Scalable Computing Systems Laboratory

Gradient Guessing in Federated Learning

by Mohamed Yassine Boukhari

Bachelor Project Report

Professor Anne-Marie Kermarrec
Project Advisor

Dr. Rafael Pires and Akash Dhasade
Project Supervisors

EPFL IC IINFCOM SaCS
BC Building, Office BC 347
CH-1015 Lausanne
Switzerland

January 7, 2022

Contents

Chapter 1

Introduction

Federated Learning (FL) is a distributed machine learning technique that trains a shared model collaboratively between different nodes while keeping the training data on the nodes and thus removes the need to store it or share it with a central server. This machine learning method has gained popularity due to its ability to train smarter and more scalable models while providing privacy guarantees to the users. Despite its appealing characteristics, Federated learning is hampered by many technical and algorithmic challenges that impact substantially its speed of convergence. The main challenges faced by FL are expensive communication, statistical heterogeneity and systems heterogeneity.

GeL, the Guess and Learn algorithm, aims to address systems heterogeneity constraints which are caused mainly by imbalanced computational capabilities between participating devices and network constraints resulting in clients dropping. By boosting FL convergence at no additional computational cost, GeL tries to compensate for the decrease in convergence speed caused by struggling nodes. This algorithm recycles the already computed gradients on the clients and provides them as inputs to the client's optimizer allowing the local models to perform extra update steps for free , relying on the last computed gradient as a guess for the upcoming gradients. By doing so, GeL anticipates the direction of the next model update and moves the model in that direction. This approach provides a way of increasing the number of model updates clients can do locally with no extra expensive computations.

In this project, we provide an implementation of a centralized version of GeL based on the already existing distributed implementation. Furthermore, we try to study the impact of certain parameters on the convergence and performance of the algorithm. Mainly, we aim to answer the following questions :

- How does changing the learning rate affect GeL ?
- Is Adam the only optimizer for which GeL shows results ?
- How does GeL scale with the number of active clients per communication round ?
- Does guessing payoff in centralized machine learning settings ?

To answer these questions, we provide an extensive set of experiments and results that allow us

to do an empirical evaluation of the algorithm. In addition, we try to improve its performance by proposing some possible optimizations.

Chapter 2

Background

In this chapter, we present the central theme of this project, namely the GeL algorithm. At first, we provide a brief overview of optimization in deep learning along with two algorithms used in this project that is SGD and Adam. Then, we outline the process of federated training. At last, we introduce the idea behind GeL and its core algorithm.

2.1 Optimization in Deep Learning

The goal of optimization in deep learning is to find the optimal weights θ^* of a neural network that locally minimize some loss function J. One widely-used optimization method is gradient descent which consists of decreasing the cost function by moving in the direction of the negative gradient :

$$\theta_{k+1} = \theta_k - \epsilon \nabla J(\theta_k)$$

where θ_k and θ_{k+1} are the model weights at iterations k and $k+1$ and ϵ is the learning rate. The gradient here is computed on the entire dataset.

2.1.1 Stochastic Gradient Descent

Stochastic gradient descent and its variants are amongst the most popular optimization methods in deep learning. In SGD, the gradient is computed on a randomly selected batch of data instead of the entire dataset. SGD is cheaper than traditional Gradient Descent since it replaces the actual gradient by an estimate and allows for faster iterations. One further improvement to SGD is the use of momentum in order to accelerate convergence by accumulating an exponentially decaying moving average of past gradients and moving in their direction.

Algorithm 1 Stochastic Gradient Descent with Momentum[1]

Require: Learning rate ϵ , momentum parameter α
Require: Initial model weights θ_0 , initial velocity v_0
while stopping criterion not met **do**
Sample a minibatch b from the training set
Compute gradient estimate on b : $g \leftarrow \nabla J(\theta)$
Compute velocity update : $v \leftarrow \alpha v + \epsilon g$
Update Model : $\theta \leftarrow \theta - v$
end while

2.1.2 Adam

Adam is an adaptive learning rate optimization algorithm. It uses estimations of first and second moments of gradient to adapt the learning rate for each weight of the neural network and it takes advantage of momentum by keeping an exponentially weighted moving average of the both first order and second order moments of gradients.

Algorithm 2 Adam [1]

Require: Learning rate ϵ , small constant δ
Require: Decay rates for moment estimates , α and β
Require: Initial model weights θ_0
$v_1 \leftarrow 0$
$v_2 \leftarrow 0$
$t \leftarrow 0$
while stopping criterion not met **do**
Sample a minibatch b from the training set
Compute gradient estimate on b : $g \leftarrow \nabla J(\theta)$
$t \leftarrow t + 1$
Update biased first moment estimate : $v_1 \leftarrow \alpha v_1 + (1 - \alpha)g$
Update biased second moment estimate : $v_2 \leftarrow \beta v_2 + (1 - \beta)g \odot g$
Correct bias in first moment : $\hat{v_1} \leftarrow \frac{v_1}{1-\alpha^t}$
Correct bias in second moment : $\hat{v_2} \leftarrow \frac{v_2}{1-\beta^t}$
Update Model : $\theta \leftarrow \theta - \epsilon \frac{\hat{v_1}}{\sqrt{\hat{v_2}}+\delta}$
end while

2.2 Federated Training

In federated learning[2], the clients collaboratively train a model under the coordination of a central server without sharing the data with the server or another client. The training process is divided into communication rounds and for most federated learning algorithms, a training round follows the upcoming template:

1: **Client Selection**: the server chooses a subset of clients to participate in the training round.
2: **Server-to-client broadcast**: the server broadcasts the global model to the selected clients.
3: **Local client update**: clients train the received model on their local data.
4: **Client-to-server upload**: clients upload their updated model to the server.
5: **Aggregation and global model update**: the server aggregates the client updates and updates the global model.

2.3 GeL : Boost Federated Learning For Free

In this section we present an overview of the GeL [3] algorithm.
By design, GeL relies on the use of momentum in the Adam optimizer to guess future model updates based on the already computed gradients. For Adam($\alpha = 0.9,\ \beta = 0.999$), the newly computed gradients contribute to only 10% and 0.1% of the new values of v_1 and v_2 (lines 11 and 12 of Algorithm 2).

Since the accumulated estimates of the first and second moments of gradients are the major contributors to the local model update, using an approximate gradient for upcoming updates should produce a good update. In GeL, we use the last computed gradient as an approximate gradient of the following update steps.

On the other hand, GeL relies on the averaging performed by the server: same as FEDAVG, the central server distributes the parameters to each client, collects periodically the updated parameters from clients and aggregates them using arithmetic averaging.
In Algorithm 3, we present a high level overview of GeL.

As we can see in Algorithm 3, given the computational budgets $[u_1, u_2, ..., u_{n_{clients}}]$ and the number of guesses $[g_1, g_2, ..., g_{n_{clients}}]$ of the selected clients, each client k performs u_k actual gradient updates followed by g_k updates using the last computed gradient. This approach allows every node k to perform $u_k + g_k$ local model update step instead of only u_k. In the first u_k steps, the Adam optimizer accumulates moments based on the actual gradient computations. Starting from step $u_k + 1$ we use the last computed gradient to perform g_k extra free updates relying on the previously accumulated momentum to continue on the same direction.
These extra steps allow the model to move further towards the client's local minimum and therefore boost convergence.

Algorithm 3 GeL. The $n_{clients}$ clients are indexed by k; u_k and g_k are the computational budget and number of guesses for client k, and B is the local mini-batch size.

Server Executes:

Initialise global model: $\theta \leftarrow \theta_0$
Training round number: $t \leftarrow 0$
while stopping criterion not met **do**
    C clients check in with the server
    $S_t \leftarrow$ Server selects randomly a subset of $n_{clients} < C$ clients for training.
    Server assigns a number of guesses g_k for every clients in S_t
    for k in S_t in $parallel$ **do**
        $\theta_k \leftarrow ClientUpdate(k,\theta,g_k)$
    end for
    $\theta \leftarrow \frac{1}{n_{clients}} \sum_{n=1}^{n_{clients}} \theta_k$
    $t \leftarrow t + 1$
end while

ClientUpdate(k, θ, g_k):

Initialise local optimizer: $optimizer \leftarrow Adam()$
Initialise local model: $\theta_{local} \leftarrow \theta$
for $i \leftarrow 1$ **to** u_k **do**
    $b_i \leftarrow$ Sample a random batch of size B
    $grad_i \leftarrow \nabla J(\theta_{local}, b_i)$
    $\Delta\theta_{local} \leftarrow optimizer.gradientStep(grad_i, \theta_{local})$
    $\theta_{local} \leftarrow \theta_{local} + \Delta\theta_{local}$
end for
Use last computed gradient as proxy: $proxy_{grad} \leftarrow grad_{u_k}$
for $g' \leftarrow 1$ **to** g_k **do**
    $\Delta\theta_{local} \leftarrow optimizer.gradientStep(proxy_{grad}, \theta_{local})$
    $\theta_{local} \leftarrow \theta_{local} + \Delta\theta_{local}$
end for
return θ_{local}

Chapter 3

Implementation

To be able to test the GeL algorithm in a traditional centralized machine learning scenario, we implemented a centralized version of the algorithm based on the original federated implementation. In this chapter, we provide first a brief overview of the already existing distributed system. Then, we present our implementation of central GeL.

3.1 Federated Setup

The system was implemented using Python3, on top of the TensorFlow 2 and TensorFlow Federated frameworks.

3.1.1 System Configuration

Since the system can be applied to different datasets and with different computational budgets, number of active clients per round, etc,we first need to specify these parameters. To run the system, we need to set the following parameters:

- **dataset**: FEMNIST, Shakespeare, CelebA, Synthetic, Sent140, Reddit
- **batch size**: batch size used for model training in each training step.
- **optimizer**: the algorithm used for ML optimization
- **learning rate**: step size for the optimizer.
- **number of clients**: number of clients to be selected in every training round.
- **budgets interval**: determines the number of local model update steps the client can perform when participating in a training round.For every client, we assign a budget value from this interval.
- **number of guesses**: number of extra update steps the local model performs after exhausting its budget.

- **selection scheme**: how to select clients to participate in a training round.
- **evaluate every**: frequency of evaluation on test set.
- **initial model weights, client sampling and budget sampling**

3.2 Centralized Setup

Centralized GeL was implemented in Python3 using the TensorFlow framework. Similar to the decentralized version, the system allows us to simulate different scenarios.

3.2.1 System Configuration

Before running the program, we need to configure the following:

- **dataset**: FEMNIST, CelebA, Synthetic.
- **batch size**: batch size used for model training in each training step.
- **optimizer**: the algorithm used for ML optimization[Adam,SGD]
- **learning rate**: step size for the optimizer.
- **computational budget**: the number of actual gradient steps performed by the model before guessing.
- **number of guesses**: number of extra update steps the model performs after exhausting its budget.
- **evaluate every**: frequency of evaluation on test set.
- **initial model weights**

3.2.2 Central GeL Algorithm

In our centralized system, we implemented the following guessing algorithm:

Algorithm 4 Central GeL. u is the computational budget, g is the number of guesses , B is the is mini-batch size, and opt is the optimizer[SGD or Adam]

Initialise optimizer: $optimizer \leftarrow new\ opt()$
Initialise model: $\theta \leftarrow \theta_0$
while stopping criterion not met **do**
    for $i \leftarrow 1$ **to** u **do**
        $b_i \leftarrow$ Sample a random batch of size B
        $grad_i \leftarrow \nabla J(\theta, b_i)$
        $\Delta\theta_{local} \leftarrow optimizer.gradientStep(grad_i, \theta)$
        $\theta \leftarrow \theta + \Delta\theta$
    end for
    Use last computed gradient as proxy: $proxy_{grad} \leftarrow grad_u$
    for $g' \leftarrow 0$ **to** g_k **do**
        $\Delta\theta \leftarrow optimizer.gradientStep(proxy_{grad}, \theta)$
        $\theta \leftarrow \theta + \Delta\theta$
    end for
end while
return θ

Chapter 4

Evaluation

In this chapter, we provide an evaluation of GeL in different setups and we study the impact of various parameters on its convergence. We layout our experimental methodology as well as a wide set of experiments that each test the impact of a distinct parameter. We start with the learning rate ϵ and measure the observed changes in GeL behavior when we train with different learning rates. Thereupon, we try guessing with a different optimization algorithm then Adam and asses if we still observe boosts in performance when we rely on the SGD with momentum optimizer. Afterwards, we take a look at the scalability of GeL and study the impact of varying the number of active clients per round . Finally, we remove the federated setup and test whether GeL still works in a centralized traditional machine learning scenario where we have only one machine and no averaging scheme.

4.1 Methodology

We run our experiments on three different FL benchmarks from LEAF[4] representing different ML tasks, techniques and models. To evaluate the performance, we measure, parse and log the shared model train accuracy, test accuracy, train loss and test loss for every third round of training. The training ends when the global model reaches a specific target accuracy for every different benchmark or when the number of rounds reaches a certain threshold. The performance is measured as the number communication rounds until convergence and the total number of gradient computations. To account for the high variance of some datasets, we run every experiment four times and we average the results over all the runs. In addition, We fix the client sampling and the initial model weights for every set of comparable runs. We note that all the experiments have been conducted with homogeneous computational budget and non-IID data.

4.1.1 Datasets

We use three benchmarks from the LEAF framework to test our implementation. We use the following datasets and their corresponding provided machine learning models :

- Federated Extended MNIST (FEMNIST), which is built by partitioning the data in Extended MNIST based on the writer of the digit/character
- CelebA, which partitions the Large-scale CelebFaces Attributes Dataset by the celebrity on the picture.
- A Synthetic dataset .

Table 4.1: FL benchmarks used in this report

DATASET	TASK	MODEL	BATCH SIZE	TARGET ACC
CelebA	Image Classification	4 Conv2D Layers	5	88.5%
FEMNIST	Image Classification	2 Conv2D Layers	20	77%
Synthetic	Cluster Indentification	Logistic Regression	5	81%

4.1.2 Metrics

In the interest of evaluating the performance of the system, we use the following metrics :

- train & test losses
- train & test accuracies
- number of rounds until convergence
- total number of gradient computations

4.1.3 Test Environment

We use the machines labostrex120-128@iccluster. All the machines have the same specifications with processor Intel Xeon E5-2630 v3 at 2.40GHz and 128GB of memory running Ubuntu 20.04.2 LTS 5.4.0-72. The tests for the centralized setup were simulated with the TensorFlow framework and the distributed experiments were simulated with TensorFlow Federated simulation environment.

4.2 Experiments

4.2.1 Varying the Learning Rate

Motivation

In the background chapter, we briefly explained how GeL relies on the last computed gradient as a proxy for the upcoming gradients. In the optimization landscape some regions have approximately the same slope, so it makes sense to the previous gradient as a substitute for the future gradients.

The learning rate or step size dictates how much we move on that landscape: a bigger learning rate means more significant model updates and more shifting in the optimization landscape. An intriguing direction to explore is the effect of the learning rate on the guessing in GeL.

Experiments

To answer the above questions, we train our model on the FEMNIST and Synthetic datasets with the Adam optimizer using learning rates with different magnitudes and with the default recommended values of the parameters α and β (0.9 and 0.999 respectively). The learning rates were chosen such that baseline and target converge in a reasonable number of rounds. We fix the number of selected clients per round $n_{clients}$ to **20**. For every dataset, we set a client computational budget value u' as a baseline and we study the convergence of GeL across a range of client guesses per round g' and we compare it to target which is the performance of the model if we replace the guesses with actual gradient computations. The following table presents the set of experiments we conducted for this part:

Table 4.2: Experiments conducted using Adam with $\alpha = 0.9$, $\beta = 0.999$ and $n_{clients} = 20$.

DATASET	**BASELINE** u'	**GUESSES** g'	**LEARNING RATES** ϵ
FEMNIST	6	[2,3,4,6]	[5×10^{-4},1×10^{-3},2×10^{-3},5×10^{-3}]
Synthetic	4	[1,2,3,4,5]	[1×10^{-3},5×10^{-2},$1 \times 10-1$,5×10^{-1}]

Results

Figure 4.1: Test accuracy of GeL, baseline and target for different learning rates ϵ and guesses g' on the FEMNIST dataset

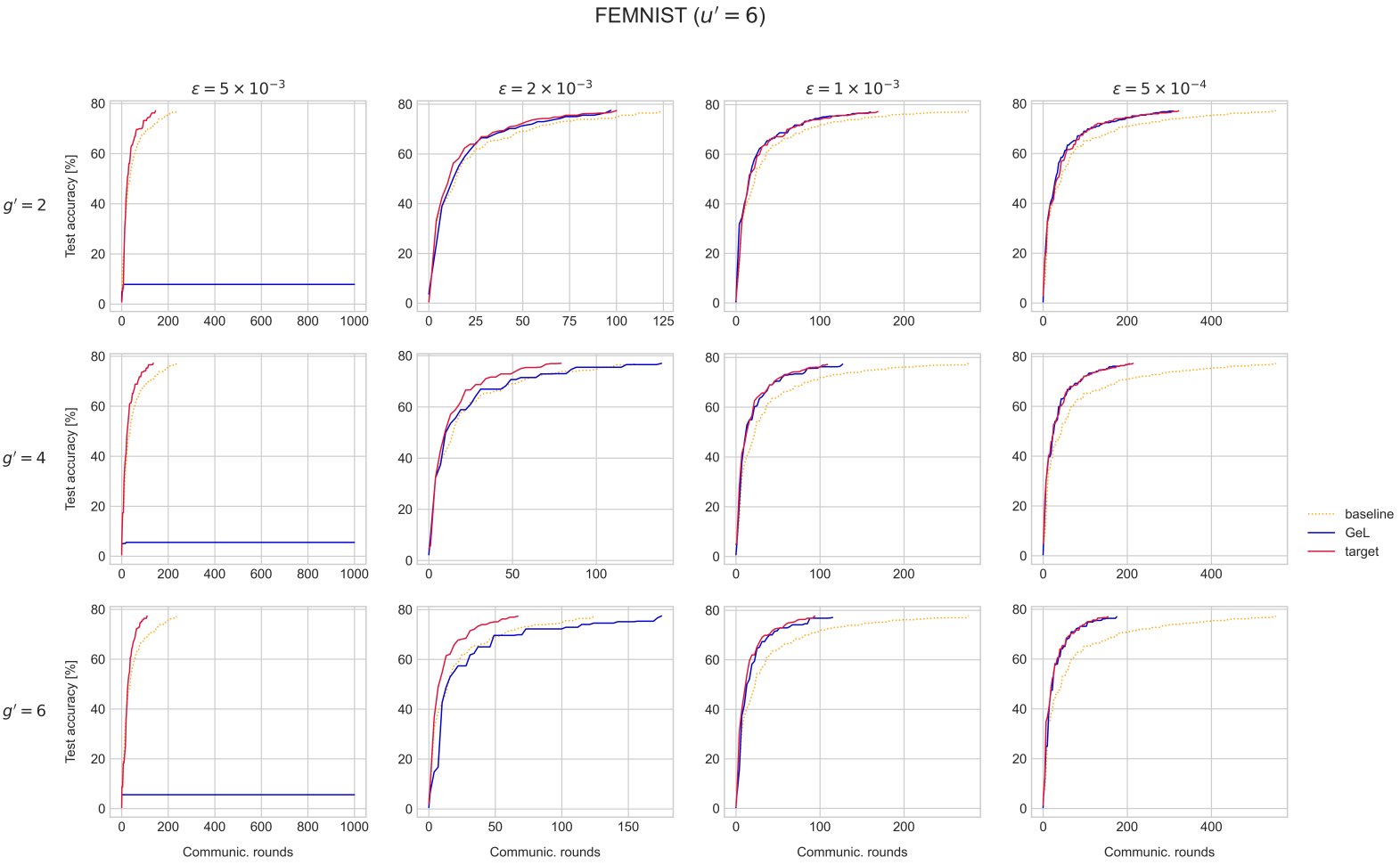
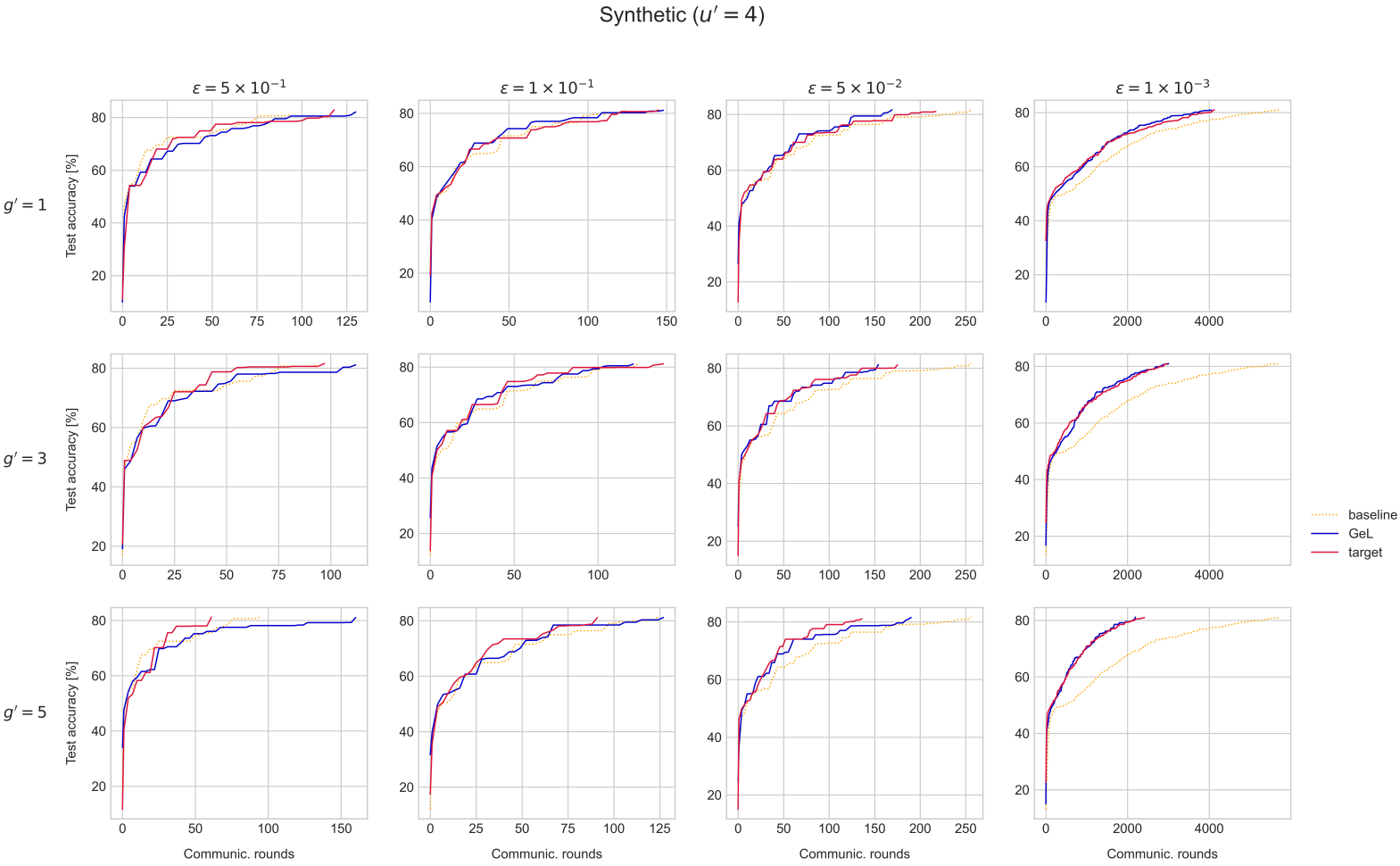


Figure 4.2: Test accuracy of GeL, baseline and target for different learning rates ϵ and guesses g' on the Synthetic dataset



As we can see in Figure 4.1 which represents the convergence of GeL on FEMNIST, for a big enough value of the learning rate ($\epsilon = 5 \times 10^{-3}$) both baseline and target reach the target accuracy but GeL fails to converge even when the number of guesses is low ($g' = 2$). When we decrease the step size ϵ, GeL starts performing better: we observe in the second column ($\epsilon = 2 \times 10^{-3}$) that GeL overlaps with target for $g' = 2$ and with baseline for $g' = 4$ and $g' = 6$.
For the smallest learning rate we exprimented with ($\epsilon = 5 \times 10^{-4}$) GeL outperforms baseline in the three scenarios and overlaps with target even for $g' = 6$.
We notice a similar pattern with the Synthetic dataset: the smaller the learning rate, the closer GeL gets to target and the more guesses we can do.

Figure 4.3: Speedup achieved by GeL as a function of the learning rate ϵ for different values of g'.

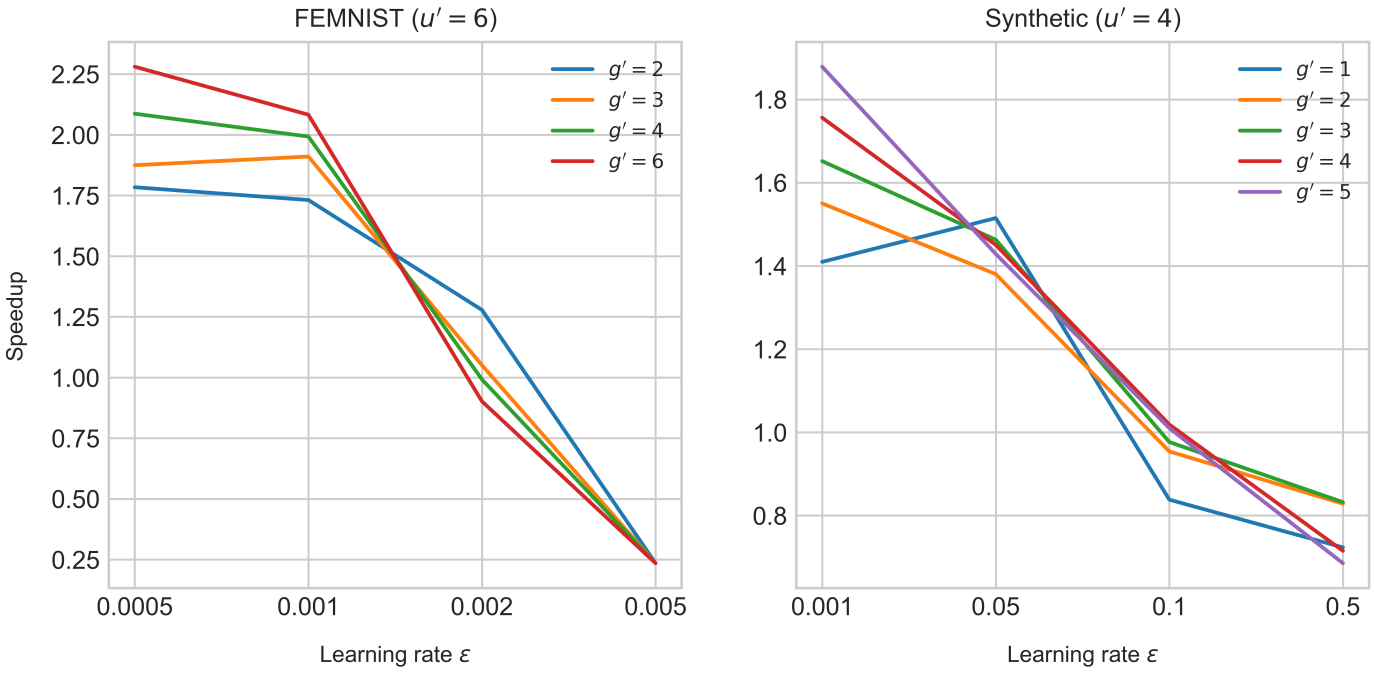


Figure 4.3 shows the impact of the learning rate on the speedup achieved by GeL. The two graphs reveal that independently of the number of guesses, the speedups reached with smaller learning rates are more significant than those attained with bigger learning rates. We can also notice that higher values of the step size can hamper GeL and start slowing down the convergence.

Discussion

The results obtained in the previous subsection show that GeL behaves better with a smaller learning rates. This can be explained by the fact that a big step size produces more movement in the optimization landscape. As we take bigger steps and move further away from the point where the last gradient was computed, the slope changes and the similarity between consecutive gradients decreases. In this case, the previously computed gradient is not a good proxy for the future gradients any more.

4.2.2 Training with SGD with Momentum

Motivation

As we mentioned in the background chapter, GeL relies on the Adam optimizer to do the guessing. With 0.9 and 0.999 as default values for α and β respectively, the new computed gradients contribute only to 10% and 0.1% of the the first and second moments estimates. One interesting question to ask is which components of Adam allow GeL to work? Is momentum the only necessary element or does the adaptative learning rate play an essential role in the algorithm. In order to isolate the momentum

component, we modify GeL to use SGD with momentum instead of Adam as an optimizer. In SGD with momentum, we retain only the the exponential average of the gradients. If we use SGD with the learning rate $\epsilon = 0.01$ and the momentum parameter $\alpha = 0.9$, the update rule becomes as follows:

$$\theta \leftarrow \theta - 0.01\, g + 0.9\, \Delta\theta$$

We can see that the old gradients contribute to nearly 99% of the update. It makes sense to use the old gradient as a guess for the future uncomputed gradients in this case as well.

Experiments

In this first experiment, we train our model using SGD with momentum as a client optimizer on FEMNIST and Synthetic datasets. On every round ,we instantiate a new optimizer on each selected client with the hyperparameters α and ϵ set to **0.9** and **0.01**.
We fix the number of selected clients per round $n_{clients}$ to **20**. For every dataset, we set a client computational budget value u' as a baseline and we study the convergence of GeL across a range of client guesses per round g' and we compare it to target which is the performance of the model if we replace the guesses with actual gradient computations. The following table presents the set of experiments we conducted for this part:

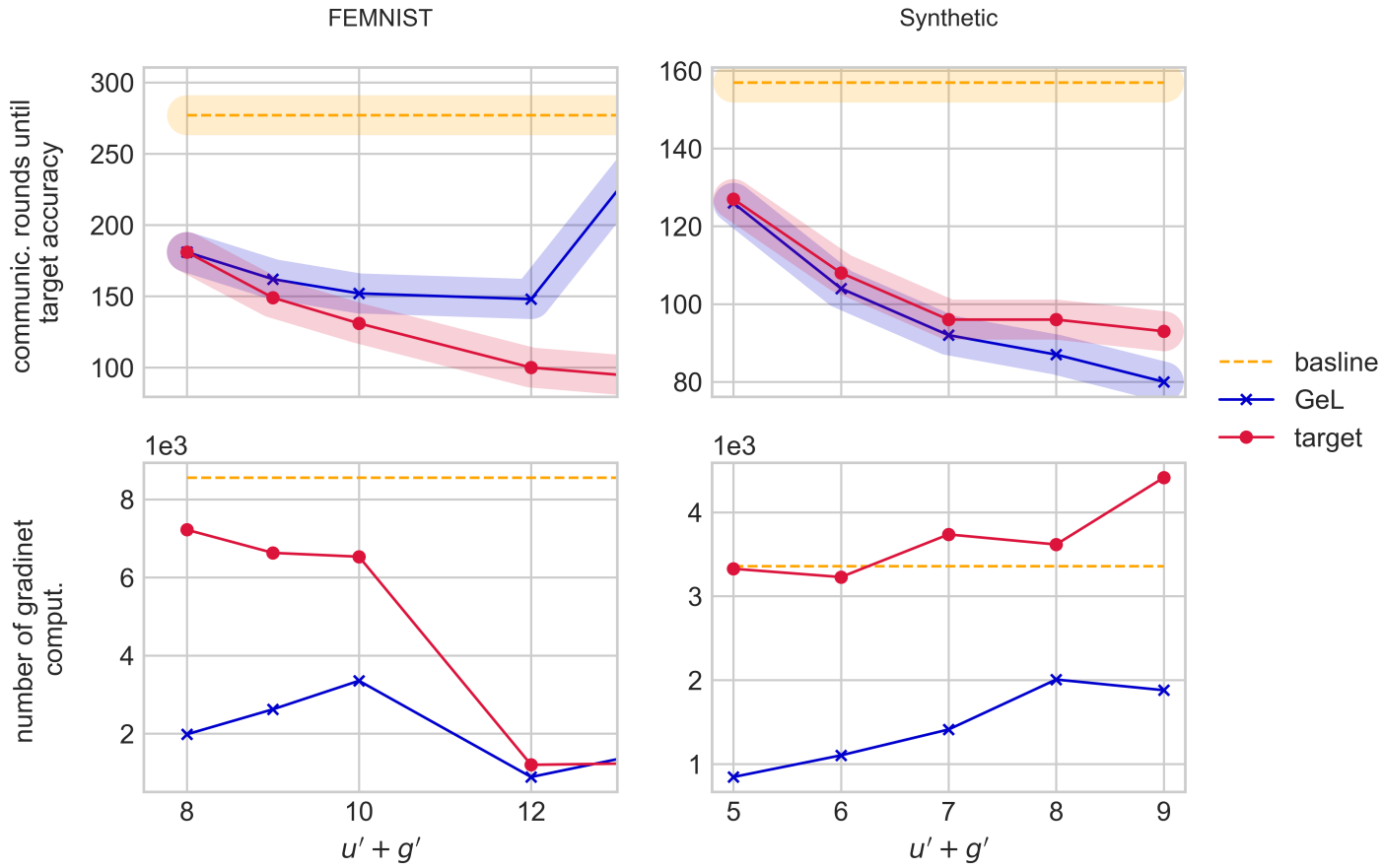
Table 4.3: Experiments conducted using SGD with momentum with $\alpha = 0.9$, $\epsilon = 0.01$ and $n_{clients} = 20$.

DATASET	**BASELINE** u'	**GUESSES** g'
FEMNIST	6	[2,3,4,6,8]
Synthetic	4	[1,2,3,4,5]

Results

In the following figures and tables [figure 4.4 and 4.5] [table 4.4 and 4.5] we present the results of the experiments mentioned in table 4.3.

Figure 4.4: Performance of GeL across a range of number of guesses



As observed from the plots in figure 4.4, using SGD with momentum with homogeneous budget and $n_{clients} = 20$, GeL still increases the speed of convergence compared to the baseline case for both datasets.
For FEMNIST, we can take a number of guesses equal to 25%, 50%, 75% and, 100% of the computational budget until guessing starts negatively affecting convergence when the number of guesses reaches 125% of u'. For $g' = 2$ and $g' = 3$ GeL meets target for the number of communication rounds until target accuracy with significantly fewer required gradient computations. Starting from $g' = 4$, target becomes faster and the gap in the required number of gradient computations between the latter and GeL shrinks.
For Synthetic, we observe that as we increase the number of guesses, the speed of convergence increases up until 125% of the computational budget. GeL overlaps with target for smaller values of g' and even outperforms it for $g' = 4$ and $g' = 5$. In addition, GeL consistently requires less gradient computations than both baseline and target for all the tested values of g'.

Table 4.4: GeL Speedup compared to baseline

DATASET	u'	g'	**BASELINE** CR_B	**GeL** CR_G	**SPEEDUP** $S_{peedup} = CR_B/CR_G$
FEMNIST	6	3	277	162	1.70x
Synthetic	4	5	157	80	1.95x

The table 4.4 shows the speedup achieved by GeL compared to baseline in terms of required communication rounds until convergence. GeL achieves a speedup of 1.70x on FEMNIST and cuts the required number of communication rounds nearly in half for Synthetic reaching a speedup up to 1.95x.

Table 4.5: Number of saved gradient computations with GeL.

DATASET	u'	g'	**GeL**		**TARGET**		**SAVINGS**
			CR_G	$G = CR_G \times u'$	CR_T	$T = CR_T \times (u' + g')$	$S = (T - G) \times n_{clients}$
FEMNIST	6	3	162	972	149	1341	7380
Synthetic	4	5	80	320	93	837	11060

In table 4.5, we present the number of saved gradient computations when using GeL instead of target. We can see that even though the number of communication rounds until convergence is relatively low, 162 for FEMNIST and 80 for Synthetic, and the number of active clients per round is small, GeL still allows us to save few thousands of expensive gradient computations. For FEMNIST, we achieve slightly slower convergence than target using 28% less gradient computations. For Synthetic, GeL impressively increases the convergence speed 1.16x using 61% less gradient computations.

Figure 4.5: Test loss and test accuracy of GeL, baseline and target.

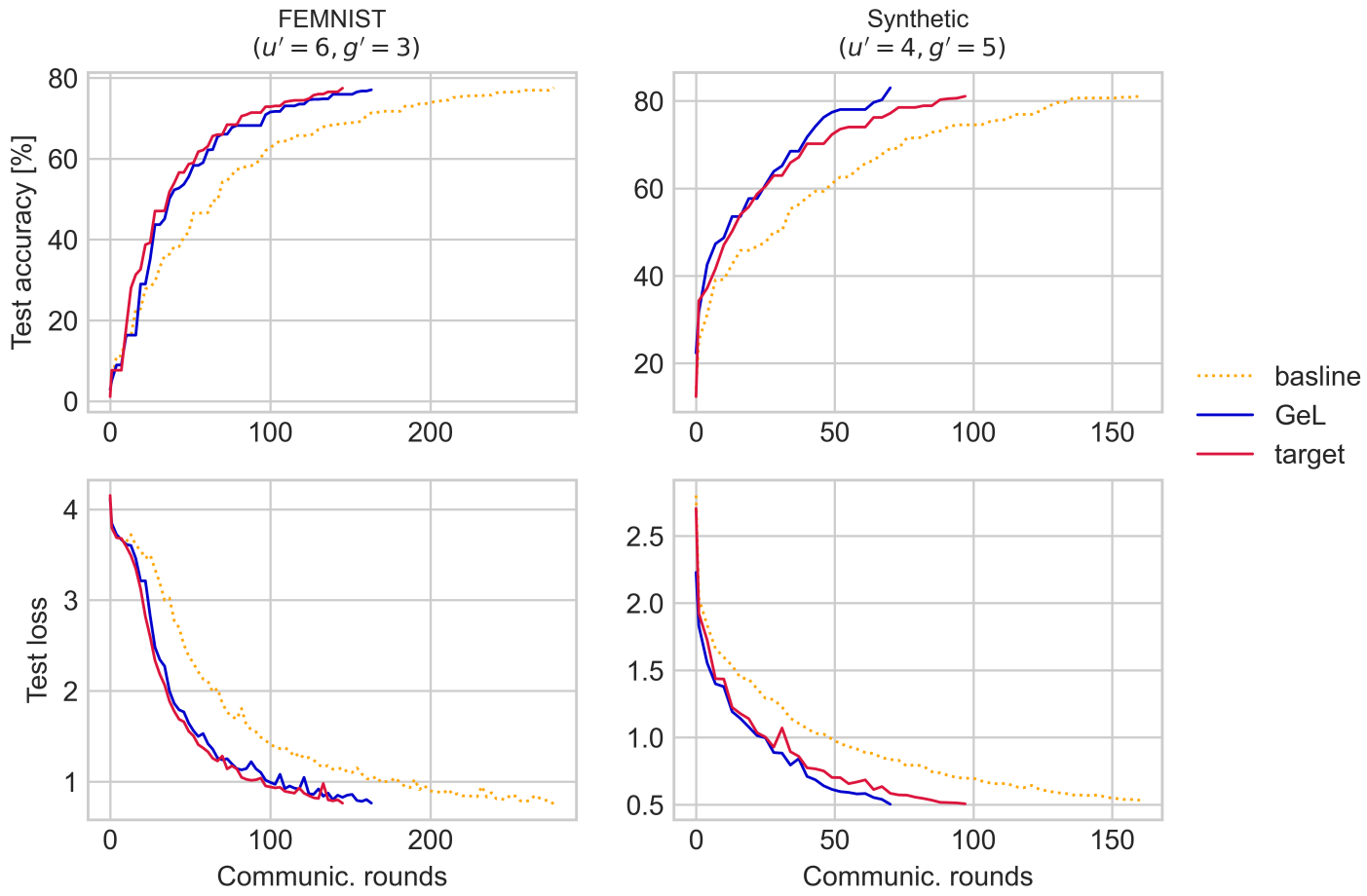


Figure 4.5 depicts the convergence of GeL in terms of test loss and test accuracy compared to baseline and target. We can observe that for both datasets, GeL achieves higher accuracy and lower test loss than baseline all along the training. After the few first rounds, GeL starts showing between 10% and 20% more accuracy than baseline for the same communication round and this gap is maintained until the target accuracy is reached. For FEMNIST, GeL overlaps with target across all rounds. The Synthetic accuracy and loss plots reveal that GeL overlaps with target across the majority of the training, but starts performing slightly better in the final rounds.

Discussion

The previous experimental results indicate that when using SGD with momentum($\alpha = 0.9$, $\epsilon = 0.01$), GeL still achieves a significant boost in performance on both datasets. This boost in performance is expressed in terms of a lower number of required communication rounds until convergence and less performed gradient computations. In some cases, guessing even allowed for faster convergence than actual gradient steps. From this empirical evaluation, we can deduce that GeL relies heavily on the use of momentum rather than the Adam optimizer itself.

4.2.3 Increasing parallelism : Number of Active Clients per Training Round

Motivation

The averaging is a fundamental component of the GeL algorithm: in order to create a global model, the FL server averages the models sent by the clients and broadcasts the new aggregated model to start a new round of training. GeL uses plain averaging to aggregate the local models following the below formula :

$$\theta \leftarrow \frac{1}{n_{clients}} \sum_{n=1}^{n_{clients}} \theta_k$$

with θ_k the model returned by client k and $n_{clients}$ the number of active clients per training round. One interesting angle is to see how the guessing interacts with the averaging by modifying the hyperparameter $n_{clients}$ and observe the impact of increasing and decreasing parallelism on the convergence of GeL. In addition, increasing the number of active clients per round will allow us to study the scalability of the algorithm.

Experiments

In this experiment, we train our model using Adam(with the default parameters) as a client optimizer on FEMNIST and Synthetic datasets.
For every dataset, we set a client computational budget value u' as a baseline and we study the convergence of GeL across a range of client guesses per round g' and we compare it to target. We repeat the previous experiment for a wide range of active clients per round $[2..200]$. The following table presents the set of experiments we conducted for this part:

Table 4.6: Experiments conducted using Adam with the default recommended hyperparameters.

DATASET	**BASELINE** u'	**GUESSES** g'	**Number of active clients per round** $n_{clients}$
FEMNIST	6	[2,3,4,6,8]	[2,5,10,50,75,100,200]
Synthetic	4	[1,2,3,4,5]	[2,5,10,50,75,100,200]

Results

Figure 4.6: Test accuracy of GeL, baseline and target for different numbers of active clients $n_{clients}$ and guesses g' on the FEMNIST dataset

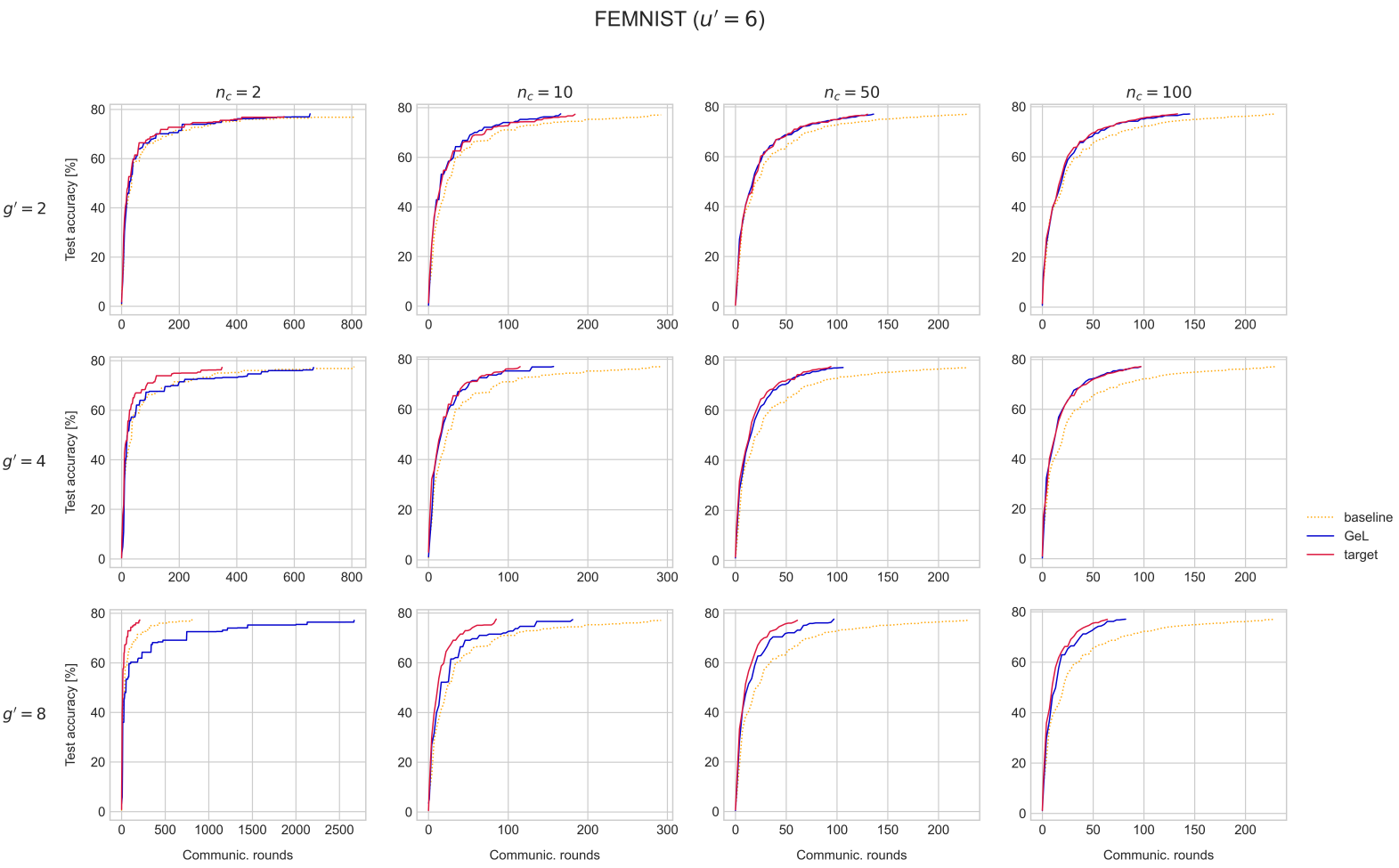
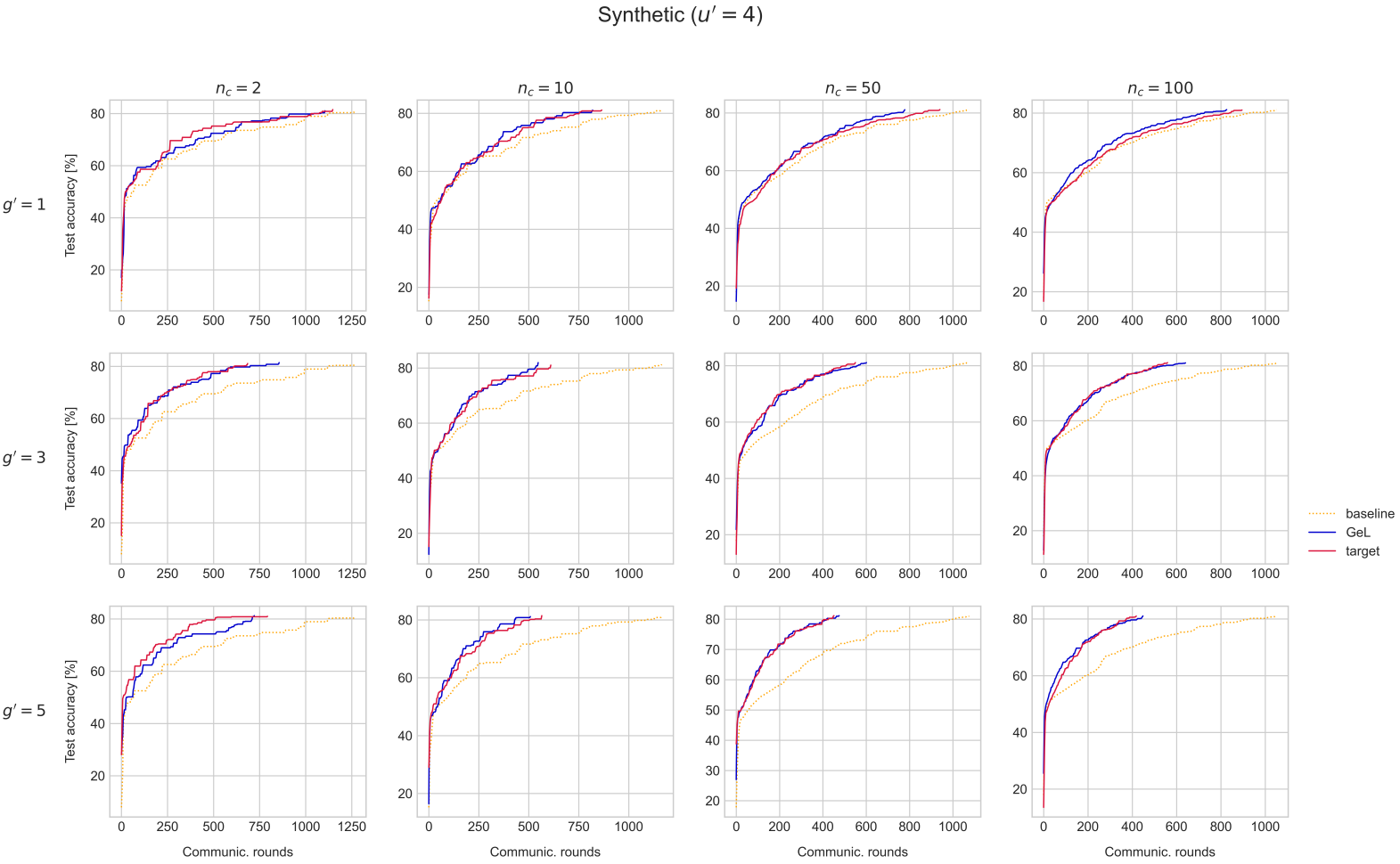


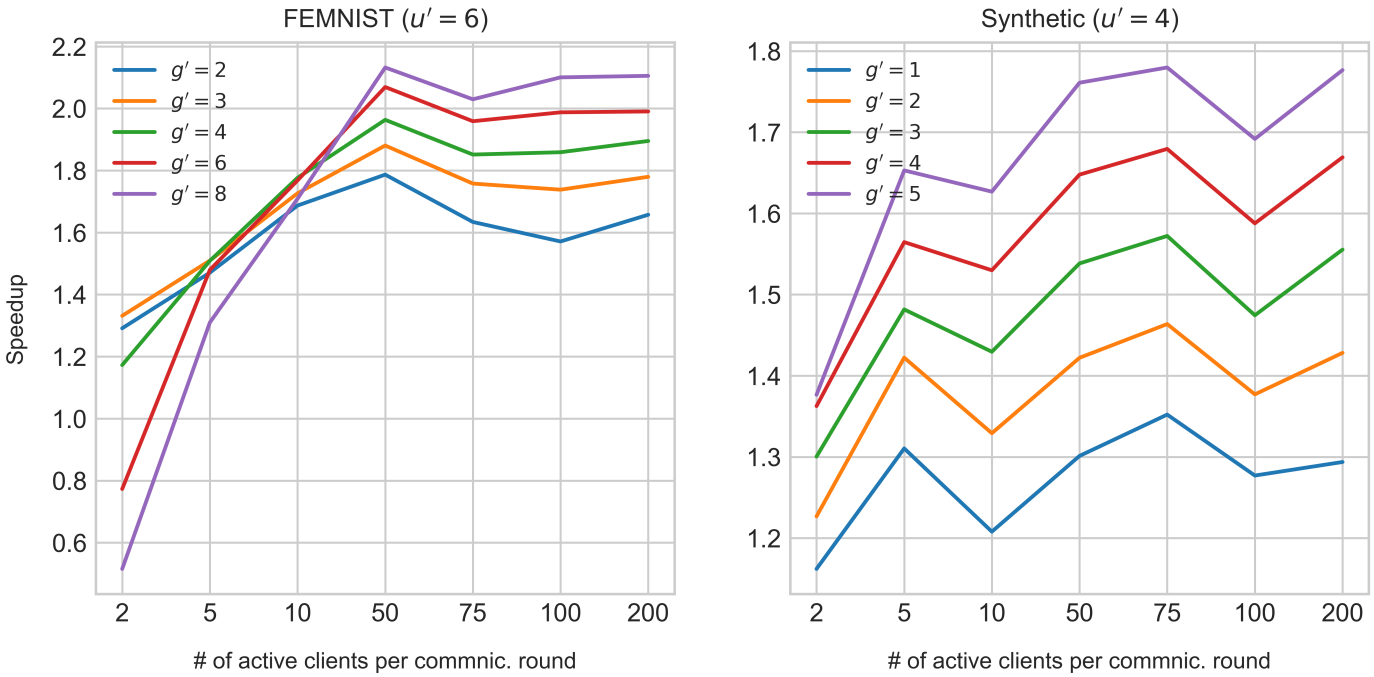
Figure 4.7: Test accuracy of GeL, baseline and target for different numbers of active clients $n_{clients}$ and guesses g' on the Synthetic dataset



As we can see from the plots in figure 4.6, for FEMNIST dataset, when the number of clients is minimal ($n_{clients} = 2$) and the number of guesses is small ($g' = 2$) GeL doesn't boost convergence that much. We can even say that GeL is overlapping with baseline for the majority of the training rounds. For bigger values of g', GeL appears to negatively affect convergence. As we increase the number of participating nodes, GeL starts to meet with target and the number of guesses required to negatively affect convergence becomes bigger. For $n_{clients} = 100$ and $g' = 8$, we are able to get close to the convergence speed of a target computational budget of 14 using a computational budget of 6.

We can observe similar behavior with Synthetic. Even though this dataset allows for high values of g' for a low number of clients, increasing parallelism improves the positive impact GeL has on the convergence.

Figure 4.8: Speedup achieved by GeL as a function of $n_{clients}$ for different values of g'.

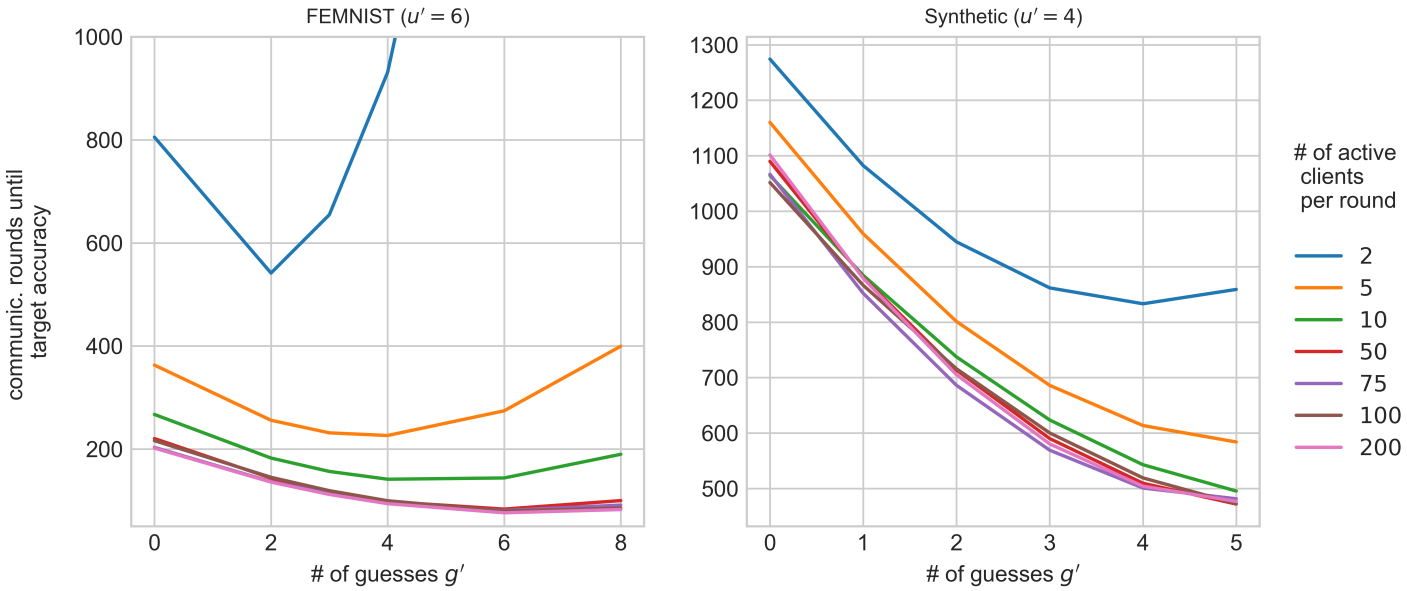


In figure 4.8, we plot the speedup achieved by GeL for different values of $n_{clients}$ for both datasets. Every line corresponds to a different value of g'.
For both FEMNIST and Synthetic, we can see that increasing the number of clients for the same value of g' gives significantly higher speedups until a certain number of clients (in this case, around 50-75 clients). From that point on, adding more participating nodes doesn't considerably affect the speedup. It is important to note that the more guesses we make per training round, the more impact varying $n_{clients}$ has on the speedup: the following table demonstrates the latter observation:

Table 4.7

DATASET	**BASELINE** u'	**Speedup difference for** $g' = 0.25\ u'$ $S^{n_c=2}_{peedup} \rightarrow S^{n_c=200}_{peedup}$	**Speedup difference for** $g' = 1.25\ u'$ $S^{n_c=2}_{peedup} \rightarrow S^{n_c=200}_{peedup}$
FEMNIST	6	$1.3 \rightarrow 1.7$	$0.5 \rightarrow 2.1$
Synthetic	4	$1.1 \rightarrow 1.3$	$1.4 \rightarrow 1.8$

Figure 4.9: Number of communication rounds until target accuracy as a function of g' for different values of $n_{clients}$



From figure 4.9, we try to find out how much we can guess depending on the number of active clients per round. The two plots reveal the same trend: As we increase the number of devices, the maximum number of guesses we can perform before starting to notice a deterioration in convergence shifts to the right and the curve $rounds_{convergence} = f(g')$ becomes flatter. In addition, we observe that as we add more parallelism, the curves $rounds_{convergence} = f(g')$ are getting close to each other until they finally overlap.

Discussion

The above results show that guessing is heavily affected by averaging. This can be seen especially when the number of participating clients is very low and the impact of guessing is either negative or insignificant. We can also deduce that adding more participating nodes (to a certain extent) improves the performance of guessing in terms of speedup and convergence regularity and allows even for more guessing which enables us to aim for higher targets.

4.2.4 Guessing in Centralized Settings

Motivation

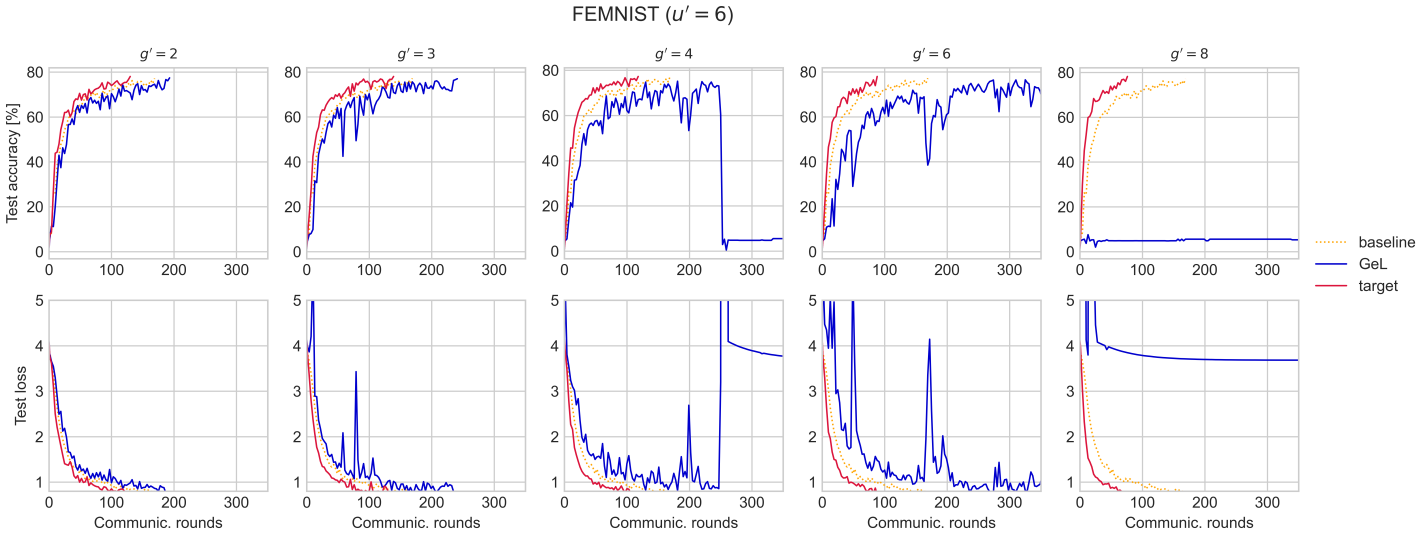
As we discussed in the previous subsection, averaging plays an important role in GeL: It allows us to ensure better convergence and to aim for higher targets. In this part of the evaluation, we remove the averaging scheme by testing GeL in traditional Centralized machine learning settings.

Experiments

We test a centralized version of GeL on the FEMNIST dataset. Since we don't have multiple nodes in this experiment, we sample an IID version of the dataset and we don't separate it into a federated dataset. We try different values of the computational budget and we use the following percentages of u' as guesses : [25%,50%,75%,100%,125%].

Results

Figure 4.10: Test loss and test accuracy of GeL, baseline and target on FEMNIST for different values of g'



The figure 4.10 shows the results we obtained in the previous experiments for a computational budget $u' = 6$. We note that similar results were observed for different budgets up until $u' = 20$. As we can see from plots, even for $g' = 0.25\ u'$, GeL is performing worse than baseline and target and fails to achieve any speedup. When we increase the number of guesses, both the test accuracy and test loss curves start to display more variance and the the number of rounds until target accuracy rises until it finally diverges.

Discussion

The above results confirm our hypothesis that the averaging is a key component of the guessing mechanism we use in GeL. With the absence of model aggregation in traditional machine learning settings, GeL can't be utilized to boost the convergence of such systems.

Chapter 5

Possible Optimizations

5.1 Adaptive Guessing

Intuition

As we explained in the discussion of subsection 4.2.1, having a step size that allows for moving in the optimization landscape without drastically changing the direction between consecutive computed gradients is very important to the effectiveness of the GeL algorithm.
Motivated by these observations, we measure cosine similarity between consecutive gradients in cases where GeL is performing well and in other scenarios where it is slowing down convergence. We remark that GeL achieves more significant speedups the higher the similarity between consecutive gradients is. Inspired by this finding, we try to propose a possible algorithm of Adaptive guessing.

Implementation

Using cosine similarity, we aim to determine when the optimization landscape is favorable for guessing. We introduce a similarity score based on two criteria:

1. How similar are consecutive gradients: a high similarity between the most recent gradients means that we are in a region where the slope is pretty much the same and guessing is more likely to achieve good results.
2. How the cosine similarity is changing from iteration to iteration: an increasing similarity over time means that the gradients are getting closer to each other and this will determine the extent to which we can guess.

In the following algorithm, we present a high level overview of an adaptive version of GeL (AdaGeL).

Algorithm 5 AdaGeL. The $n_{clients}$ clients are indexed by k; u_k and g_k are the computational budget and number of guesses for client k, and B is the local mini-batch size.

Server Executes:

Initialise global model: $\theta \leftarrow \theta_0$
Training round number: $t \leftarrow 0$
while stopping criterion not met **do**
    C clients check in with the server
    $S_t \leftarrow$ Server selects randomly a subset of $n_{clients} < C$ clients for training.
    for k in S_t in $parallel$ **do**
        $\theta_k \leftarrow ClientUpdate(k,\theta)$
    end for
    $\theta \leftarrow \frac{1}{n_{clients}} \sum_{n=1}^{n_{clients}} \theta_k$
    $t \leftarrow t + 1$
end while

ClientUpdate(k, θ):

Initialise local optimizer: $optimizer \leftarrow Adam()$
Initialise local model: $\theta_{local} \leftarrow \theta$
Initialise similarity score: $score \leftarrow 0$
for $i \leftarrow 1$ **to** u_k **do**
    $b_i \leftarrow$ Sample a random batch of size B
    $grad_i \leftarrow \nabla J(\theta_{local}, b_i)$
    $\Delta\theta_{local} \leftarrow optimizer.gradientStep(grad_i, \theta_{local})$
    if $i > 1$ **then**
        Compute Cosine similarity: $sim_i \leftarrow cosineSimilarity(grad_i, grad_{i-1})$
        Update similarity score :$score \leftarrow updateScore(sim_i)$
    end if
    $\theta_{local} \leftarrow \theta_{local} + \Delta\theta_{local}$
end for
Use last computed gradient as proxy: $proxy_{grad} \leftarrow grad_{u_k}$
Determine the number of guesses based of the similarity score : $g_k \leftarrow numberOfGuesses(score)$
for $g' \leftarrow 1$ **to** g_k **do**
    $\Delta\theta_{local} \leftarrow optimizer.gradientStep(proxy_{grad}, \theta_{local})$
    $\theta_{local} \leftarrow \theta_{local} + \Delta\theta_{local}$
end for
return θ_{local}

Chapter 6

Conclusion

In this project, we improved the gradient guessing API by enabling the guessing outside of the federated setup. We studied the impact of many parameters on the performance and behavior of the system using many experiments on three datasets.

We observed how a relatively small learning rate is essential for the guessing mechanism to achieve performance boosts. When guessing with higher learning rates, the behavior of GeL is very unpredictable and it doesn't accomplish any speedup. We have also remarked that for big enough step sizes GeL diverges even though both baseline and target succeed to reach the target accuracy. We can conclude that GeL is not robust to changes in the learning rate hyperparameter.Inspired by these results, we try a propose a modification to the algorithm which allows to dynamically determine the number of guesses based on the current optimization landscape.

We found that the guessing system doesn't depend entirely on the Adam optimizer itself but on one of its components of which is the momentum. We trained our models with momentum SGD on many datasets and obtained very significant improvements in terms of speedup and reduction of gradient computations.

Next, we experimented with the active clients fraction in the goal of understanding the impact of averaging on GeL and testing its scalability. We deduced that having a sufficient number of participating nodes allows for a more considerable performance boost in terms of communication rounds until target accuracy and even tolerates more guessing which produces more local update steps and leads to faster convergence.

Finally, we removed the aggregation scheme completely by running GeL in a centralized setup. The results obtained on FEMNIST prove that averaging is a key component to the algorithm: in a traditional machine learning scenario, gradient guessing fails to achieve any performance improvement and in many cases affects negatively the convergence.

6.1 Future works

An interesting direction to take at is how make the gradient guessing efficient in a centralized setup. In a data center setup with an abundance of data and parallelism, gradient guessing can be mounted on top of a system that uses Parallel Training of Deep Neural Network and Model Averaging [5]. In this scenario, due to the existence of an averaging scheme, GeL can achieve promising results in terms of speedup and savings.

Bibliography

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[2] H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. "Federated Learning of Deep Networks using Model Averaging". In: *CoRR* abs/1602.05629 (2016). arXiv: 1602.05629. URL: http://arxiv.org/abs/1602.05629.

[3] Akash Dhasade, Anne-Marie Kermarrec, and Rafael Pires. *Guess what? You can boost Federated Learning for free*. 2021. arXiv: 2110.11486 [cs.LG].

[4] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečný, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. *LEAF: A Benchmark for Federated Settings*. 2019. arXiv: 1812.01097 [cs.LG].

[5] Hang Su and Haoyu Chen. *Experiments on Parallel Training of Deep Neural Network using Model Averaging*. 2018. arXiv: 1507.01239 [cs.LG].