

---

# Federated Deep Reinforcement Learning

---

Hankz Hankui Zhuo<sup>1</sup> Wenfeng Feng<sup>1</sup> Yufeng Lin<sup>1</sup> Qian Xu<sup>2</sup> Qiang Yang<sup>2</sup>

## Abstract

In deep reinforcement learning, building policies of high-quality is challenging when the feature space of states is small and the training data is limited. Despite the success of previous transfer learning approaches in deep reinforcement learning, directly transferring data or models from an agent to another agent is often not allowed due to the privacy of data and/or models in many privacy-aware applications. In this paper, we propose a novel deep reinforcement learning framework to federatively build models of high-quality for agents with consideration of their privacies, namely Federated deep Reinforcement Learning (FedRL). To protect the privacy of data and models, we exploit Gaussian differentials on the information shared with each other when updating their local models. In the experiment, we evaluate our FedRL framework in two diverse domains, Grid-world and Text2Action domains, by comparing to various baselines.

## 1. Introduction

In deep reinforcement learning, building policies of high-quality is challenging when the feature space of states is small and the training data is limited. In many real-world applications, however, datasets from clients are often privacy sensitive (Duchi et al., 2012) and it is often difficult for such a data center to guarantee building models of high-quality. To deal with the issue, Konecny et al. propose 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 setting (c.f. (Yang et al., 2019)), 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 deep Reinforcement Learning* (FedRL), which aims to learn a private Q-network policy for each agent by sharing limited information (i.e., output of the Q-network) among agents. The information is “encoded” when it is sent to others and “decoded” when it is received by others. We assume that some agents have *rewards* corresponding to states and actions, while others have only observed states without *rewards*. Without rewards, those agents are unable to build decision policies on their own information. 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 the 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 on their own is often difficult due to their limited businesses and lack of rewards (for some factories). It is thus helpful for them to learn decision policies 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 FedRL 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 (or each agent receives an individual reward but shares it with other agents) (Tampuu et al., 2015; Leibo et al., 2017; Foerster et al., 2016). FedRL assumes agents do not share their partial observations and some agents are unable to receive rewards. Our FedRL framework is also different from transfer learning in reinforcement learning, which aims to transfer experience gained in learning to perform one task to help

---

<sup>1</sup>zhuohank@mail.sysu.edu.cn, fengwf2014@outlook.com, linyf23@mail2.sysu.edu.cn, Sun Yat-Sen University, Guangzhou, China; <sup>2</sup>{qianxu,qiangyang}@webank.com, WeBank, Shenzhen, China. Correspondence to: Hankz Hankui Zhuo <zhuo-hank@mail.sysu.edu.cn>.

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 FedRL assumes states cannot be shared among agents.

Our FedRL framework functions in three phases. Initially, each agent collects output values of Q-networks from other agents, which are “encrypted” with Gaussian differentials. Furthermore, it builds a shared value 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 the shared value network 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 FedRL framework, and then present the problem formulation of FedRL. After that, we introduce our FedRL framework in detail. Finally we evaluate our FedRL 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 (Konečný 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 (Konečný 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 FedRL 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, and  $A$  is the action space.  $T$  is the 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  $\pi : S \rightarrow A$ , the value function  $V^\pi(s)$  and the Q-function  $Q^\pi(s, a)$  at step  $t + 1$ , can be updated by their  $t$  step:

$$V_{t+1}^\pi(s) = r(s) + \sum_{s' \in S} T(s, \pi(s), s') V_t^\pi(s'),$$

and

$$Q_{t+1}^\pi(s, a) = r(s) + \sum_{s' \in S} T(s, a, s') V_t^\pi(s'),$$

for  $t \in \{0, \dots, K - 1\}$ . The solution to an MDP problem is the best policy  $\pi^*$  such that  $V^{\pi^*}(s) = \max_\pi V^\pi(s)$  or  $Q^{\pi^*}(s, \pi^*(s)) = \max_\pi Q^\pi(s, \pi(s))$ . In DQN (Deep Q-Network), given transition function  $T$  is unknown, the Q-function is represented by a Q-network  $Q(s, a; \theta)$  with  $\theta$  as parameters of the network, and updated by

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

as done by (Mnih et al., 2015). To learn the parameters  $\theta$ , one way is to store transitions  $\langle s, a, s', r \rangle$  in replay memories  $\Omega$  and exploit a mini-batch sampling to repeatedly update  $\theta$  (Mnih et al., 2015). Once  $\theta$  is learnt, the policy  $\pi^*$  can be extracted from  $Q(s, a; \theta)$ ,

$$\pi^*(s) = \arg \max_{a \in A} Q(s, a; \theta).$$

We define our *federated* deep reinforcement learning problem by: *given transitions*  $\mathcal{D}_\alpha = \{\langle s_\alpha, a_\alpha, s'_\alpha, r_\alpha \rangle\}$  *collected by agent*  $\alpha$ , *and pairs of states and actions*  $\mathcal{D}_\beta =$

$\{s_\beta, a_\beta\}$  collected by agent  $\beta$ , **we aim to federatively build policies  $\pi_\alpha^*$  and  $\pi_\beta^*$  for agents  $\alpha$  and  $\beta$ , respectively.** Note that in this paper we consider the federation with two members for simplicity. The setting can be extended to many agents by exploiting the same federated mechanism **between each two agents**. We denote states, actions, Q-functions, and policies with respect to agents  $\alpha$  and  $\beta$ , by “ $s_\alpha \in S_\alpha$ ,  $a_\alpha \in A_\alpha$ ,  $Q_\alpha$ ,  $\pi_\alpha^*$ ” and “ $s_\beta \in S_\beta$ ,  $a_\beta \in A_\beta$ ,  $Q_\beta$ ,  $\pi_\beta^*$ ”, respectively.

In our federated deep reinforcement learning problem, we assume:

**A1:** The feature spaces of states  $s_\alpha$  and  $s_\beta$  are *different* between agents  $\alpha$  and  $\beta$ . For example, a state  $s_\alpha$  denotes a patient’s cardiogram in hospital  $\alpha$ , while another state  $s_\beta$  denotes the same patient’s electroencephalogram in hospital  $\beta$ , indicating the feature spaces of  $s_\alpha$  and  $s_\beta$  are different.

**A2:** Transitions  $\mathcal{D}_\alpha$  and  $\mathcal{D}_\beta$  cannot be shared directly between  $\alpha$  and  $\beta$  when they learning their own models. The *correspondences* between transitions from  $\mathcal{D}_\alpha$  and  $\mathcal{D}_\beta$  are, however, known to each other. In other words, agent  $\alpha$  can send the “ID” of a transition to agent  $\beta$ , and agent  $\beta$  can use that “ID” to find its corresponding transition in  $\mathcal{D}_\beta$ . For example, in hospital, “ID” can correspond to a specific patient.

**A3:** The output of functions  $Q_\alpha$  and  $Q_\beta$  *can* be shared with each other under the condition that they are protected by some privacy protection mechanism.

Based on **A1-A3**, we aim to learn policies  $\pi_\alpha^*$  and  $\pi_\beta^*$  of high-quality for agents  $\alpha$  and  $\beta$  by preserving privacies of their own data and models.

## 4. Our FedRL Approach

In this section we present our FedRL framework in detail. An overview of FedRL is shown in Figure 1, where the left part is the model of agent  $\alpha$ , and the right part is the model of agent  $\beta$ . Each model is composed of a local Q-network with parameters  $\theta_\alpha$  for agent  $\alpha$ ,  $\theta_\beta$  for agent  $\beta$ , and a global MLP module with parameters  $\theta_g$  for both  $\alpha$  and  $\beta$ .

**Basic Q-networks** We build two Q-networks for agents  $\alpha$  and  $\beta$ , denoted by  $Q_\alpha(s_\alpha, a_\alpha; \theta_\alpha)$  and  $Q_\beta(s_\beta, a_\beta; \theta_\beta)$ , respectively, where  $\theta_\alpha$  and  $\theta_g$  are parameters of the Q-networks. **The outputs of these two basic Q-networks are not directly used to predict the actions, but taken as input of the MLP module.**

**Gaussian differential privacy** To avoid agents “inducing” models of each other according to repeatedly received outputs of other’s Q-network during training, we consider using differential privacy (Dwork & Roth, 2014) to pro-

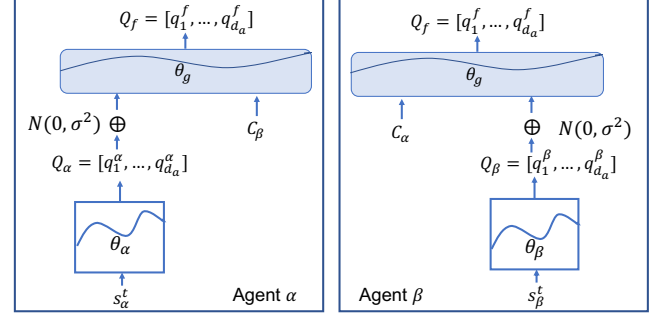


Figure 1. The networks of agents  $\alpha$  and  $\beta$

tect the output of agent’s Q-network. There are various mechanisms of differential privacy such as the Gaussian mechanism (Abadi et al., 2016) and Binomial mechanism (Agarwal et al., 2018). In this paper, we exploit Gaussian mechanism since the output of the MLP with Gaussian input is Gaussian itself. In previous federated learning settings, the mechanism is applied to the gradients of agents (clients) before being sent to the server or other agents (clients). In our FedRL framework, **we send the output of one agent’s Q-network to another, so the Gaussian noise is added to the output rather than the gradients.** The mechanism is defined by

$$\hat{Q}_\alpha(s_\alpha, a_\alpha; \theta_\alpha) = Q_\alpha(s_\alpha, a_\alpha; \theta_\alpha) + N(0, \sigma^2) \quad (1)$$

$$\hat{Q}_\beta(s_\beta, a_\beta; \theta_\beta) = Q_\beta(s_\beta, a_\beta; \theta_\beta) + N(0, \sigma^2) \quad (2)$$

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

**Federated Q-network** We build a new Q-network, namely **federated Q-network** denoted by  $Q_f$ , to leverage the outputs of the two basic Q-networks with Gaussian noise based on MLP, which is defined by

$$Q_f(\cdot; \theta_\alpha, \theta_\beta, \theta_g) = \text{MLP}([\hat{Q}_\alpha(s_\alpha, a_\alpha; \theta_\alpha) | \hat{Q}_\beta(s_\beta, a_\beta; \theta_\beta)]; \theta_g) \quad (3)$$

where  $\theta_g$  is the parameters of MLP and  $[\cdot | \cdot]$  indicates the concatenation operation. Note that parameters of an MLP can be shared between agents. Once MLP is updated, the updated parameters are shared with the other agent.

With respect to agents  $\alpha$  and  $\beta$ , we define each agent’s federated Q-networks by viewing the other agent’s basic Q-network (with Gaussian noise) as a fixed constant when updating its own basic Q-network, as shown below,

$$Q_f^\alpha(\cdot, C_\beta; \theta_\alpha, \theta_g) = \text{MLP}([\hat{Q}_\alpha(\cdot; \theta_\alpha) | C_\beta]; \theta_g) \quad (4)$$

$$Q_f^\beta(\cdot, C_\alpha; \theta_\beta, \theta_g) = \text{MLP}([C_\alpha | \hat{Q}_\beta(\cdot; \theta_\beta)]; \theta_g) \quad (5)$$

where  $C_\alpha = \hat{Q}_\alpha(s_\alpha, a_\alpha; \theta_\alpha)$  and  $C_\beta = \hat{Q}_\beta(s_\beta, a_\beta; \theta_\beta)$  are fixed constants when updating agent  $\beta$ ’s basic Q-network and agent  $\alpha$ ’s basic Q-network, respectively.

The Q-networks are trained by minimizing the square error loss  $L_\alpha^j(\theta_\alpha, \theta_g)$  and  $L_\beta^j(\theta_\beta, \theta_g)$  of agents  $\alpha$  and  $\beta$ ,

$$L_\alpha^j(\theta_\alpha, \theta_g) = \mathbb{E} \left[ (Y^j - Q_f^\alpha(s_\alpha^j, a_\alpha^j, C_\beta; \theta_\alpha, \theta_g))^2 \right] \quad (6)$$

$$L_\beta^j(\theta_\beta, \theta_g) = \mathbb{E} \left[ (Y^j - Q_f^\beta(s_\beta^j, a_\beta^j, C_\alpha; \theta_\beta, \theta_g))^2 \right] \quad (7)$$

where  $Y^j = r^j + \gamma \max_a Q_f^\alpha(s_\alpha^j, a, C_\beta; \theta_\alpha, \theta_g)$ . Note that agent  $\beta$  is unable to compute  $Y^j$  since it does not have reward  $r$ .  $Y^j$  is computed by agent  $\alpha$  and shared with agent  $\beta$ .

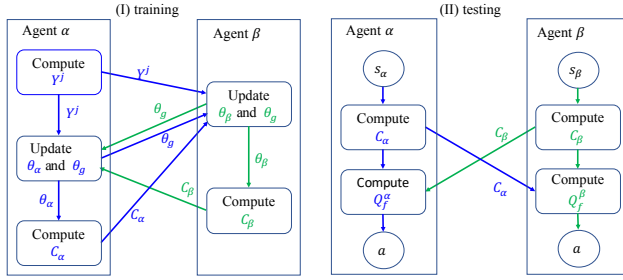


Figure 2. The training and testing procedures

**Overview of training and testing** The training and testing procedures of agents  $\alpha$  and  $\beta$  are shown in Figure 2. When training, agent  $\alpha$  computes  $Y^j$  and sends it to agent  $\beta$ . Agent  $\beta$  updates  $\theta_\beta$  and  $\theta_g$ , computes  $C_\beta$ , and sends  $\theta_g$  and  $C_\beta$  to agent  $\alpha$ . After that, agent  $\alpha$  updates  $\theta_\alpha$  and  $\theta_g$ , computes  $C_\alpha$ , and sends  $\theta_g$  and  $C_\alpha$  to agent  $\beta$ . When testing, both agents compute and send  $C_\alpha$  and  $C_\beta$  to each other for computing  $Q_f^\alpha$  and  $Q_f^\beta$ .

The detailed training procedure can be seen from Algorithms 1 and 2. In Steps 1 and 2 of Algorithm 1, we initialize the basic Q-network and replay memory of agent  $\alpha$  and the MLP module. In Step 3 of Algorithm 1, we call the function from Algorithm 2 to initialize the Q-network and replay memory of agent  $\beta$ . In Step 6 of Algorithm 1, we obtain observation of agent  $\alpha$ 's state  $s_\alpha^t$ . In Step 7 of Algorithm 1, we call the function in Algorithm 2 to calculate the output of basic Q-network of agent  $\beta$  and obtain observation of agent  $\beta$  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  $\epsilon$ -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  $\beta$  based on the index  $j$ . In Steps 14 and 15 of Algorithm 1, we update parameters  $\theta_\alpha$  and  $\theta_g$ . In Steps 16 and 17 of Algorithm 1, we compute the output of basic Q-network of agent  $\alpha$  and pass it to agent  $\beta$  and call

the function  $UpdateQ$  of agent  $\beta$  to update basic Q-network of agent  $\beta$  and MLP, as shown in Steps from 19 to 21 of Algorithm 2. Note that Algorithms 1 and 2 are executed by agents  $\alpha$  and  $\beta$  separately.

---

#### Algorithm 1 FedRL-ALPHA

---

**Input:** state space  $S_\alpha$ , action space  $A_\alpha$ , rewards  $r$

**Output:**  $\theta_\alpha, \theta_g$

```

1: Initialize  $Q_\alpha, Q_f$  with random values for  $\theta_\alpha, \theta_g$ 
2: Initialize replay memory  $D_\alpha$ 
3: Call FedRL-BETA.Init()
4: for episode = 1:  $M$  do
5:   repeat
6:     Observe  $s_\alpha^t$ 
7:     Call  $C_\beta = \text{FedRL-BETA}.ComputeQBeta()$ 
8:     Select action  $a^t$  with probability  $\epsilon$ 
9:     Otherwise  $a^t = \arg \max_a Q_f^\alpha(s_\alpha^t, a, C_\beta; \theta_\alpha, \theta_g)$ 
10:    Execute action  $a^t$ , obtain reward  $r^t$  and state  $s^{t+1}$ 
11:    Observe  $s_\alpha^{t+1}$ , store  $(s_\alpha^t, a^t, r^t, s_\alpha^{t+1})$  in  $D_\alpha$ 
12:    Sample  $(s_\alpha^j, a_\alpha^j, r^j, s_\alpha^{j+1})$  from  $D_\alpha$ 
13:    Call  $C_\beta = \text{FedRL-BETA}.ComputeQBeta(j)$ 
14:     $Y^j = r^j + \gamma \max_a Q_f^\alpha(s_\alpha^j, a, C_\beta; \theta_\alpha, \theta_g)$ 
15:    Update  $\theta_\alpha, \theta_g$  according to Eq. (4), (6)
16:     $C_\alpha = \hat{Q}_\alpha(s_\alpha^j, a; \theta_\alpha)$ 
17:    Call  $\theta_g = \text{FedRL-BETA}.UpdateQ(Y^j, j, C_\alpha, \theta_g)$ 
18:  until terminal  $t$ 
19: end for
```

---



---

#### Algorithm 2 FedRL-BETA

---

**Input:** state space  $S_\beta$ , action space  $A_\beta$

**Output:**  $\theta_\beta, \theta_g$

```

1: function Init()
2:   Initialize  $Q_\beta$  with random values for  $\theta_\beta$ 
3:   Initialize replay memory  $D_\beta$ 
4: end function
5: function ComputeQBeta()
6:   Observe  $s_\beta$ 
7:   Select  $a_\beta \in A_\beta$  with probability  $\epsilon$ 
8:   Otherwise  $a_\beta = \arg \max_{a_\beta} Q_\beta(s_\beta, a_\beta; \theta_\beta)$ 
9:   Store  $(s_\beta, a_\beta)$  in  $D_\beta$ 
10:  Let  $C_\beta = \hat{Q}_\beta(s_\beta, a; \theta_\beta)$ 
11:  return  $C_\beta$ 
12: end function
13: function ComputeQBeta(j)
14:  Select  $(s_\beta, a_\beta)$  from  $D_\beta$  based on index  $j$ 
15:  Let  $C_\beta = \hat{Q}_\beta(s_\beta, a_\beta; \theta_\beta)$ 
16:  return  $C_\beta$ 
17: end function
18: function UpdateQ( $Y^j, j, C_\alpha, \theta_g$ )
19:  Select  $(s_\beta^j, a_\beta^j)$  from  $D_\beta$  based on index  $j$ 
20:  Update  $\theta_\beta, \theta_g$  based on Eq. (5), (7)
21:  return  $\theta_g$ 
22: end function
```

---



## 5. Experiments

In the experiment, we evaluate FedRL<sup>1</sup> in the following two aspects. Firstly, we would like to see if agent  $\alpha$  can learn better policies by joining the federation than without joining the federation (agent  $\alpha$  can build policies by itself since it has complete transitions  $\{(s, a, s', r)\}$ , while agent  $\beta$  cannot build policies without joining the federation since it has only pairs of states and actions  $\{(s, a)\}$ ). Secondly, we would like to see if our FedRL approach can learn policies of high-quality which are close to the ones learnt by directly combining data of both agent  $\alpha$  and  $\beta$ , neglecting the data-privacy issue between  $\alpha$  and  $\beta$ . To do this, we compared our FedRL approach to the following baselines:

- **DQN-alpha:** which is a deep Q-network (Mnih et al., 2015) trained with agent  $\alpha$ 's data only. It takes observations  $s_\alpha$  as input and outputs actions corresponding to  $s_\alpha$ .
- **DQN-full:** which is a deep Q-network trained by directly putting data together from both agents  $\alpha$  and  $\beta$ , i.e., neglecting data privacy between agents  $\alpha$  and  $\beta$ .
- **FCN-alpha:** which is a fully convolutional network (FCN) trained with agent  $\alpha$ 's data only, similar to DQN-alpha. FCN-alpha aims to build policies via supervised learning (Furuta et al., 2019), i.e., viewing states as input and actions as labels.
- **FCN-full:** which is a convolutional network trained with all data of agent  $\alpha$  and  $\beta$  put together directly, similar to DQN-full.

In our experiment, we used the kernel of size  $3 \times 3$  in CNN and two fully-connected layers with the size of  $32 \times 4$  in MLP. We set the standard deviation in Gaussian differential privacy  $\sigma$  to be 1. We adopted the Adam optimizer with learning rate 0.001 and the ReLU activation for all models. We evaluated our FedRL with comparison to baselines in two domains, i.e., Grid-World and Text2Action.

### 5.1. Evaluation in Grid-World Domain

We first conducted experiments in Grid-World domain with randomly placed obstacles (Tamar et al., 2016). The two agents  $\alpha$  and  $\beta$  were randomly put in the environment as well, as shown in Figure 3. The two agents aim to navigate through optimal paths (i.e. the shortest paths without hitting obstacles) to meet with each other. In this domain, we define states, actions and rewards as follows.

- **States:** The domain is represented by a  $N_g \times N_g$  binary-valued matrix (0 for obstacle, 1 otherwise), where  $N_g$

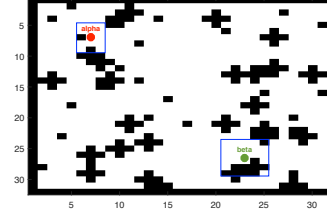


Figure 3. A  $32 \times 32$  grid-world domain with agents  $\alpha$  and  $\beta$ . The rectangles in blue are boundaries of observations.

is the size of the domain. We evaluated our approach with respect to different sizes, i.e.,  $N_g = 8, 16, 32$ . The observed state of agent  $\alpha$ , denoted by  $s_\alpha$ , was set to be a  $3 \times 3$  matrix with the current position of agent  $\alpha$  in the center of the matrix. The observed state of agent  $\beta$ , denoted by  $s_\beta$ , was set to be a  $5 \times 5$  matrix with the current position of agent  $\beta$  in the center of the matrix.

- **Actions:** There are 4 actions for each agent, i.e. going towards 4 directions, denoted by  $\{east, south, west, north\}$ .
- **Rewards:** The reward is composed of two parts, i.e., local reward  $r_l$  and global reward  $r_g$ , respectively. When an agent hits an obstacle,  $r_l$  is set to be -10; when an agent meets the other agent,  $r_l$  is set to be +50; otherwise,  $r_l$  is set to be -1. Considering the goal of the task is to make the two agents eventually meet each other, we exploited an additional reward regarding the distance between two agents, namely global reward, i.e.,  $r_g = c/md(\alpha, \beta)$ , where  $md(\alpha, \beta)$  is the Manhattan distance between the two agents, and  $c$  is a regularization factor which is set to be the dimension of the domain, i.e.  $c = N_g$ . The final reward  $r$  is calculated by  $r = r_l + r_g$ .
- **Dataset:** We generated 8000 different maps (or matrices) for each size of  $8 \times 8, 16 \times 16$  and  $32 \times 32$ . In each map, we randomly chose two positions for the two agents, and computed the optimal path (shortest path) which was compared to the paths predicted by our FedRL and baselines. We randomly split the 8000 maps into 6400 for training, 800 for validation, and 800 for testing.
- **Criteria:** In each training episode, we first took the initial state and predicted an action with a model. We then got a new state and took it as the new input of the model at the second time step. We repeated the procedure until the two agents met each other, which is indicated as a *successful episode*, or it exceeded the maximum time step  $T_m$ , which is indicated as a *failure episode*. We set  $T_m$  to be twice the length of the longest optimal path, i.e.  $T_m = 38, 86, 178$  for each size of

<sup>1</sup>The source code and datasets are available from <https://github.com/FRL2019/FRL>

$8 \times 8$ ,  $16 \times 16$  and  $32 \times 32$ , respectively. We finally computed the measures of successful rate  $SuccRate$ , and average reward  $AvgRwd$ , i.e.,

$$SuccRate = \frac{\#SuccessfulEpisodes}{\#TotalEpisodes},$$

and

$$AvgRwd = \frac{TotalCumulativeReward}{\#TotalEpisodes},$$

where  $\#SuccessfulEpisodes$  indicates the number of successful episodes,  $\#TotalEpisodes$  indicates the total number of episodes,  $TotalCumulativeReward$  indicates the total reward of all episodes, respectively.

Table 1. Comparison with baselines in Grid-World

Metric	Method	Domain w.r.t. various sizes		
		$8 \times 8$	$16 \times 16$	$32 \times 32$
<i>SuccRate</i>	FCN-alpha	69.73%	48.04%	41.73%
	DQN-alpha	88.27%	76.20%	71.41%
	FedRL-1	92.52%	79.83%	77.88%
	FedRL-2	<b>95.06%</b>	<b>84.31%</b>	<b>82.02%</b>
	FCN-full	72.16%	56.44%	50.15%
	DQN-full	93.69%	83.40%	79.73%
<i>AvgRwd</i>	DQN-alpha	13.781	-112.084	-285.946
	FedRL-1	18.152	-94.193	-226.583
	FedRL-2	<b>19.101</b>	<b>-84.139</b>	<b>-189.756</b>
	DQN-full	31.286	-38.114	-52.72

**Experimental Results w.r.t. Domain Sizes** We ran our FedRL and baselines five times and calculated an average of  $SuccRate$  (as well as  $AvgRwd$ ). We calculated two results of our FedRL, denoted by FedRL-1 and FedRL-2, which correspond to “adding Gaussian differential privacy on the local Q-network” and “not adding Gaussian differential privacy on the local Q-network” in the testing data, respectively. The results are shown in Table 1. From Table 1, we can see that  $SuccRate$  of both FedRL-1 and FedRL-2 is much better than DQN-alpha and CNN-alpha in all three different sizes of domains, **which indicates agent  $\alpha$  can indeed get help from agent  $\beta$  via learning federatively with agent  $\beta$** . Comparing to DQN-full, we can see that the  $SuccRate$  of both FedRL-1 and FedRL-2 is close to DQN-full, which indicates our federated learning framework can indeed take advantage of both training data from agents  $\alpha$  and  $\beta$ , even though they are protected locally with Gaussian differential privacy (the reason why FedRL-2 is slightly better than DQN-full is that the hierarchical structure of our federated learning framework with three components may be more suitable for this domain than a unique DQN framework). In addition, we can also see that the  $SuccRate$  generally decreases when the size of the domain increases, which is consistent with our intuition, since the larger the domain is,

the more difficult the task is (which requires more training data to build models of high-quality).

From the metric of  $AvgRwd$ , we can see that both FedRL-1 and FedRL-2 are better than DQN-alpha, which indicates agent  $\alpha$  can indeed get help from agent  $\beta$  in gaining rewards via federated learning with agent  $\beta$ . We can also find that FedRL-2 outperforms FedRL-1 in both  $AvgRwd$  and  $SuccRate$ . This is because our model is trained based on Gaussian differential privacy, indicating  $SuccRate$  on testing data with Gaussian differential privacy, as done by FedRL-2, should be better than on testing data without Gaussian differential privacy, as done by FedRL-1.

**Experimental Results w.r.t. History Length** To study when FedRL works, we consider the amount of information that an agent uses. 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 we have, and the more complicated the neural network is. We fixed the structures of all models and only changed the length of the history observations. We tested the length of history from **2 to 32** for all domains. The results are shown in Figure 4. In the first row of Figure 4, we can observe is that the success rates are improving with the increment of the history length. In  $8 \times 8$  and  $16 \times 16$  domains, the results converge when history length is longer than 16, while in  $32 \times 32$  domain, they have not converged even at the history length of 32. The reason is that  $32 \times 32$  domain is more complicated than the other two domains, so it needs more information (i.e.  $H > 32$ ) to learn a model of high-quality. We can also find that when the history length is short (i.e.  $H = 2$  for  $16 \times 16$  and  $H \leq 4$  for  $32 \times 32$ ), DQN-alpha, DQN-full and FedRL-1 perform poorly. FedRL-1 and DQN-full do not show their advantages although they take as input both  $s_\alpha$  and  $s_\beta$  directly or indirectly. However, FedRL-2, **which applies differential privacy to both training and testing samples**, shows its great scalability even with limited amount of history. In small domains, such as  $8 \times 8$ , a few steps are enough to explore the whole environment. Therefore, the DQN-alpha which takes only single observation as input can performs as well as DQN-alpha and FedRL models.

## 5.2. Text2Actions Domain

In this experiment, we evaluated our FedRL in another domain, i.e., Text2Action (Feng et al., 2018), which aims to extract action sequences from texts. For example, consider a text “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 is to extract words composing an action sequence “cook(rice), keep(rice, cold)”, or “use(leftover rice), keep(rice, cold)”. We define

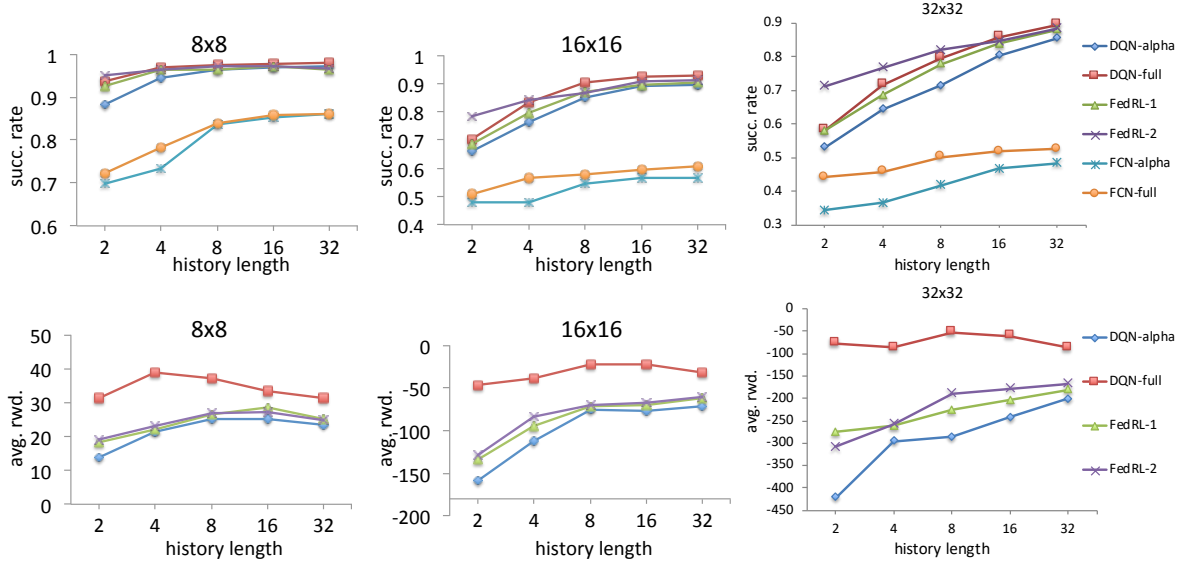


Figure 4. Results about the impact of history length.

states, actions and rewards as follows.

Table 2. Comparison with baselines in Text2Action

Metric	Method	Dataset		
		WHS	CT	WHG
F1	FCN-alpha	86.19%	65.44%	55.18%
	DQN-alpha	92.11%	74.64%	66.37%
	FedRL-1	93.76%	<b>85.05%</b>	75.64%
	FedRL-2	<b>94.41%</b>	84.92%	<b>75.85%</b>
	FCN-full	91.42%	79.03%	68.93%
	DQN-full	94.55%	83.39%	74.63%
AvgRwd	DQN-alpha	54.762	47.375	46.510
	FedRL-1	55.623	<b>50.472</b>	48.359
	FedRL-2	<b>55.894</b>	50.452	<b>48.373</b>
	DQN-full	56.192	50.307	48.154

- **States:**  $s_\alpha \in \mathbb{R}^{N_w \times K_1}$  is real-valued matrix that describes the part-of-speech of words, and  $s_\beta \in \mathbb{R}^{N_w \times K_2}$  is real-valued matrix that describes the embedding of words.  $N_w$  is the number of words in the text,  $K_1$  is the dimension of a part-of-speech vector, and  $K_2$  is the dimension of word vectors. In our experiments, part-of-speech vectors are randomly initialized and trained together with the Q-network, while word vectors are generated from the pre-trained word embeddings and will not be changed during the training of the Q-network.
- **Actions:** There are two actions for each agent, i.e.,  $\{select, neglect\}$ , indicating selecting a word as an action name (or a parameter), or neglecting a word (which means the corresponding word is neither an action name nor a parameter).

- **Rewards:** The instant reward includes a basic reward and an additional reward, where the basic reward indicates whether the agent selects a word correctly or not, and the additional reward encodes the priori knowledge of the domain, i.e., the proportion of words that are related to action sequences (Feng et al., 2018). We assume that only agent  $\alpha$  knows the rewards, while agent  $\beta$  does not know them.

- **Dataset:** We conducted experiments on three datasets, i.e., “Microsoft Windows Help and Support” (WHS) documents (Branavan et al., 2009), and two datasets collected from “WikiHow Home and Garden”<sup>2</sup> (WHG) and “CookingTutorial”<sup>3</sup> (CT), respectively. In CT, there are 116 labeled texts and 134,000 words, with 10.37% being action names and 7.44% being action arguments. In WHG, there are 150 labeled texts and 34,000,000 words, with 7.61% being action names and 6.3% being action arguments. In WHS, there are 154 labeled texts and 1,500 words, with 19.47% being action names and 15.45% being action arguments.

- **Criteria:** For evaluation, we first fed texts to each model to obtain selected words. We then compared the outputs to their corresponding ground truth and calculated  $\#TotalTruth$  (total ground truth words),  $\#TotalRight$  (total correctly selected words), and  $TotalSelected$  (total selected words). After that we computed metrics  $precision = \frac{\#TotalRight}{\#TotalSelected}$ ,

<sup>2</sup><https://www.wikihow.com/Category:Home-and-Garden>

<sup>3</sup><http://cookingtutorials.com/>

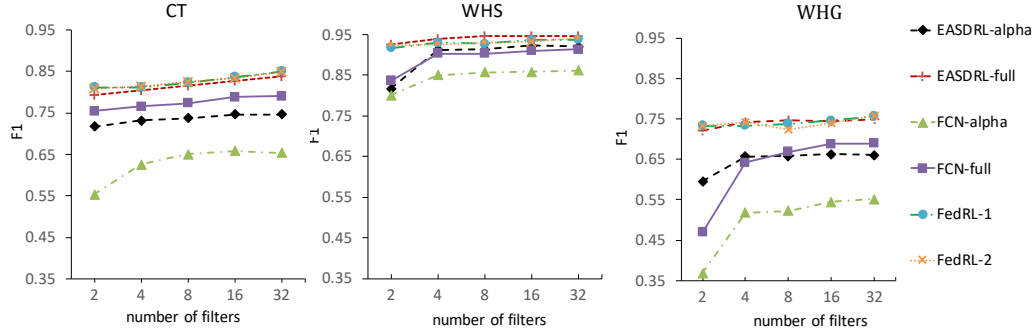


Figure 5. Results about the impact of the number of convolutional kernels.

$recall = \frac{\#TotalRight}{\#TotalTruth}$ , and

$$F1 = \frac{2 \times precision \times recall}{precision + recall}.$$

We used **F1-metric for all baselines**. For reinforcement learning methods, we also computed 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.

We adopted 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  $m = 50$ . 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  $128 \times 2$ , where 128 equals to  $4 \times 32$ , indicating 4 types of  $n$ -grams and 32 kernels for each  $n$ -grams, and 2 is the size of the action space. The MLP module of FedRL is composed of two fully-connected layers with size  $4 \times 2$ . The standard deviation of Gaussian differential privacy  $\sigma$  was set to be 1. For all models, we adopted the Adam optimizer with learning rate 0.001 and ReLU activation<sup>4</sup>.

**Experimental Results** We ran our FedRL and baselines five times to calculate an average of F1 (as well as *AvgRwd*). The results are shown in Table 2. From Table 2 we can see that both FedRL-1 and FedRL-2 outperform

both FCN-alpha and DQN-alpha in all three datasets under both F1 and *AvgRwd* metrics, which indicates agent  $\alpha$  can learn better policies via federated learning with agent  $\beta$  than learning by itself. Comparing our FedRL-1 and FedRL-2 with DQN-full in both F1 and *AvgRwd*, we can see that their performances are close to each other, suggesting our federated learning framework performs as good as the DQN model directly built from all training data from both agents  $\alpha$  and  $\beta$ .

To see the impact of the model complexity, we varied the number of convolutional kernels from 2 to 32 with other parameters fixed. The results are shown in Figure 5. We can see that all models generally perform better when the number of convolutional kernels (filters) increases, and become stable after 8 kernels. We can also see that our FedRL models outperform DQN-alpha, FCN-alpha and FCN-full with respect to the number of kernels, which indicates agent  $\alpha$  can indeed learn better policies via federated learning with agent  $\beta$ . Both FedRL-1 and FedRL-2 are close to DQN-full with respect to the number of filters, suggesting that our federated learning framework is effective even though the data of agents  $\alpha$  and  $\beta$  are not shared with each other.

## 6. Conclusion

we propose a novel reinforcement learning approach to **considering privacies and federatively** building Q-network for each agent with the help of other agents, namely Federated deep Reinforcement Learning (FedRL). To protect the privacy of data and models, we exploit **Gaussian differentials on the information shared with each other when updating their local models**. We demonstrate that the proposed FedRL approach is effective in building high-quality policies for agents under the condition that training data are not shared between agents. In the future, it would be interesting to study more members in the federation with background knowledge represented by (probably incomplete) action models (Zhuo et al., 2014; Zhuo & Yang, 2014; Zhuo & Kambhampati, 2017).

<sup>4</sup>Detailed setting can be found from the source code: <https://github.com/FRL2019/FRL>



## 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.
- Branavan, S. R. K., Chen, H., Zettlemoyer, L. S., and Barzilay, R. Reinforcement learning for mapping instructions to actions. In *ACL*, 2009.
- 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 *ICML*, pp. 1009–1018, 2018.
- Duchi, J. C., Jordan, M. I., and Wainwright, M. J. Privacy aware learning. In *NIPS*, pp. 1439–1447, 2012.
- 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.
- Foerster, J. N., Assael, Y. M., de Freitas, N., and Whiteson, S. Learning to communicate with deep multi-agent reinforcement learning. In *NIPS*, pp. 2137–2145, 2016.
- Furuta, R., Inoue, N., and Yamasaki, T. Fully convolutional network with multi-step reinforcement learning for image processing. In *AAAI*, pp. 3598–3605, 2019.
- Konecný, J., McMahan, B., and Ramage, D. Federated optimization: Distributed optimization beyond the datacenter. *CoRR*, abs/1511.03575, 2015.
- 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 Mor-datch, 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., Venness, 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 *NIPS*, 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 *NIPS*, pp. 6182–6192. 2018.
- Yang, Q., Liu, Y., Chen, T., and Tong, Y. Federated machine learning: Concept and applications. *ACM TIST*, 10(2): 12:1–12:19, 2019.
- Zhuo, H. H. and Kambhampati, S. Model-lite planning: Case-based vs. model-based approaches. *Artif. Intell.*, 246:1–21, 2017.
- Zhuo, H. H. and Yang, Q. Action-model acquisition for planning via transfer learning. *Artif. Intell.*, 212:80–103, 2014.
- Zhuo, H. H., Muñoz-Avila, H., and Yang, Q. Learning hierarchical task network domains from partially observed plan traces. *Artif. Intell.*, 212:134–157, 2014.