Asynchronous Multitask Reinforcement Learning with Dropout for Continuous Control

Zilong Jiao
EECS, Syracuse University
Syracuse, NY
zijiao@syr.edu

Jae Oh
EECS, Syracuse University
Syracuse, NY
jcoh@syr.edu

***Abstract*—Deep reinforcement learning is sample inefficient for solving complex tasks. Recently, multitask reinforcement learning has received increased attention because of its ability to learn general policies with improved sample efficiency. In multitask reinforcement learning, a single agent must learn multiple related tasks, either sequentially or simultaneously. Based on the DDPG algorithm, this paper presents Asyn-DDPG, which asynchronously learns a multitask policy for continuous control with simultaneous worker agents. We empirically found that sparse policy gradients can significantly reduce interference among conflicting tasks and make multitask learning more stable and sample efficient. To ensure the sparsity of gradients evaluated for each task, Asyn-DDPG represents both actor and critic functions as deep neural networks and regularizes them using Dropout. During training, worker agents share the actor and the critic functions, and asynchronously optimize them using task-specific gradients. For evaluating Asyn-DDPG, we proposed robotic navigation tasks based on realistically simulated robots and physics-enabled maze-like environments. Although the number of tasks used in our experiment is small, each task is conducted based on a real-world setting and posts a challenging environment. Through extensive evaluation, we demonstrate that Dropout regularization can effectively stabilize asynchronous learning and enable Asyn-DDPG to outperform DDPG significantly. Also, Asyn-DDPG was able to learn a multitask policy that can be well generalized for handling environments unseen during training.**

***Index Terms*—Deep reinforcement learning, Multitask reinforcement learning, Asynchronous method, Continuous control, Partial observability**

I. Introduction

Recently, deep reinforcement learning has been able to learn policies that can constantly exceed human performance in various task domains, such as Go [16], Atari games [11] and continuous control [9]. Even though the results are impressive, the existing work normally trains a specialized policy from scratch for one task at a time, and each task requires training a different policy instance. Learning a specialized policy for complex task in a rich environment usually require tremendous amount of time and agent experience, which make the deep reinforcement learning methods sample inefficient. To address this issue, researchers in the reinforcement learning community shifted their attention to multitask reinforcement learning which could estimate a general policy by learning a series of related tasks, either sequentially or simultaneously. Compared to single-task reinforcement learning, one would expect that, in multitask reinforcement learning, learning each individual task requires much less data, and combining solutions of multiple tasks enables a policy to have better asymptotic performance and generalizability.

Learning simple tasks individually does not make the learning in a multitask setting simpler. Instead, it poses at least two stressing issues for learning effective policies. First, the processes of learning individual tasks often interfere with each other. When policy parameters are jointly optimized based on multiple tasks, it is likely that the gradients evaluated in one task can override the gradients evaluated in another task. Without special treatments, this would make a multitask reinforcement learning method sample inefficient. Second, a multitask policy can have unbalanced performance on learned tasks. Since tasks can be learned based on rewards with different scale or distribution, some of the tasks can be more *salient* than the other during training [7]. Addressing those issues, recently parallel multitask reinforcement learning [5], [7] has demonstrated remarkable effectiveness in Atari games [11] and DeepMind Lab [2]. In those environments, agents operate in discrete action spaces. On contrary, multitask reinforcement learning for continuous control is still under-explored.

Agents with continuous actions are commonly involved in robotic control tasks, such as autonomous driving [14], UAV control [3] and object manipulation [8], [12]. Unlike game environments [2], [11], robotic control is often constrained on physical factors, e.g. limited sensing ranges, high-dimensional sensor data, and limited acceleration of motion. This paper studies multitask reinforcement learning for continuous control. In particular, we focus on agents with continuous actions and partial observability. Based on Deep Deterministic Policy Gradient (DDPG) algorithm [9], we propose an asynchronous method, Asyn-DDPG, for learning a shared policy and q-function with multiple simultaneous worker agents. Considering partial observability of agents, Asyn-DDPG represents the shared policy and q-function as recurrent neural networks which allow agents to take actions based on a sequence of recent sensor observation.

We empirically found that ensuring the sparsity of the gradients applied to the shared policy and q-function can reduce conflicts in learning competing tasks, as well as avoiding unbalanced learning. To this end, we regularize the shared policy and q-function using Dropout and ensure the

978-1-7281-4550-1/19/$31.00 ©2019 IEEE
DOI 10.1109/ICMLA.2019.00099

gradients generated through back-propagation to be sparse. As each agent needs to evaluate its own sparse gradients, applying different Dropout to the same neural network requires synchronization. To solve this issue, Asyn-DDPG let each worker agent maintain up-to-date copies of the shared policy and q-function in its own memory and independently apply Dropout regularization to those local copies. During training, each worker agent asynchronously updates the shared policy and q-function using the gradients evaluated based on its regularized local policy and q-function.

We evaluate Asyn-DDPG in physic-enabled environments based on robotic simulation. In experiments, we let worker agents simultaneously learn different navigation tasks in a small number of maze-like continuous environments. Experimenting with those environments provides the first step to understand how Dropout regulation affects learning performance of agents in a multitask setting. It also allows us to analyze the performance of the policy learned by Asyn-DDPG on all those learned tasks in detail. We demonstrate that Dropout regularization can effectively reduce the interference among competing tasks and enable a learned policy to have balanced performance on individual tasks. With extensive evaluation, the policy learned by Asyn-DDPG can significantly outperform the specialized policies learned by DDPG in all test environments. In addition, the policy learned by Asyn-DDPG is able to handle more complex navigation tasks that are unseen by agents during training

II. Related Work

In multitask reinforcement learning, a single agent must learn multiple tasks, either sequentially or simultaneously. In terms of learning multiple tasks sequentially, many work has been studied under the topics of Lifelong Learning [1], [20] and Curriculum Learning [6], [10]. Recently, simultaneous multitask reinforcement learning has been actively studied, and impressive results have been achieved in discrete environments [5], [7]. On contrary, multitask reinforcement learning in continuous environments are less focused, and recent work on this topics are [4], [22].

For learning a multitask policy, Deisenroth et al. [4] studied multitask reinforcement learning in the context of robotics for continuous control. In their work, a multitask policy is represented as a function of agent states and tasks, and, at each optimization step, a single agent optimizes the policy using the policy gradients averaged over all tasks. In contrast, our work does not rely on task identification, instead it enables worker agents to asynchronously optimize the shared policy using regularized sparse gradients. This allows our method to be more sample efficient than the aforementioned method.

The closest work to ours is multi-DDPG [22] which learns simple robotic control tasks with multiple DDPG actors. With a single shared critic, multi-DDPG learns tasks specific actors for each continuous control task. In contrast, our work allows agents to jointly learn a single actor (i.e. a shared policy) to handle all continuous control tasks. This is made possible by using Dropout regularization to resolve conflicts among competing tasks.

Using regularization to enable multitask learning has been studied in recent literature. Teh et al. [19] applied γ-discounted KL divergents to task-specific policies, in order to simultaneously distill them into a central policy. To learn tasks with different reward scales in parallel, Hessel et al. [7] regularize the gradient update of a shared value function by applying PopArt normalization [21] to task-specific state values. Comparing to aforementioned methods, Dropout regularization is simple but effective technique for enabling asynchronous multitask reinforcement learning. It avoids having need of task specific information (e.g., task IDs) for synthesizing knowledge learned in individual tasks into a meta policy, which sheds light on a better direction for learning a multitask policy.

III. Background

A. Policy Gradient Methods

Policy gradient methods optimize a parameterized policy with respect to its expected reward using gradient decent algorithms. Let S define a state space of an agent; A be a set of actions the agent can take in each state $s \in S$. μ_θ is a policy with a parameter vector θ. When μ_θ is stochastic, the parameter gradients for optimizing its expected reward can be calculated as

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_{s\sim\rho^{\mu_\theta}, a\sim\mu_\theta}\{\nabla_\theta \log \mu_\theta(a \mid s) Q(s,a)\} \quad (1)$$

where ρ^{μ_θ} denotes the probability distribution that an agent visits each $s \in S$ using μ_θ. As a special case of Equation 1, Equation 2 computes the gradients for optimizing a deterministic policy [17].

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_{s\sim\rho^{\mu_\theta}}\{\nabla_\theta \log \mu_\theta(s) \nabla_a Q(s,a) \mid_{a=\mu_\theta(s)}\} \quad (2)$$

$\nabla_\theta \log(\mu_\theta(s))$ is a Jacobian matrix where an entry in row i and column j represents the gradient of the ith parameter for the jth action. $\nabla_a Q(s,a)$ is a vector of gradients with respect to the Q-function for the action selected by μ_θ in a state. In this paper, we use Q-functions to estimate policy gradients, and alternative estimations of policy gradients can be found in [15].

If a Q-function (or a value function) is unknown, an agent must fit the unknown Q-function based on its state-action trajectories collected online while calculating policy gradients for optimizing its policy. A method alternating between fitting a Q-function and optimizing a policy is called an actor-critic method.

B. Sparse Gradients through Dropout Regulation

We revisit Dropout [18] in the context of both feed-forward neural networks and recurrent neural networks. We present how sparse gradients are obtained when Dropout is applied to both types of neural networks.

Dropout in Feed-Forward Neural Network. Suppose a policy μ_θ is represented as a feed-forward neural network with parameters θ. Let $L = \{1, 2, \ldots, l\}$ be the indexes of hidden

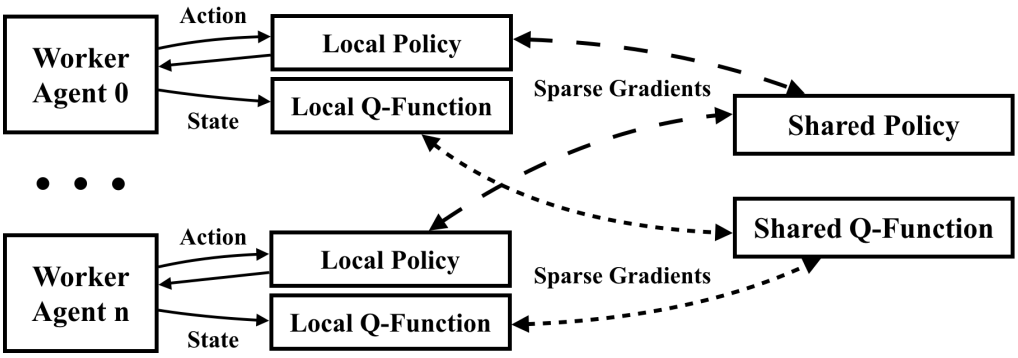


Fig. 1: The overview of Asyn-DDPG

layers. z^l and y^l are the input and output of hidden layer l. In forward operation, dropout is applied to each y^l, s.t.

$$\tilde{y}^l = r^l * y^l$$

r^l is a vector whose components are independently sampled from Bernoulli distribution with a probability of P being 0, and P is called a dropout rate. $*$ denotes element-wise multiplication. $\tilde{y}^l$ is input to the hidden layer $l+1$ according to the following equation:

$$z^{l+1} = W^{l+1}\tilde{y}^l + b^{l+1} \qquad y^{l+1} = \tau(z^{l+1})$$

where, $\tilde{y}^l$ is a column vector, and τ is non-linear activation function. W^{l+1} is a $n \times m$ weight matrix, where n and m are the number of neurons in hidden layers $l+1$ and l. For computing z^{l+1}, it's equivalent to zero out ith column of W^{l+1}, when ith component of r^l is 0. When the dropout rate P is sufficiently large, all weight matrices in a forward neural network can be sparse. While evaluating gradients for the weight matrices through back-propagation, the entries which were zeroed out in the forward operation are restricted to have gradients of 0. Therefore, the resulting gradient vector is sparse.

Dropout in Recurrent Neural Network. Taking Long Short Term Memory (LSTM) as an example, we present that Dropout can be applied to a recurrent neural network to produce sparse gradients. Let x_t and h_t be the input and the output of LSTM at time t. Dropout is applied to LSTM in the follow way

$$\tilde{h}_t = r^h * h_t \qquad \tilde{x}_t = r^x * x_t$$

r^h and r^x are dropout vectors as what was explained in the feed-forward neural network case. t indicates the time step of a input sequence. With Dropout, LSTM is given by the equations below:

$$\begin{array}{ll} \underline{i} = sigm(U_i\tilde{h}_{t-1} + W_i\tilde{x}_t) & \underline{f} = sigm(U_f\tilde{h}_{t-1} + W_f\tilde{x}_t) \\ \underline{o} = sigm(U_o\tilde{h}_{t-1} + W_o\tilde{x}_t) & \underline{g} = sigm(U_g\tilde{h}_{t-1} + W_g\tilde{x}_t) \\ c_t = \underline{f} * c_{t-1} + \underline{g} * \underline{i} & h_t = \underline{o} * tanh(c_t) \end{array}$$

Let $\mathbf{W} = \{W_i, W_f, W_o, W_g\}$ and $\mathbf{U} = \{U_i, U_f, U_o, U_g\}$ be weight matrices of a LSTM. h_t is a column vector that is the output at time t. Similar to the feed-forward neural network case, h_t can be calculated by zeroing out ith column of each $W \in \mathbf{W}$, if the ith component of $\tilde{h}_t$ is 0. Similarly, when the ith component of $\tilde{x}_t$ is 0, ith column of $U \in \mathbf{U}$ can be zeroed out for computing $\tilde{x}_t$. While evaluating gradients for each $W \in \mathbf{W}$ and each $U \in \mathbf{U}$, we restrict the entries that are zeroed out to have gradients of 0. When the dropout probability P is sufficiently large, the resulting gradient vector can be sparse.

IV. Asyn-DDPG

Based on DDPG algorithm [9], we propose an asynchronous actor-critic method, Asyn-DDPG, for learning multiple continuous control tasks with worker agents. The propose method enables multiple worker agents asynchronously optimize a shared policy and q-function. The key to the proposed method is maintaining the sparsity of the gradients applied to the shared policy and q-function. To this end, we let each agent maintain up-to-date copies of the shared policy and q-function in its local memory and apply independent Dropout regularization to those copies. Figure 1 shows the overview of Asyn-DDPG.

A. Shared Policy with Dropout Regularization

Considering robotic applications in practice, we assume that, at each time step, an agent perceives a high-dimensional feature vector from its surrounding environment through an on-board sensor (e.g. a camera or LiDar). The limited sensing capability make the environment where the agent operates partially observable. To overcome the partial observability, in Asyn-DDPG, a state of an agent contains a sequence of sensor observation perceived the past l time steps. In addition, the state of an agent also contains other information, including locations and velocities. We represent the shared policy and q-function as deep recurrent neural networks with dropout regularization. Figure 2 summarizes their structures.

In Asyn-DDPG, agents independently apply Dropout regularization to the local copies of the share policy and q-function. This allows agents to independently evaluate sparse gradients for optimizing both functions during training. Note that agents do not apply Dropout to input of LSTM, since at each optimization step the states input to the local policy and

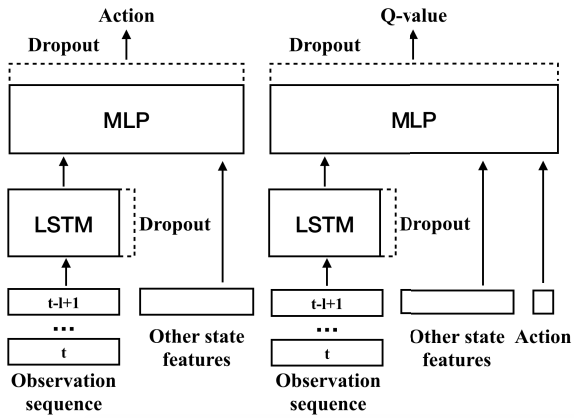
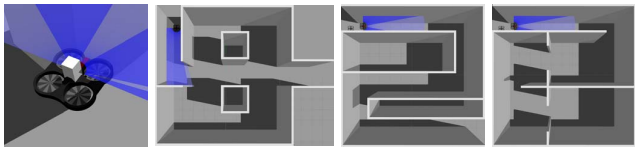


Fig. 2: The shared policy (left) and q-function (right).



(a) Agent (b) Env0 (c) Env1 (d) Env2

Fig. 3: The local patterns used for training. For each local pattern, a circle is a initial location, and a star is a goal.

q-function of an agent must always be the same. Although agent could apply different configurations of dropout rates to its local policy and q-function, we found that having all worker agents adopt the same configuration of dropout rates can already let Asyn-DDPG converge at a robust solution.

B. Asynchronous Update

Algorithm 1 summarize the asynchronous update procedural followed by all agents. $\mu(\theta_i^t)$ and $Q(\omega_i^t)$ are the agent i's local copies of the shared policy and q-function. At time t, the worker agent i evaluates Deterministic Policy Gradients and Temporal Difference Gradients. Since those evaluated gradients are sparse, agents can follow the Hogwild! strategy [13] to asynchronously apply their gradients to the shared policy and q-function.

Because of the copying mechanism, during training each worker agent can transfer their knowledge learned in different tasks to the other agents. This allows each work agent to improve the share policy upon the work done by the other agents, which is essential for the agents to synthesize their behavior into a consistent meta policy (i.e. the shared policy). As the gradient evaluation does not depend on task specific information, the policy jointly learned by all worker agents can be generalized to unfamiliar and potentially more complex tasks.

Algorithm 1 A robot applies gradients to shared networks.

function ASYNUPDATE()
  copy $w^\star$ and $\theta^\star$ to w and θ
  while the shared policy hasn't converge **do**
    deploy the robot at a fixed initial location
    while an episode is not terminated **do**
      store experience to replay buffer
      sample a batch of experience
      evaluate policy gradient δ_ω
      evaluate temporal-difference gradients δ_θ
      apply δ_w and δ_θ to $w^\star$ and $\theta^\star$
      copy $w^\star$ and $\theta^\star$ to w and θ
    end while
  end while
end function

V. EXPERIMENTS

A. Experimental Setup

Agents: We simulate worker agents as unmanned aerial vehicles using ROS and Gazebo. An worker agent observes an environment through a LiDar sensor, and it scans the area in its front with 360 evenly spaced lasers at a constant rate. An agent detects a collision if any laser measures a range less than 0.2m. A state of a worker agent is defined as $(\boldsymbol{o}, d, v)$. $\boldsymbol{o}$ is observation sequence with length of 16. d is the normalized direction from the agent's present location to its goal, and v is the agent's present velocity. In experiments, a worker agent moves on the XY-plane at a constant speed in its heading direction. We define an action of a worker agent to be its rotational velocity, $y \in [-\pi, \pi]$, on the XY-plane.

Tasks: In experiments, worker agents learn three navigation tasks in maze-like environments, as presented in Figure 3. In each environment, a worker agent has a pre-defined initial location and goal. It terminate an episode of navigation, if it collides with an obstacle or reaches its goal. In each episode, an worker agent receives rewards given by

$$r = \begin{cases} -|\omega| & d'_g \leq d_g \\ d'_g - d_g - |\omega| & d'_g > d_g \\ -1 & \text{close to an obstacle} \\ 1 & \text{reached the goal} \end{cases}$$

d'_g and d_g are the Euclidean distances from an agent's previous and present locations to its goal. ω is an agent's present rotational velocity. The policy jointly learned by all worker agents needs to master navigation tasks in all environments.

B. Learning Performance of Asyn-DDPG

In the experiment, worker agents uses the same configuration of dropout rates: 0.4 for LSTM and 0.2 for fully connected layers. For both shared policy and q-function, each fully connected layer has 1024 neurons with ReLu activation functions, and the hidden state of LSTM has the size of

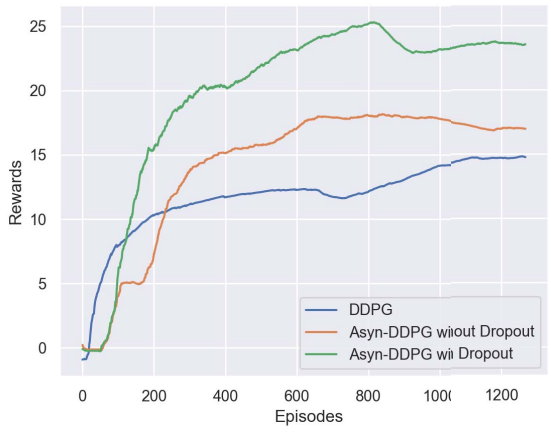


Fig. 4: Learning performance of DDPG and Asyn-DDPG in terms of episodic rewards. Both methods are evaluated in Env3.

128. Figure 3 summarizes the learning performance of Asyn-DDPG, in comparison with the standard DDPG and the Asyn-DDPG without dropout regularization. We train the Asyn-DDPPG policy with and without dropout regularization in Env0 to Env2 and train the DDPG policy directly in Env3. For fair comparison, during training we sample the shared policy jointly learned by Asyn-DDPG every 5 seconds (in wall-clock time) and evaluate its performance in Env3. In the Figure, the reward distribution are the moving average of 100 episodic rewards. It shows that the multitask policy learned by Asyn-DDPG achieves better asymptotic performance, compared the single-task policy learned by DDPG.

To better understand the effectiveness of dropout regularization, we compare the performance of Asyn-DDPG with and without Dropout regularization in Env0 to Env2. Figure 5 presents the impact of dropout regularization on the Asyn-DDPG. The results are moving averages of 100 episodic rewards. Based on the results, Dropout is able to effectively stabilize the performance of Asyn-DDPG on each task. This supports our claim that ensuring sparsity of gradients through Dropout regularization can resolve the interference of learning competing tasks.

C. Performance Balance of the Learned Multitask Policy

In this section, we evaluate the policy learned by Async-DDPG in two aspects: 1) if the learned policy performs equally well in training tasks; 2) if the learned multitask policy has better performance than task specific policies learned by DDPG.

With balanced performance, the multitask policy jointly learned by all worker agents is expected to complete navigation tasks in all training environments. For the first aspect of the evaluation, we measure the performance of a policy learned by Async-DDPG in different concatenations of training environment. Those environments are presented in Figure 6a to 6d, together with the trajectories planned by both Async-DDPG and DDPG policies. We trained those DDPG policies in each of those environments separatly, while we only trained one multi-task policy through the Asyn-DDPG based on Env0 to Env2. Table I summarizes the average performance of the trained policies during 50 episodes.

According to the results, the multitask policy learned by Asyn-DDPG can complete the navigation task in all test environments with sufficiently high success rates. The experiment results provides consistent evidence showing that the multitask policy learned by Asyn-DDPG has balanced performance in all training tasks. Without balanced performance, the learned multitask policy will result in collisions in one of the training environments during navigation. In addition, the multitask policy constantly achieve better performance in all environments, compared to the policies learned by DDPG. This confirms what we concluded in the previous section: the policies learned by Asyn-DDPG has better asymptotic performance, compared to the single-task policies learned by DDPG.

Env3	Env4	Env5	Env6	Env7	Env8
0.88	0.9	0.9	0.9	0.86	0.92

TABLE I: The success rates of a Asyn-DDPG policy in all test environments based on 50 episodes.

D. Testing Policy Performance in Unfamiliar Tasks

We evaluate the generality of the policy learned by Asyn-DDPG in another two environments which were unseen by worker agents during training. Those environments are presented in Figure 6e and 6f, together with the trajectories planed by the Asyn-DDPG policy. Table I shows the success rates of the multi-task policy in all test environments (i.e., Env3 to Env8) based on the average performance of 50 episodes. According to the results, the multitask policy learned by Asyn-DDPG can be well generalized to handle unfamiliar tasks.

VI. Conclusion

In this paper, we presented an asynchronous multitask reinforcement learning method, Asyn-DDPG. Relying on Dropout regularization, Asyn-DDPG was able to effectively resolve the interference among the processes of learning competing tasks and asynchronously learn a multitask policy with balanced performance and good generalizability. In experiments, we evaluate Asyn-DDPG in robotic navigation tasks based on realistically simulated robots and physics-enabled environments. Although the number of tasks used in our experiments is small, we designed each individual task based on a real-world setting, and each task posted a more challenging continuous environment, compared to the continuous control tasks studied in the previous multitask reinforcement learning literature [22]. With extensive evaluation of the multitask policy learned by Asyn-DDPG, our work provided the first step to understand the effects of Dropout regularization on asynchronous multitask

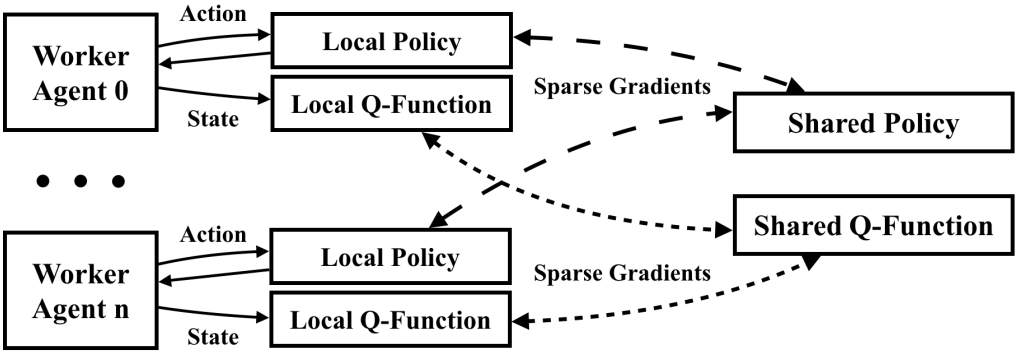


Fig. 1: The overview of Asyn-DDPG

layers. z^l and y^l are the input and output of hidden layer l. In forward operation, dropout is applied to each y^l, s.t.

$$\tilde{y}^l = r^l * y^l$$

r^l is a vector whose components are independently sampled from Bernoulli distribution with a probability of P being 0, and P is called a dropout rate. $*$ denotes element-wise multiplication. $\tilde{y}^l$ is input to the hidden layer $l+1$ according to the following equation:

$$z^{l+1} = W^{l+1}\tilde{y}^l + b^{l+1} \qquad y^{l+1} = \tau(z^{l+1})$$

where, $\tilde{y}^l$ is a column vector, and τ is non-linear activation function. W^{l+1} is a $n \times m$ weight matrix, where n and m are the number of neurons in hidden layers $l+1$ and l. For computing z^{l+1}, it's equivalent to zero out ith column of W^{l+1}, when ith component of r^l is 0. When the dropout rate P is sufficiently large, all weight matrices in a forward neural network can be sparse. While evaluating gradients for the weight matrices through back-propagation, the entries which were zeroed out in the forward operation are restricted to have gradients of 0. Therefore, the resulting gradient vector is sparse.

Dropout in Recurrent Neural Network. Taking Long Short Term Memory (LSTM) as an example, we present that Dropout can be applied to a recurrent neural network to produce sparse gradients. Let x_t and h_t be the input and the output of LSTM at time t. Dropout is applied to LSTM in the follow way

$$\tilde{h}_t = r^h * h_t \qquad \tilde{x}_t = r^x * x_t$$

r^h and r^x are dropout vectors as what was explained in the feed-forward neural network case. t indicates the time step of a input sequence. With Dropout, LSTM is given by the equations below:

$$\begin{array}{ll} \underline{i} = sigm(U_i\tilde{h}_{t-1} + W_i\tilde{x}_t) & \underline{f} = sigm(U_f\tilde{h}_{t-1} + W_f\tilde{x}_t) \\ \underline{o} = sigm(U_o\tilde{h}_{t-1} + W_o\tilde{x}_t) & \underline{g} = sigm(U_g\tilde{h}_{t-1} + W_g\tilde{x}_t) \\ c_t = \underline{f} * c_{t-1} + \underline{g} * \underline{i} & h_t = \underline{o} * tanh(c_t) \end{array}$$

Let $\mathbf{W} = \{W_i, W_f, W_o, W_g\}$ and $\mathbf{U} = \{U_i, U_f, U_o, U_g\}$ be weight matrices of a LSTM. h_t is a column vector that is the output at time t. Similar to the feed-forward neural network case, h_t can be calculated by zeroing out ith column of each $W \in \mathbf{W}$, if the ith component of $\tilde{h}_t$ is 0. Similarly, when the ith component of $\tilde{x}_t$ is 0, ith column of $U \in \mathbf{U}$ can be zeroed out for computing $\tilde{x}_t$. While evaluating gradients for each $W \in \mathbf{W}$ and each $U \in \mathbf{U}$, we restrict the entries that are zeroed out to have gradients of 0. When the dropout probability P is sufficiently large, the resulting gradient vector can be sparse.

IV. Asyn-DDPG

Based on DDPG algorithm [9], we propose an asynchronous actor-critic method, Asyn-DDPG, for learning multiple continuous control tasks with worker agents. The propose method enables multiple worker agents asynchronously optimize a shared policy and q-function. The key to the proposed method is maintaining the sparsity of the gradients applied to the shared policy and q-function. To this end, we let each agent maintain up-to-date copies of the shared policy and q-function in its local memory and apply independent Dropout regularization to those copies. Figure 1 shows the overview of Asyn-DDPG.

A. Shared Policy with Dropout Regularization

Considering robotic applications in practice, we assume that, at each time step, an agent perceives a high-dimensional feature vector from its surrounding environment through an on-board sensor (e.g. a camera or LiDar). The limited sensing capability make the environment where the agent operates partially observable. To overcome the partial observability, in Asyn-DDPG, a state of an agent contains a sequence of sensor observation perceived the past l time steps. In addition, the state of an agent also contains other information, including locations and velocities. We represent the shared policy and q-function as deep recurrent neural networks with dropout regularization. Figure 2 summarizes their structures.

In Asyn-DDPG, agents independently apply Dropout regularization to the local copies of the share policy and q-function. This allows agents to independently evaluate sparse gradients for optimizing both functions during training. Note that agents do not apply Dropout to input of LSTM, since at each optimization step the states input to the local policy and