INVESTIGATING LORA AGGREGATION IN FEDERATED LEARNING

Matthew Meyer
matthew.meyer@epfl.ch

Supervised by Dongyang Fan

ABSTRACT

We study different methods for Low-Rank Adaptation in Federated Learning. This project aims to serve as a guide to give insight into the behavior of these methods depending on factors such as data heterogeneity and noise in the dataset. We also look at different ways of initializing and sharing the LoRA modules in order to understand the respective roles of LoRA-A and LoRA-B and how to better make use of them.

1 INTRODUCTION

Machine Learning models, and in particular Large Language Models, are becoming more prevalent by the day. However, collecting data and training them poses significant challenges. The first is computational: training is expensive, and having the necessary resources is not always guaranteed. The second is due to privacy concerns: it is not always safe or desirable to gather data in one central server. Next word prediction in messaging apps can be taken as an example, as these data points leak the content of private conversations.

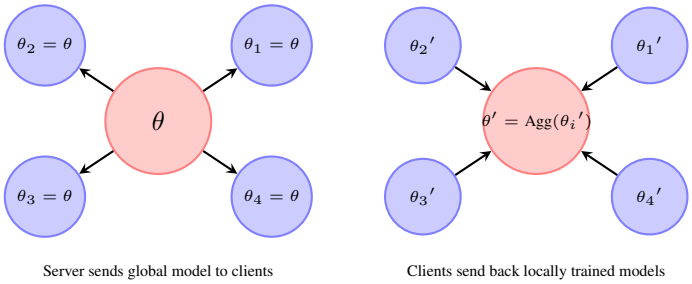
A solution to both of these concerns is Federated Learning (FL), and in particular, FL using Low-Rank Adaptation (LoRA). This technique allows for a model to be jointly fine-tuned by separate clients, each preserving ownership over its private data. What's more, compute requirements are low, allowing computations to be done on the clients' respective servers, rather than on the central server. We will now present FL and LoRA. The focus of this paper finds itself at the crossroads of both of these areas of research, and our goal is to study methods that combine the two, and determine practical guidelines to using them.

1.1 FEDERATED LEARNING

The need for federated learning (FL) arises when data is distributed, and scattered in small amounts. This scenario arises for example with mobile devices, where every user has their own local data, and for privacy and/or efficiency rationales it is preferred to not aggregate data on a central server with the aim of training a machine learning model. With federated learning, every client first trains the model locally. A priori, each of the local models has not learned much, as modern ML models require large amounts of data for training or fine-tuning. The next step is to somehow combine the local models in order to construct a new global model that encompasses all of the collective knowledge learned by the local clients.

This approach is a different paradigm with respect to traditional ML: instead of collecting data into one large dataset used to train a model, here, the data and the computations are kept *separate*.

Let's introduce some terminology and notations. Throughout this paper, we will reserve use of the variable n to refer to the number of clients in the FL setting. In addition, a *communication round* is the process of training locally for a certain number of iterations of stochastic gradient descent, aggregating the clients' models (using a process to be determined), and then redistributing models back to the clients (again, to be determined). A full training (or fine-tuning) run consists of a certain number of communication rounds.



An illustration of Federated Learning

In the figure above, we illustrate federated learning with four clients. At first, the global model θ is sent to the clients. Thus, before local training, the clients' models are given by $\theta_1 = \theta_2 = \theta_3 = \theta_4 = \theta$. After a certain number of local training iterations, the clients' models are then sent back to the global server, and aggregated. Determining how this aggregation is achieved is precisely the question that research in FL seeks to answer.

1.2 Parameter-Efficient Fine-Tuning (PEFT) and Low-Rank adaptation

Large language models (LLMs) are incredibly expensive to train from scratch, and when in possession of an LLM that was trained on a task similar to one we wish to do, then it makes sense to *fine-tune* the already trained (or pre-trained) model. However, even if we are only fine-tuning and not training from scratch, fine-tuning a full model can lead to problems. First, it remains prohibitively expensive. Second, it can lead to *catastrophic forgetting*. This happens when training on the new task causes the model to forget what it learned during pretraining.

In order to mitigate these two difficulties, Hu et al. (2021) propose *Low-Rank Adaptation*, or LoRA, for short. The idea is as follows: given a matrix before fine-tuning $W^{(0)} \in \mathbb{R}^{d\times k}$, keeping $W^{(0)}$ constant, one can write $W^{(1)} = W^{(0)} + \Delta W$, and we approximate $\Delta W \approx BA$, where $B \in \mathbb{R}^{d\times r}$, $A \in \mathbb{R}^{r\times k}$, where r is considerably smaller than d and k. Thus, the task consists in training B and A and keeping $W^{(0)}$ frozen, in such a way that the number of trainable parameters is $r(d+k)$, much fewer than the original dk. The value r is called the *LoRA-rank*.

A typical initialization of B and A are zero and a normal distribution, respectively. The idea is that BA will start as zero, and then during training, will become as good as possible an approximation of the optimal ΔW that will best fit the data. Below is a figure from the original LoRA paper, that perfectly illustrates what is done. Here, we have $h = (W + BA)x$, where x is an input and h an output, and only B and A are trained during fine-tuning, while W was pre-trained beforehand. Also in the illustration, we have $k = d$.

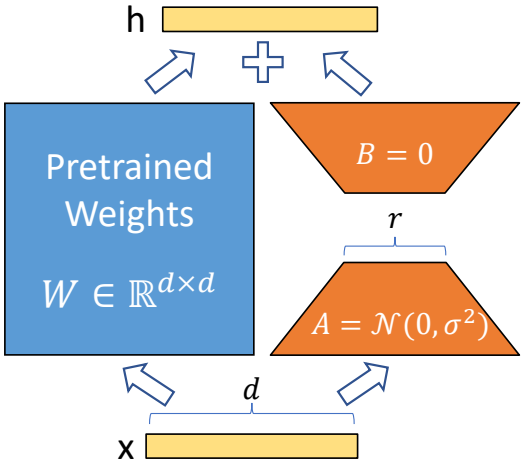


Figure 1: Illustrating Low-Rank Adaptation (from Hu et al. (2021))

1.3 Combining Federated Learning with LoRA

Now, we look at how to combine what we just discussed. What if, when doing FL, we are using parameter-efficient fine-tuning, and in particular LoRA? Then, we can say the following two things:

- It is not necessary to send the entire model back and forth between the global server and the clients. We assume that the frozen parameters are known by all from the beginning, and since they stay constant, it will never be necessary to communicate them. Only the trained parameters LoRA-A and LoRA-B are shared for collaboration.
- Different clients may not all have the same computational resources. In order to make the most of the compute that they possess, we would like to have every client i use a local LoRA-rank r_i that corresponds to their compute power. However, in that case, the first step in FL is not as straightforward as before: we cannot simply set the local clients' models as being equal to the global model - we must find a way to send lower-rank models to the clients with fewer resources. In the same vein, when local training is finished, our aggregation function must be able to handle clients having different ranks.

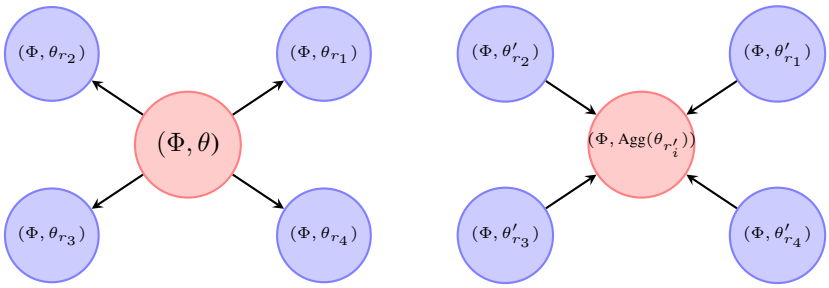


Illustration of FL with LoRA

In this illustration, again with four clients, we see how LoRA is taken into account - Φ represents the frozen parameters, and θ is the initial LoRA parameter set. We will then look at methods that construct θ_{r_i} for client i, such that the clients' models correspond to their respective local ranks r_i. Then, after local training, the aggregation function takes as input models with potentially differing LoRA ranks.

1.4 Aim of this project

In this project, we look at different existing methods of implementing LoRA in the FL setting. We will compare them from both a performance and efficiency point of view, in order to guide the user in their choice of a method.

2 Background

2.1 Federated Average (FedAvg)

A first approach to aggregating the clients learned weights is the first that comes to mind: simply average the LoRA modules. This method, the Federated Average, or *FedAvg*, was presented by McMahan et al. (2016). Thus, at communcation round t, we have the update step $B^{(t+1)} = \frac{1}{n}\sum_i B_i^{(t)}$ and $A^{(t+1)} = \frac{1}{n}\sum_i A_i^{(t)}$, where $B_i^{(t)}, A_i^{(t)}$ are the B and A LoRA modules of client i at the end of training round t, respectively.

This approach has a flaw: it requires every client to use the same LoRA rank, i.e. the rank of its lora modules. However, in practical settings this may not be the case as clients could have access to unequal resources with regards to compute. We will now look at two methods, HetLoRA and Reconstruction+SVD (FlexLoRA) that aim to solve this issue.

2.2 HETLORA

The *HetLoRA* method, by Cho et al. (2024), is similar to FedAvg, but allows for clients to have different ranks. Let r be the largest rank among the ranks of the clients. We let the global model have rank r. The steps are as follows:

1. At the end of communication round t, when distributing the global LoRA modules to the clients, truncate them to their local rank. More precisely, in our setup the global modules have dimensions $B^{(t)} \in \mathbb{R}^{d\times r}$ and $A^{(t)} \in \mathbb{R}^{r\times k}$, and client i has local LoRA rank $r_i \leq r$. Then client i will receive truncated matrices $B^{(t)}[:, : r_i]$ and $A^{(t)}[: r_i, :]$.
2. Now, every client has LoRA modules corresponding to its local rank. Local training ensues.
3. When it is time to communicate again, it is necessary for every client to send modules with rank r to the central server. In order to do so, we do *zero-padding*. That is, we add $r - r_i$ columns with zeros at the end of $B_i^{(t)}$ and $r - r_i$ rows at the end of $A_i^{(t)}$. Thus, the matrices sent to the central server are
$$\left(\begin{array}{c|c} B_i^{(t)} & 0 \end{array} \right)$$
and
$$\left(\begin{array}{c} A_i^{(t)} \\ \hline 0 \end{array} \right)$$
both with rank r, as required.
4. Now comes the last step: aggregation. Here, we do similarly as with FedAvg. However, we do not simply average the padded LoRA modules, but weight them with the Frobenius norm of $\Delta W_i^{(t)} = B_i^{(t)} A_i^{(t)}$. Let $S_i^{(t)}$ be the value of the Frobenius norm of $\Delta W_i^{(t)}$. Thus, the updated global weights are given by $B^{(t+1)} = \frac{1}{Z} \sum_i S_i^{(t)} B_i^{(t)}$ and $A^{(t+1)} = \frac{1}{Z} \sum_i S_i^{(t)} A_i^{(t)}$, where $Z = \sum_i S_i^{(t)}$.

The idea behind this weighting is to account for a problem that could arise in the different-ranks setting: since lower-rank clients will have been padded with zeros, they will have less weight in the aggregation than higher-rank clients. However, we do not want rank-size to be a deciding factor in this regard, but rather the quality of the training data of each client. In the same vein, HetLoRA also adds a regularization term while training, in order to rank-prune larger rank clients. This term for client i is $\text{Prune}(i) = \lambda \sum_j \|B_{i,j}[:, r_i\gamma : r_i]\| \cdot \|A_{i,j}[r_i\gamma : r_i, :]\|$ where j is an index that iterates over all of the LoRA matrices of client i. Here, $\gamma \in [0, 1]$ is a term that dictates how aggressively we will prune the ranks of larger rank clients. When γ is close to 1, pruning is weak, and the closer it is to 0, the more we prune. The point of this regularization is that larger-rank clients may overfit to noise present in the data.

We will not delve further into weighting with the Frobenius norm and pruning, but as mentioned, the aim of doing this is mitigating the effect of heterogeneous ranks among the clients. In the same-rank case, the methods are almost identical, and perform accordingly. Thus, when we are working in the same-rank setup, we will use FedAvg instead of HetLoRA.

The idea of first aggregating LoRA modules separately and then multiplying may seem counterintuitive - the optimal $W_i^{(t)}$ are in theory given by $B_i^{(t)} A_i^{(t)}$, which we should then aggregate. However, in general, $\frac{1}{n} \sum_i B_i^{(t)} A_i^{(t)} \neq \left(\frac{1}{n} \sum_i B_i^{(t)} \right) \left(\frac{1}{n} \sum_i A_i^{(t)} \right)$. This second approach is what we will look at now

2.3 RECONSTRUCTION + SVD (FLEXLORA)

We will now discuss another method, presented in Bai et al. (2024). The method, coined *FlexLoRA*, consists in first aggregating the local $\Delta W_i = B_i A_i$ matrices, and then using SVD to redistribute modules to the clients for local training, in such a way that all clients receive LoRA A and B corresponding to their respective local ranks. Let's go through it in more detail:

1. At communication round t, each client i computes its local update $\Delta W_i^{(t)} = B_i^{(t)} A_i^{(t)}$ and sends it to the central server. The clients have ranks r_i and the global model has rank $\max_{1 \leq i \leq n} r_i$

2. The central server computes $\Delta W^{(t+1)} = \frac{1}{N_1 + \cdots + N_n} \sum_{i=1}^{n} N_i \Delta W_i^{(t)}$, where N_i is the size of the dataset used by client i (size is determined by the number of tokens). In other words, it simply performs a weighted sum over the local updates.

3. We then perform SVD to obtain $\Delta W^{(t+1)} = U \Sigma V^T$. Now, we send modules back to the clients. Client i receives $B_i^{(t+1)} = U[:, : r_i] \Sigma[: r_i, : r_i]$ and $A_i^{(t+1)} = V[: r_i, :]^T$, such that $B_i^{(t+1)} A_i^{(t+1)} = U[:, : r_i] \Sigma[: r_i, : r_i] V[: r_i, :]^T \approx W^{(t+1)}$.

The main difference between FedAvg/Hetlora and FlexLoRA is the order of operations when the clients send their local LoRA modules to the central server: in the former, we first aggregate then multiply, and in the second, we first multiply and then aggregate.

2.4 Federated Freeze A LoRA (FFA-LoRA)

The authors of Sun et al. (2024) came up with this method in the context of *privacy-preserving federated learning*. This is not our focus here, but they also claim that even when we do not perform privacy-preserving learning, FFA is superior than FedAvg and Reconstruction+SVD.

As its name suggests, in FFA, we freeze the A module and only train B. Then, the global model will be updated as $B^{(t+1)} = \frac{1}{n} \sum_{i=1}^{n} B_i^{(t)}$. Moreover, the $A^{(0)}$ matrices must be initialized identically across all clients. This method, like FedAvg, is only usable in the same-ranks case.

What is the motivation for doing this? Well, first of all, FFA is very efficient. Since only one LoRA module is trained instead of two, much less compute is required. Secondly, as is explained in the paper, FedAvg amplifies the noise added for privacy-preserving learning, and FFA helps mitigate this. Even though we are not adding noise, every dataset has some inherent noise, and we can hope that the effect of FFA will be similar than in the privacy-preserving setting. Finally, this method solves an issue that existed with FedAvg: in general, $\frac{1}{n} \sum_i B_i^{(t)} A_i^{(t)} \neq \left(\frac{1}{n} \sum_i B_i^{(t)} \right) \left(\frac{1}{n} \sum_i A_i^{(t)} \right)$, but the equality becomes true if all of the $A_i^{(t)}$ are equal, which is the case for FFA.

2.5 Federated Share-A LoRA (FedSA)

Federated Share-A LoRA, or *FedSA* for short, was first described in Guo et al. (2024). It is similar to FedAvg, except in one key point: only the A module is shared. That is, the B module is only trained locally for each client, and is never shared with the other clients. The idea being that B will learn features pertaining to the local data and A will learn general features about the aggregated dataset.

2.6 Computational Cost Analysis

In a practical scenario, one will not only consider the performance of the methods but also their computation efficiency. Consider a simplified setup, where a model consists of a pretrained module $W^{(0)} \in \mathbb{R}^{d \times k}$, and we wish to approximate the fine-tuned model $W = W^{(0)} + \Delta W \approx W^{(0)} + BA$ with $B \in \mathbb{R}^{d \times r}$ and $A \in \mathbb{R}^{r \times k}$. For simplicity, consider that all clients have the same local rank. For all methods except for FFA, local training between communcation rounds is identical, so we will only have a look at the aggregation costs. Also, like before, let n be the number of clients in our setup.

2.6.1 FedAvg

The only operations we need to do are sum up the LoRA A and B modules. Thus, there are $n(r \cdot d + r \cdot k) = nr(d + k)$ floating point operations.

2.6.2 RECONSTRUCTION + SVD

- Reconstruction: For each client, there is a matrix multiplications to evaluate: $2 \cdot n \cdot d \cdot r \cdot k$ FLOPs.
- SVD: This of course depends on the algorithm used to perform SVD, but in general the complexity will be of the order $\mathcal{O}(\max(d, k)^3)$ FLOPs.

2.6.3 FFA AND FEDSA

For both FFA and FedSA, the costs incurred for communication are identical, as the same operations are performed. We only need to sum up the A modules, thus there are nrk FLOPs. However, FFA is far cheaper since only roughly half the parameters need to be trained between communication rounds.

2.6.4 WHICH METHODS ARE MORE EFFICIENT?

	RECONSTRUCTION+SVD	FEDAVG	FEDSA	FFA
FLOPS	$2ndrk + \mathcal{O}(\max(d, k)^3)$	$nr(d + k)$	nrk	nrk
ALL LORA PARAMS TRAINED?	YES	YES	YES	NO

Table 1: Comparing Computation Costs for Each Method

Thus Reconstruction + SVD is the most expensive, and FFA is by far the cheapest.

3 EXPERIMENTAL SETUP

3.1 DATASETS

In our experiments, we wish to explore the effects of two factors: *data heterogeneity* and *noise*.

Data homogeneity refers to the similarity in the distribution of data types between clients. If the data can be classified into different categories (by language, for example, in the context of textual data), then a data distribution will be considered more heterogeneous if each client has data belonging to mostly one category. In this case, each client will specialize in that one category but will not gain much insight into the others. At the other extreme, clients could have equal parts of all categories. In this case, the data distribution is *homogeneous*.

How to quantify heterogeneity? We use the *Dirichlet distribution*. The Dirichlet distribution $\text{Dir}(\alpha)$ of order $K \geq 2$ takes as support the set $\{(x_1, \ldots, x_K) \in \mathbb{R}^K \mid 0 \leq x_i \leq 1 \, \forall i, \; x_1 + \cdots + x_K = 1\}$. In other words, if we have K kinds of data, then sampling with a Dirichlet distribution will give us a way to distribute the data among the clients. The α parameter is what allows us to control homogeneity. When α is close to 0, then the distribution will be very uneven - one of the x_i will be close to 1 and the other values close to zero. Inversely, when α is large, the distribution is quite even - each of the x_i will be near to $1/K$.

In our experiments, in order to get a view over different data distributions, we repeated experiments with α ranging over $\alpha \in \{0, 0.1, 0.3, 1, 3, 10\}$. The distribution is not defined for $\alpha = 0$, but in our experiments, we define it as the situation where each client sees only one kind of data.

The **Noise** of a training set is also an important element that has an impact on performance in ML. We thus choose two datasets such that one is noisier than the other.

We use the following two datasets: Slim Pajama (Soboleva et al. (2023)) and Wikipedia Multilingual (Guo et al. (2020)).

Slim Pajama is a 6 billion token dataset with English-language data from several different sources. We use four sources, GitHub (code), Books (prose), ArXiv (scientific papers), and StackExchange (questions and answers). These different sources will play the role of the "categories" as described above. Wikipedia Multilingual is, as the name suggests, a dataset consisting of Wikipedia articles in different languages. Again, we separate the data into different categories, here differentiated by language. We use four languages: French, Italian, German, and Dutch. The reason we do not use English is that our model was pretrained on English data (more on the model later) and we wish to prevent leakage.

Both datasets are high in quality, but in our setup, Wikipedia Multilingual is much more noisy. This is because the base model we use (see below) was pretrained on English-language data, and learning a signal from foreign languages is inherently more difficult.

We will also run experiments using the Fineweb dataset (Penedo et al. (2024)). This high-quality dataset developed with the purpose of pretraining LLMs will provide more insight coupled with the results from our other experiments.

3.2 BASELINES, MODEL AND EXPERIMENT DETAILS

For our experiments, we use a large language model, OpenAI's GPT2 (Radford et al. (2019)) with 137 million parameters pretrained on English text, because of resource limitations. The task we train (and test) on is *next token prediction*. Before running experiments, it was necessary to figure out the optimal learning rate for each method. For each method, we sweeped through the values $\{0.0001, 0.0003, 0.001, 0.003, 0.01\}$ in order to find the best one. The following table shows what they are:

METHOD	LOCAL TRAINING	FEDAVG	HETLORA	RECONSTR.+SVD	FFA	FEDSA
OPTIMAL LEARNING RATE	0.0003	0.0003	0.0003	0.001	0.0001	0.001

TABLE 2: LEARNING RATES FOR EACH METHOD

Local training is the scenario in which every client trains on its local data and no communication takes place. This is what we use as our baseline: a method for federated learning should always be better than local training, as the point of collaborating is to put in common what was learned by the clients.

How much data should we use in our experiments? Let us remember that in a practical setting, federated learning makes sense when each client has little data, and collaborating allows data to be shared indirectly. Thus, in order to recreate this, we want to be in the following situation: local training overfits, but collaborating does not. Testing different data quantities leads us to use 500'000 tokens per client.

A few more details about our experimental setup: we used 12 clients - our limit being the capacity of the computational resources. In addition, in the same-ranks case, we used a LoRA rank of 8. The hyperparameter LoRA-α is set to be the double of the LoRA rank. In addition, every run was performed three times with three different seeds. Every run was done over 500 iterations, with a communication round every 25 iterations.

Finally, we get to the experiments. We did experiments in two cases: same ranks and different ranks.

- Same-ranks case: We use the methods FedAvg, Reconstruction+SVD, FFA and FedSA. In using the datasets Slim Pajama and Wikipedia Multilingual, we repeat runs over data distributions with the parameter Dirichlet-α varying over $\{0, 0.1, 0.3, 1, 3, 10\}$. With Fineweb we do not do this as we cannot easily separate it into different categories. As mentioned above, every run is repeated three times with three different seeds.
- Different-ranks case: Here, we only use the methods HetLoRA and Reconstruction+SVD, as they are the only ones that support this setup. The heterogeneous-ranks setup is not the main focus of this project, but it is important to have a look at it as a control, to make sure that straying from the same-ranks setup does not have a significant effect on the results.

Let's summarize the choices we made in the following table:

HYPERPARAMETER	#CLIENTS	#ITERATIONS	#ITERATIONS BETWEEN COMMUNICATION ROUNDS	#TOKENS PER CLIENT	MODEL
VALUE	12	500	25	500'000	GPT2

TABLE 3: EXPERIMENTAL SETUP HYPERPARAMETERS

3.3 EVALUATION

As a reminder, the task we are evaluating is *next token prediction*. We use perplexity as a metric for evaluation.

4 RESULTS AND ANALYSIS

4.1 SOME TOY EXPERIMENTS

Before getting to the actual results, let's try and get some intuition to better understand what we may expect. In particular, we will study the impact of *noise* in the training set.

We set up a toy experiment without any actual training, but that should yield some insight. Here is how we proceed:

Let $\texttt{inherent\_rank}$ be the inherent rank of the data distribution. Consider $P \in \mathbb{R}^{d \times \texttt{inherent\_rank}}$ and $Q \in \mathbb{R}^{\texttt{inherent\_rank} \times d}$. We represent the "true signal" as $W_{\texttt{true}} = PR$. However, we want to simulate noise. Thus, for each client i, we compute $P_i' = P + \texttt{noise} \times \widetilde{A}_i$ and $Q_i' = Q + \texttt{noise} \times \widetilde{B}_i$, where $\widetilde{A}_i$ and $\widetilde{B}_i$ are sampled from a Gaussian distribution, and $\texttt{noise}$ is a parameter that allows us to control how much noise we introduce. Let $W_i = P_i' Q_i'$.

For simplicity, consider that all clients have the same LoRA rank, r. We use singular value decomposition to retrieve the simulated B_i and A_i such that $B_iA_i = W_i$. We now simulate the effects of FedAvg and Reconstruction+SVD in aggregating the local LoRA modules.

- FedAvg: The global model is constructed as $W_{\texttt{FedAvg}} = \left(\frac{1}{n}\sum_i B_i\right)\left(\frac{1}{n}\sum_i A_i\right)$.
- Reconstruction+SVD: Here, we compute $W_{\texttt{Reconstruction}} = \frac{1}{n}\sum_i W_i = \frac{1}{n}\sum_i B_iA_i$.

In both cases, we will look at the error $\|W_{\texttt{true}} - W_{\texttt{method}}\|$ (we use the Frobenius norm). In our test case, we set $\texttt{inherent\_rank} = 5$ and $d = 20$. We vary $\texttt{noise}$ along the values $\{0.1, 1, 3, 5\}$, and for each of these, vary r from 1 to 20 (referred to in the graphs below as "intermediary rank").

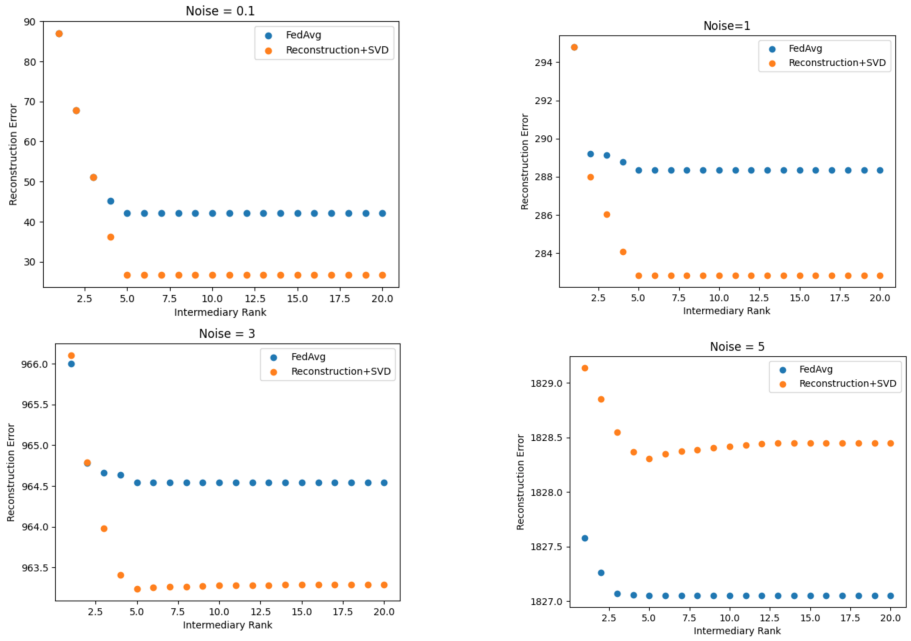


Figure 2: Examining the effect of noise in a toy experiment

This suggests that more noise benefits FedAvg, whereas a better quality signal will allow Reconstruction+SVD to get better results.

4.2 Presentation of results

4.2.1 Same-ranks case

We now show the results from our experiments in the same-rank case for FedAvg, Reconstruction+SVD, FFA and FedSA, for a range of Dirichlet-α distributions over the Slim Pajama and Wikipedia Multilingual datasets, experiments with the Fineweb dataset, averaged over three seeds.

α	0	0.1	0.3	1	3	10
FedAvg	18.94(±0.16)	17.65(±0.42)	17.41(±0.23)	14.08(±0.24)	14.14(±0.15)	13.65(±0.14)
Reconstr.+SVD	**17.74(±0.10)**	**16.68(±0.22)**	16.65(±0.17)	13.87(±0.21)	14.09(±0.16)	13.71(±0.15)
FFA-LoRA	20.18(±0.24)	19.87(±0.06)	16.73(±0.22)	14.02(±0.16)	14.21(±0.07)	13.79(±0.29)
FedSA-LoRA	18.63(±0.09)	18.47(±0.18)	**15.22(±0.17)**	**13.28(±0.19)**	**13.05(±0.19)**	**12.55(±0.16)**
Local Training	20.74(±0.22)	19.54(±0.31)	20.16(±0.19)	16.89(±0.30)	17.05(±0.16)	16.40(±0.21)

Table 4: Perplexity as a function of Dirichlet-α and used method for LoRA aggregation
Dataset used: Slim Pajama

With the Slim Pajama dataset, we see that when there is stronger data heterogeneity, then Reconstruction+SVD performs best, but when the data distribution becomes homogeneous, FedSA-LoRA is better (though Reconstruction+SVD, FedAvg and FFA are all strong contenders). This shows that the optimal method is not absolute, but depends on the distribution of the data. Now, let's have a look at what happens with the Wikipedia Multilingual dataset.

α	0	0.1	0.3	1	3	10
FEDAVG	27.39($\pm$0.12)	26.50($\pm$0.05)	24.13($\pm$0.48)	23.36($\pm$0.19)	21.98($\pm$0.44)	21.76($\pm$0.11)
RECONSTR.+SVD	37.05($\pm$0.25)	29.92($\pm$0.04)	26.72($\pm$0.35)	25.86($\pm$0.21)	24.25($\pm$0.39)	24.16($\pm$0.09)
FFA-LORA	37.71($\pm$0.06)	36.25($\pm$0.55)	32.73($\pm$0.73)	30.69($\pm$0.45)	28.20($\pm$0.32)	28.08($\pm$0.36)
FEDSA-LORA	25.92($\pm$0.29)	**25.25($\pm$0.16)**	**23.45($\pm$0.32)**	**22.08($\pm$0.15)**	**20.85($\pm$0.10)**	**20.52($\pm$0.02)**
LOCAL TRAINING	**22.86($\pm$0.18)**	25.48($\pm$0.06)	25.34($\pm$0.14)	29.61($\pm$0.18)	29.59($\pm$0.16)	30.26($\pm$0.04)

Table 5: Perplexity as a function of Dirichlet-α (the higher the more homogenous the dataset are) and used method for LoRA aggregation. Dataset used: Wikipedia Multilingual.

Again, when the data distribution is more heterogeneous, FedSA-LoRA comes out on top. Interestingly, our baseline, local training, is the best method in the purely heterogeneous scenario. This can be explained by the following: the different data types are so different (different languages), that the information imparted by other clients with a language different with one's own will not add anything useful, and will only dilute the what the client has learned. This differs somewhat from the Slim Pajama dataset, where what was learned from books, for example, could still be useful in Q&A type settings.

Notice that our findings correspond to what we saw in the toy experiments: in the noisier dataset (Wikipedia Multilingual), FedAvg did considerably better than Reconstruction+SVD.

METHOD	FEDAVG	RECONSTR.+SVD	FFA-LORA	FEDSA-LORA	LOCAL TRAINING
PERPLEXITY	31.81($\pm$0.28)	**31.49($\pm$0.31)**	32.96($\pm$0.36)	32.01($\pm$0.23)	36.80($\pm$0.35)

Table 6: Performance of Methods on the Fineweb Dataset

The best performing method for the Fineweb dataset is Reconstruction+SVD, with FedAvg, FFA-LoRA and FedSA-LoRA not far behind. With this in mind, and looking at the results from Table 4, we see that the comparative performances of methods are similar than that of the experiments with Slim Pajama. Thus, we can deduce that the noisiness in Fineweb is similar to that of Slim Pajama.

4.2.2 HETEROGENOUS-RANKS CASE

Now, let's have a look at what happens in the heterogenous-ranks case. This scenario is not the main focus of this work, but we wish to see if there is a significant difference when some clients have more compute than others.

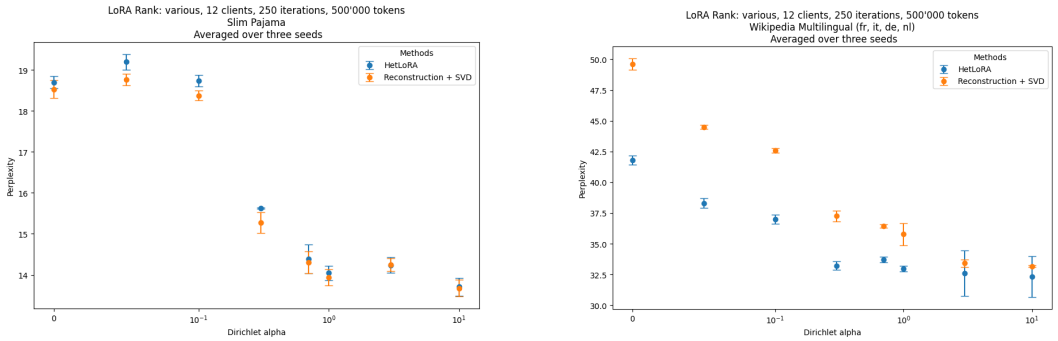


Figure 3: Comparing HetLoRA and FlexLoRA in the heterogenous-ranks setup

We see that the results are in line with what we saw before: For Slim Pajama, both HetLoRA and Reconstruction+SVD perform roughly equally, whereas for Wikipedia Multilingual, HetLoRA fares better.

4.3 Ablation studies

We will have a further look at FFA and FedSA, and seek to further understand the role that the LoRA modules play. We will experiment with variants of FFA-LoRA and FedSA-LoRA in order to better grasp how these methods function.

4.3.1 Examining the role of the frozen module in FFA

FFA-LoRA, or Federated Freeze-A LoRA consists of initializing the A module with a normal distribution, initializing B with zeros, and then freezing A. A natural question to ask is the following: what is we inverse their initializations and freeze B instead?

This is precisely the problem studied in Hayou et al. (2024). The claim made in that paper is that when B is frozen, then in order to achieve stable learning, the learning rate must be smaller, and thus the model learns slower.

In order to test this theory, we did hyperparameter testing for both datasets Slim Pajama and Wikipedia Multilingual, and found that the optimal learning rate was actually larger. However, the conclusion was the same - FFA, when freezing A, performs much better. In the following graphs, we compare the two. We will use the denomination FFB-LoRA to refer to the "freezing B" scenario (Federated Freeze-B LoRA). The learning rate used for FFA-LoRA is 0.0001 and the learning rate used for FFB-LoRA is 0.001.

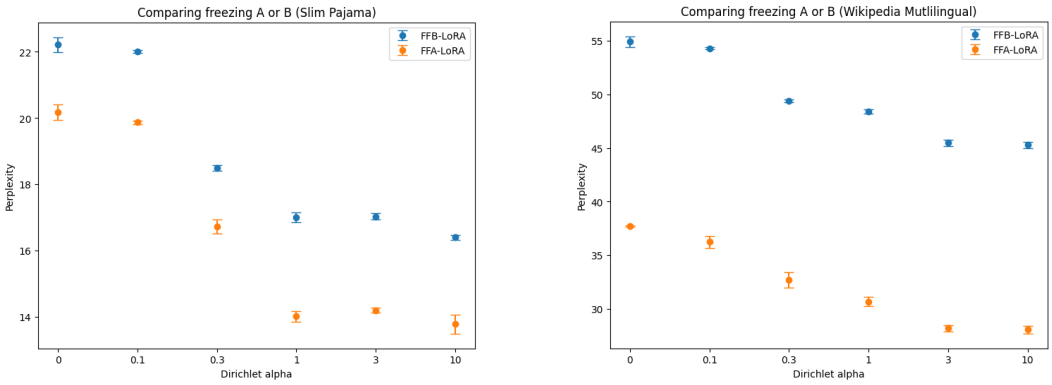


Figure 4: Comparing FFA-LoRA and FFB-LoRA

4.3.2 Which module to share in FedSA?

In FedSA-LoRA, as the name Federated Share-A LoRA suggests, clients collaborate their respective A modules, and the B module is trained locally without collaboration. We could imagine a variant of this method, Federated Share-B LoRA, or FedSB-LoRA, where the roles are inversed - B is shared, and A is kept locally. The claim in the FedSA paper is that there is a difference, and that sharing A leads to better results. We run some experiments in order to verify this.

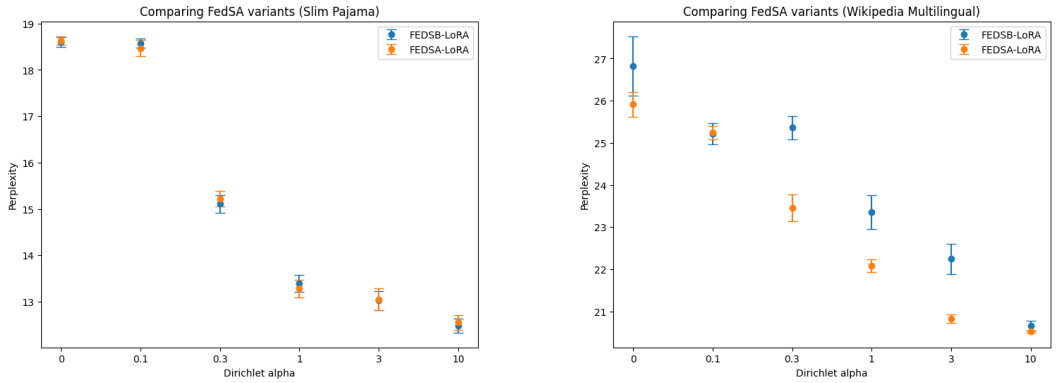


Figure 5: Comparing FedSA-LoRA and FedSB-LoRA

Our experiments confirm the hypothesis: sharing the LoRA-A module is better than the alternative.

5 Conclusion

Our final aim in this project was to provide a guide to using LoRA in the federated setting. From our experimental results, we give the following recommendations:

- Consider the impact of data heterogeneity: when data less heterogeneous (large Dirichlet-α), Federated Share-A LoRA (FedSA) performs better. However, when data is more heterogeneous, it is necessary to estimate the noisiness of the data at hand. In the less noisy case, as we had for Slim Pajama, Reconstruction+SVD (FlexLoRA) is a strong contender. However, when the data is noisier, one may want to opt for FedSA, FedAvg, or even Local Training.
- If one wishes to prioritize computation efficiency: FFA is the most efficient method from this point of view, even though it is never the best results-wise. Compared to other methods, the best scenario for employing FFA is when noise is not too prevalent.
- When freezing one of the LoRA modules, it is more effective to freeze the A rather than B (this is FFA). Moreover, when one communicates only one of the LoRA modules and trains the other one locally, it is better to share A and not B, as is done in FedSA.

We have set up a code base to test the methods we studied that can be found in this Github repository.

Acknowledgments

I would like to thank Dongyang Fan, who supervised the project, and was always available to advise throughout. Also thanks to the Machine Learning & Optimization laboratory that provided all the necessary computational resources.

References

Jiamu Bai, Daoyuan Chen, Bingchen Qian, Liuyi Yao, and Yaliang Li. Federated fine-tuning of large language models under heterogeneous tasks and client resources, 2024. URL https://arxiv.org/abs/2402.11505.

Yae Jee Cho, Luyang Liu, Zheng Xu, Aldi Fahrezi, and Gauri Joshi. Heterogeneous lora for federated fine-tuning of on-device foundation models, 2024. URL https://arxiv.org/abs/2401.06432.

Mandy Guo, Zihang Dai, Denny Vrandečić, and Rami Al-Rfou. Wiki-40B: Multilingual language model dataset. In Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Khalid Choukri,

Christopher Cieri, Thierry Declerck, Sara Goggi, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asuncion Moreno, Jan Odijk, and Stelios Piperidis (eds.), *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pp. 2440–2452, Marseille, France, May 2020. European Language Resources Association. ISBN 979-10-95546-34-4. URL https://aclanthology.org/2020.lrec-1.297/.

Pengxin Guo, Shuang Zeng, Yanran Wang, Huijie Fan, Feifei Wang, and Liangqiong Qu. Selective aggregation for low-rank adaptation in federated learning, 2024. URL https://arxiv.org/abs/2410.01463.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. The impact of initialization on lora finetuning dynamics, 2024. URL https://arxiv.org/abs/2406.08447.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. Federated learning of deep networks using model averaging. *CoRR*, abs/1602.05629, 2016. URL http://arxiv.org/abs/1602.05629.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale, 2024. URL https://arxiv.org/abs/2406.17557.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama. https://www.cerebras.net/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama, 2023. URL https://huggingface.co/datasets/cerebras/SlimPajama-627B.

Youbang Sun, Zitao Li, Yaliang Li, and Bolin Ding. Improving lora in privacy-preserving federated learning, 2024. URL https://arxiv.org/abs/2403.12313.