Federated Reinforcement Learning

Benjamin BOURBON - Thierry NAGELLEN

Orange

February 26, 2023

Contents

I. Introduction to Federated Reinforcement Learning

i. Introduction

Federated Learning (**FL**) was introduced by McMahan et al. (2017) which has attracted increasing interest of **ML** researchers. They proposed a decentralized collaborative approach that uses multiple devices to train a global model without sharing their local data. In most cases, devices are limited in resources such as computation and communication and restricted by the usage of the user. Federated Learning can meet expectations of specific fields such as natural language processing, computer vision, health care, transportation, finance, smart city, robotics, networking, blockchain and others. In other words, the fact of training a model without exchanging local data gives an overview of new possibilities for taking advantage of each other while keeping privacy.

However, Li et al. (2020) analyze the convergence of `FedAvg`, algorithm based on Stochastic Gradient Descent (**SGD**) which was suggested by McMahan et al. (2017) on **non-IID** data (**I**ndependent and **I**dentically **D**istributed). Indeed, on realistic settings, there is no guarantee to have **IID** data. This paper establishes a convergence rate of $O(\frac{1}{T})$ for strongly convex and smooth problems, where T is the number of **SGD**s. Moreover there are also multiple problems that are not covered with `FedAvg` approach namely making **FL** a completely distributed method, dealing with fairness of contributions or minimizing the energy consumption. The section iv will cover different challenges on Federated Learning.

In this study, we will focus on Reinforcement Learning (**RL**) which is a branch where the agent and environment are the basic components of it. The agent interacts with the environment by taking actions and receives rewards according to the impact of the action taken. The objective of the agent is to learn to behave in such a way as to maximize the expected rewards in the long term. **RL** demonstrates a great potential in various applications namely communications. The section iii will introduce basic knowledge of Reinforcement Learning.

When we contextualize the kind of application of **FL**, there are different settings that might block or reduce the efficiency of traditional methods of **FL**. More precisely, when local data are not labeled or also when the distribution of data is **non-IID**, **RL** could help to deal with this type of issue. Through the observation of the environment, Qi et al. (2021) put forward that **FL** can be divided into two categories according to the distribution characteristics of data. More details will be explained in section i.1.

Federated Reinforcement Learning (**FRL**) is the combination of **RL** and **FL**. Generally, the dimension of sample, feature and label in **FL** can be replaced by environment, state and action respectively in **FRL**. On one hand, **RL** can be implemented directly in clients' devices which takes the place of the model for each client. It can be a solution when data are not labeled or when data are located in particular environment according to the usage of user. For instance, the usage of mobile phones are not the same in cities and in rural areas. On the other hand, **RL** can play the role of a *scheduler* in order to select clients a better way. It turns out the agent is able to solve a optimization problem by finding parameters that minimize the objective function.

ii. Federated Learning

Federated Learning is a method where multiple devices (called *clients*) are led to collaborate together with communications to a central *server* in order to train a global model while protecting privacy. In other words, instead of sharing the local data of users, only updates of local models on clients' devices are communicated to the server.

FL can be applied in various situations. For instance, mobile phones represent a large amount of data, which much of it, is private by nature. The data could be acquired through GPS locations, microphones or cameras for example. If we want to build a global model such as takes account all contributions of all mobile phones, there are risks and responsibilities to deal with private data. **FL** aims to build a joint Machine Learning model (**ML**) without sharing local data. This technique could help diverse fields to collaborate together in order to train a global model that would be used by all participants. For instance, in hospitals, where privacy about health data, is the most important criterion, they could build an organization to perform a **ML** model in collaboration and every of them would take advantage of each others while keeping privacy. Another way to see **FL** application is when the implementation of the algorithm is fully decentralized. In other words, when **FL** architecture is based on **peer-to-peer**, communications between the server and devices are not required. For example, this approach offers new types of applications such as attack detection model that could be develop jointly by multiple banks.

Even if **FL** suggests an approach which protects privacy of users, there is still potential attacks which could pick up the parameters of the global model and rebuild information or data. To avoid such situations, Abadi et al. (2016) suggest **differential privacy** technique and demonstrate that we can train deep neural networks (**DNN**) with non-convex objectives. In order to go further, **FL** is going to be defined.

Formulation Consider N parts $F_i, \forall i \in [1, N]$ which are established and are used to train the cooperative **ML** model. Also, there are N datasets $D_i, \forall i \in [1, N]$. The **FL** model is a set of parameters w_i which are learned base on datasets D_i. The loss function of each dataset D_i at part F_i can be defined as follow :

$$F_i(w) = \frac{1}{|D_i|} \sum_{j \in D_i} f_j(w)$$

where $f_i(w)$ represents the loss function of sample j with the given model parameter vector w and $|\cdot|$ represents the size of the dataset. Now, we can define the global loss function thanks to multiple parts put together to train the global model without sharing a dataset :

$$F_g(w) = \sum_{i=1}^{N} \eta_i F_i(w)$$

where η_i indicates the relative impact of each part of global model.

Qi et al. (2021) present a metric to access the quality between the expected model $\mathcal{M}_{SUM}$ trained on the dataset containing all local data and the federated model $\mathcal{M}_{FED}$:

$$|\nu_{SUM} - \nu_{FED}| < \delta$$

where ν is the estimated performance such as accuracy, recall and F1-score, etc, and δ is a non-negative real number.

Despite good performance of **FL**, this method has difficulty to fit in dynamics environments specifically cooperative and optimal decision-making. To solve these real-world challenges, **RL** and Deep Reinforcement Learning (**DRL**) have proved excellent performance in many problems.

iii. Reinforcement Learning

Machine learning challenges are grouped in supervised, unsupervised or Reinforcement Learning depending on the situation. In supervised learning, training data set is labeled and the objective of the model is to build a relation between the input, output and system parameters. In unsupervised learning, the model has no feedback due to unlabeled training data set. Therefore, the objective of the algorithm is to find similarities between the input samples in order to approximately learn some patterns from untagged data. Finally, Reinforcement Learning takes place for a goal-oriented learning tool. In Reinforcement Learning (**RL**), an agent interacts with an environment. At each time-step, the agent observes its environment and selects an action A_t. The agent receives a reward R_{t+1} depending of its action on the environment. The environment changes over the time with state S_{t+1} (see figure 1). Generally speaking, the agent behaves like a decision maker which learns through a policy $\pi(s)$. Its goal is to optimize a long-term reward by interacting with the environment. **RL** includes all work done in all areas such as psychology, computer science, economic, and so on.

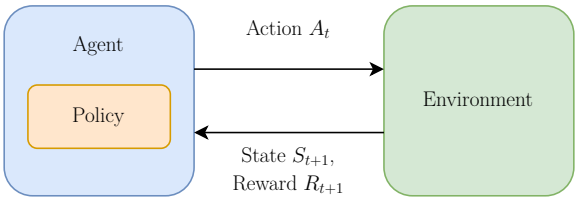


Figure 1: *Basic overview of Reinforcement Learning model*

Formulation We define the future discounted return R at the time t as :

$$R = \sum_{t=0}^{\infty} \gamma^t r_t$$

where r_t is the reward at the step t and $\gamma \in [0, 1[$ is the discount-rate. γ is less than 1, which allows to weight less distant future events than events in the immediate future. For instance, in case we would like to reduce the energy consumption of devices on Federated Learning, the agent is rewarded properly when the impact of the actions are beneficial. And on the contrary, the agent is punished (negative rewards) when the consumption of energy is worse than previous state.

Bellman (1957), who worked on Markov Decision Process (**MDP**), found a way to estimate the optimal-value $Q^*(s, a)$ which is the maximum expected return achievable by following any strategy, after seeing some sequence s and then taking some action a :

$$Q^*(s, a) = \max_{\pi} \mathbb{E}\left[R_t | s_t = s, a_t = a, \pi\right]$$

where π is a policy mapping sequences to actions.

The optimal action-value function obeys the **Bellman equation**. If the optimal value $Q^*(s', a')$ of the sequence s' at the time-step was known for all possible actions a', then the optimal strategy to select the action a' maximising the expected value of $r + \gamma Q^*(s', a')$:

$$Q^*(s, a) = \mathop{\mathbb{E}}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s', a') | s, a\right]$$

where $\mathcal{E}$ is the environment.

After getting the optimal Q values, it is simple to define the optimal policy $\pi^*(s)$. In the state s, the agent must choose the action which has the highest Q value in this state. In other words, $\pi^*(s) = \underset{a}{\text{argmax}}\ Q^*(s,a)$

We can use the **Bellman equation** as an iterative update $Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma Q_i(s',a')|s,a\right]$ in order to converge to Q^* when $i \to +\infty$. A function for approximation is defined to estimate the action-value function $Q(s,a;\theta) \approx Q^*(s,a)$. A neural network can be trained by minimizing a sequence of loss functions $L_i(\theta_i)$ that change at each iteration i :

$$L_i(\theta_i) = \underset{s,a\sim p(s,a)}{\mathbb{E}}\left[(y_i - Q(s,a;\theta_i))^2\right]$$

where $y_i = \underset{s'\sim\mathcal{E}}{\mathbb{E}}\left[r + \gamma\ \max_{a'} Q(s',a';\theta_{i-1}|s,a)\right]$ is the target for iteration i and $p(s,a)$ is a probability distribution over sequences s and actions a that refers to the *behaviour distribution*.

iv. Overview of trade-offs of Federated Learning

As mentioned in the introduction, **FL** presents several challenges to solve.

Client challenges

1. In real-world applications, data are not necessarily *labeled*. On this condition, local models and global model have to learn through unsupervised or reinforcement techniques. For instance, if we would like to use the dataset of three different sales-oriented clothing applications, on the one hand, each application manages its data as they were designed for. They are not supposed to follow the same design or *"codification"*. Thus, it is harder to find a way to identify corresponding features between datasets. On the other hand, we are not able to label data knowing that the scope of each application is not the same with each other. The objective is to gather information of applications in order to compute a local update without having to do the preprocessing of the data.

2. Even in the case of supervised learning with labeled training dataset, there is still no guarantee of the data distribution. In other words, some clients could present specific characteristics like their location or their habits which influence local dataset. It involves to deal with **non-IID** data and moreover, it shows that at a global viewpoint, data could be unbalanced, which means classes are not distributed identically.

3. As well, Internet of Things (**IoT**) or embedded devices have often limited resources. In some cases, the computation or the communication with the server can be slowed down or even interrupted during the activity. We can think when battery is discharged for computation or when there is a loss of connection on WiFi or 4G network for communication. This problem is called *the straggler problem*.

Communication challenges

1. Communication plays a main role in **FL**. It must be minimize to speed up the training of the global model because the *number of communications* represents the bottleneck of **FL**, mainly in cross-device configuration.

2. Besides, the *energy consumption* is closely related to the number of communications and the amount of information transferred between the device and the server. The energy consumption must also be minimized for the computation but, we assume, on the same embedded device, the more we reduce the energy consumption, the more the training is slowed down.

3. During communications, global model parameters and local updates are communicated between participants and the server. It introduces a risk of potential attacks because, not only privacy still remains in local updates but also because attackers are able to build information from parameters. *Privacy* must be kept protected during communications.

Due to different habits of clients, local updates impact differently the global model according to their contributions. In other words, clients, who have spent more time using their device, participate more and therefore the global model will be more dependent on their usage. Then, the *fairness* is not respected because of the imbalance of contributions of participants.

Server challenges

1. Federated Learning algorithms often start with the assumption of a centralized server. However, in order to apply those algorithms in a fully distributed way, the server has to be removed and replaced by a *peer-to-peer* network.

2. In some situations, a centralized way remains a better solution than distributed network. One way to reduce communication is spread the flow through Ultra Edge Computing, Edge Computing and Cloud Computing. Each level of computing could take charge of different services in the interest of reducing the workload.

Challenges of **FL** and some technical solutions to solve them are illustrated on figure 2. Horizontal Federated Learning and Vertical Federated Learning are described in Qi et al. (2021) with more details in section i.1.

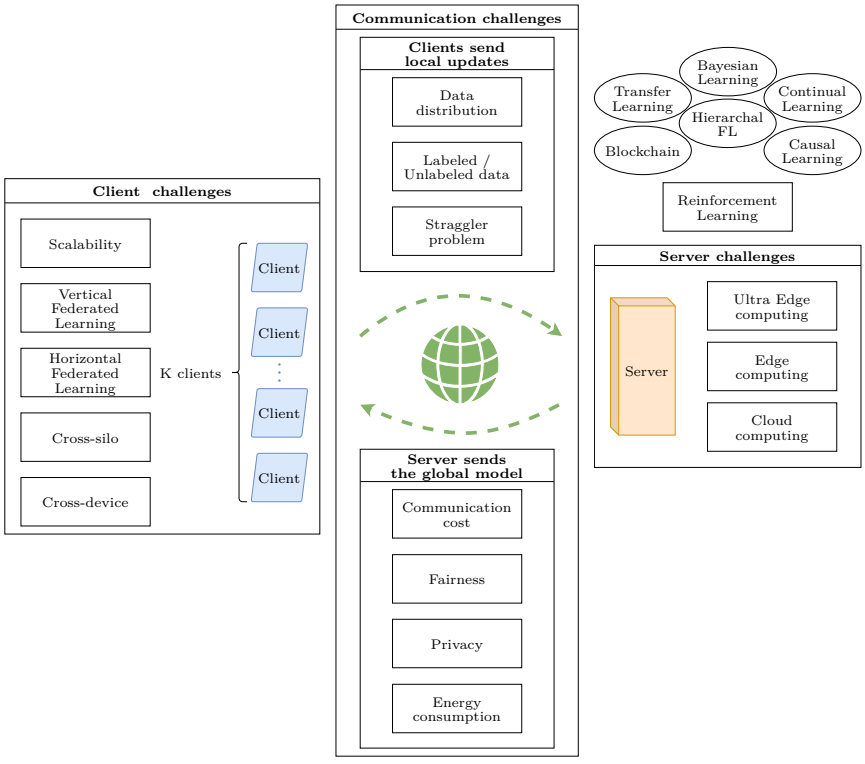


Figure 2: *Overview of Federated Learning with principal dimensions to deal with*

v. Motivation

Various papers propose solutions to achieve tremendous results with **IID** data whereas in real-world situations, we find more often **non-IID** data. Moreover, working on local data of users is dealing with **non-IID** data because each user has his own habits and his own interests. Furthermore, in some situations, data are unlabeled. This problem can be illustrated when local data are collected on mobile phones through different applications. Applications propose different services and then collect and give them according to the application was built. Because of this, it is not necessary possible to merge datasets in order to generate a unique local dataset for each client. There are some difficulties when the application does not provide features, because the analysis and preprocessing of data are not feasible through a generic algorithm. Moreover, it involves models in clients' devices should be able to learn through unsupervised learning algorithms.

In another point of view, due to the protection of privacy of users, the biggest part that can be improved is server part. In other words, the server receives all updates from clients and decides how to gather them and train the global model. While an additional process which it is called the *scheduler*, is in charge of selecting the clients for each round.

This paper aims to propose a solution for applications which tends to **unsupervised learning**. We assume that settings is known as cross-device learning, which means the number of clients is very large. It presupposes that only a fraction of clients is available at any time. We assume the primary bottleneck is often communication knowing that generally, devices have heterogeneous connections.

II. Related work

i. General ideas

i.1 HFRL and VFRL

The comprehensive survey by Qi et al. (2021) focuses on Federated Reinforcement Learning to put forward ideas where the Reinforcement Learning could solve specific situations where **FL** could be improve "to deal with cooperative control and optimal decision-making in dynamic environments". They introduce the two scenarios : Horizontal Federated Reinforcement Learning (**HFRL**) and Vertical Federated Reinforcement Learning (**VFRL**).

In **HFRL** scenario, the environment that each agent interacts with, is independent of the others, while the state space and action space of different agents are aligned to solve similar problem. A typical example is when the autonomous driving system where various environment information is collected to train the autonomous driving models locally. However, multiple factors such as weather conditions or driving regulations may differ from a country to another even if vehicles have same operations (braking, acceleration, steering, etc.). Users could share their learned experience of their model with each other without revealing their driving data.

In opposition, in **VFRL** scenario, multiple agents interact with the same global environment but each can only observe limited state information in the scope of its view. For example, for photovoltaic management company, the balance between the power generation and consumption should be optimized. It depends on the electrical utilities of household users and the decisions of the power company on the power dispatch of generators. The electricity bill can be seen as "rewards" for household users while the power company get "rewards" from the management of power generation. Both don't want to share their private data to others. Instead, **VFRL** can help to improve policy decisions without exposing specific data.

i.2 Integration of Federated Learning within Edge

Internet of Things or the emergence of multimedia services over mobile networks produce a huge rise in the traffic and computation for mobile users and devices over the recent years. Edge computing can significantly extend the capacity of cloud. Indeed, the improvement of the content deliveries and the quality of mobile services are at the core of network edges. In order to optimize edge systems, Wang et al. (2019) explain how to integrate Deep Reinforcement Learning techniques in Federated Learning framework aiming to bring more intelligent edge systems. They also discuss on User Equipments (**UEs** or also called Ultra Edge computing) for pushing computation and storage resources to the geographical proximity.

Beforehand, they introduce Deep Learning solutions with their benefits and their drawbacks on figure 3. To reduce traffic between **UEs** and the cloud, the edge part equipped with AI computation could combine massive **UEs** and the cloud together intending to form a powerful AI entity. We can imagine that each edge node support AI tasks to ensure global optimization and meet the expectation of the local dynamic system. Thus, the edge node needs to deal with the cloud and the **UEs**. The cloud server coordinates all edge nodes while they keep a **DRL** agent and updates it by their own local training data. Furthermore, in a scenario of computation, the edge nodes receive instructions from **UEs** through wireless channel. According to the demanded task and the energy consumption, edge nodes are able to share the amount of tasks. In other words, they are dedicated to a specific task fulfilled by the inference result of the agent in it. Such infrastructures offer the ability to deal with a large-scale data even if **UEs** as mobile phones, industrial **IoT** devices and smart vehicles are limited by the computation and memory resources.

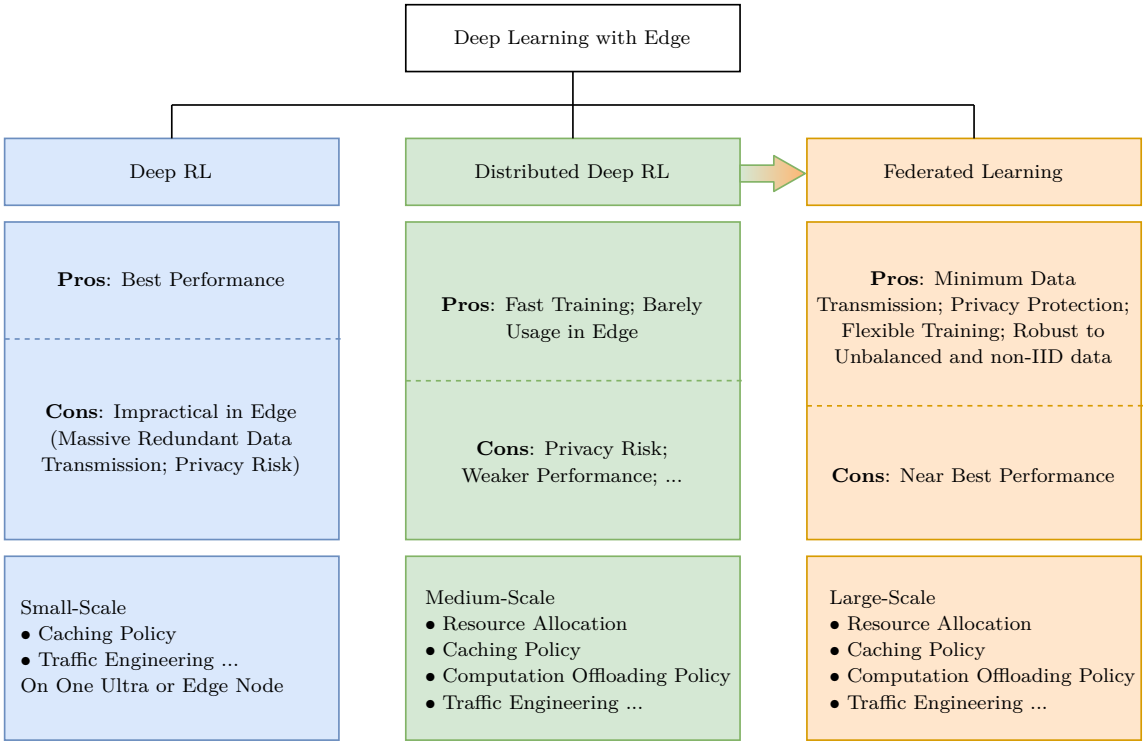


Figure 3: *Taxonomy of applying Deep Reinforcement Learning in mobile edge system by Wang et al. (2019)*

Then when **FL** is implemented in such infrastructures, through distributed training, it provides the quality of knowledge across a large number of devices. One challenge that they emphasize, is the idea of not deploying directly neural networks with at the random state (initialization of weights). Indeed, knowing that learning takes long time of training, it should be better to start **FL** with pretrained neural networks. One way to solve this challenge is to boost the training process in edge computing systems using transfer learning. Then, in the case of Reinforcement Learning, the **DRL** agent should be trained offline before being distributed in the system.

FL process with **DRL** agent is illustrated on figure 4. The iterative process solicits clients depending on actions chosen by the agent. They download the parameters of the global model from a certain server. Then, they replace their local model by the downloaded one. After that, they perform the training process and upload their local updates to the server. The server aggregates the local updates on the global model. According to various collected information, which could be the time spent on computation, the bandwidth or characteristics of the local data (mean, variance, entropy, ...), the agent observes a new state and takes an action. The action reflects which next clients should participate or not. The agent here is described as a Deep Q-Network (**DQN**) with a replay memory.

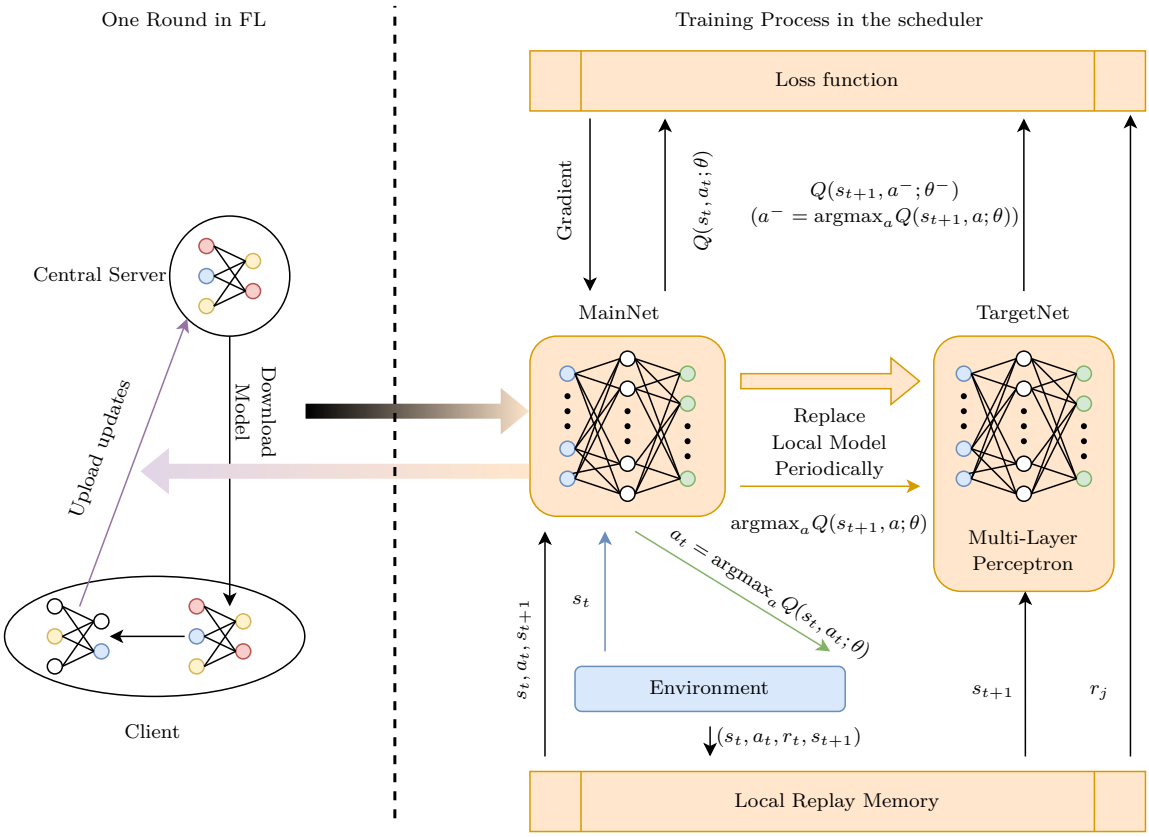


Figure 4: *General process of **DRL** with Federated Learning over mobile edge system (figure inspired by Wang et al. (2019))*

i.3 Other ideas

To tackle the lack of self-adaptive ability in dynamic environments and the centralized **FL**, Wang et al. (2020) develop a federated deep-reinforcement-learning-based cooperative edge caching (`FADE`) approach which is decentralized federated learning supervised by a **DRL** model. The edge aggregator disperses the global model parameters to devices through the policy of the Reinforcement Learning agent. Also, it plays the role to obtain the global loss function from the collected parameters of participants.

Even if the local data are not exchanged between participants and the server, there are still some potential risks of attacks to intercept communications to get parameters and sensitive information through them. To avoid this issue, Martinez et al. (2019) describe a workflow including *blockchain* which aims to establish data security. The main idea is, once all local gradients are collected, the algorithm follows the process of training the next model, validates the transactions and paying the users.

Furthermore, Yu et al. (2020) suggest to use a two-timescale deep reinforcement learning and blockchain into communication networks. The main goal is to optimize communication costs by minimizing the total offloading delay, computation offloading, resource allocation and service caching placement. The secure blockchain communication is applied to ensure the efficient and reliable cooperation in networks.

Federated Learning still remains a learning technique which asks many resources and especially energy. The papers in the literature (Zhan et al. (2020), Zarandi and Tabassum (2021), Chen et al. (2021)) that deal with the energy consumption, formulate an optimization problem where several units (computation delay, communication delay, bandwidth, ...) are taken into account. They introduce a **RL** algorithm where its goal is to find the optimized parameters that minimize the objective function given the current state of the environment.

In some scenarios, data obtained at the client-side are not labeled due to high labeling cost or difficulty of annotation for instance. `FSSL` (Federated Semi-Supervised Learning) was introduced by Jeong et al. (2021) tries to solve two challenges : when the clients have both labeled and unlabeled data and when the labeled data is only available on the server. Instead of using Reinforcement Learning technique, the authors base their idea on *inter-client consistency loss* which is one of the most popular approaches to learn unlabeled examples in a semi-supervised learning setting. As well, to find the parameter decomposition for disjoint learning (supervised and unsupervised learning), they minimize the loss term depending on the cross entropy between labeled data and the model output.

However, in most settings, the distribution of the data in local dataset of devices is not **IID**. Ghosh et al. (2019) propose sparse ternary compression which is specifically designed for unbalanced and **non-IID** data. Their algorithm does not use **RL** and outperforms on *CIFAR dataset* in unbalanced and **non-IID** settings other algorithms.

ii. Reduction of communication cost

The section iv highlights that **FL** exposes the number of communications to the main bottleneck of the algorithm. To solve this problem, one idea is to accelerate federated learning through momentum gradient descent (Liu et al. (2019a)). This method allows us to take advantage of local gradients to improve the speed of convergence and thus it reduces the communication cost. Also, Smith et al. (2018) put forward federated multi task learning where they manage to separate computation across the nodes. Their algorithm `MOCHA` is able to deal with high communication cost, stragglers, and fault tolerance for distributed learning.

ii.1 Deep Reinforcement Learning assisted Federated Learning algorithm

Recent papers have tried to solve this problem thanks to **RL** playing the role of scheduler. Zhang et al. (2021) are interested in industrial Internet of Things (**IIoT**) and model the training time cost and the training quality cost. The total cost is given by adding both of them that the agent minimizes by selecting the best **IIoT** equipment nodes each round. Formally, let consider C^t_{time} the total time cost of training for all equipment :

$$C^t_{time} = \frac{1}{N_e}\sum_{i=1}^{N_e}(c^t_l(i) + c^t_c(i))$$

where N_e denotes the number of equipment which participate for the training, $c^t_l(i)$ represents the local training time of the device i at the time t and $c^t_c(i)$ represents the communication cost of the device i at the time t. Moreover, the total of training quality cost for all equipment C^t_{qu} is considered as :

$$C^t_{qu} = \sum_{i=1}^{N_e} \underbrace{\sum_j \text{Loss}(y_j - \hat{w}^t(x_j))}_{\sigma_i}$$

where σ_i quantifies the quality of the network model and $\hat{w}^t$ represents the training model aggregated at the time t. Thus, the total cost in the time slot t is :

$$C^t = C^t_{time} + C^t_{qu}$$

A **RL** agent is used to find the optimal solution to minimize the total cost. The agent observes the current state which is characterized by the performance of **IIoT** equipment, the quality of local data and the gradient loss of these **IIoT** equipment. Then, the agent takes action a_t which after all, is a vector with a size of the number of nodes, filled with 0 and 1 where when the node i is selected, $\delta^t_i = 1$, and otherwise $\delta^t_i = 0$. And the evaluation method is a weighted average as follows:

$$r(s_t, a_t) = -\frac{\sum_{i=1}^{n} C^t_i \cdot a^t_i}{\sum_{i=1}^{n} a_i}$$

The performance evaluation of the solution of this paper shows results about the accuracy to reduce communication costs on the MNIST data set with **IID** data and **non-IID** data. However, the evolution of the total reward was not put forward and it would essentially not prove the efficiency of the **RL** agent. Even if the convergence of the loss value was pointed out during the federated learning, there is a lack of explanation on how where selected the participants. We could conclude the agent figures out that the best way to minimize the communication costs is to select the most fastest devices and with the best quality. In other words, the algorithm faces two issues :

- the fairness is not taken into account during the training of the global model. In other words, the agent would move aside some participants and prioritize other ones even though its goal is to minimize the communication costs.
- there is no reason to believe that the agent would be able to find new participants over the time if the agent selects the same participants to maximize its rewards. For instance, if devices change behaviors over the time because of their usage, the agent could be stuck in local minima, because of a lack of exploration over the time.

ii.2 A Multi-agent Reinforcement Learning approach in Federated Learning

Another approach to reduce the communication costs is to define a cooperative Multi-Agent Reinforcement Learning (**MARL**). Zhang et al. (2022) propose to set N agents to be trained to produce optimal actions in order to maximize a team reward. The method is called `FedMARL`. The main idea of this strategy aims to tackle the **non-IID** training data problem which represents a real-world problem. So, it improves the model accuracy with much lower processing latency and communication cost. A set of N agents is defined in such a way where each one of them must maximize the total expected discounted reward. Each agent selects the action a^* with the maximum Q-value. To take the *final action*, a joint Q-function $Q_{tot}(s_t, a_t) = \sum_n Q^\theta_n(s^t_n, a^t_n)$ is defined where s_t and a_t are the set of states and the set of actions collected from all agents at the time t.

Like the previous paper Zhang et al. (2021), the problem to solve is an optimization problem. They supposed that "a subset of clients will be early rejected by the central server while the rest clients will continue to finish their local training process". They are interested in the communication latency $H^u_{t,n}$ for uploading the local model from a client n. As well, B^t_n represents the communication cost for sending updates from client n to the central server. The local training time for client n is denoted as $H^c_{t,n}$ H_t represents the total processing latency of a training round t as well as the total communication cost B_t for the round t :

$$H_t = \max_{1 \le n \le N} (H^u_{t,n} + H^c_{t,n}) a^t_n \quad \text{and} \quad B_t = \sum_n B^t_n a^t_n$$

Finally, the problem is to optimize :

$$\max_{A} E\left[\underbrace{w_1 Acc(T)}_{Obj_1} - \underbrace{w_2 \sum_{t=1}^{T} H_t}_{Obj_2} - \underbrace{w_3 \sum_{t=1}^{T} B_t}_{Obj_3}\right]$$

where $Acc(t)$ represents the accuracy and $A = [a_n^t]$ is a $T \times N$ matrix for client selection, with T the number of rounds and N the number of workers. w_1, w_2, w_3 are chosen by the user to design the **FL** application. Obj_1 must be maximized in order to get the best accuracy while Obj_2 must be minimized to reduce communication latency and Obj_3 must be also minimized to reduce communication cost.

Now, the state s_n^t of agent n at the round t is defined as follows:

$$s_n^t = \left[L_n^t, H_{t,n}^p, H_{t,n}^u, B_n^t, D_n, t\right]$$

where D_n is the size of the local dataset and t is the round index. On the central server, each **MARL** agent n produce a binary action $a_n^t \in [0, 1]$ where 0 indicates the device will be terminated after the probing training and vice versa. We indicate the reward r_t at training round t as :

$$r_t = w_1 \left[U(Acc(t)) - U(Acc(t-1))\right] - w_2 H_t - w_3 B_t$$

with H_t the processing latency of round t which is defined as :

$$H_t = \max_{1 \leq n \leq N} \left(H_{t,n}^p\right) + \max_{\substack{1 \leq n \leq N \\ \text{with } a_n^t = 1}} \left(H_{t,n}^{rest} + H_{t,n}^u\right)$$

Note $U(\cdot)$ is a utility function defined to ensure that the difference $U(Acc(t)) - U(Acc(t-1))$ is no-null when $Acc(t)$ improvement is small near the end of **FL** process. Also, we recall for each client n, $H_{t,n}^p$ represents the training latency, $H_{t,n}^{rest}$ is the time required to finish the local training process and $H_{t,n}^u$ is the communication latency. B_t is the total communication cost as it was defined at the beginning of the problem formulation.

According to their results, `FedMARL` outperforms classical algorithms such as `FedAvg`, `FeteroFL`, `FedNova` or `FedProx`. Note that agents are multi-layer perceptrons (**MLP**) with only two layers which is cheap to implement. Moreover, the figure 5, where dots represent a client device, illustrates that agents do not select same participants over the time. It indicates how the scheduler can adapt its choices over the time to find the optimal participants at each step of the time.

However, this approach would be harder to implement in peer-to-peer configuration. Indeed, the loss function to train the main agent depends on $Q_n^\theta(s_n^t, a_n^t)$ which means that all $Q_n^\theta(s_n^t, a_n^t)$ of agent n has to be collected. Also, the contributions of participants were not studied to show how the global model influences the accuracy on local dataset to reveal the decrease of communications. In other words, if the global model has always good impacts on local accuracy compared to the local model, thus the global model will converge quicker. It is obviously an ideal configuration, but by minimizing negative impacts, it improves the convergence of the global model.

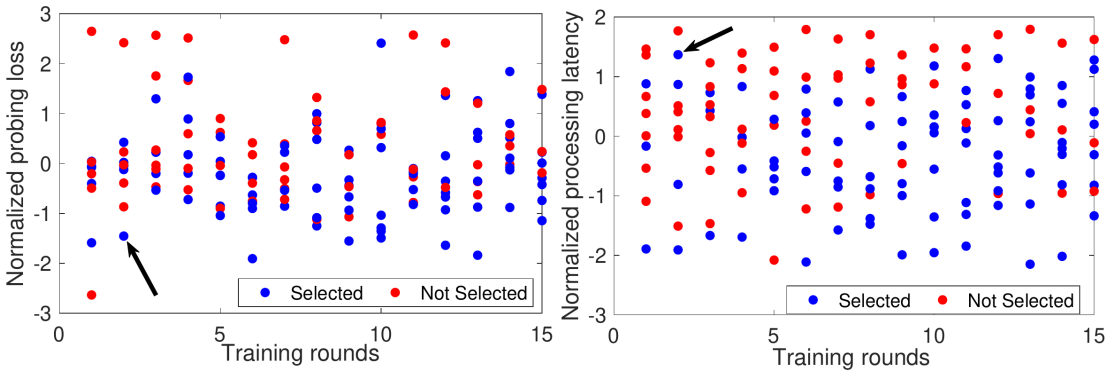


Figure 5: *Decisions made by FedMarl. Each dot represents a client device. The blue dot means the client is selected for local training, the red dot means the client is early rejected (results from Zhang et al. (2021))*

iii. Improvement of aggregation

In this part, we will discuss about ideas where the management of learning process during aggregation is improved aiming to enhance the convergence of local models. For instance, Liang et al. (2019) demonstrate through transfer Reinforcement Learning, that it is possible to share knowledge between participant agents. This work highlights the potential to make agents to learn cooperatively through each other in very different environments. Another idea is to avoid to change too much the new model from the previous model where there is a new update. That is what Lim et al. (2020) propose where independent **IoT** device finds its optimal learning in its environment taking advantage of other devices and with only benefits. A critic model learns to evaluate if the action taken by the actor model (*supervisor*) led the device to be in a better state or not. It leads to a better optimization of local models. At this time, several papers bring up interesting ideas.

iii.1 Cross-gradient aggregation

Esfandiari et al. (2021) are interested in *cross-gradient aggregation* where the algorithm is based on quadratic programming projection. The algorithm does not use a **RL** agent; however it is intended for **non-IID** data. Two algorithms are proposed: the basic cross-gradient aggregation algorithm and a better version of the first one in terms of communication costs; compressed cross-gradient aggregation algorithm (**compressed `CGA` algorithm**). Contrary to the averaging approach which succeeds most of the time on **IID** data, they seek a descent direction which is close to $g^{jj} = \nabla_x f_j(\mathcal{D}_j; x^j)$ and simultaneously is positively correlated with all the cross-gradients. $\mathcal{D}_j$ represents the dataset of the client j and x_j the model parameter copy of the client j. The cross-gradient is defined as $g^{jl} = \nabla_x f_l(\mathcal{D}_l; x^j)$. Then, the algorithm aims to minimize the quadratic programming projection as follows :

$$\min_z \frac{1}{2} z^T z - g^T z + \frac{1}{2} g^T g \quad \text{where} \quad (g^{jl}) z \leq 0$$

Note for information only that the compressed CGA algorithm uses the Error Feedback SGD (see Stich and Karimireddy (2021)) to compress gradients.

At the end, they show good results of the method on **non-IID** especially with the basic approach (non compressed). Unfortunately, there is no recommendation or specific information to help us to choose correctly the hyperparameters α and β mentioned in the algorithm. Moreover, the

communication rounds for CGA are pretty important : in the experiments, twice compared to `SGP` and 4 times the communication rounds of `SwarmSGD`.

iii.2 Lifelong Federated Reinforcement Learning

Speaking about **RL** makes often reference to robots. It is common to use Reinforcement Learning algorithms for mobile robotic navigation in order to reach target position and avoid obstacles. Liu et al. (2019b) propose a **L**ifelong **F**ederated **R**einforcement **L**earning architecture (**LFRL**) to reduce training time, storing data, over the time for application of Reinforcement Learning in navigation. Even if the application is specific for navigation applied to mobile robots, the idea suggested is still relevant for other applications.

First of all, to start quicker the training time, the initial Q-network in robots has the ability to reach the target and avoid some types of obstacles. To avoid random initial weights, a transfer learning of a model could be trained on anonymized data for privacy purposes. Secondly, the key of the algorithms result on *knowledge function algorithm* and *transferring approaches*.

Knowledge function algorithm In robotic field, inputs are generally based on sensor data attributes. The degree of confirmation is defined on which action (output) the robot chooses to perform as the "confidence value". This idea is basically to highlight when there is a significant differentiation in the evaluation of Q-values, which it is called the *scoring process*. For instance, the values $(20, 19, 16, 14, 20)$ are less different than the values $(5, 4, 80, 8, 10)$. The information entropy is chosen to characterize the confidence as a quantitative function of robotic confidence:

$$c_j = -\frac{1}{\ln m}\sum_{i=1}^{m}\left(\frac{\text{score}_{ij}}{\sum_{i=1}^{m}\text{score}_{ij}}\cdot\ln\left(\frac{\text{score}_{ij}}{\sum_{i=1}^{m}\text{score}_{ij}}\right)\right)$$

where m is the action size of robot j, n is the number of private networks. Also the memory weight of robot j is specified as:

$$w_j = \frac{(1-c_j)}{\sum_{j=1}^{n}(1-c_j)}$$

and the knowledge fusion function which is the proportion of the confidence of robots as :

$$\text{label}_j = \text{score} \times \left(c_1, c_2, \ldots, c_m\right)^T$$

Also, in the cloud, the loss function along with the optimal parameter are defined as :

$$y_i = \sum_{j=1}^{\text{num}} c_j \cdot w_j$$

$$L(y, h_\theta(x_i)) = \frac{1}{N}\sum_{i=1}^{N}(y_i - h_\theta(x_i))^2$$

$$\theta^* = \text{argmin}_\theta \frac{1}{N}\sum_{i=1}^{N} L(y_i \cdot h_\theta(x_i))$$

Transferring approaches One idea is to take the shared model as initial actor network while another idea is to use the shared model as a feature extractor. When the shared model is the initial actor network, it turns out that it is unstable even if good scores are observed at the beginning. However, the feature extractor increases the dimension of features and improves the effect stably.

One highlighted problem is when there is a structural difference between input layer of the shared network and private network. For example, most of neural networks in robot are convolutional neural networks (**CNN**) because **CNN** process images as input. The output of the **CNN** are described as features. But, if a non-image sensor such as a laser radar, is used, the Q-network is not considered as **CNN**. Then, the output of the entire network is used as additional features.
To summarize, each private neural network and the shared network generate actions which are called scores. They are stored and organized as a *score matrix* where each column corresponds to scores of a specific network. After normalizing each column, the knowledge fusion is applied which means the computation of *confidence* (c_j) and *labels* ($label_j$). In other words, all sample data labels are generated. Finally, a network for the generation $(k+1)$, is generated and fits the sample data as much as possible.

Results and discussion The major benefit of this approach is presented as a kind of unsupervised learning technique, where from unlabeled data, the algorithm is able to label sample data based on the confidence. Moreover, the transferring idea is designed as a feature extractor in order to improve the shared network. However, there are several issues to be pointed out :

- there is no information about how the shared network is supposed to have access to data
- to construct the *score matrix*, there is no reason that the dimension of samples is the same between each neural network, which means the number of scores between neural networks should vary depending of its environment (i.e. its local data at the time t)
- the formula of the output y_i which is based on the memory weight w_j is not completely justified to be used in the loss function as the target

iii.3 Federated REINFORCE client contribution evaluation

Intending to improve the aggregation, Zhao et al. (2021) propose a method to evaluate the contribution of federated participants on horizontal Federated Learning systems. The idea is to set up an evaluator which estimates the values of gradients sent by clients before aggregating client updates. `F-RCCE` algorithm is used as decision-maker and selects or discards clients gradients. It is able to perform task-specific evaluation that provides designated data distribution. The **RL** problem is presented as :

- *State space*: The feasible set of θ_G
- *Action space*: $S^t = \left[s_1^t, \ldots, s_n^t\right]$ where $s_i^t \in \{0, 1\}$ (0 : discarded; 1: selected)
- *Reward function*: $r(S^t)$

The reward function is related to the global model's performance on validation set.

$$r(S^t) = \frac{1}{m_v} \sum_{k=1}^{m_v} L_v(f_{\theta_G}(x_k^v), y_k^v) - \delta$$

where L_v is the loss function of global model on validation set D_v and δ is a baseline calculated by moving average of previous loss L_v with moving average window $T > 0$.

The global model parameters are updated as:

$$\theta_G^{t+1} \leftarrow \theta_G^t - \frac{\alpha_\theta}{\sum_{i=1}^N s_i^t} \sum_{i=1}^N s_i^t \nabla \theta_i^t$$

The probability of θ_i^t is introduced as $P(s_i^t = 0) = \omega_i^t$ Considering a parameterized stochastic policy $\pi_\theta(a|s)$, the probability of trajectory (with S_t the state at the time-step t and A_t the action at the

time-step t) is :

$$p(\tau|\pi) = \rho_0(S_0) \prod_{t=0}^{T} p(S_{t+1}|S_t, A_t)\pi(A_t, S_t)$$

with $\rho_0(S_0)$ the initial state distribution and τ the trajectory (sequel of state - action - reward). Then, the probability of S_t of the policy ϕ is defined as :

$$p(S^t|\phi) = \prod_{t=1}^{T} \left[{\omega_i^t}^{s_i^t} \cdot (1 - \omega_i^t)^{1-s_i^t} \right]$$

Using the log-derivative trick:

$$\nabla_\phi p(S^t|\phi) = p(S^t|\phi) \nabla_\phi \log p(S^t|\phi)$$

where in this specific case, the expression $\nabla_\phi \log p(S^t|\phi)$ is :

$$\nabla_\phi \log p(S^t|\phi) = \sum_{i=1}^{n} s_i^t \nabla_\phi \log \omega_i^t + (1 - s_i^t) \nabla_\phi \log(1 - \omega_i^t)$$

The learning objective is to maximize the expected cumulative reward :

$$\begin{aligned} J(\pi_\theta) &= \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)] \\ &= \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} R_t \right] \\ &= \sum_{t=0}^{T} \mathbb{E}_{\tau \sim \pi_\theta} [R_t] \end{aligned}$$

and :

$$\begin{aligned} \nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta} [R_t] &= \nabla_\theta \int_{\tau_t} R_t p(\tau_t|\theta) d\tau_t \\ &= \int_{\tau_t} R_t \nabla_\theta p(\tau_t|\theta) d\tau_t \\ &= \int_{\tau_t} R_t p(\tau_t|\theta) \nabla_\theta \log p(\tau_t|\theta) d\tau_t \\ &= \mathbb{E}_{\tau \sim \pi_\theta} [R_t \nabla_\theta \log p(\tau_t|\theta)] \end{aligned}$$

Then, the policy gradient can be given as :

$$\nabla_\phi J(\phi) = \mathbb{E}_{S^t \sim \pi_\phi(\theta^t, \cdot)} \nabla_\phi \log p(S^t|\phi) r(S^t)$$

Then the evaluator's model parameters ϕ can be optimized by gradient ascent method with learning α:

$$\phi^{t+1} \leftarrow \phi^t + \alpha_\phi \sum_{i=1}^{n} r(S^t) \nabla_\phi \log p(S^t|\phi)|_{\phi^t}$$

The algorithm F-RCCE offers promising results. Indeed, the authors of the paper have simulated **FL** systems with 50 to 500 clients. The total number of iterations of the model is around 1000 iterations. From 100 to 500 clients, the time of cost only increases by 6% (in comparison to the baseline leave-one-out (**LOO**) which increases linearly). The paper highlights that removing the

highest contribution slows down the model convergence. Moreover, removing the lowest contribution also puts forward negative impacts because they still have positive effects on the model. When the amount of clean data is large, thus the calculated gradient is relatively stable which is correlated with the client's contribution. However, the task model in the experiment is a logistic regression classifier which means the size of gradients is not large contrary to a deep learning algorithm. In other words, if the task model requires many parameters, then the size of gradients times the number of task models (i.e. the number for participants) could be a significant size of input for the evaluator's model. Moreover, the assumption of having a dataset in the server to compute the reward function, rises some questions. There is no information about the size or the distribution of the dataset on the server even if the dataset is supposed to be anonymized for keeping privacy.

III. Methodology

The chapter will cover explanations for the algorithms tested in experiments. Firstly, an introduction of fundamental knowledge of Reinforcement Learning will clarify how algorithms are built. Secondly, the first algorithm is designed to evaluate in a better way the contribution of participants for the aggregation. Finally, the second algorithm is designed to optimize different criteria by selecting in a better way the participants each round.

i. Basic knowledge of Reinforcement Learning

The environment of the agent is defined as a state S_t and the agent takes an action A_t to interact with it. Then the agent receives a reward R_t. Thus the trajectory τ[1] can be defined such as $\tau = (S_0, A_0, R_0, S_1, A_1, R_1, \dots)$. Also, the initial state follows a start-state distribution $S_t \sim \rho_0(\cdot)$ and the transition process is defined as $S_{t+1} \sim P(S_{t+1}|S_t, A_t)$ (for a stochastic process[2]).

The element of state transition of the tuple (S, A, P, R, γ) in a *Markov Decision Process* is :

$$P(S'|S, A) = P(S_{t+1} = S'|S_t = S, A_t = A)$$

The immediate reward is $R_t = R(S_t, A_t)$ where A_t is a finite set of actions. γ is the reward discount factor and $\gamma \in [0, 1]$

The discounted return is the weighted sum of rewards which gives more weights to the closer steps.

$$G_{t\in[0,T]} = R(\tau) = \sum_{t=0}^{T} \gamma^t R_t$$

The agent follows a policy π and the objective of the agent is to maximize the expected reward through this policy. The policy is defined as[3] :

$$\pi(A|S) = P(A_t = A|S_t = S), \forall t$$

The probability of the trajectory τ according to the policy π is:

$$P(\tau|\pi) = \rho_0(S_0) \prod_{t=0}^{T-1} P(S_{t+1}|S_t, A_t)\pi(A_t|S_t)$$

The value function[4] represents the expected return at the state S :

$$V(S) = \mathbb{E}_{\tau\sim\pi}\left[R(\tau)|S_0 = S\right]$$

The action-value function[5] represents the expected return at the state S and at the action A[6] :

$$Q^\pi(S, A) = \mathbb{E}_{\tau\sim\pi}\left[R(\tau)|S_0 = S, A_0 = A\right]$$

[1] $\tau \sim \pi$ are the trajectories τ sampled according to the policy π

[2] For a deterministic transition process, the next state S_{t+1} follows a deterministic function $S_{t+1} = f(S_t, A_t)$

[3] The policy can be also written as $\pi(A|S) = \underset{A\in\mathcal{A}}{\operatorname{argmax}}\, q^\pi(S, A)$

[4] The approximation value function can be written as $v(S) = \mathbb{E}_{S'\sim P(\cdot|S)}\left[r + \gamma v(S')\right]$

[5] $q^\pi(S, A) = \mathbb{E}_{S'\sim P(\cdot|S)}\left[R(S, A) + \gamma\, \mathbb{E}_{A'\sim\pi(\cdot|S')}\left[q^\pi(S', A')\right]\right]$ represents the approximation action-state function

[6] There is a relation between the optimal approximated value function and the optimal approximated action-state function $v^*(S) = \max_{A\in\mathcal{A}} q^*(S, A)$

The expected return of the policy π for any trajectories τ and any reward function R is defined as :

$$J(\pi) = \int_{\tau} P(\tau|\pi)R(\tau) = \mathbb{E}_{\tau\sim\pi}\left[R(\tau)\right]$$

The optimal policy is the policy which the agent aims to get to maximize the expected return and it is defined as:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\ J(\pi)$$

ii. Management of participation through a contribution evaluation

ii.1 Understanding of the algorithm

Based on the paper of (Zhao et al., 2021), the algorithm aims to improve the aggregation of local models from participants. One difficulty from the initial idea is that the size of gradients could be significantly large. For instance, for a convolutional neural network for the dataset MNIST with two convolutional layers and two linear layers, the total of parameters of gradients is almost 1.2 millions of parameters. Therefore, if the state is designed as the gradients of each local model $\nabla\theta_i$ where i is the index of the participant, then the state is composed of $n \times 1.2 \times 10^6$ components where n is the number of participants. In that case, the amount of information becomes tough to deal with for achieving to have a good policy and for computation with limit resources even if the agent is on the server.

In order to simplify this approach, the state is represented with a collection of accuracies computed after a training step. The agent outputs a vector of probability where the probability p_i represents the probability of the participant i to be chosen. Then, probabilities are sampled according to the Bernoulli law to produce a vector with the same size as the number of participants where **1** denotes to select the participant for aggregation and **0** denotes to discard the participant. The update of the global task model is done as the average of selected local task models of participants :

$$\theta_G^t \leftarrow \theta_G^t + \frac{\sum_i^k a_i\theta_i^t}{\sum_i^k a_i}$$

where θ_G^t is the parameters of the global task model at the time t, θ_i^t is the local task model of the participant i at the time t after training step, $a_i \in \{0, 1\}$ is the stochastic action given the probabilities of the output of the agent and k is the number of participants. The reward is designed to push the agent to find the best contribution configuration in order to improve the accuracy:

$$R_t = Acc_G^t - \delta_t = \frac{1}{\sum_i^k a_i}\sum_i^k a_i Acc_i(\theta_i^t) - \delta_t$$

where $Acc_i(\cdot)$ is the accuracy of the participant i and δ_t is the exponential moving average of previous accuracies Acc_G. Acc_G is an approximation of the accuracy of the global task model. The exponential moving average applies weighting factor which decrease exponentially. In other words, the recent accuracies have more impact than older accuracies.

The algorithm of Reinforcement Learning is the `REINFORCE` algorithm which will be explained theoretically below. This algorithm encourages the agent to take action that has greater cumulative reward weighting the gradient by the cumulative reward of each action.

ii.2 Gradient-Based Optimization

The agent is modeled as a neural network. The parameters of the neural network θ are updated according to :

$$\theta_{t+1} \leftarrow \theta_t + \Delta\theta \quad \text{with} \quad \Delta\theta = \alpha \nabla_\theta J(\pi_\theta)$$

By definition, the goal of the agent is to maximize the expected return which is formulated as:

$$J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[R(\tau)\right] = \mathbb{E}\left[\sum_{t=0}^{T} R_t\right] = \sum_{t=0}^{T} \mathbb{E}\left[R_t\right] \quad \text{because} \quad \mathbb{E}(aX+bY) = a\mathbb{E}(X) + b\mathbb{E}(Y)$$

The expectation of R_t is given by the integral:

$$E_{\tau_t\sim\pi_\theta}[R_t] = \int_{\tau_t} P(\tau_t|\theta) R_t \, d\tau_t$$

because R_t only depends on $\tau_t = (S_0, A_0, R_0, \ldots, S_t, A_t, R_t, S_{t+1})$. The derived expectation can be written as :

$$\begin{aligned}
\iff \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R_t] &= \nabla_\theta \int_{\tau_t} P(\tau_t|\theta) R_t \, d\tau_t \\
&= \int_{\tau_t} \nabla_\theta P(\tau_t|\theta) R_t \, d\tau_t \quad \text{because the integral does not depend on } \theta \\
&= \int_{\tau_t} P(\tau_t|\theta) \, \nabla_\theta \log(P(\tau|\theta)) \, R_t \, d\tau_t \quad \text{because} \quad \nabla_\theta \log(P(\tau|\theta)) = \frac{\nabla_\theta P(\tau|\theta)}{P(\tau|\theta)} \\
&= \mathbb{E}_{\tau\sim\pi_\theta}\left[\nabla_\theta \log(P(\tau|\theta)) R_t\right]
\end{aligned}$$

Since the probability of the trajectory τ according to the policy π is $P(\tau_t|\pi_\theta) = \rho_0 \prod_{t'=0}^{t} P(S_{t'+1}|S_{t'}, A_{t'})\pi(A_{t'}|S_{t'})$, thus the logarithmic expression of the probability is :

$$\log\big[P(\tau_t|\pi_\theta)\big] = \log\big[\rho_0(S_0)\big] + \sum_{t'=0}^{t} \Big(\log\big[P(S_{t'+1}|S_{t'}, A_{t'})\big] + \log\big[\pi_\theta(A_{t'}|S_{t'})\big]\Big)$$

The expression can be differentiated according to θ :

$$\nabla_\theta \log P(\tau_t|\pi_\theta) = \sum_{t'=0}^{t} \nabla_\theta \log \pi_\theta(A_{t'}|S_{t'})$$

Thus, the derived expected return can be expressed as:

$$\implies \nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} R_t \sum_{t'=0}^{t} \nabla_\theta \log \pi_\theta(A_{t'}|S_{t'})\right]$$

An equivalence of indices is explained in the following expression :

$$\begin{aligned}\sum_{t=0}^{T} A_t \sum_{t'=0}^{t} B_{t'} &= A_0\, B_0 \\ &\quad + A_1(B_0 + B_1) \\ &\quad + A_2(B_0 + B_1 + B_2) \\ &\quad + \dots \\ &\quad + A_T(B_0 + B_1 + B_2 + \dots + B_T) \\ \\ &= B_0(A_0 + A_1 + A_2 + \dots + A_T) \\ &\quad + B_1(A_1 + A_2 + \dots + A_T) \\ &\quad + B_2(A_2 + \dots + A_T) \\ &\quad + \dots \\ &\quad + B_T A_T \\ &= \sum_{t'=0}^{T} B_{t'} \sum_{t=t'}^{T} A_t\end{aligned}$$

The previous equivalence allows to write differently the indices of sums :

$$\boxed{\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t'=0}^{T} \nabla_\theta \log \pi_\theta(A_{t'}|S_{t'}) \sum_{t=t'}^{T} R_t\right]}$$

By introducing the discount factor γ, the expression can be completed :

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t'=0}^{T} \nabla_\theta \log \pi_\theta(A_{t'}|S_{t'}) \sum_{t=t'}^{T} \gamma^t R_t\right] \quad \left(\gamma^t = \gamma^{t-t'} \times \gamma^{t'}\right) \\ &= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t'=0}^{T} \nabla_\theta \log \pi_\theta(A_{t'}|S_{t'}) \left(\gamma^{t'} \sum_{t=t'}^{T} \gamma^{t-t'} R_t\right)\right]\end{aligned}$$

$\gamma^{t'}$ is removed to prevent an excessive update of recent rewards. A reference $b(S_{t'})$ is introduced for reducing the large variance when the gradient is estimated.

The complexity of the model and the random value of the reward increase exponentially when the size of the trajectory increases

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t'=0}^{T} \nabla_\theta \log \pi_\theta(A_{t'}|S_{t'}) \left(\sum_{t=t'}^{T} \gamma^{t-t'} R_t - b(S_{t'})\right)\right] \\ &= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t'=0}^{T} \nabla_\theta \log \pi_\theta(A_{t'}|S_{t'}) \sum_{t=t'}^{T} \gamma^{t-t'} R_t\right] - \sum_{t'=0}^{T} \mathbb{E}_{\tau\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(A_{t'}|S_{t'}) b(S_{t'})\right]\end{aligned}$$

The following statement $\mathbb{E}_{\tau\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(A_{t'}|S_{t'}) b(S_{t'})\right] = 0$ can be verified. Any normalized distribution follows:

$$\begin{aligned} & \int_x P_\theta(x)\, dx && = 1 \\ \iff \quad & \nabla_\theta \int_x P_\theta(x)\, dx && = \nabla_\theta 1 = 0 \\ \iff \quad & \int_x \nabla_\theta P(x)\, dx && = 0 \\ \iff \quad & \int_x P_\theta(x) \nabla_\theta \log P_\theta(x)\, dx && = 0 \\ \iff \quad & \mathbb{E}_{x\sim P(\cdot|\theta)}\left[\nabla_\theta log P_\theta(x)\right] && = 0 \end{aligned}$$

$$\implies \boxed{\mathbb{E}_{A_{t'}\sim\pi(\cdot|\theta)}\Big[\nabla_\theta \log \pi_\theta(A_{t'}|S_{t'})b(S_{t'})\Big] = 0}$$

Moreover, it is better to choose $b(S_t)$ with a variance $Var(b(S_t))$ close to 0 and strongly correlated to $\sum_{t=t'}^{T} \gamma^{t-t'} R_t$ because if X and Y are two random distributions then

$$Var(X - Y) = Var(X) + Var(Y) - 2\,cov(X, Y)$$

with $cov(X, Y)$ the covariance between X and Y. If Y has a variance close to 0 then :

$$\boxed{Var(X - Y) \approx Var(X) - 2cov(X, Y) \leq Var(X)}$$

The intuition of adding a bias allows to reduce the variance. The bias $b(S_t)$ is often the value function $V(S_t)$ for an actor-critic algorithm.

ii.3 Application of a binary policy

Having a binary policy means that the probability of the action A_t is defined as:

$$P(A_t|\theta) = \prod_{i=1}^{n} {p_i^t}^{a_i^t} \cdot (1 - p_i^t)^{1-a_i^t}$$

This distribution can be verified to be a normalized distribution. For convenience, S is the sum $\sum_{A_t\in\mathcal{A}} \prod_{i=1}^{n} {p_i^t}^{a_i^t} \cdot (1 - p_i^t)^{1-a_i^t}$ and $\mathcal{A}$ is the set of all possible actions, $\mathcal{A} = \big\{(a_1, \ldots, a_k) \mid \forall i \in [1, k], a_i \in \{0, 1\}\big\}$. Then, the expression can be organized differently to verify the result.

$$\begin{aligned}
S = {p_1^t}^0 \cdot (1 - p_1^t)^1 \times {p_2^t}^0 \cdot (1 - p_2^t)^1 \times \cdots \times {p_n^t}^0 \cdot (1 - p_n^t)^1 \\
+ {p_1^t}^1 \cdot (1 - p_1^t)^0 \times {p_2^t}^0 \cdot (1 - p_2^t)^1 \times \cdots \times {p_n^t}^0 \cdot (1 - p_n^t)^1 \\
+ {p_1^t}^0 \cdot (1 - p_1^t)^1 \times {p_2^t}^1 \cdot (1 - p_2^t)^0 \times \cdots \times {p_n^t}^0 \cdot (1 - p_n^t)^1 \\
+ {p_1^t}^1 \cdot (1 - p_1^t)^0 \times {p_2^t}^0 \cdot (1 - p_2^t)^1 \times \cdots \times {p_n^t}^0 \cdot (1 - p_n^t)^1 \\
+ \ldots \\
+ {p_1^t}^1 \cdot (1 - p_1^t)^0 \times {p_2^t}^0 \cdot (1 - p_2^t)^1 \times \cdots \times {p_n^t}^1 \cdot (1 - p_n^t)^0
\end{aligned}$$

$$\begin{aligned}
\iff S = {p_1^t}^0 \cdot (1 - p_1^t)^1 \times [\\
{p_2^t}^0 \cdot (1 - p_2^t)^1 \times \cdots \times {p_n^t}^0 \cdot (1 - p_n^t)^1 + \\
{p_2^t}^1 \cdot (1 - p_2^t)^0 \times \cdots \times {p_n^t}^0 \cdot (1 - p_n^t)^1 + \\
\cdots + \\
{p_2^t}^1 \cdot (1 - p_2^t)^0 \times \cdots \times {p_n^t}^1 \cdot (1 - p_n^t)^0 \\
] + {p_1^t}^1 \cdot (1 - p_1^t)^0 \times [\\
{p_2^t}^0 \cdot (1 - p_2^t)^1 \times \cdots \times {p_n^t}^0 \cdot (1 - p_n^t)^1 + \\
{p_2^t}^1 \cdot (1 - p_2^t)^0 \times \cdots \times {p_n^t}^0 \cdot (1 - p_n^t)^1 + \\
\cdots + \\
{p_2^t}^1 \cdot (1 - p_2^t)^0 \times \cdots \times {p_n^t}^1 \cdot (1 - p_n^t)^0 \\
]
\end{aligned}$$

$$
\begin{aligned}
\iff S = \Big[{p_1^t}^0 \cdot (1-p_1^t)^1 + {p_1^t}^1 \cdot (1-p_1^t)^0\Big] \times [\\
{p_2^t}^0 \cdot (1-p_2^t)^1 \times [\\
{p_3^t}^0 \cdot (1-p_3^t)^1 \times \cdots \times {p_n^t}^0 \cdot (1-p_n^t)^1 + \\
{p_3^t}^1 \cdot (1-p_3^t)^0 \times \cdots \times {p_n^t}^0 \cdot (1-p_n^t)^1 + \\
\cdots + \\
{p_3^t}^1 \cdot (1-p_3^t)^0 \times \cdots \times {p_n^t}^1 \cdot (1-p_n^t)^0 \\
] + {p_2^t}^1 \cdot (1-p_2^t)^0 \times [\\
{p_3^t}^0 \cdot (1-p_3^t)^1 \times \cdots \times {p_n^t}^0 \cdot (1-p_n^t)^1 + \\
{p_3^t}^1 \cdot (1-p_3^t)^0 \times \cdots \times {p_n^t}^0 \cdot (1-p_n^t)^1 + \\
\cdots + \\
{p_3^t}^1 \cdot (1-p_3^t)^0 \times \cdots \times {p_n^t}^1 \cdot (1-p_n^t)^0 \\
] \\
]
\end{aligned}
$$

$$
\begin{aligned}
\iff S = & \Big[{p_1^t}^0 \cdot (1-p_1^t)^1 + {p_1^t}^1 \cdot (1-p_1^t)^0\Big] \\
& \times \Big[{p_2^t}^0 \cdot (1-p_2^t)^1 + {p_2^t}^1 \cdot (1-p_2^t)^0\Big] \\
& \times \dots \\
& \times \Big[{p_n^t}^0 \cdot (1-p_n^t)^1 + {p_n^t}^1 \cdot (1-p_n^t)^0\Big] \\
& = 1 \times 1 \times \cdots \times 1 = 1
\end{aligned}
$$

Thus, the probability $P(A_t|\theta)$ follows a normalized distribution :

$$\boxed{\sum_{A_t \in \mathcal{A}} \prod_{i=1}^{n} {p_i^t}^{a_i^t} \cdot (1-p_i^t)^{1-a_i^t} = 1}$$

ii.4 Neural network architecture for the agent

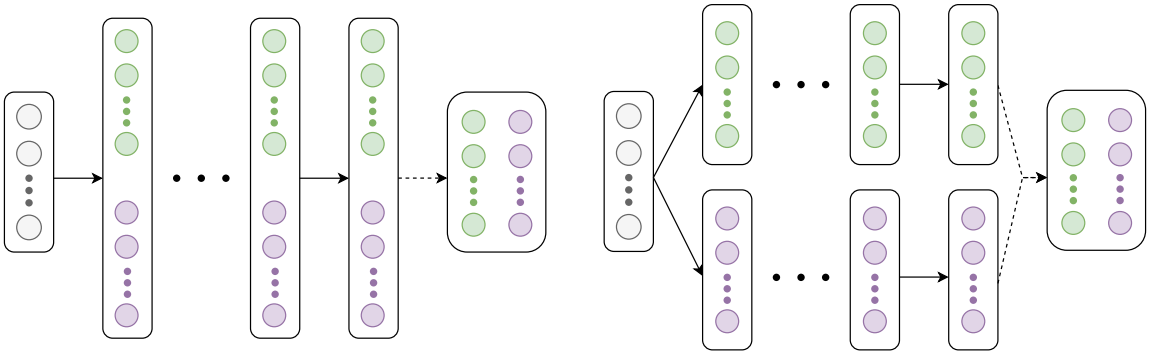
The output of the neural network of the agent is a vector of probabilities. In order to get probabilities from a neural network, usually the *softmax* function is applied after computing the output of the last layer. *Softmax* is defined as :

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp x_j}$$

Therefore the elements of the output belong to the range $[0, 1]$ and sum to 1.

Nevertheless, it means that when the probabilities are distributed in a homogeneous way, the probability for one participant to be selected depends strongly on other probabilities of participants. For example, with 100 participants, a uniform distribution would be that the probability to be selected is $\frac{1}{100}$. Consequently, next participants would have few chances to be selected for the next round and it probably leads to an action where all participants are discarded for the next round.

With the aim of preventing this difficulty, the output would not be a vector with k elements where k is the number of participants but a $k \times 2$ matrix where the first column corresponds to the probability of being selected while the second column corresponds to the probability of being discarded. Therefore, the *softmax* function is applied on rows and the sum of probabilities per row is 1. The architecture of the neural model must enable to have an output with $k \times 2$ values. Two ideas are suggested on the figure 6 :



(a) *Information are treated globally and hidden layers do not take account of the independence of probabilities of selection and probabilities of discard*

(b) *Information are treated separately and allows the neural network to focus on probabilities of selection and probabilities of discard*

Figure 6: *Different architectures of neural networks to get an output with two columns. Green color represents neurons for selection and violet color represents neurons for discard.*

The architecture of the left side of the figure 6 puts forward several difficulties to understand. The output of the neural network should be a $k \times 2$ matrix, however to get this specific shape on this specific architecture, the last output must be reshaped from $(2k) \times 1$ to $k \times 2$. There is no better reason for choosing the k first elements for the first column and the k last elements for the second column for the output shape than choosing randomly elements to fill the output matrix for having the correct shape. Also the size of hidden layers could be *doubled* to have enough parameters. Whereas the architecture of the right side on the figure 6 separates selection and discard which makes the information easier to deal with. This architecture is then used for the neural network of the agent.

ii.5 Moving batch

To train a neural network, batch could significantly speed up model training because batch sizes would allow to make parallel computations to a greater degree. However batch tends to generalize worse to test data. In the situation of limited experiments and time, speeding up model training is important but having a batch means collecting enough states, actions and rewards over the time. The moving batch involves to collect b elements by b elements which overlap between each other where b is the size of the batch and also denotes a window parameter. For instance, if $b = 3$, the agent learns only starting from the third round after collecting $(S_0, A_0, R_0, S_1, A_1, R_1, S_2, A_2, R_2)$. Then at the 4^{th}, the batch is composed of $(S_1, A_1, R_1, S_2, A_2, R_2, S_3, A_3, R_3)$. The figure 7 allows to visualize an example of the idea of moving batch.

ii.6 Details on the algorithm

To train the agent, for each experiment, also called *episode*, there are r rounds performed. Each round, participants download the global task model θ_G. Then they train their local model on their

$$\begin{array}{cccc} \text{Time} & t & t+1 & t+2 \\ \text{State} & S_t & S_{t+1} & S_{t+2} \\ \text{Action} & A_t & A_{t+1} & A_{t+2} \\ \text{Reward} & R_t & R_{t+1} & R_{t+2} \end{array} \longrightarrow \begin{array}{ccc} t+1 & t+2 & t+3 \\ S_{t+1} & S_{t+2} & S_{t+3} \\ A_{t+1} & A_{t+2} & A_{t+3} \\ R_{t+1} & R_{t+2} & R_{t+3} \end{array}$$

Figure 7: *Moving batch example with a size of 3 states*

local data set. Participants estimate the accuracy of their local model θ_i. After this step, the server downloads at its turn all local models and the accuracies of them. The agent, on the server, observes a state S_t, a vector of k accuracies $(Acc_0, \ldots, Acc_k)$ and takes an action A_t, a matrix of probabilities where the first column is the probability of being selected and the second column is the probability of being discarded. The action is then translated by a vector of k elements, 0 for discarded and 1 for selected generated by applying the Bernoulli law on the first column of the previous matrix. Depending of his decision, the agent receives a reward R_t. The moving batch is updated by removing the oldest transition and adding a new transition (S_t, A_t, R_t) Then the agent collects for training with moving batch. When the moving batch is full, the model of the agent can be trained by using the following formula :

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t'=0}^{T} \nabla_\theta \log \pi_\theta(A_{t'} | S_{t'}) \sum_{t=t'}^{T} \gamma^{t-t'} R_t \right]$$

The algorithm `EvaluatorFL` is summarized here:

Algorithm 1 `EvaluatorFL` There are k participants; B_l is the local minibatch size, E is the number of local epochs, and η is the learning rate of the task model, R is the number of rounds, B_a is the size of the moving batch, α_{lr} is the agent model and α_{exp} is the parameter for exponential moving average. The algorithm is described for **one episode**.

procedure SERVER WITH THE AGENT EXECUTES:
  Initialize θ_G^0 ▷ Global task model
  Initialize the Moving Batch $\mathcal{B}$
  Initialize θ_a ▷ Agent model
  Initialize $\delta = 0$
  for each round r from 1 of R **do**
    for each client $k \in S_t$ **in parallel do**
      Update local model $\theta_i^t \leftarrow \theta_G^t$ ▷ Participants download global task model
      $\theta_i^{t+1} \leftarrow \text{ClientUpdate}(\theta_i^t)$
      $Acc_i^{t+1} \leftarrow \text{Accuracy}(\theta_i^t)$
    The server collects local gradients θ_i^{t+1} and accuracies Acc_i^{t+1}
    $S_t \leftarrow (Acc_1^{t+1}, \ldots, Acc_k^{t+1})$ ▷ State
    $A_t \leftarrow \theta_a(S_t)$ ▷ Action
    $Sel^t \leftarrow \text{Bernoulli}(A_t^0)$ ▷ Selection using first column of probabilities
    $p \leftarrow \sum_{i=1}^{k} Sel_t^i$
    $\hat{R} \leftarrow \frac{1}{p}\left(\sum_{i=1}^{k} Sel_i^t \cdot Acc_i^{t+1}\right)$
    $R_t \leftarrow \hat{R} - \delta$
    $\delta \leftarrow \delta + (1 - \alpha_{\text{exp}})(\hat{R} - \delta)$
    Collects $\{S_t, A_t, R_t\}$ and update $\mathcal{B}$
    $\theta_G^{t+1} \leftarrow \frac{1}{p}\sum_{i=1}^{k} Sel_t^i \cdot \theta_i^{t+1}$
    if $\mathcal{B}$ is full **then**
      $\hat{A}_{t',j} \leftarrow \sum_{t=t'}^{T} \gamma^{t-t'} R_t$
      Compute the policy gradient :

$$J(\theta_a) \leftarrow \frac{1}{B_a} \sum_{j=1}^{B_a} \sum_{t'=0}^{T} \nabla_\theta \log \pi_{\theta_a}(A_{t',j}|S_{t',j}) \hat{A}_{t',j}$$

      Update $\theta_a^{t+1} \leftarrow \theta_a^t + \alpha_{\text{lr}} \cdot J(\theta_a)$

iii. Management of participants for the next round

iii.1 Understanding of the situation

The method for choosing the next participants for the algorithm `FederatedAveraging` involves to choose the next participants randomly by imposing a percentage of participants. For instance, if there are 100 workers, to insure the convergence of the task model, 10% of workers are selected randomly each round. Several questions are coming up :

- how to insure the fairness of participation to avoid that the same workers contribute every round ?
- is it possible to select the fastest workers to make the learning quicker ?
- how to take into account the distribution of data over all local data of workers ?

iii.2 Definition of the state

In order to solve this problem, Reinforcement Learning algorithms can be design to find the best way to optimize these criteria. In fact, the state of the environment can be modeled as a vector which contains for each worker : the previous participation, a criterion of performance and a value which represents the distribution of local data obtained through the assumption of *Federated Analytics*. For this study, in order to simplify the challenge, only the time is used to defined the criterion of performance. The state is then defined with the computation time T^c, the time for uploading the model to the server T^u and the time to download the model from the server T^d where :

$$T^c = v \times \frac{|D|}{B_s} \times E$$
$$T^u = \frac{P \times |u|}{B_u}$$
$$T^d = \frac{P \times |u|}{B_d}$$

with v the time spent to compute a batch (e.g. 0.5 s/batch), $|D|$ the size of local data, B_s the batch size, P the number of parameters, $|u|$ the size of a parameter unit (e.g. a float is 32 bits), B_u the bandwidth for uploading and B_d the bandwidth for downloading. The total time is the sum of previous times :

$$T^t = T^c + T^u + T^d$$

The state is normalized each round following a standard normalization:

$$X_{\text{normalized}} = \frac{X - m}{\sigma}$$

where m is the mean of X and σ is the standard deviation of X. Note the state is composed of 3 components for each worker which means there are 3 means and 3 standard deviations to compute and one normalization per component.

iii.3 Definition of the reward

The goal of the agent could be to minimize the variance of participation. For instance, the table 1 illustrates the evolution of the variance of participation which must be minimized by the agent to insure the fairness of participation.

Moreover, the agent could be used to minimize the learning time while speeding up the evolution of the accuracy. There are several ways to reward the agent : it could be by giving a positive reward

Time	Worker 1	Worker 2	Worker 3	Worker 4	Variance
t	4	7	9	3	7.583
$t+1$	5	7	10	3	8.917
$t+k$	10	10	10	10	0.000

Table 1: *Example of the evolution of the variance of participation with 4 workers*

when the accuracy is increased and the total of time is decreased between two consecutive rounds and by punishing it in the opposite case. Else, for instance, the reward could increase exponentially when the time would be more and more minimized while the accuracy would be more and more maximized. Finally, the distribution of data could be taken as a weight which influences the reward. In case of **non-IID**, an illustration of the intention, would be when a worker with 300 samples of the label A should have a different impact on the reward than a worker with 4000 samples of the same label A.

However, in this study, the reward is based on the criterion of performance, then the objective of the agent is to minimize the total time T^t for each round. The reward is established as the difference between the maximum of time of participants and the previous maximum time:

$$R_t = (-\max(T^a) - \nu)$$

where T^a is the set of all total times of participants of the current round and ν is the previous maximum time. Note the total times are normalized and to normalize the reward, it is better to take the mean of times than the sum of times (i.e. the total times).

iii.4 Choice of the algorithm and explanations

Different algorithms were explored[7] :

- **TRPO** algorithm (**T**rust **R**egion **P**olicy **O**ptimization) where the step size is properly handled in policy gradient based on the idea of trust region. In other words, the algorithm helps to manage the step size depending on the curvature of the reward landscape. The algorithm is complex and would take too much time to be implemented.
- **PPO** algorithm (**P**roximal **P**olicy **O**ptimization) is an improvement of the **TRPO** algorithm because **TRPO** is complicated and suffers of a computation burden in computing the natural gradient. In the same way, the algorithm still stays elaborate to be used for Federated Learning and was not retained.
- **AC** algorithm (**A**ctor **C**ritic) is a simple algorithm which is an improvement of the policy-based methods (for example, `REINFORCE` algorithm) where the state value is used as a baseline to reduce the variance of the estimated gradient in the vanilla policy gradient method. The main advantage of this algorithm is its simplicity, however, after hundreds of experiments the algorithm does not converge to an acceptable solution. Most of the time, the probabilities diminish until being close to zero and only one or two participants are selected for the next round.

Only the **DDPG** algorithm (**D**eep **D**eterministic **P**olicy **G**radient) was validated. The idea of the algorithm is to improve the **DQN** algorithm (**D**eep **Q** **N**etwork) to be a deterministic policy gradient. **DDPG** establishes a Q function (critic) and a policy function (actor) simultaneously. Temporal-difference methods are used to update the Q function (critic) which is the same with

[7] A small description of several algorithms can be found in Appendix ii

DQN. The action-value function $Q(s,a|\theta^Q)$, where θ^Q denotes for the critic network parameters, is learned using Bellman equation (see Bellman (1957)). The action-value function **given any policy** π is defined as :

$$\begin{aligned} Q^\pi(S,A) &= \mathbb{E}_{\tau\sim\pi}\left[R(\tau)|S_0=S,A_0=A\right] \\ &= \mathbb{E}_{A_t\sim\pi(\cdot|S_t)}\left[\sum_{t=0}^{\infty}\gamma^t R(S_t,A_t)|S_0=S,A_0=A\right] \end{aligned}$$

The Bellman equation **given any policy** π can be derived as follows:

$$\begin{aligned} Q^\pi(S,A) &= \mathbb{E}_{A\sim\pi(\cdot|S),S'\sim P(\cdot|S,A)}\left[R(\tau_{t:T})|S_t=S,A_t=A\right] \\ &= \mathbb{E}_{A\sim\pi(\cdot|S),S'\sim P(\cdot|S,A)}\left[R_t+\gamma R_{t+1}+\gamma^2 R_{t+2}+\cdots+\gamma^T R_T|S_t=S,A_t=A\right] \\ &= \mathbb{E}_{A\sim\pi(\cdot|S),S'\sim P(\cdot|S,A)}\left[R_t+\gamma(R_{t+1}+\gamma R_{t+2}+\cdots+\gamma^{T-1}R_T)|S_t=S,A_t=A\right] \\ &= \mathbb{E}_{S_{t+1}\sim P(\cdot|S_t,A_t)}\left[R_t+\gamma\mathbb{E}_{A\sim\pi(\cdot|S),S'\sim P(\cdot|s,a)}[R_{\tau_{t+1:T}}]|S_t=S\right] \\ &= \mathbb{E}_{S_{t+1}\sim P(\cdot|S_t,A_t)}\left[R_t+\gamma\mathbb{E}_{A_{t+1}\sim pi(\cdot|S_{t+1})}[Q^\pi(S_{t+1},A_{t+1})]|S_t=S\right] \\ &= \mathbb{E}_{S_{t+1}\sim P(\cdot|S_t,A_t)}\left[R(S,A)+\gamma\mathbb{E}_{A'\sim pi(\cdot|S')}[Q^\pi(S',A')]\right] \end{aligned}$$

where $S'=S_{t+1}$ and $A'=A_{t+1}$.

The action-value function $Q^\pi(S_t,A_t)$ in the `DDPG` algorithm **given any policy** π is then written as :

$$\boxed{Q^\pi(S_t,A_t)=\mathbb{E}\left[R(S_t,A_t)+\gamma Q^\pi(S_{t+1},\pi(S_{t+1}))\right]}$$

Then the Q value of the critic network θ^Q is computed and uses to minimize the loss function :

$$L=\frac{1}{N}\sum_i\left(Y_i-Q(S_i,A_i|\theta^Q)\right)^2 \quad \text{with} \quad Y_i=R_i+\gamma Q(S_{t+1},\pi(S_{t+1}|\theta^{\pi'})|\theta^{Q'})$$

The policy of the actor network is deterministic thus it will be noted as μ for a deterministic policy to make the difference with stochastic policy. The performance objective for the deterministic policy μ follows the same expected discounted reward definition than the stochastic policy:

$$\begin{aligned} J(\mu) &= \mathbb{E}_{S_t\sim\rho^\mu,A_t=\mu(S_t)}\left[\sum_{t=1}^{\infty}\gamma^{t-1}R(S_t,A_t)\right] \\ &= \int_{\mathcal{S}}\int_{\mathcal{S}}\sum_{t=1}^{\infty}\gamma^{t-1}\rho_0(S)P(S'|S,t,\mu)\cdot R(S',\mu(S'))\ dS\ dS' \\ &= \int_{\mathcal{S}}\rho^\mu(S)\cdot R(S,\mu(S))\ dS \quad \text{because} \quad \rho^\mu(S)=\int_{\mathcal{S}}\sum_{t=1}^{\infty}\gamma^{t-1}\rho_0(S)P(S'|S,t,\mu) \end{aligned}$$

where $P(S'|S,t,\mu)=P(S_{t+1}|S_t,A_t)P^\mu(A_t|S_t)$ is the transition probability times the probability of the action choice. The definition of the state function can be used :

$$\begin{aligned} V^\mu(S) &= \mathbb{E}\left[\sum_{t=1}^{\infty}\gamma^{t-1}R(S_t,A_t)|S_t=S;\mu\right] \\ &= \int_{\mathcal{S}}\sum_{t=1}^{\infty}\gamma^{t-1}P(S'|S,t,\mu)R(S',\mu(S'))dS' \end{aligned}$$

And the objective can be written as :

$$J(\mu) = \int_{\mathcal{S}} \rho_0 V^{\mu}(S) ds$$

With a deterministic policy, there is only a single action whereas with a stochastic policy, the Q-value is an expectation over the action distribution. Thus $V^{\mu}(S) = Q^{\mu}(S, \mu(S))$ and :

$$J(\mu) = \int_{\mathcal{S}} \rho_0 Q^{\mu}(S, \mu(S)) ds$$

There is a Bellman (1957) equation for the action-value function :

$$\begin{aligned}
Q^{\mu}(S, A) &= \mathbb{E}\left[R(\tau_{t:T})|S_t = S, A_t = \mu(S)\right] \\
&= \mathbb{E}\left[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots + \gamma^T R_T)|S_t = S, A_t = \mu(S)\right] \\
&= \mathbb{E}\left[R_t + \gamma(R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T)|S_t = S, A_t = \mu(S)\right] \\
&= \mathbb{E}\left[R_t + \gamma \mathbb{E}\left[R(\tau_{t+1:T})|A_t = \mu(S)\right]|S_t = S\right] \\
&= \mathbb{E}\left[R_t + \gamma \mathbb{E}\left[Q^{\mu}(S_{t+1}, A_t)|A_t = \mu(S))\right]|S_t = S\right] \\
&= \mathbb{E}\left[R_t + \gamma \mathbb{E}\left[Q^{\mu}(S', A_t)|A_t = \mu(S))\right]\right] \\
&= R_t + \gamma \mathbb{E}\left[Q^{\mu}(S', A_t)|A_t = \mu(S))\right] \\
&= R_t + \gamma \mathbb{E}\left[V(S')|A_t = \mu(S))\right]
\end{aligned}$$

It can be use in the policy gradient:

$$\begin{aligned}
\nabla_\theta J(\mu_\theta) &= \nabla_\theta \int_{\mathcal{S}} \rho_0(S) V^{\mu_\theta}(S) dS \\
&= \int_{\mathcal{S}} \rho_0(S) \cdot \nabla_\theta V^{\mu_\theta}(S) dS \quad \text{because the integral does not depend on } \theta
\end{aligned}$$

And :

$$\begin{aligned}
\nabla_\theta V^{\mu_\theta}(S) &= \nabla_\theta Q^{\mu_\theta}(S, \mu_\theta(S)) \\
&= \nabla_\theta \left[R(S, \mu_\theta(S) + \gamma \mathbb{E}\left[V^{\mu_\theta}(S')|A_t = \mu(S)\right]\right] \\
&= \nabla_\theta [R(S, \mu_\theta(S)] + \nabla_\theta \left[\int_{\mathcal{S}} \gamma P(S'|S, \mu_\theta(S)) \cdot V^{\mu_\theta}(S') dS'\right] \\
&= \nabla_\theta A + \nabla_\theta B
\end{aligned}$$

So, the two expressions $\nabla_\theta A$ and $\nabla_\theta B$ can be developed:

$$\begin{aligned}
\nabla_\theta A &= \nabla_\theta [R(S, \mu_\theta)] \\
&= \nabla_\theta \mu_\theta(S) \times \nabla_A R(S, A)_{|A=\mu_\theta(S)}
\end{aligned}$$

And:

$$\begin{aligned}
\nabla_\theta B &= \nabla_\theta \left[\int_{\mathcal{S}} \gamma P(S'|S, \mu_\theta(S)) \times V^{\mu_\theta}(S') dS'\right] \\
&= \int_{\mathcal{S}} \gamma \nabla_\theta \left[P(S'|S, \mu_\theta(S)) \times V^{\mu_\theta}(S')\right] dS'
\end{aligned}$$

By posing $u(\theta) = P(S'|S\mu_\theta(S))$ and $v(\theta) = V^{\mu_\theta}(S')$, the formula $\nabla_\theta[u(\theta) \times v(\theta)] = \nabla_\theta u(\theta) \times v(\theta) + u(\theta) \times \nabla_\theta v(\theta)$ can be used to develop the expression:

$$\nabla_\theta B = \int_{\mathcal{S}} \nabla_\theta \mu_\theta(S) \nabla_A P(S'|S, \mu_\theta(S)) \times V^{\mu_\theta}(S') dS' + \int_{\mathcal{S}} \gamma P(S'|S, \mu_\theta(S)) \times \nabla_\theta V^{\mu_\theta}(S') dS'$$

Thus, the expression $\nabla_\theta V^{\mu_\theta}(S)$ is simplified as:

$$\begin{aligned}\nabla_\theta V^{\mu_\theta}(S) &= \mu_\theta(S) \left[\nabla_\theta R(S, \mu_\theta(S)) + \int_{\mathcal{S}} \gamma \nabla_A P(S'|S, \mu_\theta(S)) \times V^{\mu_\theta}(S') dS' \right] + \int_{\mathcal{S}} \gamma P(S'|S, \mu_\theta(S)) \times \nabla_\theta V^{\mu_\theta}(S') dS' \\ &= \nabla_\theta \mu_\theta(S) \nabla_A Q^{\mu_\theta}(S, \mu_\theta(S)) + \int_{\mathcal{S}} \gamma P(S'|S, \mu_\theta(S)) \times \nabla_\theta V^{\mu_\theta}(S') dS'\end{aligned}$$

Fubini's theorem *Suppose A and B are complete measures spaces. Suppose $f(x, y)$ is $A \times B$ measurable. If*

$$\int_{A \times B} |f(x, y)| d(x, y) < \infty$$

where the integral is taken with respect to a product measure on the space over $A \times B$, then

$$\int_A \left(\int_B f(x, y) dy \right) dx = \int_B \left(\int_A f(x, y) dx \right) dy = \int_{A \times B} f(x, y) d(x, y)$$

the first two integrals being iterated integrals with respect to two measures, respectively, and the third being an integral with respect to a product of these two measures.

The last part of the expression can be iterated using the relation between $\nabla_\theta V^{\mu_\theta}(S)$ and $\nabla_A Q^{\mu_\theta}(S, \mu_\theta(S))$, and simplified by using the Fubini's theorem :

$$\begin{aligned}&\int_{\mathcal{S}} \gamma P(S'|S, \mu_\theta(S)) \times \nabla_\theta V^{\mu_\theta}(S') dS' \\ &= \int_{\mathcal{S}} \gamma P(S'|S, \mu_\theta(S)) \left[\nabla_\theta \mu_\theta(S') \nabla_A Q^{\mu_\theta}(S'|\mu_\theta(S')) + \int_{\mathcal{S}} \gamma P(S''|S', \mu_\theta(S')) \nabla_\theta V^{\mu_\theta}(S'') dS'' \right] dS' \\ &= \int_{\mathcal{S}} \gamma P(S'|S, \mu_\theta(S)) \nabla_\theta \mu_\theta(S') \nabla_A Q^{\mu_\theta}(S'|\mu_\theta(S')) dS' + \int_{\mathcal{S}} \gamma^2 P(S''|S, \mu_\theta(S)) \nabla_\theta V^{\mu_\theta}(S'') dS''\end{aligned}$$

Thus, by iterating infinitely, the expression $\nabla_\theta V^{\mu_\theta}(S)$ can be written as:

$$\nabla_\theta V^{\mu_\theta}(S) = \int_{\mathcal{S}} \sum_{t=0}^{\infty} \gamma^t P(S \to S', t, \mu_\theta(S)) \times \nabla_\theta \mu_\theta(S') \times \nabla_A Q^{\mu_\theta}(S', A) dS'$$

where $P(S \to S', t, \mu_\theta(S))$ is the probability of transition t times[8]. Then the expression can be used in the policy gradient:

$$\begin{aligned}\nabla_\theta J(\mu_\theta) &= \int_{\mathcal{S}} \rho_0(S) \cdot \nabla_\theta V^{\mu_\theta}(S) dS \\ &= \int_{\mathcal{S}} \int_{\mathcal{S}} \sum_{t=0}^{\infty} \gamma^t \rho_0(S) P(S \to S', t, \mu_\theta(S)) \nabla_\theta \mu_\theta(S') \times \nabla_A Q^{\mu_\theta}(S', A) \; dS' \; dS\end{aligned}$$

[8]Note $P(S \to S', 0, \mu_\theta(S) = 1$ for $S' = S$ and 0 when $S' \neq S$.

As $\rho^{\mu}(S) = \int_{\mathcal{S}} \sum_{t=0}^{\infty} \gamma^{t-1} \rho_0(S) P(S'|S,t,\mu) dS$, then :

$$\boxed{\nabla_\theta J(\mu_\theta) = \int_{\mathcal{S}} \rho^{\mu_\theta}(S)\ \nabla_\theta \mu_\theta(S)\ \nabla_A Q^{\mu_\theta}(S,A) dS}$$

For the algorithm, the policy function is updated by applying the chain rule to the expected return from the start distribution J. Learning from mini-batches, the loss of the actor network can be computed as:

$$\nabla_{\theta^\pi} J \approx \sum_i \nabla_A Q(S_i, \pi(S_i, |\theta^\pi)|\theta^Q) \nabla_{\theta_\pi} \pi(S_i|\theta^\pi)$$

where θ^π denotes the actor network parameters.

The network parameters are updated by exponentially smoothing rather than directly copying the parameters:

$$\theta^{Q'} \leftarrow \rho\theta^Q + (1-\rho)\theta^{Q'}$$
$$\theta^{\pi'} \leftarrow \rho\theta^\pi + (1-\rho)\theta^{\pi'}$$

where ρ represents the degree of weighting decrease, a constant smoothing factor between 0 and 1.

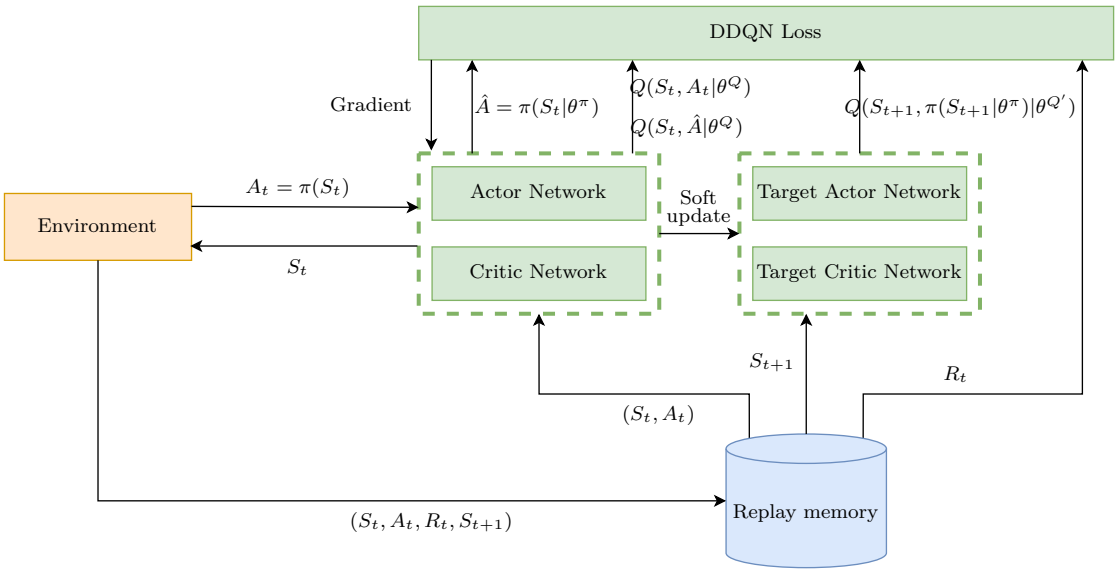


Figure 8: *DDPG algorithm with replay memory where $\pi(\cdot|\theta^\pi)$ denotes the policy function (actor) and $Q(\cdot|\theta^Q)$ denotes the Q function (critic)*

The figure 8 illustrates the DDPG algorithm and introduces the concept of *replay memory*. The replay memory stores the experiences in a large replay memory. Then, for each iteration of training, it selects randomly a sample of this memory. It allows to reduce the correlation between experiences in a training batch and it helps greatly the training along.

iii.5 Finalization of the algorithm

The algorithm SchedulerFL summarizes the DDPG algorithm. At the first round, the participants are chosen randomly given a percentage. The server sends the global model to all participants. Each participant trains their local model on local data. Every participants send back their local model and their respective times. For worker which did not participate the first round, their time are generated from the mean of each collected time (T^c, T^u, T^d). Thus for 100 workers, if only 10% have been selected, the others would have the mean of each collected time to avoid zero values. Note the first round helps to generate the first state and the time of non-participants are replaced by the mean of collected times only this round. At the end of the first round, the server aggregates the models of participants using the FedAvg algorithm.

On the next rounds, the policy network takes an action A_t depending on the current state S_t. The server sends the global model to participants and they send back its local trained model and times. Then, the server collects the new times and replaces them to update the state to make the next state S_{t+1}. The agent receives a reward R_t based on the state and its action. A replay memory stores the experience (S_t, A_t, R_t, S_{t+1}). At the end, the critic network and the actor network are trained on experiences randomly selected from the replay memory. All information are summarized in the algorithm SchedulerFL.

Algorithm 2 `SchedulerFL` There are k workers, p percentage of participants the first round, E experiences, R rounds, γ is the discount factor for reward

procedure SERVER WITH THE AGENT EXECUTES:
  Initialize θ^{π} ▷ Actor network
  Initialize $\theta^{\pi'}$ ▷ Target Actor network
  Initialize θ^{Q} ▷ Critic network
  Initialize $\theta^{Q'}$ ▷ Target Critic network
  Initialize a replay memory $\mathcal{M}$
  for each experience e from 1 of E **do**
    Initialize θ_G^0 ▷ Global task model
    Initialize S_0 as a full $3 \times k$ zero matrix ▷ State
    Initialize $\nu = 0$ ▷ Previous maximum time for reward
    Select randomly $p \cdot k$ participants
    for each participant **in parallel do**
      $\theta_i^0 \leftarrow \theta_G^0$ ▷ Participants download global task model
      $\theta_i^1 \leftarrow \text{ClientUpdate}(\theta_i^0)$ ▷ Local training
      Server collects the local model θ_i^1
      Compute the times T^c, T^u and T^d
      Update S_0 to S_1 by replacing with new times
    Complete S_1 by replacing zeros by the mean of participants times
    Aggregate local models with `FedAvg` algorithm to get θ_G^1
    for each round r from 2 of R **do**
      Normalize S_t to get $\hat{S}_t$ ▷ Using standard normalization
      Selection action $A_t = \pi(\hat{S}_t|\theta^{\pi})$
      for each participant according to A_t **in parallel do**
        $\theta_i^t \leftarrow \theta_G^t$ ▷ Participants download global task model
        $\theta_i^{t+1} \leftarrow \text{ClientUpdate}(\theta_i^t)$ ▷ Local training
        Server collects the local model θ_i^1
        Compute the times T^c, T^u and T^d
        Update S_{t+1} to S_t by replacing with new times
      Aggregate local models with `FedAvg` algorithm
      Compute the maximum time t_m and compute R_t
      $\nu \leftarrow t_m$
      Store the experience (S_t, A_t, R_t, S_{t+1}) in $\mathcal{M}$
      Set $Y_i = R_i + \gamma Q(S_{t+1}, \pi(S_{t+1}|\theta^{\pi'})|\theta^{Q'})$
      Update critic by minimizing the loss:

$$L = \frac{1}{N} \sum_i \left(Y_i - Q(S_i, A_i|\theta_Q)\right)^2$$

      Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^{\pi}} J \approx \frac{1}{N} \sum_i \nabla_A Q(S_i, \pi(S_i, |\theta^{\pi})|\theta^{Q}) \nabla_{\theta_{\pi}} \pi(S_i|\theta^{\pi})$$

      Update the target networks
      $\theta^{Q'} \leftarrow \rho\theta^{Q} + (1-\rho)\theta^{Q'}$
      $\theta^{\pi'} \leftarrow \rho\theta^{\pi} + (1-\rho)\theta^{\pi'}$

IV. Experiments

For the experiments, the distribution of data between clients is crucial to simulate **IID** cases and **non-IID** cases. In the first section, the generation of data will be covered to illustrate the different distributions of data. Then the others sections aim to test algorithms through experiments and analyse their results. The whole code for the experiments is available in the github repository `https://github.com/bourbonut/reinforcedFL`.

i. Generation of data

This section aims to cover how data are distributed in each device of clients where the assumption of *having an identically and independent distribution when all data are gathered* is applied. To achieve this assumption, the working data (e.g. **MNIST dataset** or **FashionMNIST dataset**) is initialized and has to be **IID**. Then, there are several parameters which are manageable :

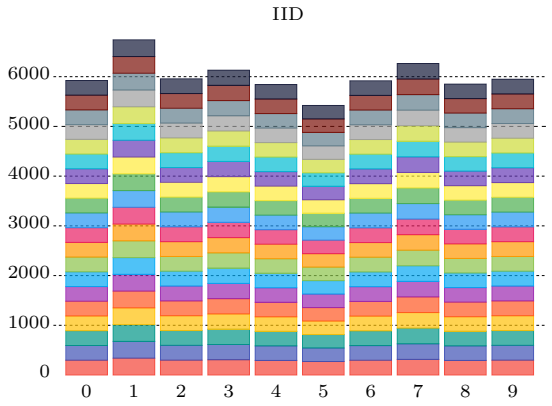
- the distribution based on the volume which means in case of **IID**, all clients have the same amount of data for each label and in case of **non-IID**, the volume of data changes from one client to another for a specific label. For instance, with two clients A and B, if both have the label l then in **IID** case, each client has half of the amount of data for label l while in **non-IID** case, the proportion owning by clients is randomly chosen, which creates an unbalanced configuration (e.g. 36% of data on client A and 64% of data on client B)
- the distribution based on the label which implies in case of **IID**, all clients have all labels and in case of **non-IID**, a parameter m allows to specify the minimum number of labels on clients and labels are randomly chosen. Another parameter b indicates if each client has at least m labels or less. For example, if there are 3 clients and 5 labels l_1, l_2, l_3, l_4, l_5, then on **IID** situation, all clients have the 5 labels, whereas in **non-IID** labels, if $m = 3$ and $b = False$, then one client has 3 labels (e.g. l_2, l_3, l_5), one client has 2 labels (e.g. l_1, l_4) and other clients shared one label (e.g. l_5). In the case when $b = True$, then all clients have at least 3 labels.

The figure 9 illustrates visually some distributions. Each color corresponds to a client and then it reflects the amount of data on each label for a specific client. The distribution of volume and the distribution of labels can be mixed. For instance, the volume can be **non-IID** while the labels are **non-IID** too. When the distribution of labels is non *identically and independent* and *balanced*, depending on the number of clients and the number of labels, the volume can not always be **IID**. On the figure, it is noticeable that the label 3 has a total of four clients where two clients which have more data than the others. Due to choosing $m = 3$ (the minimum number of labels on each client), several labels become unbalanced in order to meet the expectations of parameters.

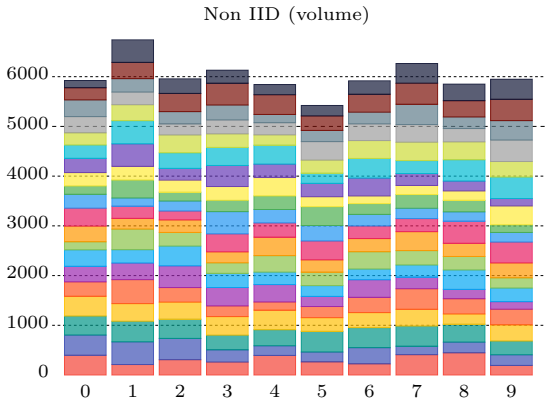
When data are generated, two types of data are given per client : the data for training and the data for testing. Necessary, taking the whole set of data, data are split to tend to the ratio 80% for training - 20% for testing. For well-known datasets as MNIST, the default configuration is kept (for MNIST, 60000 samples for training and 10000 samples of testing). Same labels and same volume of data are used between training data and testing data. In other words, for example, if a model on the client's device learns on labels 2, 4 and 7, then, the model is tested also on labels 2, 4, 7. As well, speaking of data for training, if the client's device holds 5% of label 2, 10% of label 4 and 6% of label 7, then data for testing is distributed in the same way than data for training.

ii. Baseline - Federated Averaging

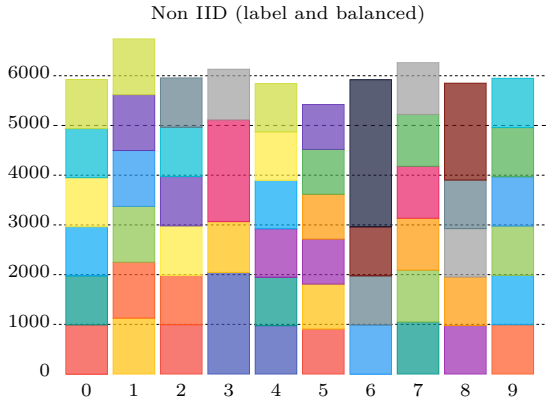
The algorithm `FedAvg` is called Federated Averaging from McMahan et al. (2017) and it is simple to implement and to see the impact of different non identically and independent distributions on the



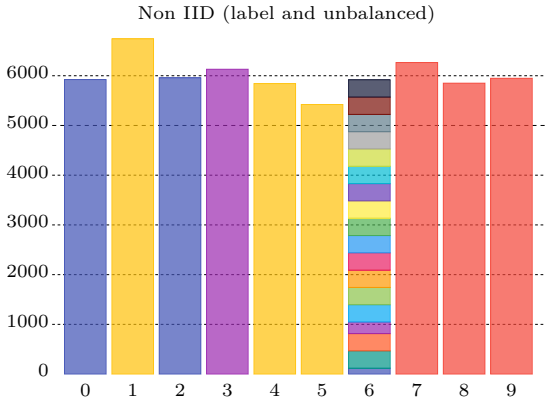
(a) *Configuration where labels and volume follow are* ***IID***



(b) *Configuration where labels are* ***IID*** *and volume is* ***non-IID***



(c) *Configuration where labels are* ***non-IID*** *(balanced* $b = True$ *and* $m = 3$*) and volume is* ***IID***



(d) *Configuration where labels are* ***non-IID*** *(unbalanced* $b = False$ *and* $m = 3$*) and volume is* ***IID***

Figure 9: *Impact on the generation of data according to the distribution of labels and the distribution of the volume with 20 clients on MNIST dataset (10 labels and 60000 samples)*

evolution of the global model. `FedAvg` is described as a weighted average of local models on clients' devices. The weights are computed based on the proportion of data for a device compared to the whole size of data. For instance, if a device holds 1500 samples and the size of the whole dataset, all data of devices taken together, is 60000, then the weight is 0.025. The total amount of data on local devices is supposed known thanks to Federated Analytics techniques, which are a collaborative data science without data collection. Then, knowing all amounts of data on local devices allows to know the whole size of data if they were gathered on a server. Thus, `FedAvg` can be computed while keeping privacy of clients.

Algorithm 3 `FederatedAveraging` The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

procedure SERVER EXECUTES:
  initialize w_0
  for each round $t = 1, 2, \dots$ **do**
    $m \leftarrow \max(C \cdot K, 1)$
    $S_t \leftarrow$ (random set of m clients)
    for each client $k \in S_t$ **in parallel do**
      $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
    $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$

procedure CLIENTUPDATE(k, w): ▷ Run on client k
  $\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)
  for each local epoch i from 1 of E **do**
    for batch $b \in \mathcal{B}$ **do**
      $w \leftarrow w - \eta \nabla L(w; b)$ ▷ $L(w; b)$ represents the loss function
  return w to server

Different experiments are presented on figure 10, figure 11, figure 12 and figure 13 which are realized on FashionMNIST dataset (60000 samples with balanced labels) working with 20 workers[9] with 10 rounds and 3 epochs, with different distributions of data. Each figure is composed of the distribution of data for training through all workers on left side and the evolution of the weighted average accuracy on the right side. Note the distribution of data for testing is similar to the distribution of data for training and therefore, each worker has same volume of labels and same labels for training and testing. In case of 10 labels, a model which learns on label 1, 5 and 6 for instance, will never be tested on label 0, 2, 3, 4, 7, 8 and 9. Each round, every worker computes the accuracy of the model established on their own local dataset. Then, the server collects all local accuracies and computes the weighted average accuracy where the evolution is illustrated the right side of each figure. Note the first *round 0* corresponds to the state where workers' models are randomly initialized. When all labels appear on each dataset of workers (case figure 10 and 11) then the method converges considerably quickly. However, when labels are non independent and identically distributed, the convergence becomes slower on the balanced (see figure 12) and is penalized when only few workers have all data of some specific labels while others have the remainder part of one label (see figure 13). Note in all figures 10, 11, 12 and 13, all workers are participants every round.

iii. Results on contribution evaluation algorithm

Due to train an agent through `REINFORCE` algorithm, the main difficulty is to know how many steps is needed for the model to converge depending on the number of participants. Two sizes are simulated : 20 participants and 100 participants. For both sizes, the number of episodes is 20 and there are 10 rounds per episode. All workers are participants which means there is no random choice of future participants. The model of agent is 4 parallel linear layers where each of them is composed of 128 neurons. The task model is convolutional neural network with 2 convolutional layers and 2 linear layers working on `MNIST` dataset. Data are generated with **non-IID** labels including 3 labels

[9] A worker is a client device.

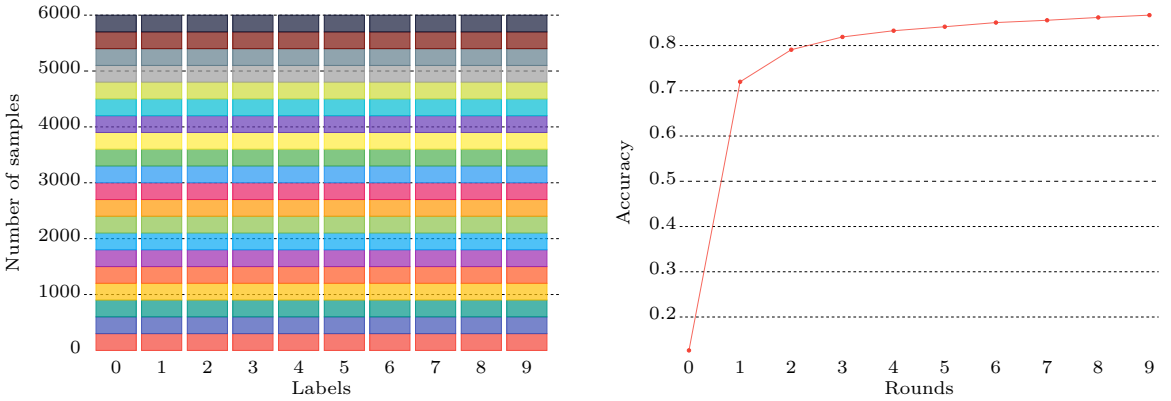


Figure 10: *Convergence of FashionMNIST model with Federated Averaging using 20 workers on **IID** configuration*

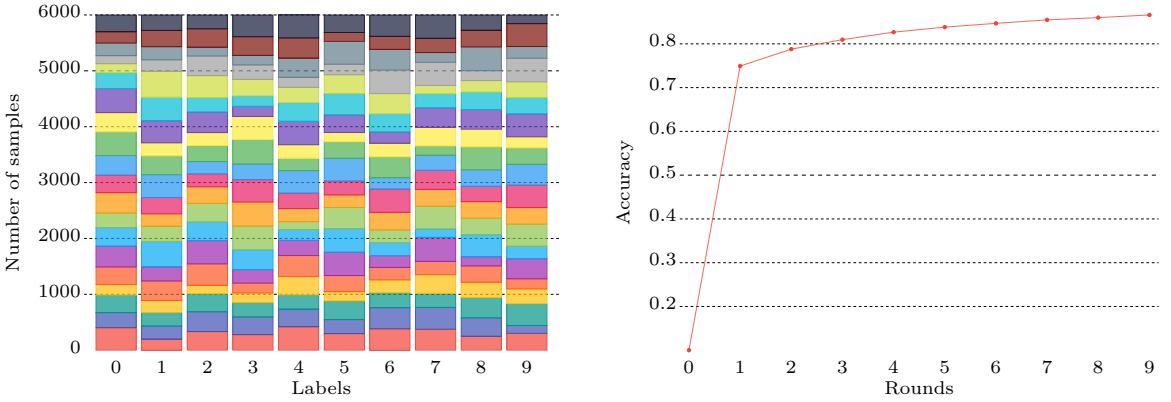


Figure 11: *Convergence of FashionMNIST model with Federated Averaging using 20 workers with **non-IID** volume*

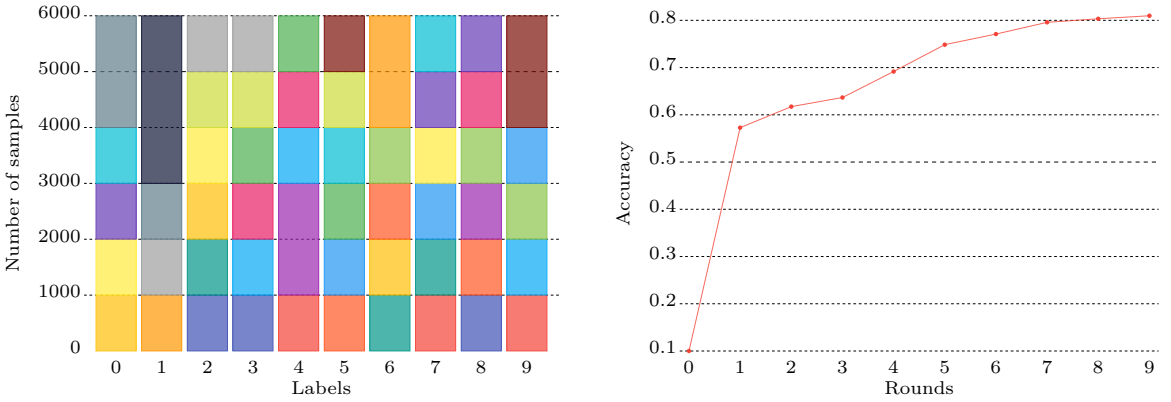


Figure 12: *Convergence of FashionMNIST model with Federated Averaging using 20 workers with **non-IID** labels (balanced and 3 labels minimum)*

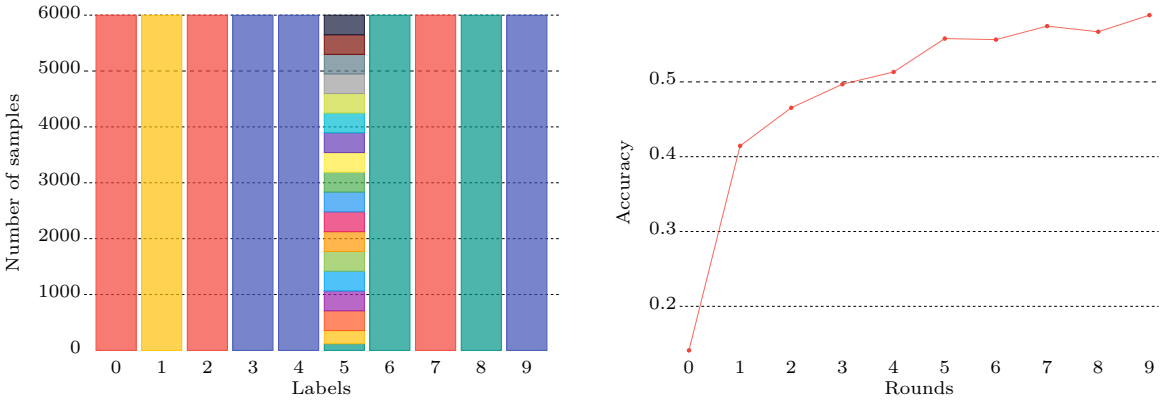


Figure 13: *Convergence of FashionMNIST model with Federated Averaging using 20 workers with* ***non-IID*** *labels (unbalanced and 3 labels minimum)*

at least per worker and with **non-IID** volume. Between each episode, workers keep the same dataset but only the task model is reinitialized globally and locally. There are two approaches realized.

iii.1 First approach

The first idea is to base the state and the reward on what was explained in section ii.6 for the algorithm `EvaluatorFL` where the state and the reward are defined as :

$$S_t = (Acc_1, \ldots, Acc_k) \quad \text{and} \quad R_t = \frac{1}{\sum_{i=1}^{k} Sel_i} \left(\sum_{i=1}^{k} Sel_i \cdot Acc_i \right) - \delta$$

where Sel_i is 1 for selected and 0 for discarded, Acc_i is the accuracy of the participant i and k is the number of participants.

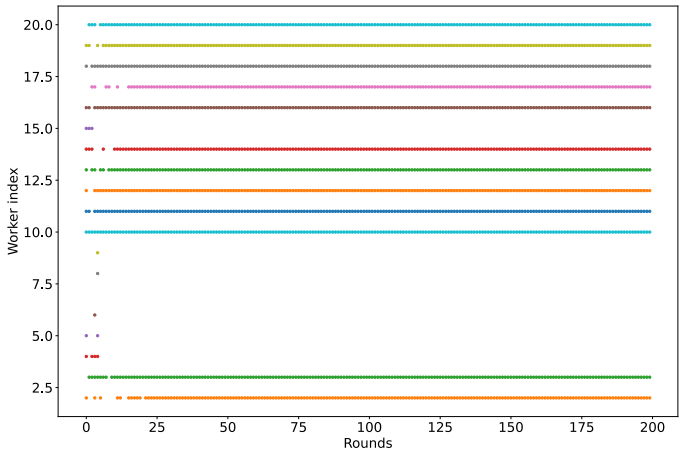


Figure 14: *Selection represented for 20 episodes with 10 rounds per episode with 20 participants*

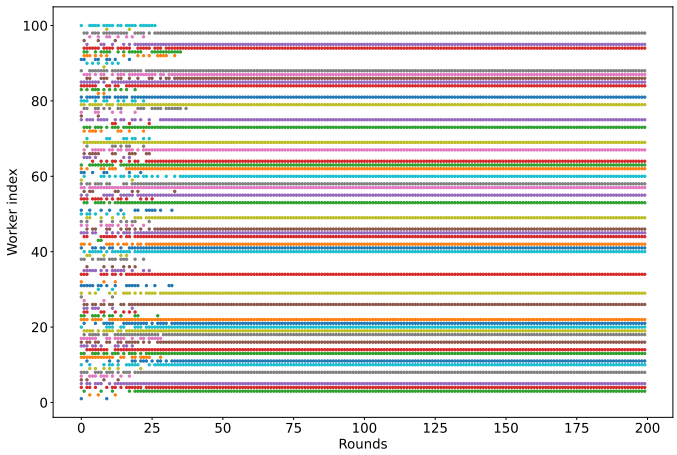
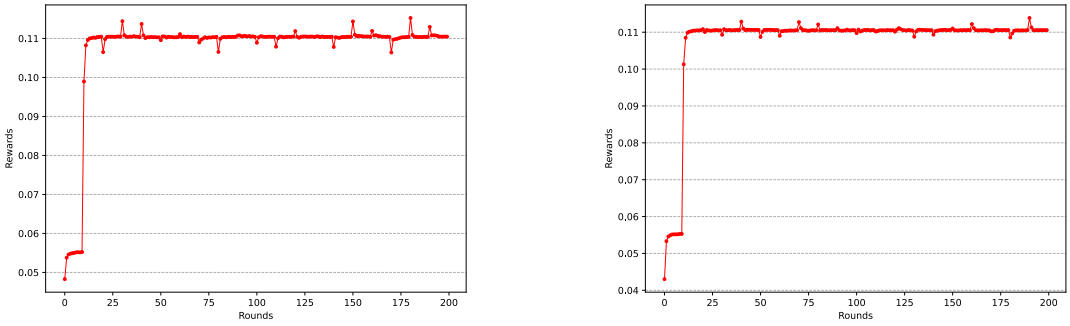


Figure 15: *Selection represented for 20 episodes with 10 rounds per episode with 100 participants*

The figure 14 and 15 illustrates the selection of contributors for aggregation. Each color is associated to a participant and when there is a colored dot, it means the participant is selected else the participant is discarded for the aggregation. On figure 14, with only one episode, the algorithm converges quickly and selects the same participants over the time. On figure 15, the number of participants is increased to 100 workers and the algorithm converges at the 18^{th} episode. A separation is significantly distinguishable between homogeneous selection similar to a random behavior and heterogeneous selection when the algorithm finds a strategy for optimizing contribution.

The following figure 16 highlights the evolution of the rewards which indicates the development of the reinforcement learning system.



(a) *Evolution of the rewards given rounds with 20 participants*

(b) *Evolution of the rewards given rounds with 100 participants*

Figure 16: *Evolution of the rewards given rounds*

The figure 16a puts forward the same behavior with rewards as the figure 14 on selection. After one episode, the evolution of the rewards reaches a limit and stabilizes around it. On the figure 16b, 100 participants impact the shape of the curve. The system increases slower to reach a limit and then oscillates. It underlines that the evolution of the reward should not be a single indicator of

convergence of the algorithm.

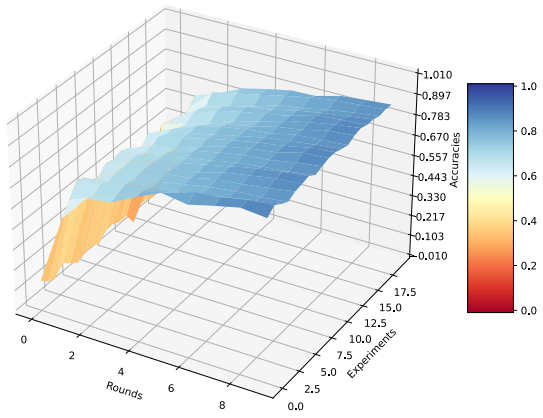


Figure 17: *Evolution of the accuracies given episodes and rounds with 20 participants*

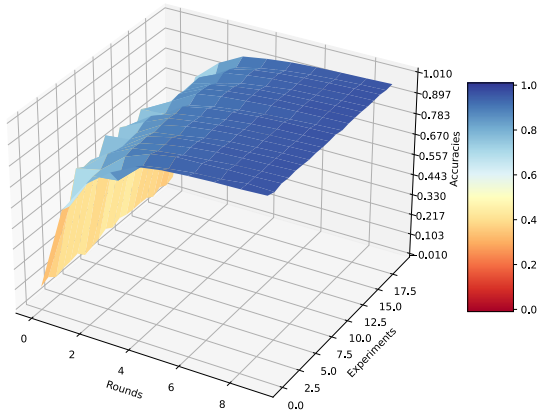


Figure 18: *Evolution of the accuracies given episodes and rounds with 100 participants*

Figure 17 and figure 18 illustrate the evolution of the accuracy per episode and per round. The evolution of the accuracy should rise faster over experiments. However in both figures, the trend remains the same and the progression of the color does not tend to change. The gradient of colors highlights that the progression of figure 18 with 100 workers based on rounds is slower than the progression of the figure 17 with 20 workers.

Even if the algorithm converges and find a group of participants which should contribute in a better way to optimize the aggregation, results point out that the contribution evaluation based on the accuracy does not improve the speed of convergence as expected.

iii.2 Second approach

Another approach is to use a state based on the difference between the accuracy of the trained local task model and the accuracy of the global task model before being trained. The reward is then computed by the difference between the weighted average of the accuracy of global task model after aggregation and the exponential moving average of previous accuracies.

$$S_t = \left(\left[Acc_1\left(\theta_1^{t+1}\right) - Acc_1\left(\theta_G^t\right)\right], \ldots, \left[Acc_k\left(\theta_k^{t+1}\right) - Acc_k\left(\theta_G^t\right)\right]\right)$$

$$\text{and} \quad R_t = \frac{1}{\sum_{i=1}^{k} d_i} \sum_{i=1}^{k} d_i \cdot Acc_i(\theta_G^{t+1}) - \delta$$

where $Acc_i(\cdot)$ is the accuracy computed using data of the worker i, θ_i^{t+1} is the local model of the worker i after a training of the global model θ_G^t, d_i is the size of the local worker data and δ is the exponential moving average of previous accuracies.

On figure 19 and figure 20, the selection of participants for the aggregation puts forward that the algorithm does not converge. In fact, with 20 workers or with 100 workers, both figures 19 and figure 20 are filled of dots which are not discernible because almost all gradients are used for the aggregation. However, the evolution of the rewards on figure 21 has almost the same shape as the reward on the first approach except some small pics. If the network would have converged, thus the reward would be smooth. But in this case, the reward oscillates, which is a type of divergence, and explains why the agent does not select a subset of gradients after the training.

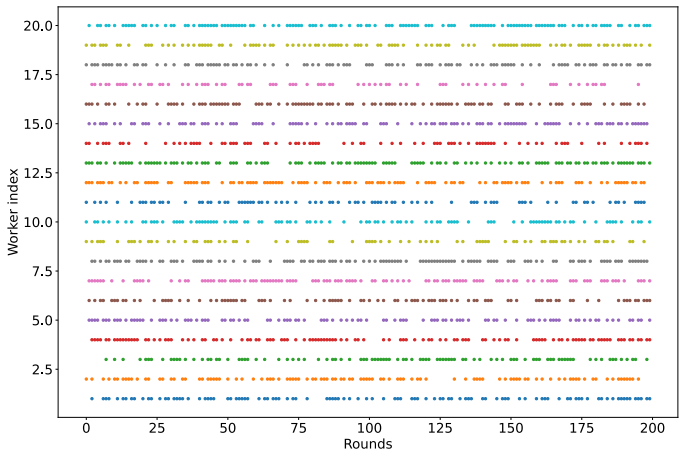


Figure 19: *Selection represented for 20 episodes with 10 rounds per episode with 20 participants*

Moreover, the evolution of the accuracy over the episodes on figures 22 and 23 reaches directly 85% for every first round in both cases (20 workers and 100 workers). It can be explained due to the number of gradients which are aggregated. Indeed, even if the distribution of data is **non-IID**, there are enough gradients to make an aggregation in the same way as a **IID** case.

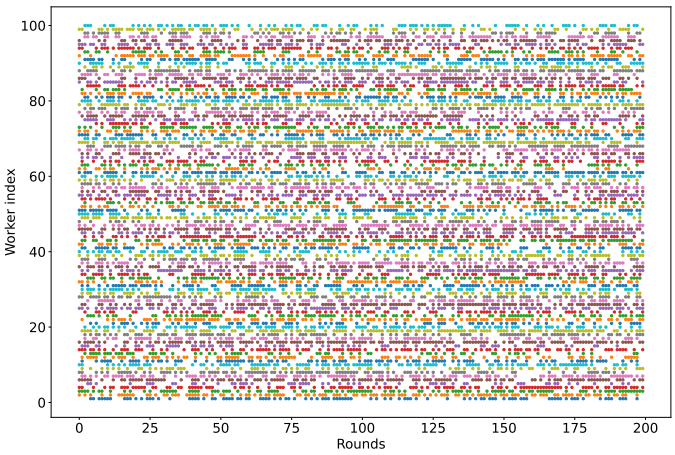
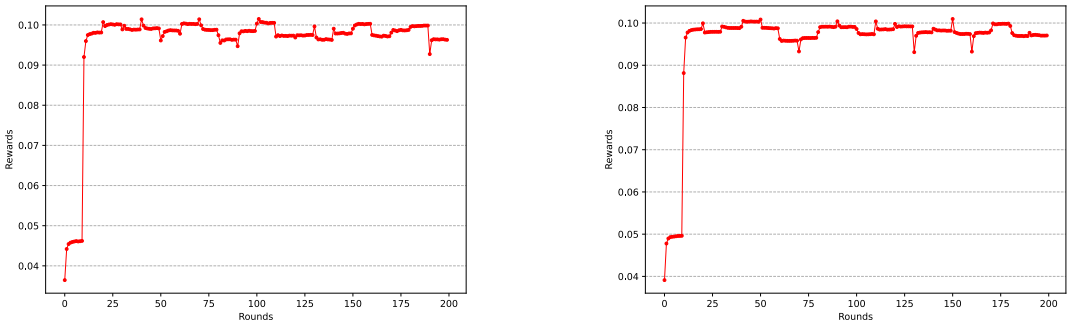


Figure 20: *Selection represented for 20 episodes with 10 rounds per episode with 100 participants*



(a) *Evolution of the rewards given rounds with 20 participants*

(b) *Evolution of the rewards given rounds with 100 participants*

Figure 21: *Evolution of the rewards given rounds*

iii.3 An exploration of Dimensionality Reduction applied for gradients

The performance of the algorithm shown in experiments in section iii does not allow to improve the aggregation. In fact, the state of the environment is based on the accuracy of the task model on local data. The accuracy does not have enough information than the gradients. Therefore, one way to improve the state of the environment is to use gradients. However, the size of gradients for neural networks is often large (e.g. here, the task model has almost 1.2 millions of values for the gradient). Hence, the dimensionality reduction of gradients can be used to extract the information of gradients. Several methods of manifold learning were explored and only the best results will be shown below.

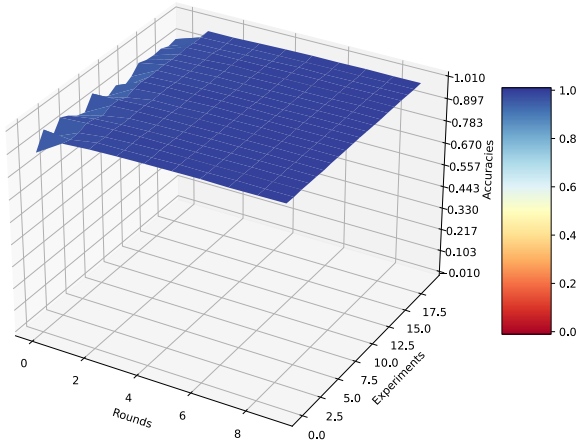


Figure 22: *Evolution of the accuracies given episodes and rounds with 20 participants*

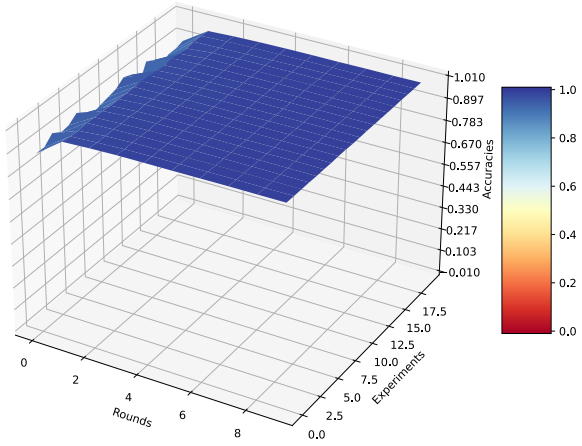


Figure 23: *Evolution of the accuracies given episodes and rounds with 100 participants*

Here is a non exhaustive list of them:

- **PCA** (**P**rincipal **C**omponent **A**nalysis)
- **MDS** (**M**ulti**d**imensional **S**caling)
- **t-SNE** (**t**-distributed **S**tochastic **N**eighbor **E**mbedding)

- Isomap embedding
- Spectral embedding for non-linear dimensionality reduction
- Locally linear embedding

The experiment was made on 40 workers on MNIST dataset where labels are swapped and are distributed to form 4 clusters. For instance, the sample of the number 3 become the label 8 for 10 workers randomly chosen. The goal is to reduce the dimension of gradients and find the clusters through visualization.

On figure 24 and figure 25, dots indicates a gradient. When different dots have with the same color, it means they belong to the same cluster. These methods allows to go from an input size $40 \times 1.2M$ to an output size 40×2.

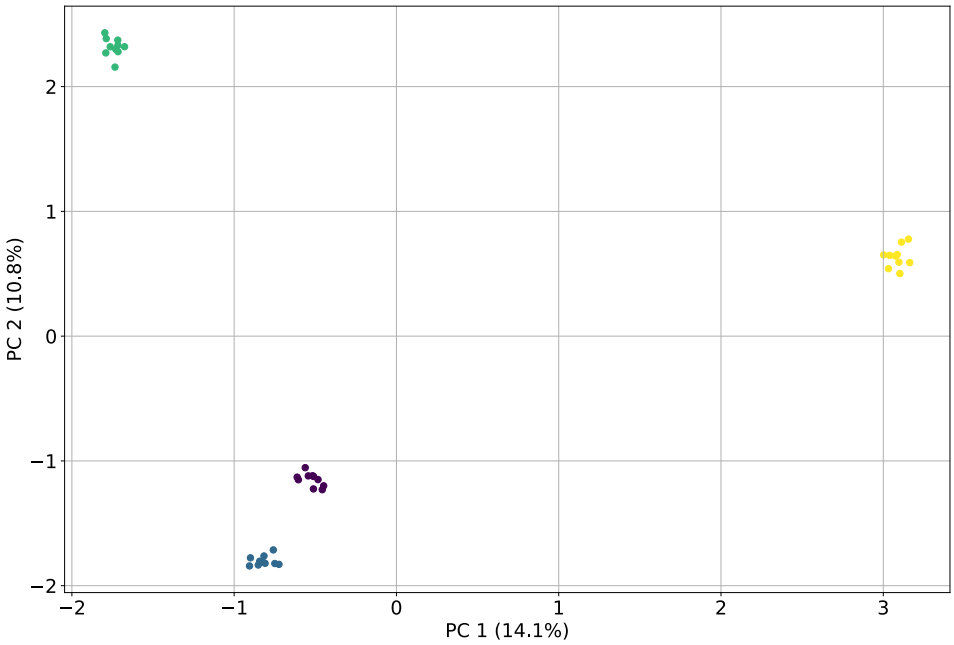


Figure 24: *PCA applied on 40 gradients with 4 clusters*

With the PCA method on figure 24, the variance ratio for 2 components is, in this case, 25%. Before the reduction, there is a total of 48 millions of values for the state with 40 workers, and after reduction, there is only a total of 80 values. Applying the PCA method with a preservation of 95% of variance gives an output size 40×37, in other words, gradients of every workers are reduced to 47 components per worker. The spectral embedding constructs an affinity matrix and applies spectral decomposition to the corresponding graph Laplacian. The affinity matrix in this case is computed based on a radial basis function kernel. The main difference between 24 and 25 is the scale of axis which is smaller for the spectral embedding method than the PCA method.

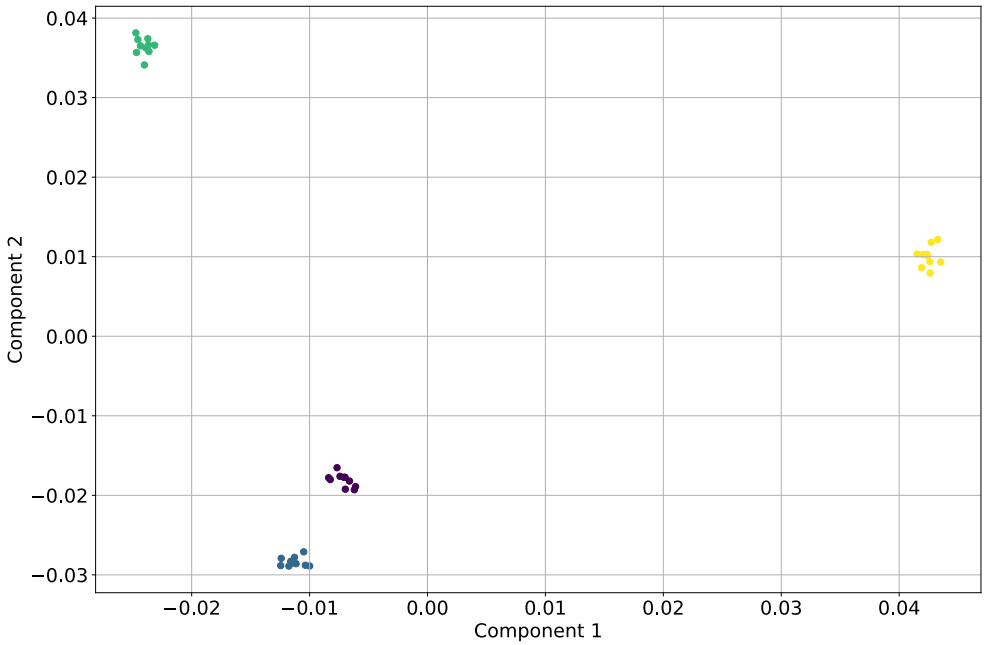


Figure 25: *Spectral embedding applied on 40 gradients with 4 clusters*

iv. Results on the selection of next participants algorithm

This experiment focuses on simulating a dynamic environment with different times between workers and estimates the algorithm for a better selection of future participants.

iv.1 Scope of the experiment

To simulate different bandwidths and speed of computation, several means and standard deviations are used as parameters for a normal law $\mathcal{N}(m, \sigma)$:

- the speed of computation (in *s/batch*) can have 3 different means : 0.5, 0.7 or 1. The standard deviation of the variable is always 0.02.
- the bandwidth of uploading (in *Mb/s*) can have 2 different means associated with its own standard deviation : (8, 2) or (0.5, 0.2) written as (m, σ)
- the bandwidth of downloading (in *Mb/s*) can have 2 different means associated with its own standard deviation : (30, 5) or (5, 1) written as (m, σ)

Note for each worker corresponds a mean and a standard deviation chosen randomly based on previous values.

For the experiments, the actor network and the critic actor are composed of 3 linear layers with 128 neurons per layer. The exploration of the algorithm can be dissociated from Federated Learning. It means there is no need to have a task model to train the agent network due to the aim of the agent : minimize the time spent by the next participants. However, to compare the performance of the agent on task model accuracy against a completely random selection, the Federated Learning is used

to keep the same environment between the agent and the baseline (random selection). There are 100 workers in this experiment with 20 rounds per experiment and a maximum of 30 experiments. The output of the agent network is a set of probabilities. The selection of next participants is made by applying the Bernoulli law $\mathcal{B}(p)$ for each probability. The shape of the selection is a vector filled with values **0** or **1** where **0** to discard the worker for the next round and **1** denotes to select the worker for the next round. When the probabilities are close to zero and no worker is selected for the next round, then the experiment ends.

The results are divided in two parts :

- a first experiment where the goal is to highlight that the agent is able to find the fastest workers in a dynamic environment
- a second experiment where the environment is kept between training step, testing step and comparison of performance with a random selection

iv.2 Training of the agent

Knowing that the objective of the agent is to maximize the reward, the evolution of the reward illustrated on figure 26 reflects the convergence of the network learning. In fact, due to the definition of the reward (the difference between the maximum of time of participants and the previous maximum time), the progression of the reward tends to zero over rounds when the selection of workers is the fastest workers.

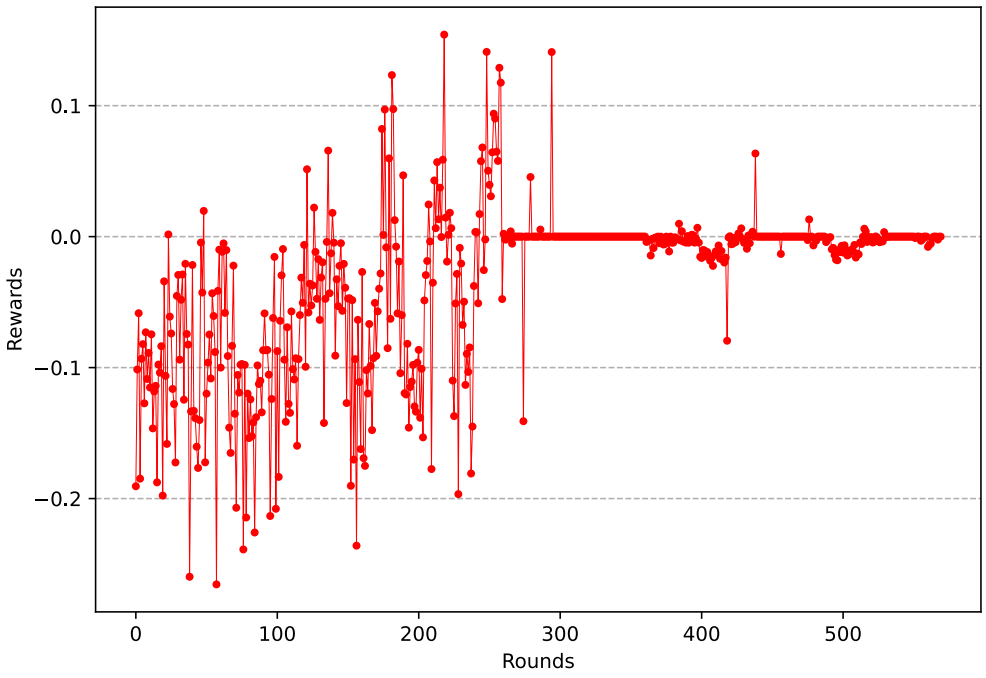


Figure 26: *Evolution of the reward over rounds*

The figure 27 puts forward how probabilities progress over rounds. They are sorted from the fastest worker to the slowest worker according to their means (speed of computation, bandwidth of

uploading and bandwidth of downloading). Each pixel of the image corresponds to the probability of a worker to be selected to the next round. Blue color indicates a probability close to zero while red indicates a probability close to one. Around the round 240, there are 10 participants with a high probability for being selected for the next round. The training should be stopped at this point in order to have enough participants to represent the distribution of date in case of Federated Learning.

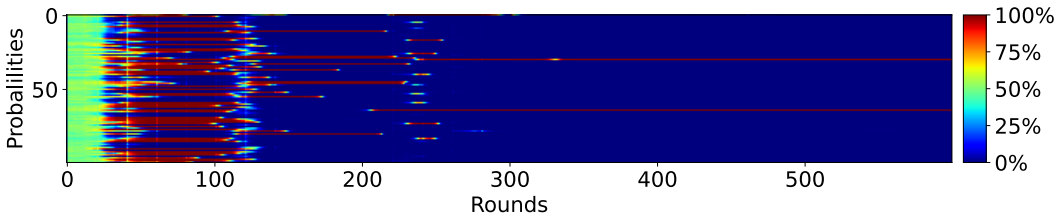


Figure 27: *Evolution of probabilities over rounds for 100 workers*

With the same idea, the figure 28 indicates the selection of workers over rounds. When there is a colored dot, it means the worker has been selected else there is a white space.

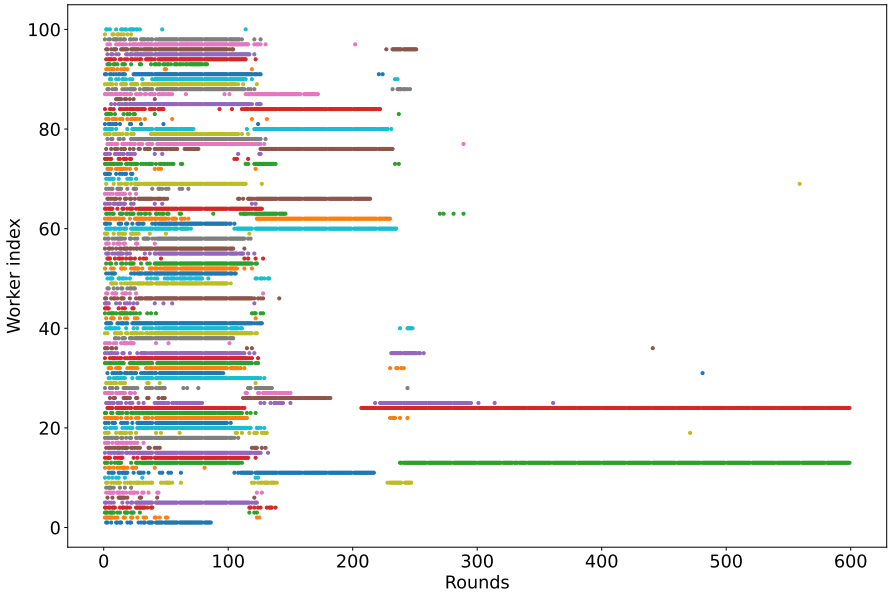
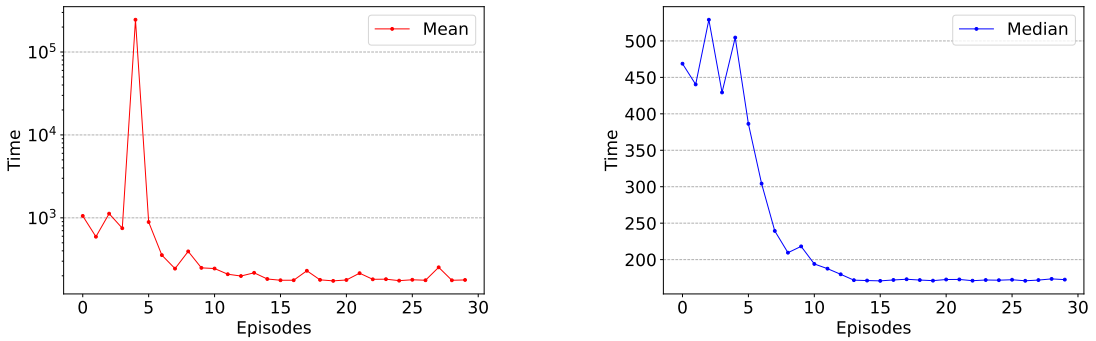


Figure 28: *Evolution of the selection of workers over rounds for 100 workers*

Finally, the maximum time of participants given the episode decreases over time on figure 29. Note on the figure 29a, the scale of the Y-axis is logarithmic to reduce the variance due to a large pic of maximum time whereas the figure 29b does not have this point.



(a) *The mean of the times for each episode*

(b) *The median of the times for each episode*

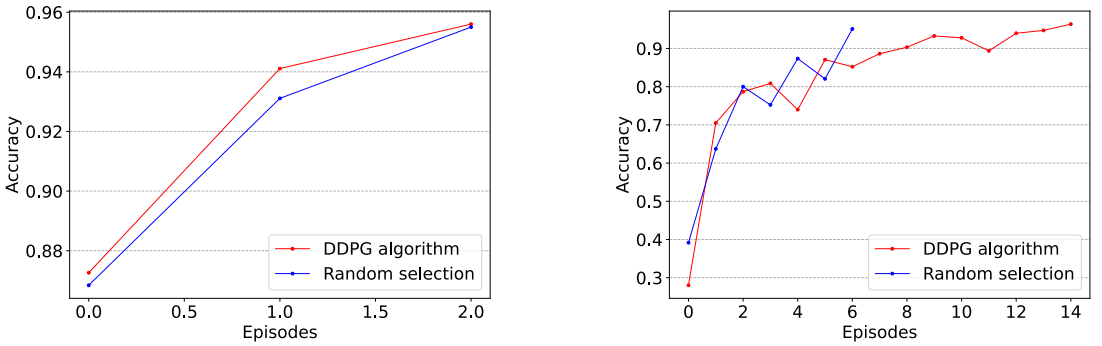
Figure 29: *Evolution of the maximum time of participants over episodes for 100 workers*

iv.3 Analysis of the performance of the agent

The two experiments which are illustrated on following figures and table, allows to compare the performance of the `DDPG` algorithm with a random selection. The comparison is made on the distribution of data on the MNIST dataset: **IID** and **non-IID**. Note the experiment is made with same values of speed of computation and bandwidths for the training, the evaluation and the baseline (random selection). The training is ended when the agent selects less than 10 workers, which represents 10% of the numbers of workers. The random selection is based on 10% of the numbers of workers. The `FedAvg` is used to aggregate for evaluation and for the baseline.

The figure 30 puts forward several points :

- in the **IID** case, DDPG algorithm converges almost with an identical speed.
- on the **non-IID** case, DDPG algorithm takes twice rounds to converge. It can be explained by the fact that the agent chooses the same participants every round which do not represent a subset of the distribution of the data.



(a) *Evolution of the accuracy on **IID** data with 100 workers*

(b) *Evolution of the accuracy on **non-IID** data with 100 workers*

Figure 30: *Comparison of the evolution of the accuracy between the selection with DDPG algorithm and the random selection with 100 workers on MNIST data*

However, the comparison of the maximum time between the two methods in tables 2 and 3,

highlights that the agent selects workers where the maximum time of the set of participants is better than the random selection in **IID** scenario and has a worse performance in **non-IID** scenario like it was mentioned above.

	Round 1	Round 2	Round 3	$m \pm \sigma$
DDPG algorithm	214.852 s	204.948 s	212.285 s	210.685 ± 5.14
Random selection	250.050 s	703.737 s	254.55 s	402.779 ± 260.647

Table 2: *Comparison of maximum time between the selection with DDPG algorithm and the random selection with 100 workers on **IID** data*

	Round 1	Round 2	Round 3	Round 4
DDPG algorithm	221.021 s	383.08 s	219.107 s	268.685 s
Random selection	241.361 s	277.299 s	287.556 s	374.664 s

	Round 5	Round 6	Round 7	Round 8
DDPG algorithm	277.683 s	325.126 s	677.49 s	330.88 s
Random selection	240.724 s	260.318 s	297.858 s	

	Round 9	Round 10	Round 11	Round 12
DDPG algorithm	264.877 s	252.359 s	224.117 s	264.161 s
Random selection				

	Round 13	Round 14	Round 15	$m \pm \sigma$
DDPG algorithm	226.657 s	209.424 s	240.285 s	292.33 ± 117.126
Random selection				282.826 ± 46.028

Table 3: *Comparison of maximum time between the selection with DDPG algorithm and the random selection with 100 workers on **non-IID** data*

V. Conclusion

The algorithm for the evaluation of contribution does not achieve better performance than the `FedAvg` algorithm. As it was explained, the description of the state is based on the accuracy which does not reflect the same amount of information than gradients for instance. However, the idea of using the accuracy rather gradients is due to the large size of gradients (almost 1.2 millions of values for a convolutional neural network model for MNIST dataset). Therefore, the dimensionality reduction of gradients with the PCA method or the spectral embedding method could allow to describe in a better way than using the accuracy. It remains to confirm these two methods with more experiments (different definitions of clusters for instance) and consider the formulation of a new definition of the reward for a better aggregation.

The algorithm for a better selection of participants puts forward the possibility of being able to optimize the selection through conditions. The experiment highlights that during the training part, the algorithm minimizes the maximum time by selecting the participants in a better way. Therefore, the algorithm should be able to optimize new conditions like the maximization of the accuracy or the fairness as well as the minimization of the time. Obviously, experiments must be done to justify this assumption.

More generally, although the Reinforcement Learning offers promising expectations to optimization, there are three majors questions to answer in order to be able to achieve the objective of the challenge:

- What is the state of the environment and how to define it ?
- How to define the reward in order to train correctly the neural network ?
- Which algorithm to choose ?

In case of Federated Learning, these questions are often hard to answer because of the constraints of the Federated Learning field. For instance, in case of **non-IID** for instance, information on the data are not available which involves to find another way to take into account them for the state or for the reward. However Federated Analytics could solve this challenge but it stays an immature field for the moment. As well, the reward significantly impacts the behavior of the network. For example, in experiments which were made, when the reward is always positive, the probabilities in output of the neural network, tend to 1 while when the reward is always negative, the probabilities tends to 0. The definition of the reward becomes a fundamental point of the algorithm. Moreover, for the study of a better selection of participants, different algorithms were tested **TRPO**, **PPO**, **AC**. The choice of the algorithm is based on two criteria: continuous or discrete action space and on-policy or off-policy. In both algorithms (evaluation of contribution and better selection), the action space is discrete and it is more favorable towards using an on-policy than an off-policy. Generally off-policy algorithms utilize a replay memory which means to store transitions over the time and train the network with samples after collecting enough transitions. Whereas the on-policy can learn directly each transition because the action, in the loss function, is taken according to the policy π while in off-policy learning, the algorithm learns from taking taking actions for the same state. At the end, the **DDPG** algorithm is an off-policy algorithm with a replay memory with for a continuous action space. The action space was transformed into a discrete action space to meet the expectations of the challenge.

In this viewpoint, the combination of **FL** with **RL** becomes a topic which asks for expertise in the both fields. To a certain extend, there are many notions to understand in **FL** and **RL**. In other words, this process is not easy to automate for the moment due to the number of parameters to test before a real use case. Furthermore, once the neural network of the algorithm is trained, there is no proof that the algorithm would work in another environment. For instance, training an agent

on workers of a simulated environment may not work on a real world. Thus, a lot of information should be collected to minimize the difference between the simulation and the real world.

The main advantage of using the Reinforcement Learning in Federated Learning is to tackle specific challenges where the environment is well-known whereas for the industry use case, the generic methods are more reliable for a large range of use cases.

A. Appendix

i. Figures

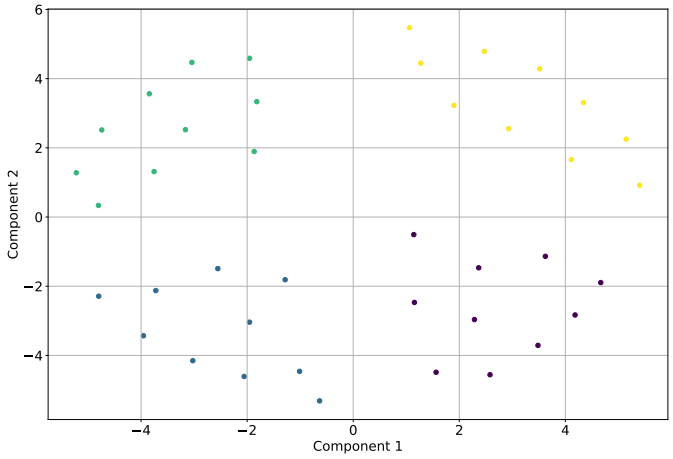


Figure 31: *Multidimensional scaling method on 40 gradients*

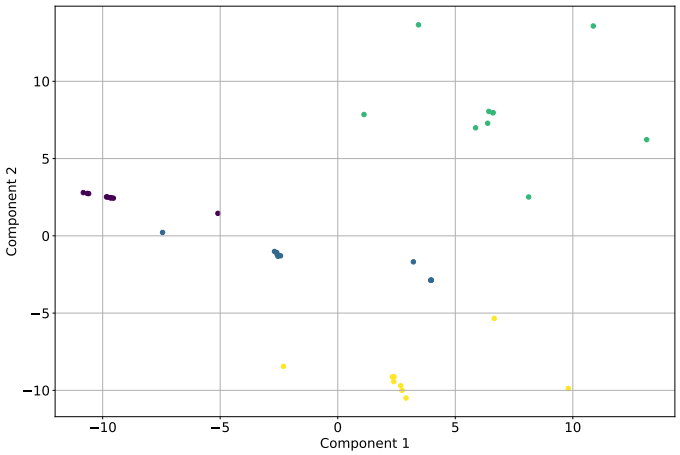


Figure 32: *Isomap embedding method on 40 gradients*

ii. Concise table on different algorithms of Reinforcement Learning

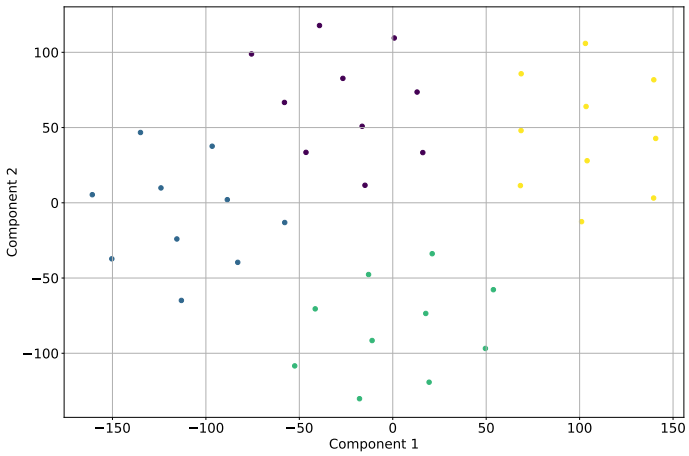


Figure 33: *T-distributed Stochastic Neighbor Embedding method on 40 gradients*

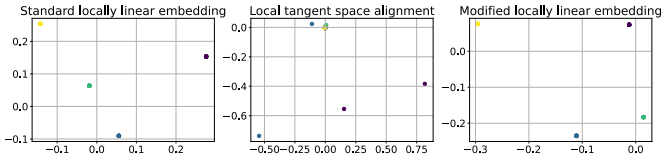


Figure 34: *Locally Linear Embedding method on 40 gradients*

Name	Type	Advantages
TD(0)	off-policy	This method is called one-step TD for looking one-step ahead. TD forms the target from observed return and an estimated state value for the next state. It can learn at every step and **TD should have less variance than MC methods** (see also 1)
TD(λ)	off-policy	λ-returns are a combination of n discounted returns with an existing estimate at the last step (see also 1)
SARSA	on-policy	(state-action-reward-state-action) Difference with TD: we transit from the state-to-state alternation to state-action pair alternation
Q-Learning	off-policy	The main difference that Q-learning has from Sarsa is that the target value now is no longer dependent on the policy being used but only on the state-action function
DQN	off-policy	It uses a deep neural network for Q-value function approximation in Q-Learning, and maintains an experience replay buffer to store transition samples during the agent–environment interactions. DQN also applies a target network QT , which is parameterized by a copy of the original network Q parameter and updated in a delayed manner, to stabilize the learning process, i.e. to alleviate the non- stationary data distribution problem in deep learning.

Double DQN	off-policy	Double DQN is an enhancement of DQN for reducing overestimation. Q is noisy, which may be caused by environment, non-stationarity, function approximation or any other reasons. The central idea of double DQN is to decorrelate the noises in selection and evaluation by using two different networks in these two stages. The Q-network in the DQN architecture provides a natural candidate for the extra network. Recall that it is the evaluation role of the target network that improves the stability more.
Dueling DQN	off-policy	The Q-values of different actions do not matter. So decoupling the action-independent value of state and Q-value may lead to more robust learning. The experiments show that dueling architectures lead to better policy evaluation in the presence of many similar-valued actions.
Rainbow DQN	off-policy	Rainbow uses the truncated n-step return $R_t^{(k)}$ from a given state S_t directly, where $R_t^{(k)}$ is defined by $R_t^{(k)} = \sum_{k=0}^{n-1} \gamma^k R_{t+k}$
Noisy DQN	off-policy	It is an alternative exploration algorithm for ϵ-greedy, especially for games requiring huge exploration. The noise is added into linear layer $y = (W\ x + b)$ by an extra noisy stream $y = (Wx + b) + ((W_{noisy} \odot \epsilon_w)\, x + b_{noisy} \odot \epsilon_b)$
REINFORCE	on-policy	The algorithm REINFORCE follows a straightforward idea of performing gradient ascent on the parameters of the policy *heta*. Despite of its simplicity, naive REINFORCE has been observed to suffer a large variance when estimating the gradient. To alleviate the large variance problem, we further introduce a baseline $b(S_i)$, where $b(S_i)$ is a function only depending on S_i (or more importantly, not dependent to A_i)
Actor-Critic	off-policy	The actor-critic method follows this idea that learns together an actor, the policy function $\pi(\cdot\|s)$, and the critic, the value function $V_\pi(s)$. Moreover, actor-critic also uses the idea of bootstrapping to estimate the Q-value
A2C	on-policy	Synchronous Advantage Actor-Critic - Similar to actor-critic but it focuses on parallel training. It works with one master, one coordinator and several workers. The master is updated like in the AC algorithm
A3C	on-policy	Asynchronous Advantage Actor-Critic - There is no coordinator (asynchronous). Each worker talks directly to the global actor and critic
TRPO	on-policy	Trust Region Policy Optimization aims to handle the step size more properly in policy gradient based on the idea of trust region
PPO	on-policy	Based on TRPO and instead of optimizing with a hard constraint, Proximal Policy Optimization tends to optimize its regularization version
DPPO	on-policy	Distributed Proximal Policy Optimization is a distributed version of the PPO algorithm.
ACKTR	on-policy	Actor Critic Using Kronecker Factor Trust Region uses the Kronecker-factored approximated curvature to compute the natural gradient

DDPG	off-policy	Deep Determinist Policy Gradient - It can be viewed as an extension of the DQN algorithm in continuous action space. It establishes a Q function (critic) and a policy function (actor) simultaneously. It distributes data collection and gradient calculation over multiple workers, which greatly reduces the learning time. Periodically, the chief updates parameters after averaging gradients passed by workers, and then passes the latest parameters to workers synchronously.
TD3	off-policy	Twin Delayed Deep Deterministic Policy Gradient is an improvement of DDPG : clipped double Q-learning for actor critic, target networks and delayed policy updated, target policy smoothing regularization
SAC	off-policy	Soft actor critic follows the idea of maximum entropy reinforcement learning (it aims to maximize its entropy regularized reward). SAC also provides a way in automatically update the regularization coefficient α
CE method	on-policy	CE method is usually faster for policy search in reinforcement learning as a non-gradient-based optimization method. However, preliminary investigations showed that the applicability of CE to reinforcement learning problems is restricted severely by the phenomenon that the distribution concentrates to a single point too fast. Therefore, its applicability in reinforcement learning seems to be limited though fast, because it often converges to suboptimal policies.

Table 4: *Set of algorithms with a small description of their advantages*

B. References

H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," *arXiv:1602.05629 [cs]*, Feb. 2017, arXiv: 1602.05629. [Online]. Available: http://arxiv.org/abs/1602.05629 2, 36

X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the Convergence of FedAvg on Non-IID Data," *arXiv:1907.02189 [cs, math, stat]*, Jun. 2020, arXiv: 1907.02189. [Online]. Available: http://arxiv.org/abs/1907.02189 2

J. Qi, Q. Zhou, L. Lei, and K. Zheng, "Federated Reinforcement Learning: Techniques, Applications, and Open Challenges," *arXiv:2108.11887 [cs]*, Oct. 2021, arXiv: 2108.11887. [Online]. Available: http://arxiv.org/abs/2108.11887 2, 3, 6, 8

M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep Learning with Differential Privacy," *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 308–318, Oct. 2016, arXiv: 1607.00133. [Online]. Available: http://arxiv.org/abs/1607.00133 3

Bellman, "Dynamic Programming," Princeton University Press, Princeton, N.J., 1957. 4, 30, 31

X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-Edge AI: Intelligentizing Mobile Edge Computing, Caching and Communication by Federated Learning," *IEEE Network*, vol. 33, no. 5, pp. 156–165, Sep. 2019, arXiv: 1809.07857. [Online]. Available: http://arxiv.org/abs/1809.07857 8, 9, 10

X. Wang, C. Wang, X. Li, V. C. M. Leung, and T. Taleb, "Federated Deep Reinforcement Learning for Internet of Things With Decentralized Cooperative Edge Caching," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9441–9455, Oct. 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9062302/ 10

I. Martinez, S. Francis, and A. S. Hafid, "Record and Reward Federated Learning Contributions with Blockchain," in *2019 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*. Guilin, China: IEEE, Oct. 2019, pp. 50–57. [Online]. Available: https://ieeexplore.ieee.org/document/8945913/ 10

S. Yu, X. Chen, Z. Zhou, X. Gong, and D. Wu, "When Deep Reinforcement Learning Meets Federated Learning: Intelligent Multi-Timescale Resource Management for Multi-access Edge Computing in 5G Ultra Dense Network," *arXiv:2009.10601 [cs]*, Sep. 2020, arXiv: 2009.10601. [Online]. Available: http://arxiv.org/abs/2009.10601 10

Y. Zhan, P. Li, and S. Guo, "Experience-Driven Computational Resource Allocation of Federated Learning by Deep Reinforcement Learning," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. New Orleans, LA, USA: IEEE, May 2020, pp. 234–243. [Online]. Available: https://ieeexplore.ieee.org/document/9139873/ 11

S. Zarandi and H. Tabassum, "Federated Double Deep Q-learning for Joint Delay and Energy Minimization in IoT networks," *arXiv:2104.11320 [cs]*, Apr. 2021, arXiv: 2104.11320. [Online]. Available: http://arxiv.org/abs/2104.11320 11

Y. Chen, Z. Liu, Y. Zhang, Y. Wu, X. Chen, and L. Zhao, "Deep Reinforcement Learning-Based Dynamic Resource Management for Mobile Edge Computing in Industrial Internet of Things," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 7, pp. 4925–4934, Jul. 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9214878/ 11

W. Jeong, J. Yoon, E. Yang, and S. J. Hwang, "Federated Semi-Supervised Learning with Inter-Client Consistency & Disjoint Learning," *arXiv:2006.12097 [cs, stat]*, Mar. 2021, arXiv: 2006.12097. [Online]. Available: http://arxiv.org/abs/2006.12097 11

A. Ghosh, J. Hong, D. Yin, and K. Ramchandran, "Robust Federated Learning in a Heterogeneous Environment," *arXiv:1906.06629 [cs, stat]*, Oct. 2019, arXiv: 1906.06629. [Online]. Available: http://arxiv.org/abs/1906.06629 11

W. Liu, L. Chen, Y. Chen, and W. Zhang, "Accelerating Federated Learning via Momentum Gradient Descent," *arXiv:1910.03197 [cs, stat]*, Oct. 2019, arXiv: 1910.03197. [Online]. Available: http://arxiv.org/abs/1910.03197 11

V. Smith, C.-K. Chiang, M. Sanjabi, and A. Talwalkar, "Federated Multi-Task Learning," *arXiv:1705.10467 [cs, stat]*, Feb. 2018, arXiv: 1705.10467. [Online]. Available: http://arxiv.org/abs/1705.10467 11

P. Zhang, C. Wang, C. Jiang, and Z. Han, "Deep Reinforcement Learning Assisted Federated Learning Algorithm for Data Management of IIoT," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 12, pp. 8475–8484, Dec. 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9372789/ 11, 12, 14

S. Q. Zhang, J. Lin, and Q. Zhang, "A Multi-agent Reinforcement Learning Approach for Efficient Client Selection in Federated Learning," *arXiv:2201.02932 [cs]*, Jan. 2022, arXiv: 2201.02932. [Online]. Available: http://arxiv.org/abs/2201.02932 12

X. Liang, Y. Liu, T. Chen, M. Liu, and Q. Yang, "Federated Transfer Reinforcement Learning for Autonomous Driving," *arXiv:1910.06001 [cs]*, Oct. 2019, arXiv: 1910.06001. [Online]. Available: http://arxiv.org/abs/1910.06001 14

H.-K. Lim, J.-B. Kim, J.-S. Heo, and Y.-H. Han, "Federated Reinforcement Learning for Training Control Policies on Multiple IoT Devices," *Sensors*, vol. 20, no. 5, p. 1359, Mar. 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/5/1359 14

Y. Esfandiari, S. Y. Tan, Z. Jiang, A. Balu, E. Herron, C. Hegde, and S. Sarkar, "Cross-Gradient Aggregation for Decentralized Learning from Non-IID data," *arXiv:2103.02051 [cs]*, Jun. 2021, arXiv: 2103.02051. [Online]. Available: http://arxiv.org/abs/2103.02051 14

S. U. Stich and S. P. Karimireddy, "The Error-Feedback Framework: Better Rates for SGD with Delayed Gradients and Compressed Communication," Jun. 2021, number: arXiv:1909.05350 arXiv:1909.05350 [cs, math, stat]. [Online]. Available: http://arxiv.org/abs/1909.05350 14

B. Liu, L. Wang, and M. Liu, "Lifelong Federated Reinforcement Learning: A Learning Architecture for Navigation in Cloud Robotic Systems," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4555–4562, Oct. 2019, arXiv: 1901.06455. [Online]. Available: http://arxiv.org/abs/1901.06455 15

J. Zhao, X. Zhu, J. Wang, and J. Xiao, "Efficient Client Contribution Evaluation for Horizontal Federated Learning," *arXiv:2102.13314 [cs, eess]*, Feb. 2021, arXiv: 2102.13314. [Online]. Available: http://arxiv.org/abs/2102.13314 16, 20

H. Dong, Z. Ding, and S. Zhang, *Deep Reinforcement Learning, Fundamentals, Research and Applications*. Springer, 2020.

C. Glossary

AC Actor Critic. 29, 52

CNN Convolutional Neural Network. 16

DDPG Deep Deterministic Policy Gradient. 29, 52

DNN Deep Neural Networks. 3

DQN Deep Q Network. 9, 29, 30

DRL Deep Reinforcement Learning. 3, 8–10

FL Federated Learning. 2, 3, 5, 6, 8–11, 13, 17, 52

FRL Federated Reinforcement Learning. 2

HFRL Horizontal Federated Reinforcement Learning. 8

IID Independent and Identically Distributed. 2, 7, 11, 12, 14, 36, 37, 39, 43, 50, 51

IIoT Industrial Internet of Things. 11, 12

IoT Internet of Things. 5, 8, 14

LFRL Lifelong Federated Reinforcement Learning. 15

MARL Multi-Agent Reinforcement Learning. 12, 13

MDP Markov Decision Process. 4

MDS Multidimensional Scaling. 45

MLP Multi-Layer Perceptron. 13

ML Machine Learning. 2, 3

PCA Principal Component Analysis. 45

PPO Proximal Policy Optimization. 29, 52

RL Reinforcement Learning. 2–4, 11, 12, 14–16, 52

SGD Stochastic Gradient Descent. 2

TRPO Trust Region Policy Optimization. 29, 52

UEs User Equipments. 8

VFRL Vertical Federated Reinforcement Learning. 8

non-IID Non Independent and Identically Distributed. 2, 5, 7, 11, 12, 14, 29, 36–40, 43, 50–52

t-SNE t-distributed Stochastic Neighbor Embedding. 45

`CGA` Cross-Gradient Aggregation. 14

`F-RCCE` Federated REINFORCE client contribution evaluation. 16, 17

FSSL Federated Semi-Supervised Learning. 11

FedAvg Federated Averaging. 2, 34, 36, 37, 50, 52

FedMARL Federated Multi-Agent Reinforcement Learning. 12, 13