also show that fine-tuning only on parameters with ”limited expressive power” regularizes the training.
Finally, we try to fine-tune BERT by reinitializing some of its parameters first. We show that the model can not create knowledge by only fine-tuning its biases, if its parameters have not been pre-trained. However, it is able to learn if only the biases are reinitialized, but its efficiency is highly task-dependant, showing that the biases are more or less important, depending on the task we wish to achieve.
The code used and the results can be found here.

I. Introduction

Nowadays, state-of-the-art language models can be extremely time and space consuming to train. A widely used method to relax the computations, is fine-tuning, here one uses an already pre-trained model and asks it to learn a new and specific task. This has the potential to greatly decrease the resources needed when the model already knows general features related to the task we wish to achieve. One of the most used pre-trained model for fine-tuning on natural language processing (NLP) tasks is BERT (Devlin et al. 2019). BERT is a transformer (Vaswani et al. 2017) based model which outperformed every other model on the GLUE benchmark (Wang et al. 2018) when it was published. As BERT is typically pre-trained on an unsupervised masked language modelling task on a large corpus, it makes it naturally task-unspecific, and thus a model of choice for fine-tuning.

Even if fine-tuning permits to reduce the computations, the question of the space allocation remains. Indeed, the base form of BERT takes already $\sim$ 400MB of memory to store, limiting the deployment and the storage of multiple models, especially for organizations with small amount of resources. A solution is to only fine-tune a subset of all the parameters, this way only those parameters need to be stored (along with one instance of BERT).

We will first test the performance, on some GLUE tasks, of two existing fine-tuning methods where the majority of the parameters are frozen during training. The first one, called the Bias-terms Fine-tuning (BitFit) method, was introduced in Ben-Zaken et al. 2020. This method, which only updates the parameters corresponding to the biases, showed good results on the GLUE tasks. In Lu et al. 2021, it has been suggested that fine-tuning transformers (Vaswani et al. 2017) only on the parameters corresponding to layer normalization layers (Ba et al. 2016) can bring good results on tasks which are not related to NLP. Because of it, we decided to use a fine-tuning method, which we will call the LayerNorm method, where only the layer normalization's parameters will be trained. We will also compare those two techniques with a fully fine-tuned model but in a low data regime. Doing so we hope to assess whether or not the training of less parameters allows the use of less data to learn a new task.
Then, we will fine-tune pre-trained BERT models, but this time the frozen parameters to fine-tune will be chosen in a purely random manner. From what we know, this has not been experimented

in the current litterature, but it could show that fine-tuning a model on a fraction of its parameters is more about the capacity of the model rather than the locality and the utility of the parameters in the model's architecture. That is to say, that the number of parameters being fine-tuned is more important than which parameters.
Finally, in Frankle et al. 2020, it has been shown that convolutional neural networks were able to learn simply by "scaling" and "shifting"the outputs of the layers, more precisely by fine-tuning only the batch normalization layers (Ioffe et al. 2015), and freezing everything else at their random initialization. Keeping the idea that a model can learn solely by "shifting" inputs, we will try to reinitialize the model's parameters before using the BitFit method. This way, we hope to assess if the model can learn by only "shifting" the linear combinations of the inputs learned during pre-training. Indeed, if the same performances can be achiveved, this would point the idea that the biases really "expose" the knowledge created through linear combinations of the inputs, i.e. the knowledge kept into the weights.

We enumerate here our different results:

- We show that the BitFit and LayerNorm fine-tunings can be competitive with a traditional fine-tuning.
- We show that fine-tuning on a fraction of the parameters do not allow better results in a low data regime.
- We show that the choice of parameters is important for fine-tuning. Training only on the bias-terms can be used as regularization for fine-tuning.
- We show that BERT cannot be fine-tuned with BitFit without pre-training. Making NLP stands out from other deep learning fields.
- We show that the importance of the pre-trained bias-terms can vary depending on the task.

II. Related Work

Different works already showed that it is possible to lower the space-allocation cost of transfer-learning when using BERT. As stated in the previous section, in Ben-Zaken et al. 2020 and Lu et al. 2021, the model is fine-tuned on a subset of all its parameters, allowing great gains in space allocation, but other methods exist.
In Houlsby et al. 2019, new layers, the adapters, are added inside the transformers. Those new layers can be trained on each task, while keeping the pre-trained model untouched. Doing so, they achieved good results on the GLUE benchmark by adding only 3.6% parameters. Even if this method proved to be efficient, 3.6% of the parameters can still be non-negligible when it comes to storage, e.g it represents $\sim$50MB when using BERT$_{LARGE}$.
In Diff-Pruning (Guo et al. 2020), the model is trained by learning what to add to the original parameters, then by imposing an L_0-norm to those additional terms, in order to encourage sparsity, they managed to obtain good results by modifying only 0.5% of all the parameters. This method allows more space saving than the previously mentioned, but the changed parameters will not be localized, and the training procedure can be more difficult due to the non-differentiablity of the L_0-norm.
Although the fine-tuning of pre-trained BERT models is a widely used method, it is known to be unstable, especially on small datasets (Mosbach et al. 2020, Zhang et al. n.d.). The instablility of a fine-tuning method is the difficulty that it has to get consistent results, 2 runs on different seeds might bring quite different performances, one might even fail to learn the task but not the other. For that matter, we will bring estimates of the standard deviations of our results.

III. Baselines

We present here the model and the different datasets we use to train it. We also present the training procedure that we will apply to get the results as well as the fine-tuning methods used during the project.

A. Model

We used the HuggingFace (Wolf et al. 2019) framework to fetch the pre-trained BERT-base model (Devlin et al. 2019) as well as the corresponding tokenizer. We decided to use the cased version of BERT, as the CoLA dataset's task needs to recognize grammatical errors. As the model performs a masked language modelling task, a Linear layer is appended at the end of the model, which allows us to use the model for classification tasks. As this layer is randomly initialized before fine-tuning, and therefore does not have any knowledge related to NLP, all its weights and biases will also be updated during training. Therefore, the classification layer should also be changed when switching between tasks.

B. Datasets

In order to evaluate the different methods implemented, we used four datasets of various sizes from the GLUE benchmark (Wang et al. 2018):

- 1 - The Corpus of Linguistic Acceptability (CoLA) dataset, where the task is to recognize if a sentence is grammatically acceptable. The training set contains 8,551 samples, and the developer set contains 527 samples. Performance is evaluated using the Matthew's correlation.
- 2 - The Recognizing Textual Entailment (RTE) dataset, where the task is to recognize whether or not the meaning of a first sentence is entailed into the second. The training set contains 2,490 samples, and the developer set contains 277 samples. Performance is evaluated using accuracy.
- 3 - The Stanford Sentiment Treebank (SST-2) dataset, made for a sentiment analysis task. The training set contains 67,349 samples, and the developer set contains 872 samples. Performance is evaluated using accuracy.
- 4 - Question-answering Natural Language Inference (QNLI), where the task is to determine whether or not the answer to a question is contained inside a context sentence. The training set contains 104,741 samples, and the developer set contains 5,461 samples. Performance is evaluated using accuracy.

Notice that the four tasks are classification tasks and that they might use distinct evaluation metrics.

As the test sets were not available to the public, we decided to use the developer/validation sets as test sets, and to split the training sets into our training and validation sets, with a split ratio of 0.8/0.2. This might lead to differences compared to analysis published on all the data available.

C. Training

We use the same training method as proposed in Ben-Zaken et al. 2020, the optimization is done using Adam (Kingma et al. 2015) with a batch size of 8 on 20 epochs. To fine-tune the whole model (Full fine-tuning), a learning rate of 3e-5 is used, and the fine-tunings using BitFit and LayerNorm are made with a learning rate searched in {1e-4, 5e-4, 1e-3}. The learning rate used might change for other methods, but we will precise it in time. We used some simple early stopping technique, where we returned the model with the best validation metric before the evaluation on the test set. As the number of epochs is quite low, we decided to make 10 evaluations on the validation set per epoch, to be sure to pick a good model. To get consistent results, we also made each experiment on 5 or 10 rounds. On the CoLA and RTE datasets, we ran 10 times the experiments, while we only ran it 5 times for the SST-2 and QNLI datasets, being limited by the computation times. Finally, no other tricks were used, such as schedulers and warm-up, even though this is known to increase performance (Mosbach et al. 2020), this might also be a reason why we did not get performances similar to previous works.

We would like to inform the reader about a mistake. In the plots of the different losses (Figure 9), we can observe spikes appearing at regular intervals. This is due to an error in the implementation, indeed, the models were set in training mode at the beginning of each epoch but were then set to evaluation mode after the first evaluation on the test set. This led the model to use Dropout only for the first 0.1% steps of each epoch, hence the periodic behaviour of those anomalies.
This error were found only late in the project, thus we could not run the analysis with the corrected framework in time. However, as we are mainly observing and comparing the behaviours of the models, we think that it should not invalidate the results.

D. Fine-tuning methods

Here we present the different fine-tuning techniques that we are going to use throughout this project. We denote by frozen parameters, the parameters which will not be updated during the optimization of the model, meaning that they will not change during the training procedure. In all those methods, the output layer's parameters, i.e. the classification's parameters, are being updated, but we will not mention it systematically.

1) Full: We will denote by Full fine-tuning the traditional fine-tuning method. Here, none of the parameters are frozen, hence the trained parameters account for 100% of all parameters.

2) BitFit: In the BitFit method from Ben-Zaken et al. 2020, only the bias terms are updated. The bias terms are the parameters which are added in each linear transformation. On the other hand, all the multiplicative terms, the weights, will be frozen. For example in the linear transformation:

$$y = Wx + \beta$$

where x is the input and y is the output, W corresponds to the weights and β to the bias terms. Here the trained parameters account for 0.1% of all parameters.

3) LayerNorm: We use the method in Lu et al. 2021, where only the output and layer normalization's parameters are updated. Note that in the study, they did not use it on NLP tasks. For more information about layer normalization see Section VI-A3. Notice that the layer normalization essentially ”rescales” and ”shifts” the inputs. Thus it does not bring extra cross-information between the inputs. In that way, we can say that using LayerNorm is similar to BitFit, but BitFit only ”shifts” the outputs of the different layers. We will sometimes denote this behaviour by the ”limited expressive power” of the parameters. Here the trained parameters account for 0.04% of all parameters.

4) Random: For this method of our own, the parameters to be frozen are selected randomly. This means that on all runs, different parameters will be trained. Note that this method is highly time inefficient (at least in our implementation), and that its random behaviour makes it quite unpractical. Here the trained parameters account for 0.1% of all parameters.

5) InitBias + BitFit: This method also uses the BitFit fine-tuning method, but all the biases terms are first reinitialized to 0 before training. The idea behind is to discard all the knowledge learned through the biases during the pre-training, keeping only the knowledge from the other parameters. Doing so, we hope to be able to assess whether or not the biases are a good proxy to ”expose” the knowledge kept into the pretrained model.

IV. Results

A. Testing BitFit and LayerNorm

To get a first glimpse at the results achievable when only fine-tuning on a fraction of the parameters, we used the BitFit and LayerNorm methods on our four datasets. They both outperformed the Full fine-tuning method on all datasets except on QNLI. However, the differences do not seem significant, with respect to the standard deviation, for all datasets and fine-tunings, with exception of the LayerNorm method on QNLI. Indeed, in this case we only achieved $88.0\%(\pm 0.3)$ accuracy, while the Full fine-tuning gave $89.0\%(\pm 0.5)$. Those results can be found in Table I and we give a visual representation in Figure 1.

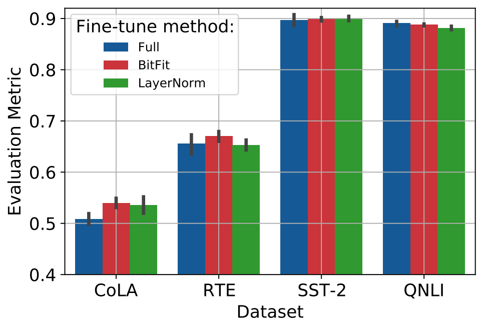
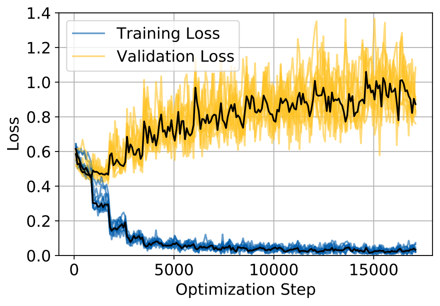
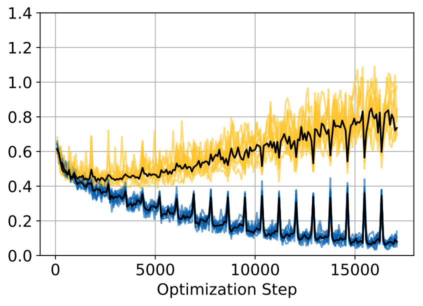


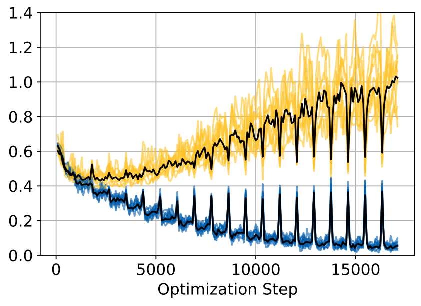
Fig. 1: Evaluation metrics on different datasets, using three existing fine-tuning methods. Notice that the minimum value for Matthew Correlation is 0, but here the y-axis starts at 0.4.



(a) Full Fine-tuning



(b) BitFit Fine-tuning



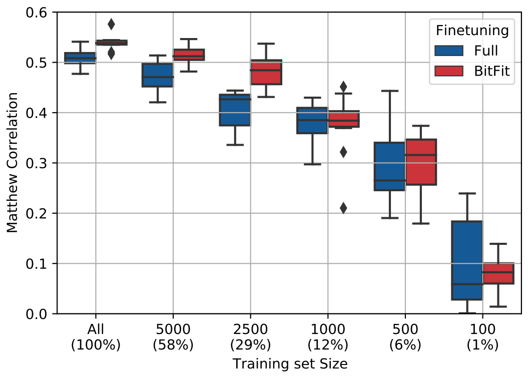
(c) LayerNorm Fine-tuning

Fig. 2: Losses from the fine-tuning of BERT using different methods with CoLA dataset. In color are the actual training and validation losses from each training round, in black are their corresponding medians. We observe a slower training for the BitFit and LayerNorm fine-tunings. We can also observe a periodic increase in the train loss, please see Section III-C for more information about it.

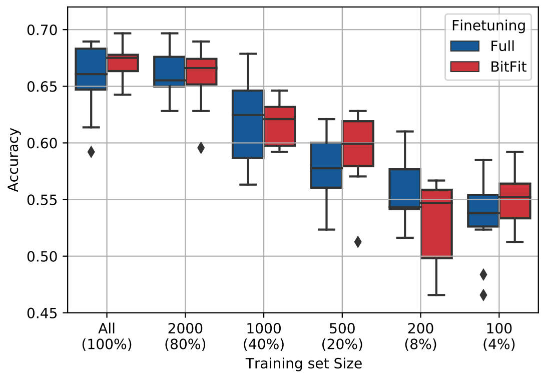
As stated in sections III-B and III-C, our results are not as good as in the original study of BitFit (Ben-Zaken et al. 2020), however we still get differences between the Full and BitFit fine-tunings that are in the same order of magnitude. For example Ben-Zaken et al. 2020 got a difference on the CoLA dataset of 2.6% and 0.2% on the SST-2 dataset, while we got 3% and 0.1% respectively, this shows that the results from Ben-Zaken et al. 2020 are indeed reproducible. Therefore, we conclude that BitFit and LayerNorm are both promising fine-tuning methods as they offer results which are comparable to a Full fine-tuning.
One last observation can be extracted out of the results. Interestingly, it seems like the optimization behaves differently when training only on a subset of the parameters or all of them. As it can be seen in Figure 2, the BitFit and LayerNorm methods show a more gentle slope when it comes to the training loss. Also, we can see that the validation metrics (Figure 10) tend to decrease after a certain number of steps when using the Full fine-tuning method, but it is not the case when using the two other techniques. This suggests that training a model using of a small fraction of its parameters tends to avoid overfitting. The reason why BitFit and LayerNorm are less prone to overfitting, seems to be the few degrees of freedoms that are left to the model, when fine-tuning only on a small subset of the parameters. Still, a question arises, is this reduction of freedom coming from the low number of parameters, or is it because the parameters trained with BitFit and LayerNorm can only ”rescale” and/or ”shift” inputs (see Section III-D) ? We will come back to it during the analysis of the Random fine-tuning method.

B. Low Data Regime

Models with a lot of parameters, hence a lot of parameters to fine-tune, are prone to overfitting when they are used on small datasets. For this reason, the use of smaller models is sometimes preferred when it comes to train on tasks with a low amount of data.
As in the BitFit and LayerNorm methods, only 0.1% of the parameters, or less, are fine-tuned, one could ask if it is possible to achieve better results when using only a fraction of the dataset, i.e. are BitFit and LayerNorm able to outperform the Full fine-tuning in the low data regime? Hence, as a first experiment, we tried to fine-tune BERT using the same techniques, but we decreased the number of training samples to see if the methods could still learn well from it. Results can be seen in Figure 3 and Tables II-III.
It can be observed that decreasing the number of training data still drastically decreases the performance of our model, even though it is learning on a small portion of its parameters. Furthermore, the decrease in performance seems to be approximately the same no matter the method used. Henceforth, even if fine-tuning a model with a lower number of parameters tends to diminish the impact of overfitting, as stated in the previous section, it seems like it is not enough to prevent the loss of performance when the number of training samples decays. It



(a) CoLA dataset



(b) RTE Dataset

Fig. 3: Effect on reducing the number of training samples on the test evaluation metric. For both figures, the results were obtained with a learning rate of 5e-4.

might be that having a large training dataset is the only way for a transformer based model to learn the inner distribution of a new and specific task. Which would mean that, no matter how much semantics of the english language the pre-trained model has learned, feeding it more data could only bring up its performances, up to a certain limit.

C. Random Fine-Tuning

In Section IV-A, we asked ourselves if the ability that BitFit and LayerNorm have to achieve a gentler optimization was a consequence of the low number of parameters to fine-tune or a consequence of the limited expressive power of those parameters in BERT's architecture. By ”limited expressive power of the parameters”, we refer to the fact that the parameters can only ”rescale” and/or ”shift” the inputs. In an attempt to answer this question, we will now analyses the fine-tuning of BERT, but this time, the subset of parameters which are not frozen will be chosen randomly along all the model's parameters. To make our new results comparable with the previous ones, we picked 0.1% of all parameters. In a more technical aspect, as most deep learning frameworks do not allow back-propagation on single entries of parameters, we used masks of 0 and 1 to freeze the gradients before the optimization step. We would like to emphasize the fact that this method is quite unpractical, indeed, the running times when using Random fine-tuning could be up to three times higher than when using other methods.

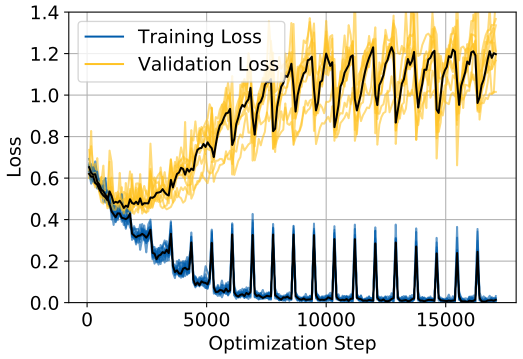


Fig. 4: Training and validation loss of fine-tuning BERT with our random method on the CoLA dataset. We notice that the train loss has a steeper slope than the BitFit and LayerNorm fine-tunings (Figure 2).

Coming back to the analysis, we can see that using the Random method gives sharper optimizations than when using the BitFit and LayerNorm methods (Figure 4). Also, the number of fine-tuned parameters are in the same order of magnitude for all the three fine-tuning methods, and the majority of the fine-tuned parameters in the Random method are non-bias parameters. Those elements suggest that it is indeed the limited expressive power of the parameters that allows the BitFit and LayerNorm methods to achieve a gentler optimization, and therefore, to reduce overfitting. Hence, fine-tuning only low-expression parameters can be used as some sort of regularization when it comes to fine-tuning a pre-trained transformer based model.

Concerning the actual performances, the results can be seen on Figure 5 (and in Table I), as the procedure was quite different this time we used learning rate in {5e-4, 1e-3, 5e-3}. Surprisingly, this random fine-tuning managed to get good results and even outperformed all the others on the SST-2 dataset. While our best result on previous methods was $90.0\%(\pm 0.4)$, this Random method achieved $90.4\%(\pm 0.5)$ of accuracy. However, the Random fine-tuning seems to have more difficulties to generalize what is has learned on the RTE task. Indeed, looking at the validation accuracies achieved on the RTE dataset (Figure 10), and noticing that lower learning rates gives lower results (Table I), we can assess that the performances dropped slightly on this particular dataset, even though it were able to learn from the training set (Figure 9).

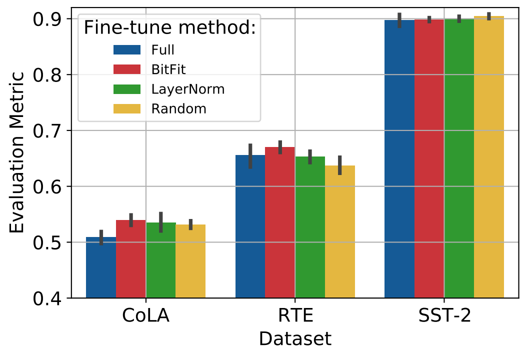


Fig. 5: Evaluation metrics on different datasets, using the fine-tuning methods introduced before. Again, the y-axis starts at 0.4, even though the Matthew correlation has a smallest value of 0.

Finally, we can notice that the losses obtained by training on the SST-2 dataset (Figure 9) are much more stable across the epochs compared to the other datasets, and that it is the only dataset where Random fine-tuning brings more stability on the validation metrics (Figure 10) than the other methods. This could have two reasons, firstly the size of training data, as fine-tunings in the large data regime are known to be more stable (Mosbach et al. 2020, Zhang et al. n.d.). Secondly, it could come from the complexity of the task to learn, indeed, if the inner distribution of the task to learn is simple, it is less likely to observe overfitting in the first place. A last experiment on the QNLI dataset using the Random method could have helped us determine which of the two has the most importance when fine-tuning. If we had not gotten good stability, then it would be of good use to assess the task's complexity before fine-tuning. Unfortunately, this experiment could not be done due to time (as one fine-tuning of BERT on QNLI can already take up to 8 hours without the Random method).

D. Reinitializing Parameters

In Frankle et al. 2020, it has been shown that randomly initialized convolutional neural networks could learn by only fine-tuning batch normalization layers (Ioffe et al. 2015). Also, in Lu et al. 2021, randomly initialized transformers are fine-tuned using the LayerNorm technique, and it has been showed that it could still learn, to some extent, various tasks. Yet, it has not been shown if similar results could be achieved on NLP related tasks. Those two results emphasize the idea that it is possible for a model to learn by simply "rescaling" and "shifting" its inputs. This means that the model is able to rearrange a randomly initialized distribution across the inputs, so that it helps it learn the actual distribution of the data. To assess whether or not this is true on NLP tasks, we will fine-tune BERT using BitFit, but this time we will first reinitialize the parameters.

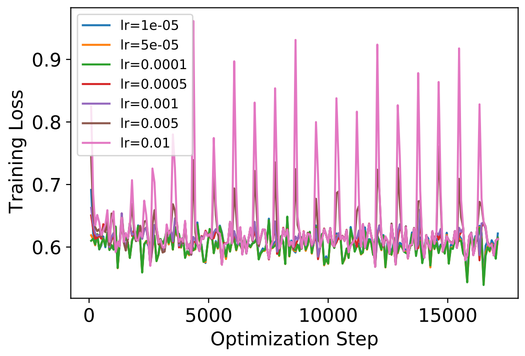


Fig. 6: Training losses using the BitFit method on CoLA, but reinitializing every parameters before training. We can see that none of the models converged. The results are given over many learning rates. Runs with higher learning rates only resulted in diverging optimizations.

First, we reinitialized every parameters of the model, and then tried to apply the BitFit fine-tuning on various learning rates from 1e-5 to 1. As a result, none of the models were able to learn the CoLA task in the slightest (Figure 6). This indicates that,

in the case of NLP, the knowledge learned from pre-training is essential in order to learn new tasks, and that only "shifting" random projections to get an approximation of the inner distribution of the task is not sufficient. Hence, we can safely say that the information created through linear combinations of the inputs, and so especially in the multi-head attention layers, is crucial in our setting.
Then, we only reinitialized the bias-terms of the pre-trained model to 0 before fine-tuning using BitFit (Init+BitFit method). Doing so, we hope to assess whether or not biases play a key role in the comprehension of the language semantics, or if their main utility is to "expose" knowledge from the other parameters. That is to say, whether or not reinitializing the biases can destroy the pre-trained knowledge of the model. Results can be found in Figure 7 and Table I.

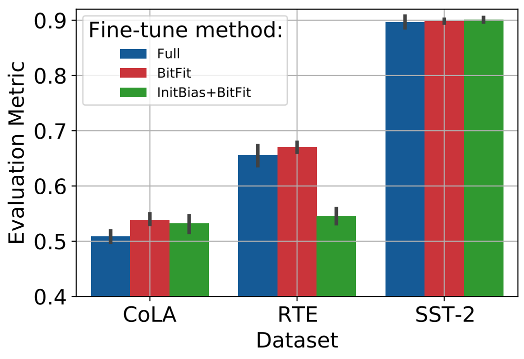


Fig. 7: Evaluation metrics on different datasets when using the InitBias+BitFit method. We show the runs for the best learning rate. Note that we have discarded the one failed run (out of 10) on the CoLA dataset, this is why the results are different compared to Table I. Once again, the y-axis starts at 0.4, even though the Matthew correlation has a smallest value of 0.

Here, we get good performance on the SST-2 dataset, as well as slightly worst results for the CoLA dataset (note that one instance did not converge, hence the high variance and the low mean value in Table I). However, this time the model nearly did not learn the RTE task, indeed, we observe an accuracy of $54.5\%(\pm 2.4)$, which is close to randomness, while BitFit achieved $67\%(\pm 1.6)$. Given those various results, it seems like the importance of the bias-terms can not be assessed easily, but we will train to bring an explanation as of why we get those results. In the two datasets where the model could still learn fairly well, i.e. SST-2 and CoLA, it might be that the semantic learned across different tokens is less important for the comprehension of the task (see Section III-B for information about the tasks). Indeed, sentiment analysis (SST-2) can only ask to recognize a word to be efficient (e.g if the words 'happy' or 'sad' appear in the sentence), but RTE's task asks for a fine comprehension of the semantics of two sentences. This could explain why reinitializing the biases to 0 before fine-tuning makes the training on RTE more difficult. As biases are also a part of the attention mechanism (Section VI-A), setting them to 0 might destroy the finest semantics learned during pre-training while still allowing the comprehension of the individual embeddings. If this is the very reason for the bad results on RTE, then it would mean that biases do play a key role in the comprehension of the language semantics.
To summarize this section, we have first showed that, given an NLP task, pre-training a transformer is mandatory before fine-tuning only on its biases. This highlights the fact that NLP stands out from others deep learning fields, as it has been shown that random transformers can be enough to fine-tune on non-NLP tasks. Then, we showed that the importance of the knowledge contained inside the bias-terms depends on the task aimed through fine-tuning. We supposed that it was related to the task's dependance on cross-relations between the tokens, i.e. its dependance to a fine comprehension of the sentence's semantic, but more work would be needed in order to assess if this affirmation is right or not.

V. Conclusion

In this project, we tried to bring some insights about the choice of parameters to train when it comes to fine-tuning a pre-trained BERT model on a fraction of its parameters. We have shown that using only the bias-terms, and more generally parameters with a low expressive power, is competitive with the traditional fine-tuning technique and that it can help reduce overfitting. We assessed the importance of having a sufficiently large dataset even when fine-tuning on a small subset of the parameters. Indeed, it seems like reducing the overfit is not sufficient to get better performances in the low data regime, emphasizing the idea that there is a specific distribution to learn when fine-tuning on a new task. Then, we showed that the knowledge acquired during the pre-training of transformers is essential for the fine-tuning of BERT on NLP tasks. This makes NLP stand out from other deep learning fields, as it has been shown that a transformer based model could learn from non-NLP tasks, simply by ”rescaling” and ”shifting” random projections of its parameters. Finally, we have shown that the importance of the pre-trained bias-terms in the comprehension of a new task, is directly related to the task itself. We suggested an explanation as of why this is the case which emphasizes the idea that the biases are not only trained to ”expose” the knowledge of the other parameters, but that they play a role in the fine understanding of the semantics of the english language.

We now propose some ideas to go further. Firstly, one could try to apply BitFit with reinitialized biases on more datasets, this should bring more insights about the biases' utility and could maybe give us the reason why trained bias-terms is important for some tasks and not others. Secondly, the same analysis could be made, but reinitializing only on a subset of the biases, this way, it could be shown if all biases are important in the semantic comprehension or only some of them (e.g. one could only set to 0 the biases in the first layers of BERT). Finally, in the methods presented here, all the outputs, and nearly all the gradients, of each layers still need to be computed, hence we did not witness a huge save in the runtimes. In Liu et al. 2021, a method was proposed to overcome this issue. Hence by combining the space-saving of the BitFit method with their method, it could be possible to achieve a fine-tuning which would be both time-saving and space-saving. This is left for future work.

References

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton (July 2016). “Layer Normalization”. In: URL: http://arxiv.org/abs/1607.06450.

Ben-Zaken, Elad, Shauli Ravfogel, and Yoav Goldberg (2020). “BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models”. In:

Devlin, Jacob et al. (Oct. 2019). “BERT: Pre-training of deep bidirectional transformers for language understanding”. In: *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*. Vol. 1. Association for Computational Linguistics (ACL), pp. 4171–4186. ISBN: 9781950737130. URL: https://github.com/tensorflow/tensor2tensor.

Frankle, Jonathan, David J. Schwab, and Ari S. Morcos (Feb. 2020). “Training BatchNorm and Only BatchNorm: On the Expressive Power of Random Features in CNNs”. In: URL: http://arxiv.org/abs/2003.00152.

Guo, Demi, Alexander M. Rush, and Yoon Kim (Dec. 2020). “Parameter-Efficient Transfer Learning with Diff Pruning”. In: URL: http://arxiv.org/abs/2012.07463.

Houlsby, Neil et al. (Feb. 2019). “Parameter-Efficient Transfer Learning for NLP”. In: *36th International Conference on Machine Learning, ICML 2019* 2019-June, pp. 4944–4953. URL: http://arxiv.org/abs/1902.00751.

Ioffe, Sergey and Christian Szegedy (Feb. 2015). “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *32nd International Conference on Machine Learning, ICML 2015*. Vol. 1. International Machine Learning Society (IMLS), pp. 448–456. ISBN: 9781510810587. URL: https://arxiv.org/abs/1502.03167v3.

Kingma, Diederik P. and Jimmy Lei Ba (Dec. 2015). “Adam: A method for stochastic optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR. URL: https://arxiv.org/abs/1412.6980v9.

Liu, Yuhan, Saurabh Agarwal, and Shivaram Venkataraman (Feb. 2021). “AutoFreeze: Automatically Freezing Model Blocks to Accelerate Fine-tuning”. In: URL: http://arxiv.org/abs/2102.01386.

Lu, Kevin et al. (Mar. 2021). “Pretrained Transformers as Universal Computation Engines”. In: URL: http://arxiv.org/abs/2103.05247.

Mosbach, Marius, Maksym Andriushchenko, and Dietrich Klakow (June 2020). “On the Stability of Fine-tuning BERT: Misconceptions, Explanations, and Strong Baselines”. In: URL: http://arxiv.org/abs/2006.04884.

Vaswani, Ashish et al. (June 2017). “Attention is all you need”. In: *Advances in Neural Information Processing Systems*. Vol. 2017-December. Neural information processing systems foundation, pp. 5999–6009. URL: https://arxiv.org/abs/1706.03762v5.

Wang, Alex et al. (Apr. 2018). “GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding”. In: *7th International Conference on Learning Representations, ICLR 2019*. URL: http://arxiv.org/abs/1804.07461.

Wolf, Thomas et al. (Oct. 2019). “HuggingFace’s Transformers: State-of-the-art Natural Language Processing”. In: URL: http://arxiv.org/abs/1910.03771.

Zhang, Tianyi et al. (n.d.). *REVISITING FEW-SAMPLE BERT FINE-TUNING*. Tech. rep. URL: https://github.com/zihangdai/xlnet/issues/96.

VI. Appendix

A. BERT

1) Architecture: As the task we want to perform is classification, we need the encoder part of BERT (Devlin et al. 2019). When the embeddings are provided, BERT is composed of N transformer layers (Vaswani et al. 2017), each of them being made of a self-attention layer followed by a feed-forward layer, which is simply a small multi-layer perceptron. Right after each of the 2 layers a skip-connection and layer normalization are applied. Here is a visual representation:

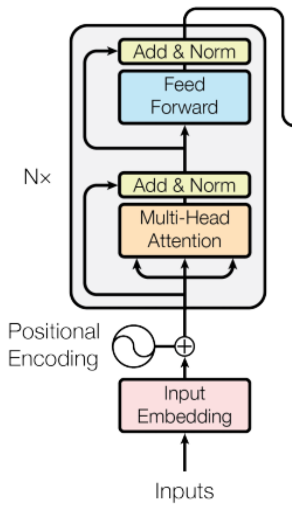


Fig. 8: Architecture of BERT's encoder.

2) Attention Mechanism: The multi-head attention layer is composed of many self-attention heads (12 in our case). In a self-attention head, mappings from the tokens to $queries$, $keys$ and $values$ are learned using Linear layers to create matrices Q, K and V respectively. Then the cross-products of the $queries$ with the $keys$ are passed into a softmax function, before being multiplied by the $values$:

$$Z = softmax(\frac{QK^T}{\sqrt{d}})V$$

where d is the dimension of the key vectors. Finally, for the Multi-Headed attention mechanism, multiple such mappings are learned, giving multiple matrices Z, those matrices are then concatenated and a Linear layer can be applied to obtain an output of the same shape as the input. In our case the number of attention heads (12) divides exactly the dimension of the embeddings (768), hence by taking $d = \frac{768}{12} = 64$, this last Linear layer can be avoided.
Note that, as this layer contains Linear layers, bias-terms are contained inside th multi-head attention.

3) Layer Normalization: Let $x \in \mathbb{R}^{S \times N}$ be a sample of features inside a batch, where S is the number of tokens and N the embeddings' size. Layer normalization (LayerNorm) first computes the mean and variance over each token $x^{(j)}$:

- Mean: $\mu^{(j)} = \frac{1}{N}\sum_{i=1}^{N} x_i^{(j)}$
- Variance: $(\sigma^{(j)})^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i^{(j)} - \mu^{(j)})^2$

and then it normalizes the features with:

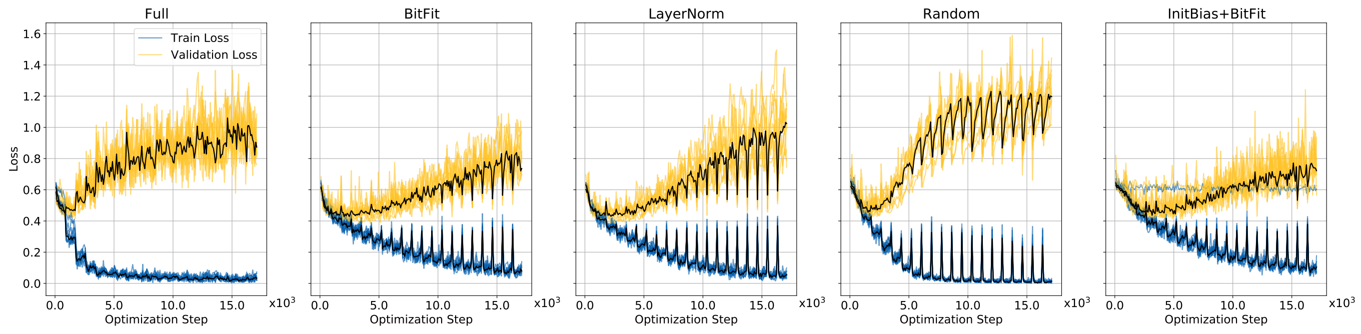
$$\hat{x}_i^{(j)} = \frac{x_i^{(j)} - \mu^{(j)}}{\sqrt{(\sigma^{(j)})^2 + \epsilon}}, \forall i \in \mathbb{R}^N, \forall j \in \mathbb{R}^S$$

where ϵ is simply a small smoothing term to avoid division by 0. Finally, the output is given by:

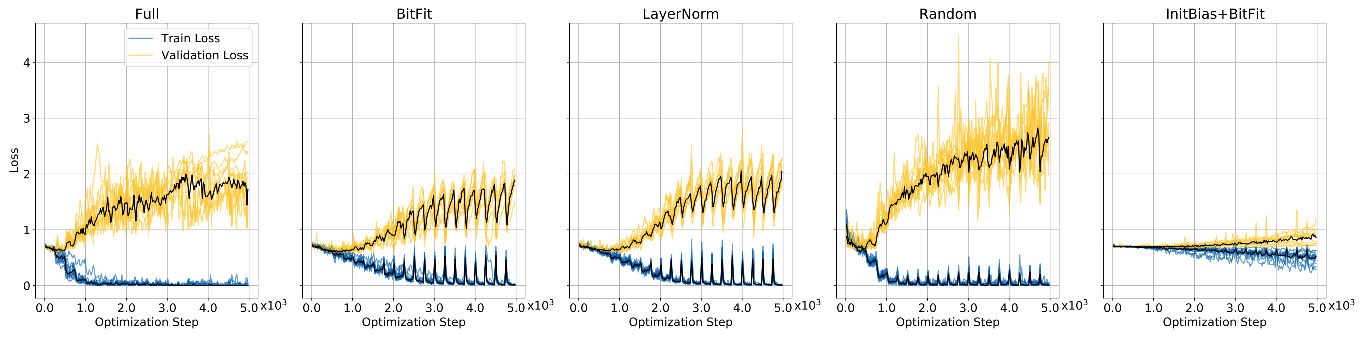
$$y_i^{(j)} = w_i \hat{x}_i^{(j)} + b_i, \forall i \in \mathbb{R}^N, \forall j \in \mathbb{R}^S$$

The y_i's and b_i's are parameters learned during training. As each transformer contains 2 LayerNorm layers, and as the BERT architecture we use has 12 transformers, they account for $48N$ parameters (without counting the LayerNorm used for the embeddings).

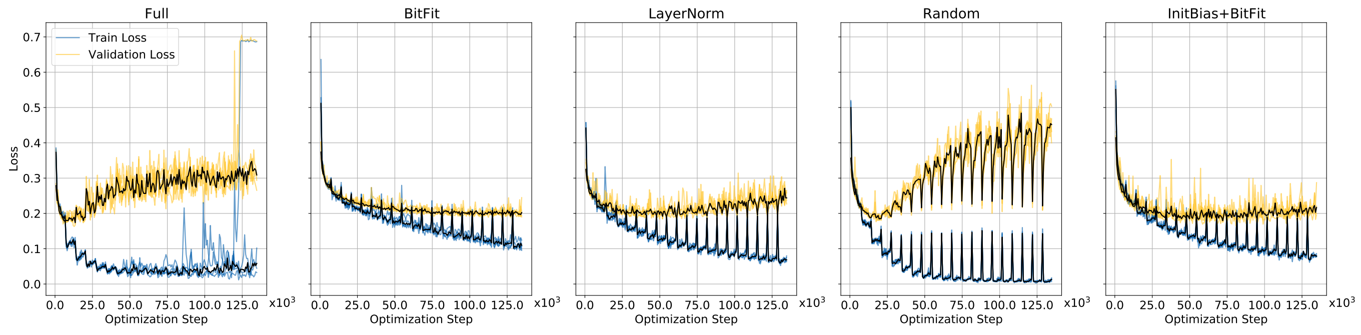
B. Figures



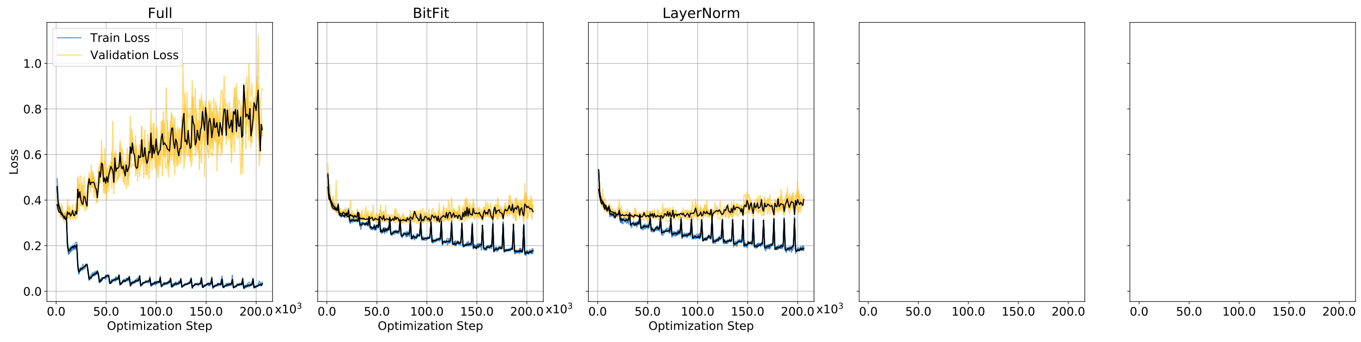
(a) CoLA Dataset



(b) RTE Dataset

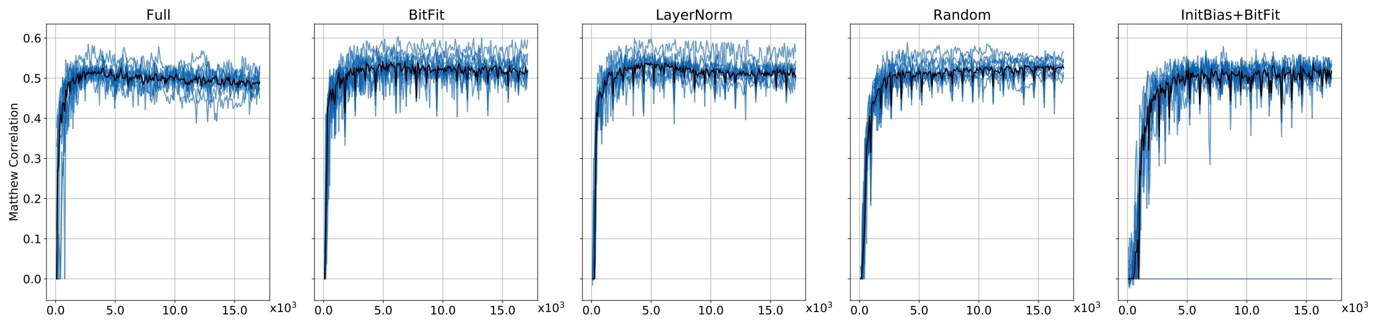


(c) SST-2 Dataset

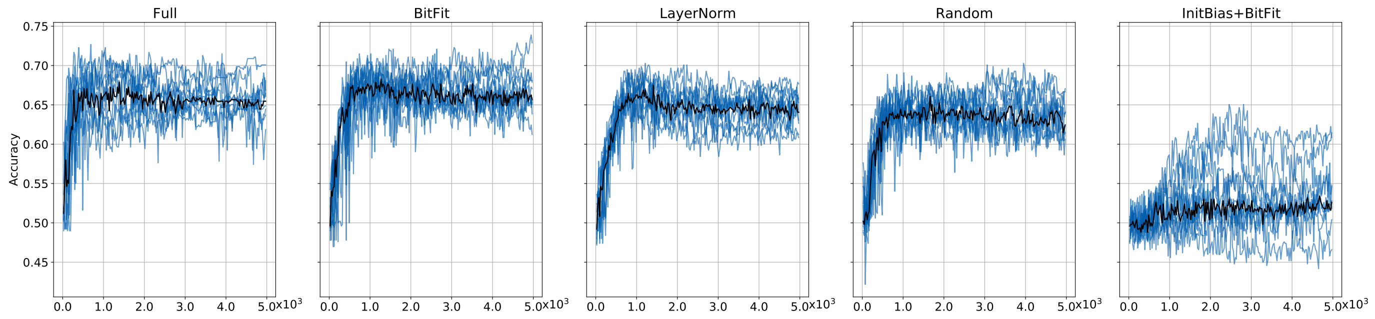


(d) QNLI Dataset

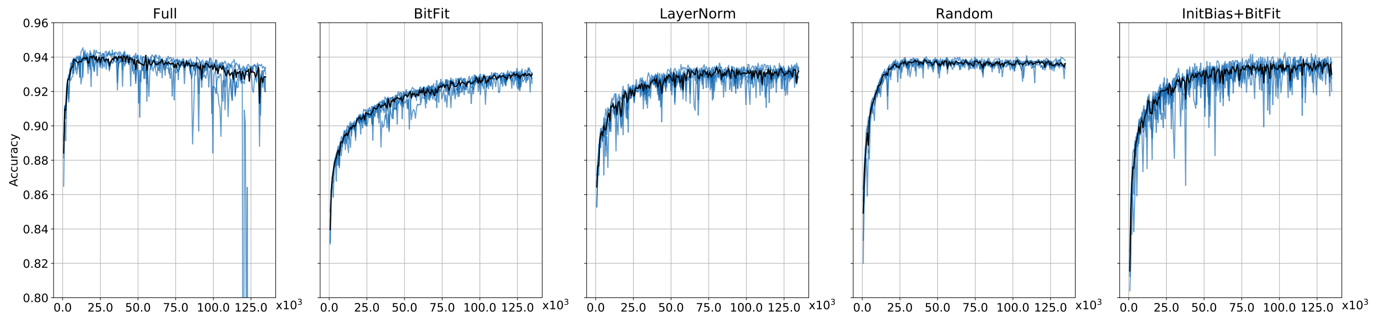
Fig. 9: **Training and validation losses over training on different datasets.** The actual losses obtained per round are in color, the black lines are the medians over the rounds. We observe that the training losses for BitFit and LayerNorm converge slower than for the Full and Random fine-tunings. Also notice that BitFit has trouble learning on the RTE dataset with reinitialized biases (InitBias), and that the Full fine-tuning led once to a total collapsing of the model on the SST-2 dataset. The learning rates were chosen given the best results per dataset and method in Table I. See the Section III-C for more information about the periodic ”spikes”.



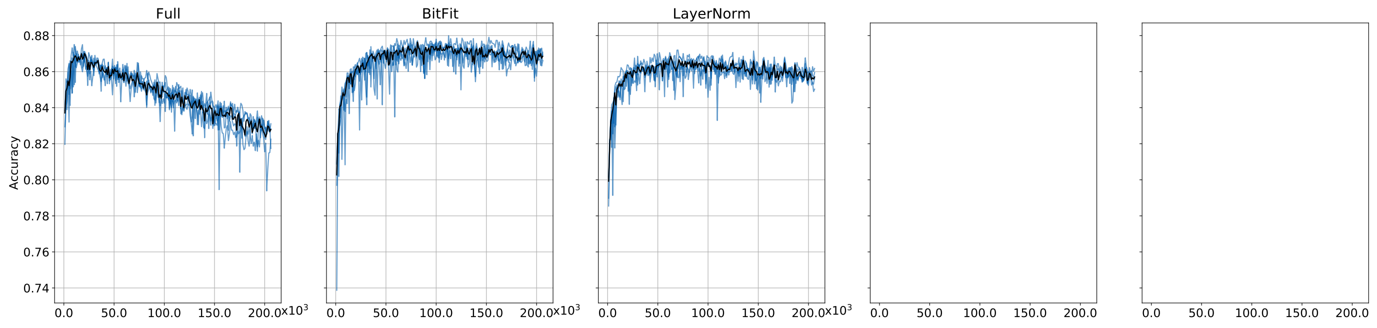
(a) CoLA Dataset



(b) RTE Dataset



(c) SST-2 Dataset



(d) QNLI Dataset

Fig. 10: **Validation metrics over training on different datasets.** The actual validation metrics obtained per round are in blue, the black line is the median over the rounds. We observe that, over the same number of steps, the validation metric with full fine-tuning tends to diminish at some point (it can be well seen on QNLI). It is not the case when we fine-tune less parameters. We can also notice the collapsing of one model for the Full fine-tuning on the SST-2 dataset, the accuracy actually dropped to 0.5, without increasing again. Finally, one instance of BitFit with reinitialized biases did not manage to learn on the CoLA dataset. The learning rates were chosen given the best results per dataset and method in Table I.

C. Tables

	% Params	CoLA	RTE	SST-2	QNLI
Full:					
lr = 3e-5	100%	50.9(±1.7)	65.6(±3.2)	89.7(±1.4)	**89.0**(±0.5)
BitFit:					
lr = 1e-4		52.6(±1.4)	65.3(±2.0)	89.8(±0.4)	88.4(±0.3)
lr = 5e-4	0.1%	**53.9**(±1.6)	**67.0**(±1.6)	89.3(±0.4)	88.8(±0.2)
lr = 1e-3		53.6(±1.8)	65.6(±3.9)	89.6(±0.7)	88.8(±0.4)
LayerNorm:					
lr = 1e-4		49.7(±1.3)	64.8(±1.1)	88.3(±0.2)	87.6(±0.1)
lr = 5e-4	0.4%	53.5(±2.7)	65.3(±1.8)	90.0(±0.6)	88.0(±0.3)
lr = 1e-3		52.9(±2.1)	64.5(±2.0)	89.8(±0.7)	88.0(±0.3)
Random:					
lr = 5e-4		49.1(±2.1)	60.7(±2.2)	89.6(±0.6)	···
lr = 1e-3	0.1%	53.1(±1.1)	60.9(±1.7)	**90.4**(±0.5)	···
lr = 5e-3		53.1(±2.4)	63.7(±2.4)	89.1(±0.5)	···
InitBias+BitFit					
lr = 1e-4		33.3(±4.9)	54.5(±2.4)	88.7(±0.2)	···
lr = 5e-4	0.1%	42.9(±13.9)	53.9(±3.5)	90.1(±0.5)	···
lr = 1e-3		48.8(±14.0)	52.6(±2.7)	85.4(±3.5)	···
lr = 5e-3		7.9(±12.1)	49.7(±2.7)	57.7(±15.2)	···

TABLE I: **Results for different fine-tuning methods.** The values are the mean test accuracies achieved with their standard deviations, for information about the training procedure, see Section III-C

	CoLA					
	All (100%)	5000 (58%)	2500 (29%)	1000 (12%)	500 (6%)	100 (1%)
Full						
3e-5	50.9(±1.7)	47.1(±3.3)	40.5(±4.0)	37.6(±4.6)	28.9(±7.8)	10.2(±9.3)
BitFit						
lr = 1e-4	52.6(±1.4)	49.5(±2.6)	42.7(±3.5)	32.3(±9.5)	19.9(±10.9)	4.7(±4.5)
lr = 5e-4	**53.9**(±1.6)	51.3(±1.8)	48.1(±3.3)	37.4(±6.8)	29.8(±6.7)	7.9(±3.6)
lr = 1e-3	53.6(±1.8)	48.1(±6.5)	48.1(±2.0)	**42.2**(±3.4)	**31.4**(±6.8)	**11.9**(±6.9)
LayerNorm						
lr = 1e-4	49.7(±1.3)	44.7(±3.3)	41.0(±5.4)	30.8(±9.1)	20.6(±8.1)	3.2(±3.3)
lr = 5e-4	53.5(±2.7)	51.1(±1.3)	46.7(±2.2)	37.8(±5.5)	28.0(±6.5)	7.6(±6.6)
lr = 1e-3	52.9(±2.1)	**51.6**(±2.0)	**48.2**(±2.7)	40.5(±3.1)	30.8(±5.0)	8.5(±6.9)

TABLE II: **Test accuracies for different fine-tuning methods, while decreasing the number of training samples.** The experiments were done on the CoLA dataset. The values are the mean of the test accuracies over 10 rounds, along with their standard deviation.

	RTE					
	All (100%)	2000 (80%)	1000 (40%)	500 (20%)	200 (8%)	100 (4%)
Full						
3e-5	65.6(±3.2)	66.1(±2.1)	62.1(±4.0)	57.7(±3.0)	**55.7**(±2.9)	53.4(±3.6)
BitFit						
lr = 1e-4	65.3(±2.0)	64.6(±1.1)	62.3(±1.7)	57.5(±2.9)	54.5(±2.7)	54.2(±2.4)
lr = 5e-4	**67.0**(±1.6)	65.8(±2.8)	61.7(±2.1)	59.3(±3.5)	53.0(±3.6)	**55.1**(±2.5)
lr = 1e-3	65.6(±3.9)	**66.3**(±2.8)	**62.7**(±2.5)	**59.5**(±2.1)	55.1(±4.1)	54.3(±3.9)
LayerNorm						
lr = 1e-4	64.8(±1.1)	64.0(±1.2)	59.3(±2.2)	55.5(±2.6)	52.9(±4.4)	53.1(±3.3)
lr = 5e-4	65.3(±1.8)	64.4(±1.8)	60.8(±2.8)	58.5(±2.5)	53.9(±3.8)	52.7(±4.8)
lr = 1e-3	64.5(±2.0)	64.5(±1.4)	61.7(±2.4)	59.4(±1.7)	55.0(±5.1)	52.6(±2.0)

TABLE III: **Test accuracies for different fine-tuning methods, while decreasing the number of training samples.** The experiments were done on the CoLA dataset. The values are the mean of the test accuracies over 10 rounds, along with their standard deviation.