PRIVACY PRESERVING MACHINE LEARNING

ECOLE CENTRALE LILLE

MASTER DATA SCIENCE

Differentially Private Fine-tuning of Language Models

Authors:
Ayoub Youssoufi
Yassin El Hajj Chehade

Professor:
Aurélien Bellet

January 10, 2023

1 Introduction

Fine-tuning pretrained language models to supervise downstream tasks has become common in the field of natural language processing where we used a model trained on a large dataset and then we adapt it to our current problem.

The main issue comes from the fact that most or all deep learning models are very sensitive to information leakage. Hence, we will train the model to obey the DP(differentially private) conditions.

In this paper the authors introduce a new meta-framework to fine tune the model while respecting the privacy condition, but with only training a small number of parameters, about 0.1% of it. Then, they compare the result with the known DPSGD method and show that one can privately fine-tune models whose utility approaches that of non-private models with only pre-training a less number of parameters, and this will be especially true for larger models.

2 Paper content

2.1 DPGSD method:

The typical example in machine learning has been that once there is an available dataset, we can build a predictive model from that dataset. However, this data is distributed among multiple parties such as financial data which is distributed between the banks and financial agencies. In a such scenario, it is very important to design an algorithm that can learn from the overall data while preserving the privacy of the individual parties. Data privacy is strong notion of privacy. Briefly, it ensures that the output of the algorithm does not allow any individual of the data to be identified. In DP stochastic gradient descent, the clipping and random noise addition are the key points of preserving the privacy of the dataset. With this approach, the most basic fine-tuning strategy is to train all the parameters of the model using DPSGD. Fine-tuning framework using the DPSGD is performed by decomposing the weight matrix $W_(t-1)$ during the iteration t to find low-rank product LR. The gradients computed on L and R are then projected back to the full parameter space to perform descent step. Hence, This method requires updating all the parameters and storing the different fine-tuning copy per task which is costly in memory. Therefore, the author proposed approach based on updating only a tiny fraction of the parameters which is discussed in the next sections.

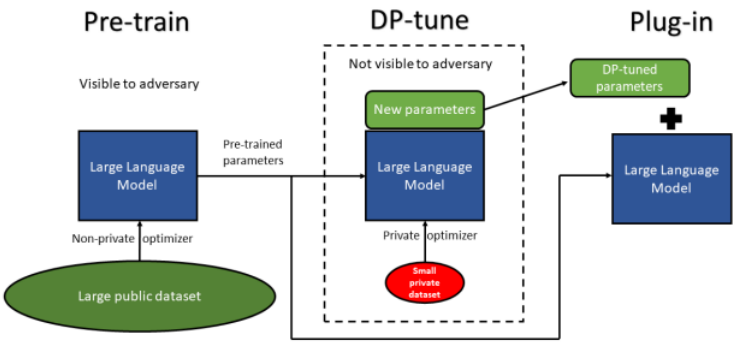

Figure 1: Illustration of the pre-training and private fine-tuning a model

2.2 Efficient parameter methods:

The author applied different methods to reduce the number of parameters in fine-tuning :

- Low Rank Adaptation (LoRA): the neural network contains many dense layers which perform matrix multiplication. The weight matrices in these layers typically have full-rank. For a pre-trained weight matrix W_0, we constrain its update by representing the matrix with a low-rank decomposition. Hence, $W_0 = W_0 + LR$ with L and R respectively of size $(d \times r)$ and $(r \times k)$. During training, W_0 is frozen and does not receive gradient updates, while L and R contain trainable parameters. (Figure 2)
- Fine-Tuning via adapters : We add to the pretrained model an adapter layers after each attention and feed-forward layer. Given an input x an adapter layer A performs : $A(x) = U(\tau(D(x))) + x$ adapter layers are designed to have few parameter by having small bottleneck dimension. Hence, when tuning, the

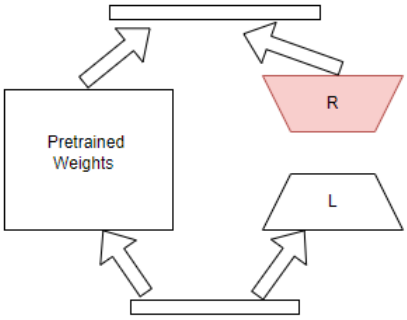

Figure 2: The fine-tuning using LoRA representation, we only train using L and R while W_0 is frozen

parameters of the original model are frozen, and only parameters of the adapter layers, as well as layer normalisation are modified.

- Fine-Tuning via compacter: An extension of adapter is compacter, which essentially parameterize the adapter layers using Kronecker products. The matrices U and D are replaced by tensor products of small matrices. $M_l = \sum A_i \otimes (S_i^l T_i^l)$

2.3 Experiments & results:

IN this paper, the author evaluates the utility of the efficient parameters methods for fine-tuning DP models in both NLU(natural language understanding) and NLG(natural language generation) tasks, using RoBERTa and GPT models, respectively. At the end, he tried to study the influence of tuning the hyperparameters of the LoRA method, and obtained interesting results that open up new avenues of study for the future.

2.3.1 Comparison between different DP fine-tuning methods

First, we compare the different fine-tuning methods defined above using state of the art models from the BERT family. Here, we did the evaluation for NLU on RoBERTa, a pre-trained model on a public data set collected from the web, as we can notice that in this paper, privacy is applied only on the fine-tune part and not on the training process. We test the model on the version, RoBERTa-Base which has 125M parameters and RoBERTa-Large with 355M parameters.

Here, for the choice of hyperparameters, the authors used suggestions from the original papers of the meta-framework papers, for LoRA and others methods.

Method		MNLI	SST-2	QQP	QNLI	Avg.	Trained params
Full	w/o DP	87.6	94.8	91.9	92.8	91.8	100%
	DP	53.1	82.6	74.4	63.9	68.5	
LoRA	w/o DP	87.5	95.1	90.8	93.3	91.7	0.24%
RGP[5]	DP	80.1	91.6	85.5	87.2	86.1	100%
Adapter	DP	83.4	**92.5**	85.6	**87.5**	**87.3**	1.4% ($r = 48$)
Compacter	DP	82.6	92.3	84.7	85.1	86.2	0.055% ($r = 96$, $n = 8$)
LoRA	DP	**83.5**	92.2	**85.7**	87.3	87.2	0.94% ($r = 16$)

Figure 3: Accuracy for fine-tuning downstream tasks with RoBERTa base

In table 3, and table 4 the author tests the accuracy of the method on MNLI, SST-2, QQP and QNLI tasks from NLU, with $\epsilon = 6.7$ and $\delta = 1e - 5$. The choice of hyperparameters for each method is mentioned in the table. In bold is indicated the best accuracy with DP. We can see that the different efficient-parameter methods achieve a very good result, and even the best accuracy , with only a tiny fraction of trained parameters.

One can notice that significant improvements are given while evaluating on larger models. Especially for the larger model, RoBERTa-Large, LoRA outperforms all other methods. Similarly, the RGP mthod which uses all trained parameters was outperformed by all other efficient-parameters methods.

The result for DPSGD on RoBERTa-Large was not reported due to memory costs and running time issues.

Method		MNLI	SST-2	QQP	QNLI	Avg.	Trained params
Full	w/o DP	90.2	96.4	92.2	94.7	93.4	100%
LoRA	w/o DP	90.6	96.2	91.6	94.9	93.3	0.23%
RGP	DP	86.1	93.0	86.7	90.0	88.9	100%
Adapter	DP	87.7	93.9	86.3	90.7	89.7	1.4% ($r = 48$)
Compacter	DP	87.5	94.2	86.2	90.2	89.5	0.053% ($r = 96$, $n = 8$)
LoRA	DP	**87.8**	**95.3**	**87.4**	**90.8**	**90.3**	0.94% ($r = 16$)

Figure 4: Accuracy for fine-tuning downstream tasks with RoBERTa large

2.3.2 Fine-tuning for NLG

Secondly, the author after comparing different fine-tuning methods on NLU tasks, deduce that approximately all efficient-parameter methods leads to a roughly similar result. For this reason, and in order to show that private fine-tuning is competitive with non-private fine-tuning for NLG tasks, he used only LoRA method.

Method	Val perp	BLEU	MET	TER
GPT-2-Small + DP	3.82	38.5	0.34	0.53
GPT-2-Medium + DP	3.30	42.0	0.36	0.51
GPT-2-Large + DP	3.10	43.1	0.36	**0.5**
GPT-2-XL + DP	**3.00**	**43.8**	**0.37**	**0.5**
GPT-2-Medium	2.67	47.1	0.39	0.46
GPT-2-Large	2.89	47.5	0.39	0.45
GPT-2-XL	2.83	48.1	0.39	0.46

Figure 5: Metrics on the DART dataset

In table 5 the author tests the validation perplexity, BLEU, MET and TER metrics, where $\epsilon = 6.8$, $\delta = 1e-5$, on the DART dataset. DART is an open-domain data-to-text that consists of 62k training samples, 6.9k validation samples, and 12k test samples. The model used is GPT-2 where GPT-2-Small contains 117M parameters, GPT-2-Medium with 345M parameters, GPT-2-Large with 774M parameters and GPT-2-XL with 1.5B parameters.

For the result, we conclude that private fine-tuning with parameter-efficient methods performs well and is close to non private results, with notable improvements as the model gets larger, in all metrics.

2.3.3 Influence of Different Privacy parameters and hyper-parameters

Finally, in order to understand the influence of the hyperparameters and DP parameters in fine-tuning models, the author performed some experiments for this purpose.

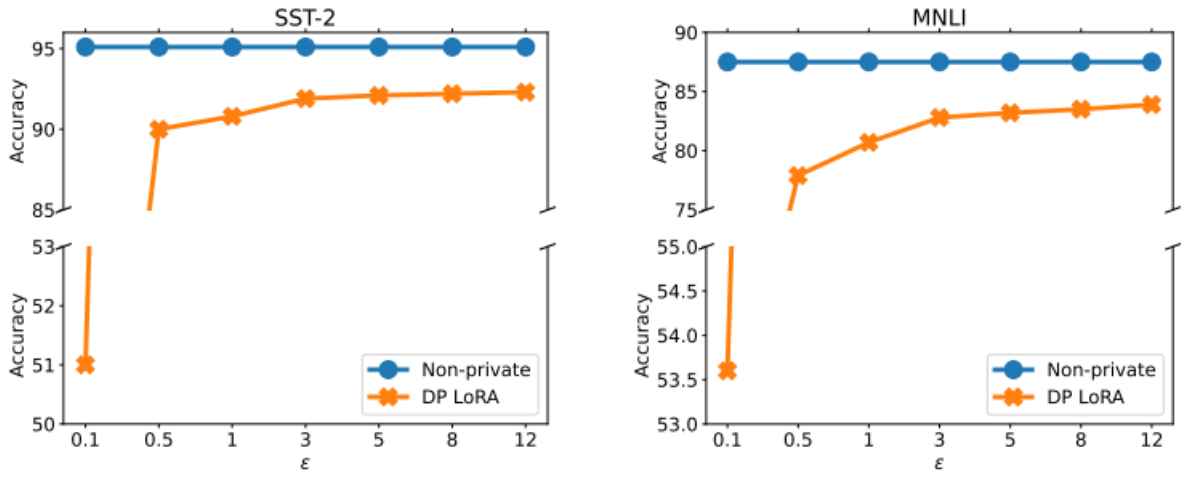

Figure 6: Test accuracy of fine-tuning the RoBERTa-Base model with respect to ϵ

In figure 6 we can see the influence of ϵ on the accuracy of the model. For DP-LoRA in both SST-2 and MNLI data set, the accuracy improves with increasing epsilon, even for a very tight parameter $\epsilon = 0.5$ we got a good result, although such value is not often tested when training model with DP.

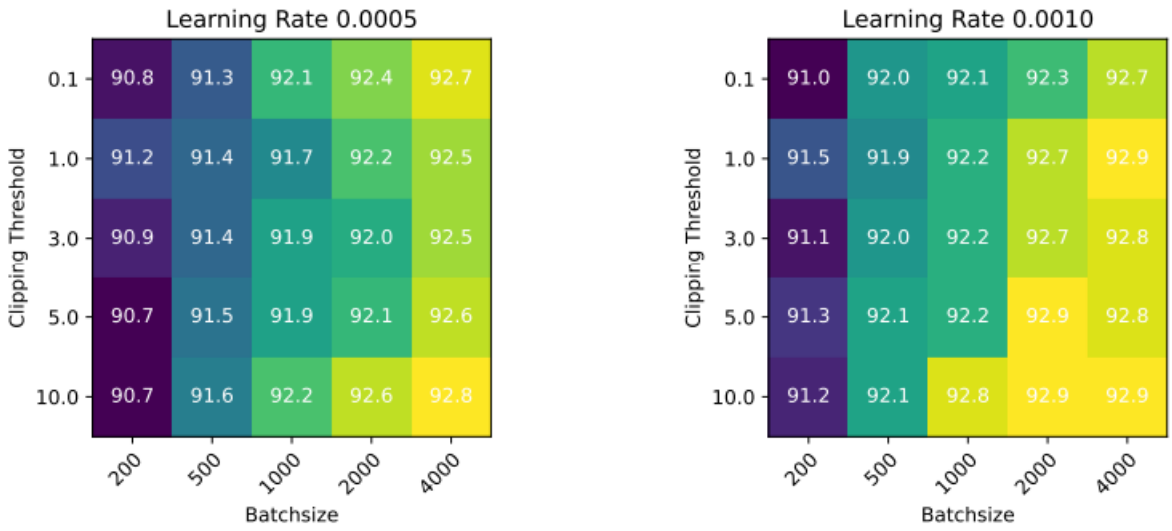

Figure 7: Test accuracy of fine-tuning the RoBERTa-Base model with respect to batchsize and clipping threshold

For figure 7 we can see that the larger the batch size, the better the accuracy obtained. Similarly, for the different clipping thresholds, we get roughly with the same result. Notice that for the other hyperparameters used in the previous section, where we compared the different efficient methods, they remain unchanged.

3 Conclusion:

Given the results of the experiments stated on this paper, fine-tuning using all the parameters of the model is expensive in terms of the hardware required and storage cost and the running speed. An alternative is been introduced in this paper to avoid applying DPSGD which is based on updating all the parameters and storing different fine-tuning copies. The author proposed LoRA, an efficient adaptation strategy which reduce the number of the parameter to fine-tune in the model while retaining high model quality. This proposed principle is generally applied to any neural networks with dense layers. There are many directions for future works, LoRA can be combined with other efficient adaptation methods such as (LoRA + Compacter) which potentially reduces more the number of parameters to fine-tune. The mechanism behind fine-tuning or LoRA is far from clear – how are features learned during pre-training transformed to do well on downstream tasks?