Federated Reinforcement Learning

Hankz Hankui Zhuo [1] **Wenfeng Feng** [1] **Qian Xu** [2] **Qiang Yang** [2] **Yufeng Lin** [1]
[1]**Sun Yat-sen University** {**zhuohank@mail; fengwf@mail2; linyf23@mail2**}**.sysu.edu.cn**
[2]**WeBank** {**qianxu; qiangyang**}**@webank.com**

Abstract

In reinforcement learning, building policies of high-quality is challenging when the feature space of states is small and the training data is limited. Directly transferring data or knowledge from an agent to another agent will not work due to the privacy requirement of data and models. In this paper, we propose a novel reinforcement learning approach to considering the privacy requirement and building Q-network for each agent with the help of other agents, namely federated reinforcement learning (`FRL`). To protect the privacy of data and models, we exploit Gausian differentials on the information shared with each other when updating their local models. In the experiment, we evaluate our `FRL` framework in two diverse domains, Grid-world and Text2Action domains, by comparing to various baselines.

1. Introduction

Conventional machine learning aims at training models from large datasets assuming they are accessible from a data center. In many real-world applications, however, datasets from clients are often privacy sensitive and it is often difficult to such a data center to guarantee building models of high-quality. To deal with the issue, Konecny et al. propose to a new learning setting, namely federated learning, whose goal is to train a classification or clustering model with training data involving texts, images or videos distributed over a large number of clients (Konecný et al., 2016; McMahan et al., 2017). Different from previous federated learning, we propose a novel federated learning framework based on reinforcement learning (Sutton & Barto, 1998; Mnih et al., 2015; Co-Reyes et al., 2018), i.e., *federated reinforcement learning* (`FRL`), which aims to learn private Q-network policy for each agent by sharing limited information (i.e., output of the Q-network) among agents, which is "encrypted" when sending the information to others and "decrypted" when receiving the information from others. We assume that some agents may have *rewards* corresponding to states and actions, while others have only observed states and actions but *rewards* which suggests these agents are unable to build decision policies by theirselves. We claim that all agents benefit from joining the federation in building decision policies.

There are many applications regarding federated reinforcement learning. *For example, in manufacturing industry, producing products may involve various factories which produce different components of the products. Factories' decision policies are private and will not be shared with each other. On the other hand, building individual decision policies of high-quality by their own is often difficult due to their limited businesses and lack of rewards (for some agents). It is thus helpful for them to learn decision polices federatively under the condition that private data is not given away. Another example is building medical treatment policies to patients for hospitals. Patients may be treated in some hospitals and never give feedbacks to the treatments, which indicates these hospitals are unable to collect rewards based on the treatments given to the patients and build treatment decision policies for patients. In addition, data records about patients are private and may not be shared among hospitals. It is thus necessitated to learn treatment policies for hospitals federatively.*

Our `FRL` framework is different from multi-agent reinforcement learning, which is concerned with a set of autonomous agents that observe global states (or partial states which are directly shared to make "global" states), select an individual action and receive a team reward (Tampuu et al., 2015; Leibo et al., 2017). `FRL` assumes agents do not share their partial observations and some agents are unable to receive rewards. Our `FRL` framework is also different from transfer learning in reinforcement learning, which aims to transfer experience gained in learning to perform one task to help improve learning performance in a related but different task or agent, assuming observations are shared with each other (Taylor & Stone, 2009; Tirinzoni et al., 2018), while `FRL` assumes states cannot be shared among agents.

Our `FRL` framework functions in three phases. Initially, each agent collects output values of Q-networks from other agents, which are "encrypted" with Gausian differentials.

Furthermore, it builds a neural network, e.g., MLP (multilayer perceptron), to compute a global Q-network output with its own Q-network output and the encrypted values as input. Finally, it updates both MLP and its own Q-network based on the global Q-network output. Note that MLP is shared among agents while agents' own Q-networks are unknown to others and should not be inferred based on the encrypted Q-network output shared in the training process.

In the remainder of the paper, we first review previous work related to our `FRL` framework, and then present the problem formulation of `FRL`. After that, we introduce our `FRL` framework in detail. Finally we evaluate our `FRL` framework in the Grid-World domain with various sizes and the Text2Actions domain.

2. Related Work

The nascent field of federated learning considers training statistical models directly on devices (Konecný et al., 2015; McMahan et al., 2017). The aim in federated learning is to fit a model to data generated by distributed nodes. Each node collects data in a non-IID manner across the network, with data on each node being generated by a distinct distribution. There are typically a large number of nodes in the network, and communication is often a significant bottleneck. Different from previous work that train a single global model across the network (Konecný et al., 2015; 2016; McMahan et al., 2017), Smith et al. propose to learn separate models for each node which is naturally captured through a multi-task learning (MTL) framework, where the goal is to consider fitting separate but related models simultaneously (Smith et al., 2017). Different from those federated learning approaches, we consider federated settings in reinforcement learning.

Our work is also related to multi-agent reinforcement learning (MARL) which involves a set of agents in a shared environment. A straightforward way to MARL is to extend the single-agent RL approaches. Q-learning has been extended to cooperative multi-agent settings, namely Independent Q-learning (IQL), in which each agent observes the global state, selects an individual action and receives a team reward (Tampuu et al., 2015; Leibo et al., 2017). One challenging of MARL is that multi-agent domains are non-stationary from agent's perspectives, due to other agents' interactions in the shared environment. To address this issue, (Omidshafiei et al., 2017) propose to explore Concurrent Experience Replay Trajectories (CERTs) structures, which store different agents' histories, and align them together based on the episode indices and time steps. Due to the action space growing exponentially with the number of agents, learning becomes very difficult due to partial observability of limited communication when the number of agents is large. (Lowe et al., 2017) thus propose to solve the MARL problem through a Centralized Critic and a Decentralized Actor, and (Rashid et al., 2018) propose to exploit a linear decomposition of the joint value function across agents. Different from MARL, our `FRL` framework assumes agents do not share their partial observations and some agents are unable to receive rewards, instead of assuming observations are sharable and all agents are able to receive rewards.

3. Problem Definition

A Markov Decision Process (MDP) can be defined by $\langle S, A, T, r \rangle$, where S is the state space, A is the action space, and T is a transition function: $S \times A \rightarrow S$, i.e., $T(s, a, s') = P(s'|s, a)$, specifying the probability of next state $s' \in S$ given current state $s \in S$ and $a \in A$ that applies on s. r is the reward function: $S \rightarrow \mathcal{R}$ where $\mathcal{R}$ is the space of real numbers. Given a policy $\pi : S \rightarrow A$ for $t \in \{0, \ldots, K-1\}$, let $V^{\pi}_{t+1}(s) = r(s) + \sum_{s' \in S} T(s, \pi(s), s') V^{\pi}_t(s')$ and $Q^{\pi}_{t+1}(s, a) = r(s) + \sum_{s' \in S} T(s, a, s') V^{\pi}_t(s')$ be its value function and Q-function, where $V^{\pi}_0(s) = 0$. The solution to a MDP problem is to find the best policy π^* such that $v^{\pi^*}_t(s) = \max_{\pi} v^{\pi}_t(s)$ or $Q^{\pi^*}_t(s, \pi^*(s)) = \max_{\pi} Q^{\pi}_t(s, \pi(s))$. In Q-networks learning, given the transition function T is unknown, the Q-function is represented by a Q-network $Q(s, a; \theta)$ with θ as the parameters, and updated by

$$Q_{t+1}(s, a; \theta) = E\Big\{ r(s) + \gamma \max_{a' \in A} Q_t(s', a'; \theta) | s, a \Big\},$$

as done by (Mnih et al., 2015). To learn the parameters θ, transitions $\langle s, a, s', r \rangle$ are stored in replay memories Ω and a mini-batch sampling can be exploited to repeatedly update θ (Mnih et al., 2015). Once the parameters θ is learnt, the policy π^* can be extracted from $Q(s, a; \theta)$: $\pi^*(s) = \arg\max_{a \in A} Q(s, a; \theta)$.

Suppose there are two agents α and β making decisions in two different environments, e.g., two doctors treating patients at two different hospitals, respectively. Agent α is able to collect reward r and new state s' from the environment given current state s and action a, i.e., agent α is capable of building complete transitions $\langle s, a, s', r \rangle$ from the environment (e.g., patients are treated in the hospital). On the other hand, agent β can only collect current state s and action a, without any information about next state s' and reward r from the environment. For example, patients that doctor β treats never come back such that β cannot collect new state s' about the patients and reward r based on the treatment (i.e., action a) β gave to the patient (i.e., current state s).

We thus define our *federated* reinforcement learning problem as follows: *given transitions* $\{\langle s_\alpha, a_\alpha, s'_\alpha, r_\alpha \rangle\}$ *col-*

lected by agent α, and $\{\langle s_\beta, a_\beta\rangle\}$ collected by agent β, we aim to build policies π^*_α and π^*_β for agents α and β, respectively. **Note that we consider the federation with two members. It is straightforward to extend it to many.** To make difference of states, actions, Q-functions, policies with respect to agents α and β, we will associate subscripts of "α" and "β" with them, i.e., "$s_\alpha \in S_\alpha$, $a_\alpha \in A_\alpha$, Q_α, π^*_α" and "$s_\beta \in S_\beta$, $a_\beta \in A_\beta$, Q_β, π^*_β", respectively. Both agents share the same reward function (which is unknown to agent β), denoted by r without any subscript.

In our federated reinforcement learning problem, we assume:

A1: The feature spaces of s_α and s_β are *different* from each other but the values are *related* with each other. For example, s_α denotes a patient's cardiogram, while s_β denotes the patient's electroencephalogram, which indicates the feature spaces are different while the values of them are related since they both refer to the same patent.

A2: s_α (or r_α) and s_β (or r_β) *cannot* be shared with each other due to data privacy protection, e.g., patients' information is not allowed to be shared to others.

A3: The output of functions Q_α and Q_β *can* be shared with each other under the condition that the networks of their own are complex enough and unknown to each other (the input of them is unknown to each other as well based on **A1**), such that α and β cannot induce each other's networks.

Based on **A1-A3**, we aim to leverage information from α and β to learn policies π^*_α and π^*_β of high-quality. We claim that, based on the *federating* of both sides, the policy π^*_α is better than the one α learns by itself, and β gains a high-quality policy π^*_β by joining in the federation, instead of "zero" by itself (β cannot build policies based on its own data $\{\langle s_\beta, a_\beta\rangle\}$).

4. Our FRL Approach

The framework of FRL is shown in Figure 1, which is composed of three main components: basic Q-networks, Gaussian differential privacy and MLP module, as described below.

Basic Q-networks: We build two Q-networks of α and β, denoted by $Q_\alpha(o_\alpha, a; \theta_1)$ and $Q_\beta(o_\beta, a; \theta_2)$, respectively, where θ_1 and θ_2 are parameters of the Q-networks. The outputs of these two basic Q-networks are not directly used to predict the actions, but are taken as input of a federated network.

Gaussian differential privacy: To protect the data better, we consider using the differential privacy (Dwork & Roth, 2014). There are various mechanisms of differential privacy such as Gaussian mechanism (Abadi et al., 2016) and Binomial mechanism (Agarwal et al., 2018). In this paper, we exploit Gaussian mechanism since the output of MLP with Gaussian input is Gaussian itself. In conventional federated learning, the mechanism is applied to the gradients of agents (clients) before transmitted to the server or other agents (clients). In our FRL framework, we transmitted Q-values of basic Q-networks, so the Gaussian noise is added to the Q-values rather than the gradients. The mechanism is defined as

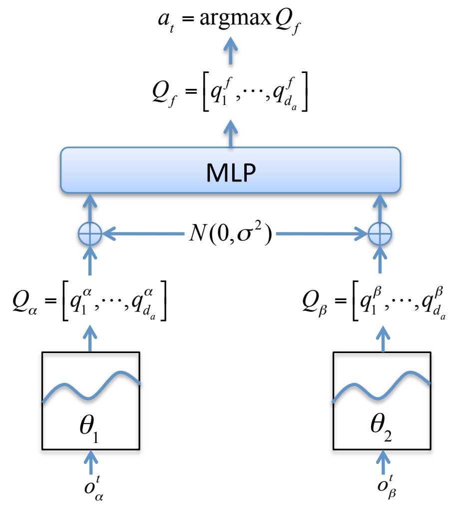

Figure 1: FRL framework. o^t_α, o^t_β are observations of agents at time step t. θ_1, θ_2 are parameters of Q-networks. Q_α, Q_β are two real-valued vectors which indicate the basic q-values corresponding to the two Q-networks, d_a is the dimension of the action space. $N(0, \sigma^2)$ is the Gaussian noise. Q_f is the federated Q-values and a_t is the predicted action at t.

$$\hat{Q}_\alpha(o_\alpha, a_\alpha; \theta_1) = Q_\alpha(o_\alpha, a_\alpha; \theta_1) + N(0, \sigma^2) \quad (1)$$
$$\hat{Q}_\beta(o_\beta, a_\beta; \theta_2) = Q_\beta(o_\beta, a_\beta; \theta_2) + N(0, \sigma^2) \quad (2)$$

where $N(0, \sigma^2)$ is the Gaussian distribution with mean 0 and standard deviation σ.

MLP module: We propose a MLP as the federated network to take as input the basic Q-values with Gaussian noise and output the final Q-values, which is defined by

$$Q_f(\cdot; \theta_1, \theta_2, \theta_3) = MLP([\hat{Q}_\alpha(o_\alpha, a_\alpha; \theta_1) | \hat{Q}_\beta(o_\beta, a_\beta; \theta_2)]; \theta_3) \quad (3)$$

where Q_f is the federated Q-network, θ_3 is the parameters of the MLP and $[\cdot|\cdot]$ indicates the concatenation operation.

We assume that agents α and β share Gaussian differential privacy and MLP modules but have their own private basic Q-networks. Due to Gaussian differential privacy protection and private basic Q-networks (both weights and structures of Q-networks are unknown to each other), Agents α

and β are unable to decrypt the Q-networks of each other when learning parameters θ_1, θ_2 and θ_3. Specifically, agent α updates its own parameters θ_1 and parameters θ_3 of MLP module by viewing the output of basic Q-networks of agent β as a constant, while agent β updates its own parameters θ_2 and parameters θ_3 of MLP module by viewing the output of basic Q-networks of agent α as a constant. We thus define the final outputs of Q-values with respect to agents α and β are:

$$Q_f^{\alpha}(\cdot, C_\beta; \theta_1, \theta_3) = MLP([\hat{Q}_\alpha(\cdot; \theta_1) | C_\beta]; \theta_3) \tag{4}$$

$$Q_f^{\beta}(\cdot, C_\alpha; \theta_2, \theta_3) = MLP([C_\alpha | \hat{Q}_\beta(\cdot; \theta_2)]; \theta_3) \tag{5}$$

, where $C_\alpha = \hat{Q}_\alpha(o_\alpha, a_\alpha; \theta_1)$ and $C_\beta = \hat{Q}_\beta(o_\beta, a_\beta; \theta_2)$ are constant outputs of basic Q-networks with Gaussian differential noise of agents β and α, respectively.

To estimate the optimal action-value function, the federated framework performs value iteration, and Q-values are updated by iteratively applying Bellman updates,

$$\begin{aligned} Q_f^{\alpha}(o_\alpha, a_\alpha, C_\beta; \theta_1, \theta_3) &= r + \\ &\gamma \max_{a'} Q_f^{\alpha}(o'_\alpha, a'_\alpha, C_\beta; \theta_1, \theta_3) \end{aligned} \tag{6}$$

$$\begin{aligned} Q_f^{\beta}(o_\beta, a_\beta, C_\alpha; \theta_2, \theta_3) &= r + \\ &\gamma \max_{a'} Q_f^{\beta}(o'_\beta, a'_\beta, C_\alpha; \theta_2, \theta_3). \end{aligned} \tag{7}$$

The Q-networks can be trained by minimizing the square error loss functions $L_\alpha^j(\theta_1, \theta_3)$ and $L_\beta^j(\theta_2, \theta_3)$ that change at each iteration j,

$$L_\alpha^j(\theta_1, \theta_3) = \mathbb{E}\left[(Y^j - Q_f^{\alpha}(o_\alpha^j, a_\alpha^j, C_\beta; \theta_1, \theta_3))^2\right] \tag{8}$$

$$L_\beta^j(\theta_2, \theta_3) = \mathbb{E}\left[(Y^j - Q_f^{\beta}(o_\beta^j, a_\beta^j, C_\alpha; \theta_2, \theta_3))^2\right] \tag{9}$$

where $Y^j = r^j + \gamma \max_a Q_f^{\alpha}(o_\alpha^j, a, C_\beta; \theta_1, \theta_3)$.

The full training processes of agents α and β are shown in Algorithms 1 and 2. In Steps 1 and 2 of Algorithm 1, we initialize the basic Q-network and replay memory of agent α and the MLP module. In Step 3 of Algorithm 1, we call the function in Algorithm 2 to initialize the Q-network and replay memory of agent β. In Step 6 of Algorithm 1, we obtain observation of agent α. In Step 7 of Algorithm 1, we call the function in Algorithm 2 to calculate the output of basic Q-network of agent β and obtain observation of agent β and select the corresponding action, as shown in Steps from 6 to 10 of Algorithm 2. In Steps from 8 and 11 of Algorithm 1, we perform the ϵ-greedy exploration and exploitation, obtain new observations and store the transitions to the replay memory. In Steps 12 and 13 of Algorithm 1, we sample a record j in the memory and call the function $ComputeQBeta(j)$ of Algorithm 2 to calculate the output of basic Q-network of agent β based on the index j. In Steps 14 and 15 of Algorithm 1, we update parameters θ_1

Algorithm 1 `FRL-ALPHA`

Input: state space S_α, action space A_α, rewards r
Output: θ_1,θ_3

1: Initialize Q_α, Q_f with random values for θ_1, θ_3
2: Initialize replay memory D_α
3: Call `FRL-BETA.`$Init()$
4: **for** episode = 1: M **do**
5:   **repeat**
6:     Observe o_α^t
7:     Call C_β=`FRL-BETA.`$ComputeQBeta()$
8:     Select action a^t with probability ϵ
9:     Otherwise $a^t = \arg\max_a Q_f^{\alpha}(o_\alpha^t, a, C_\beta; \theta_1, \theta_3)$
10:     Execute action a^t, obtain reward r^t and state s^{t+1}
11:     Observe o_α^{t+1}, store $(o_\alpha^t, a^t, r^t, o_\alpha^{t+1})$ in D_α
12:     Sample $(o_\alpha^j, a^j, r^j, o_\alpha^{j+1})$ from D_α
13:     Call C_β=`FRL-BETA.`$ComputeQBeta(j)$
14:     $Y^j = r^j + \gamma \max_a Q_f^{\alpha}(o_\alpha^j, a, C_\beta; \theta_1, \theta_3)$
15:     Update θ_1, θ_3 according to Eq. (4), (6), (8)
16:     $C_\alpha = \hat{Q}_\alpha(o_\alpha^j, a; \theta_1)$
17:     Call θ_3=`FRL-BETA.`$UpdateQBeta(Y^j, j, C_\alpha, \theta_3)$
18:   **until** terminal t
19: **end for**

Algorithm 2 `FRL-BETA`

Input: state space S_β, action space A_β
Output: θ_2, θ_3

1: **function** $Init()$
2:   Initialize Q_β with random values for θ_2
3:   Initialize replay memory D_β
4: **end function**
5: **function** $ComputeQBeta()$
6:   Observe o_β
7:   Select action $a_\beta \in A_\beta$ with probability ϵ
8:   Otherwise $a_\beta = \arg\max_{a_\beta} Q_\beta(o_\beta, a_\beta; \theta_2)$
9:   store (o_β, a_β) in D_β
10:   let $C_\beta = \hat{Q}_\beta(o_\beta, a; \theta_2)$
11:   **return** C_β
12: **end function**
13: **function** $ComputeQBeta(j)$
14:   Select (o_β, a_β) from D_β based on index j
15:   let $C_\beta = \hat{Q}_\beta(o_\beta, a_\beta; \theta_2)$
16:   **return** C_β
17: **end function**
18: **function** $UpdateQBeta(Y, j, C_\alpha, \theta_3)$
19:   Select (o_β^j, a_β^j) from D_β based on index j
20:   Update θ_2, θ_3 according to Eq. (5), (7), (9)
21:   **return** θ_3
22: **end function**

and θ_3. In Steps 16 and 17 of Algorithm 1, we compute the output of basic Q-network of agent α and pass it to agent β and call the function $UpdateQBeta$ of agent β to update basic Q-network of agent β and MLP, as shown in Steps from 19 to 21 of Algorithm 2. Note that Algorithms 1 and 2 are executed by agents α and β separately.

5. Experiments

The main objective of the experiments is to understand the benefits of federated reinforcement learning. Specifically, we want to find out when do we need FRL and how FRL works. In this section, we first compare our `FRL` model with several baselines. We then present some key findings of the federated model.

5.1. Baselines

We compare `FRL` with following baselines:

DQN-alpha: A basic DQN-model that is trained by taking the observation o_α^t as input, i.e. approximating the Q-function $Q_\alpha(o_\alpha, a; \theta_1)$ with a CNN.

DQN-full: A DQN-model that is trained by taking both observations o_α^t and o_β^t as input, i.e. approximating the Q-function $Q_{full}(o_\alpha, o_\beta, a; \theta_f)$ with a CNN.

CNN-alpha: A CNN-based framework that takes each observation o_α as input and outputs the action. It is a basic classification model.

CNN-full: A CNN-based framework that takes both observations o_α and o_β as input and outputs the action.

Our `FRL`: The proposed approach. To improve the robustness of the model, we do not apply Gaussian mechanism to all training samples, but with a noise probability p, i.e. set $\hat{Q}_\alpha(o_\alpha, a; \theta_1) = Q_\alpha(o_\alpha, a; \theta_1) + N(0, \sigma^2), \hat{Q}_\beta(o_\beta, a; \theta_2) = Q_\beta(o_\beta, a; \theta_2) + N(0, \sigma^2)$ with the probability p, otherwise set $\hat{Q}_\alpha(o_\alpha, a; \theta_1) = Q_\alpha(o_\alpha, a; \theta_1), \hat{Q}_\beta(o_\beta, a; \theta_2) = Q_\beta(o_\beta, a; \theta_2)$. When testing, we experiment two settings, without adding noise to any Q-values of the testing samples (named **FRL-1**) and with adding noise to all Q-values of the testing samples (named **FRL-2**).

Implementation details of the above-mentioned baselines will be discussed in the following subsections.

5.2. Grid-World Domain

We first conduct experiments in a synthetic grid-world domain with randomly placed obstacles (Tamar et al., 2016). In the original grid-world domain, there is only a single agent which needs to navigate itself from a given start position to a given goal position without hitting obstacles, as shown in the left part of Figure 2. We modify the settings by replacing the start and goal positions with agents α and β, as shown in the right part of Figure 2. Therefore, in our task, two agents are initially in different positions and they need to navigate themselves to meet each other through the optimal path (i.e. the shortest path without hitting obstacles). Our task is more challenging than the original task, since the original task can be seen as a special case that agent β is still all the time and agent α is movable. In the following parts, we will first describe the experimental setup in this domain in detail. Then we present the main comparative results of all models. Finally, we conduct experiments for studying when `FRL` works.

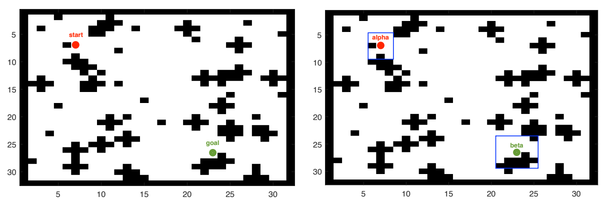

Figure 2: 32×32 sized grid world examples. The left part of the figure is an instance of the original task with a start and a goal position. The right part of the figure is an instance of our task with agent alpha and agent beta. The rectangles in blue are the boundaries of the observations.

5.2.1. Experimental Setup

In the experiment, we exploit the following settings:

State: Each full state s^t in this domain is a $N_g \times N_g$ binary-valued matrix (0 for obstacle, 1 otherwise), where N_g is the size of a grid world. We test $N_g = 8, 16, 32$ in our experiments, i.e. $8 \times 8, 16 \times 16, 32 \times 32$ grids. The observation o_α is a 3×3 grid with the current position in the center, and the observation o_β is a 5×5 grid with the current position in the center. Since the single 3×3 or 5×5 grid information may be insufficient for an agent to confirm its location and make a decision in a partially observed environment, i.e. there are many 3×3 or 5×5 grids which are the same, we adopt a sequence of observations to construct an observation, i.e. $o_\alpha^t = [o_\alpha^{t-H+1}, o_\alpha^{t-H+2}, \ldots, o_\alpha^t]$, where o_α^t is a 3×3 grid observation at time t of agent α and H is the length of history. We calculate the average length of optimal paths for each domains, and find out that each agent need to move 2.4 steps in 8×8 grid, 4.8 steps in 16×16 grid and 9.8 steps in 32×32 grid on average. Therefore, we empirically set $H = 2, 4, 8$ for $8 \times 8, 16 \times 16, 32 \times 32$ domain, respectively.

Action: In the grid-world domain, there are 4 possible actions for each agent, i.e. going towards 4 directions $\{east, south, west, north\}$.

Reward: Each agent will get a negative reward (-10) when it hits an obstacle, and get a positive reward (+50) when it meets the other agent, and get a small negative reward (-1) otherwise. The goal of the task is that two agents meet each other, so there is a distance reward $r_d^t = c/md(\alpha, \beta)$, where $md(\alpha, \beta)$ is the Manhattan distance and c is a regularization factor that equals to the dimension of state s^t, i.e. $c = N_g$. The instant reward at each step involves both agents, but only agent β can receive and take advantage of it.

Q-Network: We devise a CNN-based Q-Network, with a convolutional layer of 32 kernels of size 3×3 and paddings. The outputs of the convolutional layer are then flatten and fed to two fully connected layers with 256 and 4.

Baseline: DQN-alpha only uses the alpha Q-network. CNN-alpha has the same network structure as DQN-alpha, except that it is trained in the supervised way. DQN-full uses both networks, i.e. o_α and o_β will be input to two different convolutional layer with 3×3 sized kernels, then the outputs will be flatten, concatenated and input into the fully connected layers with size 256 and 4. CNN-full has the same network structure as DQN-full, except that it is trained in the supervised way. `FRL` federates the two networks according to Eq (1) - Eq(3), where the MLP module comprises two fully-connected layers with size 32 and 4. The noise probability p is set to 0.5, σ (the standard deviation of Gaussian noise) is set to 1. For all models, we adopt the Adam optimizer with learning rate 0.001 and the ReLU activation.

Evaluation: We generate 8000 different maps for each domain of $8 \times 8, 16 \times 16$ and 32×32 sized grids. In each map, we randomly choose two positions for the two agents, and then compute the optimal path (shortest path) which will be compared with the path that predicted by our model and baselines. For evaluation, we randomly split the 8000 maps to 6400 (training), 800 (validation) and 800 (testing). In each testing episode, we first input the initial state to each model to predict an action. We then get a new state and take it as the input at the second time step. We repeat the procedure until the two agents meet each other, *named a successful episode*, or it exceed the maximum time step T_m, *named a failure episode*. We set T_m to be double the length of the longest optimal path in training data, i.e. $T_m = 38, 86, 178$ for $8 \times 8, 16 \times 16$ and 32×32 domains, respectively. We finally compute $SuccRate = \frac{\#SuccessfulEpisodes}{\#TotalEpisodes}$, $AvgRwd = \frac{TotalCumulativeReward}{\#Episodes}$ and $TrajDiff = \frac{|LengthOfPredictedPaths - LengthOfShortestPaths|}{LengthOfShortestPaths}$, where $\#SuccessfulEpisodes$ indicates the number of successful episodes, $\#TotalEpisodes$ indicates the number of testing episodes, $TotalCumulativeReward$ indicates the total cumulative reward of all testing episodes, $LengthOfPredictedPaths$ indicates the total number of steps of predicted paths (a path is a sequence of actions which are predicted by a model), $LengthOfShortestPaths$ indicates the total number of steps of optimal paths.

Table 1: Comparative results on grid-world, where *Succ. rate* indicates the success rate (the larger the better), *Avg. rwd.* indicates the average cumulative reward of each episodes (the larger the better) and *Traj. diff.* indicates the trajectory difference between the predicted paths and the optimal paths (the smaller the better).

Metric	Method	Domain		
		8×8	16×16	32×32
Succ. rate	CNN-alpha	69.73%	48.04%	41.73%
	DQN-alpha	88.27%	76.20%	71.41%
	FRL-1	92.52%	79.83%	77.88%
	FRL-2	**95.06%**	**84.31%**	**82.02%**
	CNN-full	72.16%	56.44%	50.15%
	DQN-full	93.69%	83.40%	79.73%
Traj. diff.	CNN-alpha	4.036	9.343	11.313
	DQN-alpha	**0.378**	**0.974**	**2.046**
	FRL-1	0.579	1.513	2.434
	FRL-2	0.581	1.964	2.868
	CNN-full	3.935	8.020	9.277
	DQN-full	0.241	1.348	2.568
Avg. rwd.	DQN-alpha	13.781	-112.084	-285.946
	FRL-1	18.152	-94.193	-226.583
	FRL-2	**19.101**	**-84.139**	**-189.756**
	DQN-full	31.286	-38.114	-52.72

5.2.2. Main Results

In this part, we would like to consider two questions: (1) how much can the `FRL` model excel the models which take as input only single part of observation, and (2) how well can the `FRL` model leverage the high level information of basic Q-networks without coming to the original observations. Table 1 shows the comparative results, where each item is an average of 5 independent runs. For the first question, we can see that, under the *success rate* metric, both FRL-1 and FRL-2 significantly outperform the baselines DQN-alpha and CNN-alpha, with a large margin around $5\% \sim 20\%$ absolutely. In this domain, DQN-based models perform better than CNN-based classification models. When considering the *trajectory difference* metric, our `FRL` models outperform both CNN-alpha and CNN-full. Compared to DQN-alpha, `FRL` models induce a bit increment of trajectory difference in all domains. Meanwhile, DQN-full also induces the trajectory difference in 16×16 and 32×32 domains. Because these models try more steps to success and to get more reward in the long run, which can be demonstrated by the results of *average cumulative reward* (CNN-alpha and CNN-full do not involve in this metric). We can see that our `FRL` models gain more reward than DQN-alpha. As for the second question, although the CNN-full directly takes advantage of both o_α and o_β, it performs worse than our `FRL`, and worse even than the DQN-alpha model. Most importantly, both our FRL-1 and FRL-2 models perform very closely to the DQN-full model under the success rate metric, and underperforms only a little bit when considering the trajectory difference and the average reward, which indicates that our approach can make good use of the encrypted information of agents.

Regarding the different settings of Gaussian mechanism, the FRL-1 model performs more successful episodes and gets more reward than the FRL-2 model, while it induces a little bit more trajectory difference. The result shows that our `FRL` model is more suitable to Q-values with noise than original Q-values, which means that our approach can protect data privacy better.

5.2.3. Study of Input Information

To study when `FRL` approach works, we consider the amount of information that an agent uses. As we mentioned before, the observation input at each time is a sequence of observations, i.e. observation history. Intuitively, the longer the length of the observation history, the more information it contains, and the more complicated neural networks are needed. In this part, we fixed the structures of all models and only change the length of the history. We test the length of history from 2 to 32 for all domains, and the results are presented in Figure 3. In the first line of Figure 3, the first phenomenon we can observe is that the success rates are improving with the increment of the history length. In 8×8 and 16×16 domains, the results converge when history length is longer than 16, while in 32×32 domain, they have not converged even at the history length of 32. The reason is that 32×32 domain is more complicated than the other two domains, so it needs more information (i.e. $H > 32$) to learn a model. We can also find that when the history length is short (i.e. $H = 2$ for 16×16 and $H \leq 4$ for 32×32), models DQN-alpha, DQN-full and FRL-1 perform poorly. The FRL-1 and DQN-full approaches do not show their advantages although they take as input both o_α and o_β directly or indirectly. However, FRL-2, which applies differential privacy to both training and testing samples, shows its great scalability even with limited amount of history. In naive domains, such as 8×8, dozens of steps are enough to explore the whole environment. Therefore, the DQN-alpha which takes only single observation as input can perform as well as DQN-alpha and `FRL` models. We think that it may not be necessary to apply federated learning when the problems are too simple.

5.3. Text2Actions Domain

We now consider Text2Action domain (Feng et al., 2018) which aims to extract action sequences from texts, i.e., words that are related to actions should be selected, and other words should be neglected. For example, consider a text of action description, "*Cook the rice the day before, or use leftover rice in the refrigerator. The important thing to remember is not to heat up the rice, but keep it cold.*", which addresses the procedure of making egg fired rice. The task of the agent is to select the words that make up an action sequence "cook(rice), keep(rice, cold), or "use(leftover rice), keep(rice, cold). Figure 4 shows the overall process of the task extracting action sequences. We assume that there are two agents, α and β, where agent α has a POS-tagger that can generate the part of speech of each word from a text, agent β has a rich word embedding table (i.e. a word2vec model trained by a sufficiently large corpus). In the following parts, we will first describe the experimental setup in this domain in detail. Then we present the main comparative results of all models. Finally, we conduct experiments for studying when `FRL` works.

5.3.1. Experimental Setup

In this experiment, we exploit the following settings:

State: $o_\alpha \in \mathbb{R}^{N_w \times K_1}$ is real-valued matrix that describes the part-of-speech of words, and $o_\beta \in \mathbb{R}^{N_w \times K_2}$ is real-valued matrix that describes the embedding of words. N_w is the number of words in a text, K_1 is the dimension of a part-of-speech vector, and K_2 is the dimension of a word vector. In our experiments, part-of-speech vectors are randomly initialized and trained together with the Q-network, word vectors are generated from the pre-trained word embedding and will not be changed during the training of the Q-network.

Action: At each time step, these two agent need to cooperate to select a word or neglect a word. Therefore, they execute the same action at each time step and the action space $A = \{select, neglect\}$.

Reward: The instant reward include a basic reward and an additional reward, where the basic reward indicates whether the agent select a word correctly or not, and the additional reward encodes the priori knowledge of this domain, i.e. the proportion of words that are related to action sequences. More details can be found in (Feng et al., 2018).

Q-network: We adopt the same TextCNN structure as (Feng et al., 2018). Four convolutional layers corresponding to bigram, tri-gram, four-gram and five-gram with 32 kernels of size $n \times m$, where n refers to the n-gram and $m = K_1$ or K_2. We set $K_1 = K_2 = 50$ in our experiments. Each convolutional layer is followed by a max-pooling layer with size of $(N_w - n + 1, 1)$, where $N_w - n + 1$ is the first dimension of the outputs of the n-gram convolutional layer. The max-pooling outputs are concatenated and fed to two fully connected layers with size 128 ($128 = 4 \times 32$, 4 kinds of n-grams, each has 32 kernels) and 2 (the size of the action space). The MLP module of `FRL` approach comprises two fully-connected layers with

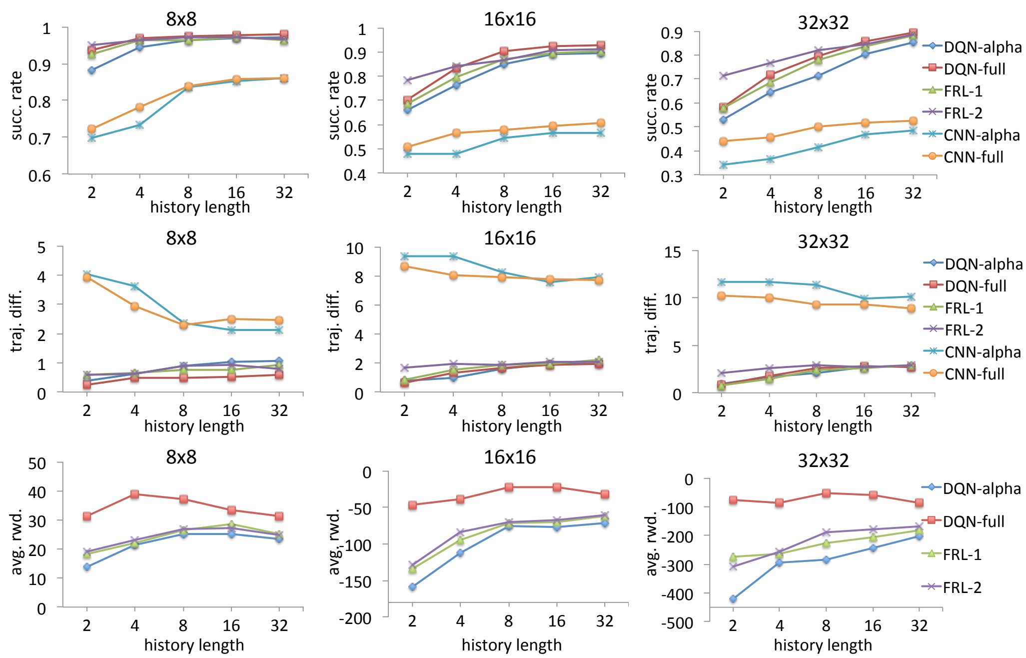

Figure 3: Results about the impact of history length.

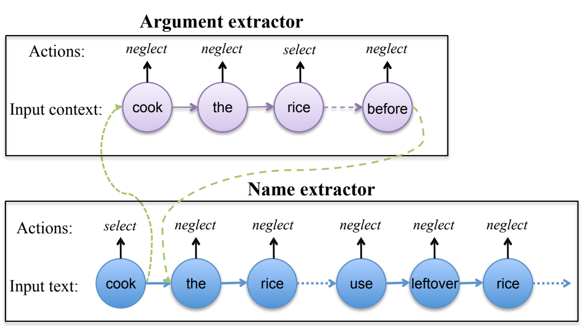

Figure 4: Illustration of the overall process of the Text2Action task. Name extractor is a module for extracting the words of action names from a given text. Argument extractor is a module for extracting the words of action arguments from a given action name and its context.

size 4 and 2, and the noise probability p is set to 0.5, σ (the standard deviation of Gaussian noise) is set to 1. For all models, we adopt the Adam optimizer with learning rate 0.001 and the ReLU activation.

Evalutaion: Following (Feng et al., 2018), we adopt ten-fold cross validation for all the three datasets, WHS, CT and WHG. For evaluation, we first feed texts to each model to obtain selected words. We then compare the outputs to their corresponding ground truth and calculate $\#TotalTruth$ (total ground truth words), $\#TotalRight$ (total correctly selected words), $TotalSelected$ (total selected words). We finally compute $precision = \frac{\#TotalRight}{\#TotalSelected}$, $recall = \frac{\#TotalRight}{\#TotalTruth}$ and $F1 = \frac{2 \times precision \times recall}{precision + recall}$. We use F1-metric for all baselines. For reinforcement learning methods, we also compute the average cumulative rewards $AvgRwd = \frac{TotalCumulativeReward}{\#TotalTimeSteps}$, where $TotalCumulativeReward$ indicates the total cumulative reward of all testing texts and $\#TotalTimeSteps$ indicates the total number of steps of all testing texts.

5.3.2. Main Results

We run 5 independent runs of all experiments and the comparative results are presented in Table 2. It shows that our approach excels the baselines DQN-alpha, CNN-alpha and CNN-full in all three datasets under both F1 score and average reward metrics. Texts of the WHS dataset are quite short and most verbs of the texts are exactly the words of actions to be selected, which makes the Text2Action task easy to solve, i.e. WHS is a naive domain. Therefore, all baselines perform well in WHS, and our `FRL` only outperforms the DQN-alpha, CNN-full and CNN-alpha by around $2\% \sim 8\%$. When processing more complicated texts which contain much more redundant verbs and sentences, e.g. texts in CT and WHG datasets, the performance of our `FRL` model is dominant, improving the F1 score around $10\% \sim 20\%$ absolutely compared to the baselines. Under all metrics, our `FRL` performs as well as DQN-full. Generally speaking, FRL-2 excels FRL-1, but the advantage is not so obvious.

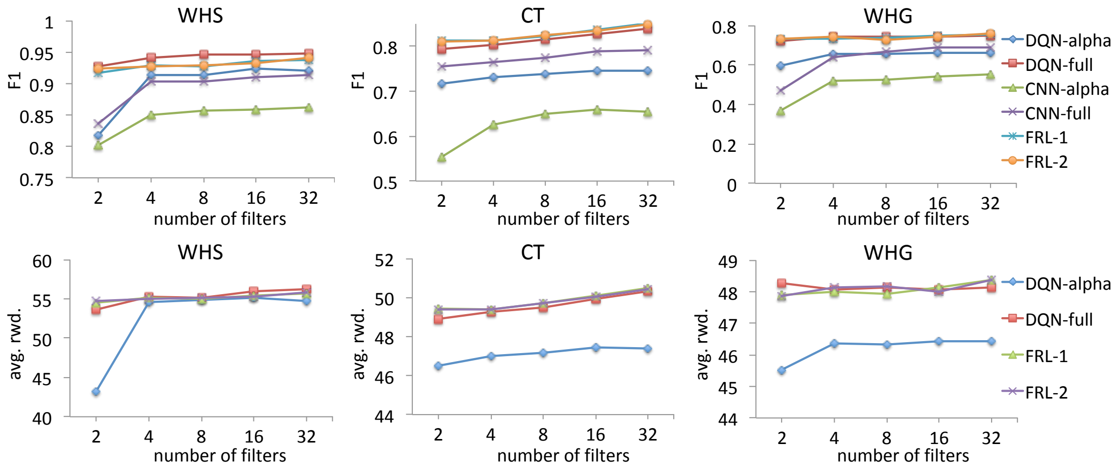

Figure 5: Results about the impact of the number of convolutional kernels.

Table 2: Comparative results of our FRL model and all baselines in Text2Action domain.

Metric	Method	Dataset		
		WHS	CT	WHG
F1 score	CNN-alpha	86.19%	65.44%	55.18%
	DQN-alpha	92.11%	74.64%	66.37%
	FRL-1	93.76%	**85.05%**	75.64%
	FRL-2	**94.41%**	84.92%	**75.85%**
	CNN-full	91.42%	79.03%	68.93%
	DQN-full	94.55%	83.39%	74.63%
Avg. rwd.	DQN-alpha	54.762	47.375	46.510
	FRL-1	55.623	**50.472**	48.359
	FRL-2	**55.894**	50.452	**48.373**
	DQN-full	56.192	50.307	48.154

5.3.3. Study of Model Complexity

To study when FRL approach works, we study the impact of the model complexity. More concretely, we change the number of convolutional kernels from 2 to 32 (the last but one fully connected layer will change from $8 = 4 \times 2$ to $128 = 4 \times 32$, respectively) and fixed all other parameters. The comparative results are shown in Figure 5. We can see that all models perform better with more convolutional kernels. Our FRL models outperform DQN-alpha, CNN-alpha and CNN-full. Both FRL-1 and FRL-2 are competitive with DQN-full.

6. Conclusion

In this paper, we propose a federated method for reinforcement learning, namely FRL. We demonstrate that the proposed FRL approach is capable of making full use of the joint observations (or states). The FRL model outperforms all the baselines with partial observation inputs and performs closely to the baselines that directly take all joint information as input. We also study when FRL works. Our experiments indicate that FRL works well in common or complicated domains rather than naive ones.

References

Abadi, M., Chu, A., Goodfellow, I. J., McMahan, H. B., Mironov, I., Talwar, K., and Zhang, L. Deep learning with differential privacy. In *ACM SIGSAC*, pp. 308–318, 2016.

Agarwal, N., Suresh, A. T., Yu, F. X., Kumar, S., and McMahan, B. cpsgd: Communication-efficient and differentially-private distributed SGD. In *NeurIPS*, pp. 7575–7586, 2018.

Co-Reyes, J., Liu, Y., Gupta, A., Eysenbach, B., Abbeel, P., and Levine, S. Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 1009–1018, 2018.

Dwork, C. and Roth, A. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

Feng, W., Zhuo, H. H., and Kambhampati, S. Extracting action sequences from texts based on deep reinforcement learning. In *IJCAI*, pp. 4064–4070, 2018.

Konecný, J., McMahan, B., and Ramage, D. Federated optimization: Distributed optimization beyond the datacenter. *CoRR*, abs/1511.03575, 2015. URL `http://arxiv.org/abs/1511.03575`.

Konecný, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., and Bacon, D. Federated learning: Strategies for improving communication efficiency. *CoRR*, abs/1610.05492, 2016.

Leibo, J. Z., Zambaldi, V. F., Lanctot, M., Marecki, J., and Graepel, T. Multi-agent reinforcement learning in sequential social dilemmas. In *AAMAS*, pp. 464–473, 2017.

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., and Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. In *NIPS*, pp. 6382–6393, 2017.

McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, pp. 1273–1282, 2017.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., and Ostrovski, G. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529–33, 2015.

Omidshafiei, S., Pazis, J., Amato, C., How, J. P., and Vian, J. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *ICML*, pp. 2681–2690, 2017.

Rashid, T., Samvelyan, M., de Witt, C. S., Farquhar, G., Foerster, J. N., and Whiteson, S. QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning. In *ICML*, pp. 4292–4301, 2018.

Smith, V., Chiang, C., Sanjabi, M., and Talwalkar, A. S. Federated multi-task learning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 4427–4437, 2017.

Sutton, R. S. and Barto, A. G. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998. ISBN 0262193981.

Tamar, A., Levine, S., Abbeel, P., Wu, Y., and Thomas, G. Value iteration networks. In *NIPS*, pp. 2146–2154, 2016.

Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J., and Vicente, R. Multiagent cooperation and competition with deep reinforcement learning. *CoRR*, abs/1511.08779, 2015.

Taylor, M. E. and Stone, P. Transfer learning for reinforcement learning domains: A survey. *J. Mach. Learn. Res.*, 10:1633–1685, December 2009. ISSN 1532-4435.

Tirinzoni, A., Rodriguez Sanchez, R., and Restelli, M. Transfer of value functions via variational methods. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 6182–6192. Curran Associates, Inc., 2018.