An Actor-Critic-Attention Mechanism for Deep Reinforcement Learning in Multi-view Environments

Elaheh Barati[1]* and **Xuewen Chen**[2]
[1]Department of Computer Science, Wayne State University, Detroit, MI, USA
[2]AIWAYS AUTO, Shanghai, China
elaheh.barati@wayne.edu, xuewen.chen@ai-ways.com

Abstract

In reinforcement learning algorithms, leveraging multiple views of the environment can improve the learning of complicated policies. In multi-view environments, due to the fact that the views may frequently suffer from partial observability, their level of importance are often different. In this paper, we propose a deep reinforcement learning method and an attention mechanism in a multi-view environment. Each view can provide various representative information about the environment. Through our attention mechanism, our method generates a single feature representation of environment given its multiple views. It learns a policy to dynamically attend to each view based on its importance in the decision-making process. Through experiments, we show that our method outperforms its state-of-the-art baselines on TORCS racing car simulator and three other complex 3D environments with obstacles. We also provide experimental results to evaluate the performance of our method on noisy conditions and partial observation settings.

1 Introduction

Distributed reinforcement learning algorithms [Mnih *et al.*, 2016; Barth-Maron *et al.*, 2018] have been proposed to improve the performance of the learning algorithm by passing copies of the environment to multiple workers. In these methods, although different workers receive different copies of the environment, observations provided by all these copies are from the same sensory input. In other words, the adoption of multiple workers in these works is rather to increase the training reward and earlier convergence, not to collectively increase the amount of observable information from the environment through multiple sensory inputs.

In an environment such as autonomous driving, an agent may have multiple views of the environment through sensory inputs such as cameras, radars, and so on. These input resources may provide different representative information about the velocity, position, and orientation of the agent. In a realistic environment, observability can be typically partial on account of occlusion from the obstacles or noise that affect the sensors in the environment. Particularly, in environments with pixel-only input, using only one camera view can result in failure, since locating a single camera in a position that can capture both the targets as well as details of agents body is difficult [Tassa *et al.*, 2018]. On the other hand, sensory inputs with less importance can sometimes provide observations which are vital for achieving rich behavior. Therefore, it is desirable to incorporate multiple sensory inputs in the decision-making process according to their importance and their provided information at each time. The incorporation of multiple sensory inputs in a reinforcement learning environment potentially reduces the sensitivity of policies to an individual sensor and makes the system capable of functioning despite one or more sensors malfunctioning. Since sensors can provide diverse views of the environment, and they are likely to be perturbed by different noise impacts, a policy is required to attend to the views accordingly.

In this paper, we propose an attention-based deep reinforcement learning algorithm that learns a policy to attend to different views of the environment based on their importance. Each sensory input, which provides a specific view of the environment, is assigned to a worker. We employ an extension of the actor-critic approach [Silver *et al.*, 2014] to train the network of each worker and make the final decision through the integration of feature representations provided by the workers based on the attention mechanism. This attention mechanism makes use of critic networks of the workers. Since these critic networks provide signals regarding the amount of salient information supplied by their corresponding views, we employ these signals in the attention module to estimate the amount of impact that each of these views should have in the final decision-making process. We decouple training of the model, so that view dependent and task-dependent layers of the network are trained under different strategies.

Unlike distributed reinforcement learning algorithms such as [Mnih *et al.*, 2016; Barth-Maron *et al.*, 2018], our method employs different views of the environment as well as different exploration strategies to provide diversity. On the other hand, through the attention mechanism, our method can prevent its decision-making process from being perturbed by degraded views of the environment. Our main contribution is hence three-fold: (1) To the best of our knowledge, we are the first to propose a deep reinforcement learning algorithm

*Contact Author

and a stabilized training process that leverages multiple views of an environment. (2) We propose a mechanism to attend to multiple views of the environment with respect to their importance in the decision-making process. (3) We propose an effective solution to reduce the impact of noise and partial observability on sensory inputs.

2 Related Work

In reinforcement learning (RL) tasks that need to deal with continuous action space, it is usually required to maintain an explicit representation of the policy. Multiple approaches have successfully learned policies in continuous domains such as Deep Deterministic Policy Gradient (DDPG) [Lillicrap *et al.*, 2015], proximal policy optimization (PPO) [Schulman *et al.*, 2017] and their variants D3PG and D4PG [Barth-Maron *et al.*, 2018], and DPPO [Heess *et al.*, 2017] in distributed settings. [Mnih *et al.*, 2016] proposed asynchronous advantage actor-critic (A3C) algorithm as a variant of policy gradient that can be applied to continuous action spaces. A3C uses multiple parallel workers (agents) that are fed with the same copy of the environment to explore the state spaces. [Mnih *et al.*, 2016] investigated maximizing diversity by adopting different policies for the exploration of the environment by different workers. [Lillicrap *et al.*, 2015] introduced DDPG algorithm, which is based on an actor-critic architecture for representing the policy and value function and makes use of replay buffer to stabilize learning. Recently, [Barth-Maron *et al.*, 2018], proposed D3PG and D4PG, as distributed versions of DDPG. Under ApeX framework [Horgan *et al.*, 2018], D3PG decouples acting and learning and shares the replay buffer among parallel actors. At each time step, a worker samples a batch of transitions from the shared replay buffer and computes the gradients. Distributed Distributional DDPG (D4PG) [Barth-Maron *et al.*, 2018] is similar to D3PG except it uses the categorical distribution to model the critic function.

In environments with multiple agents, an RL model can incorporate interaction between multiple agents in competitive or cooperative settings. By increasing the number of agents, the problem complexity enhances exponentially that makes applying traditional RL approaches in these environments to be infeasible. To address this challenge, deep multi-agent RL approaches have emerged [Nguyen *et al.*, 2018; Palmer *et al.*, 2018; Barati *et al.*, 2019]. Multi-agent RL approaches use different strategies such as optimistic and hysteretic Q-function updates [Lauer and Riedmiller, 2000; Barbalios and Tzionas, 2014; Omidshafiei *et al.*, 2017]. These methods are in cooperative settings where the actions of other agents are made to improve collective reward. [Palmer *et al.*, 2018] applied leniency to deep Q-network [Mnih *et al.*, 2015] in a cooperative setting with noisy observations. However, there are a number of multi-agent RL methods that are trained in competitive or mixed settings [Bansal *et al.*, 2018].

In this work, we take into account different views of an environment provided by different cameras in which each camera is considered as a worker. Similar to multi-agent systems, there is a common environment that provides the observations of the workers, and if one or multiple workers fail, the remaining workers can take over the task. Moreover, workers help each other to improve their performance in decision making. Our method differs from the multi-agent methods such as [Lowe *et al.*, 2017; Peng *et al.*, 2017] in that all workers take the same action as the final decision while in multi-agent systems, each agent can take its own action. Besides, our method uses the worker's reward to determine the importance of that worker in the final decision-making process. Similar to distributed methods [Barth-Maron *et al.*, 2018; Mnih *et al.*, 2016], our method achieves a better training performance of its deep network by utilizing multiple workers. However, in our method, each worker has a distinct view of the environment, while in the distributed methods all workers are fed with the same view of the environment.

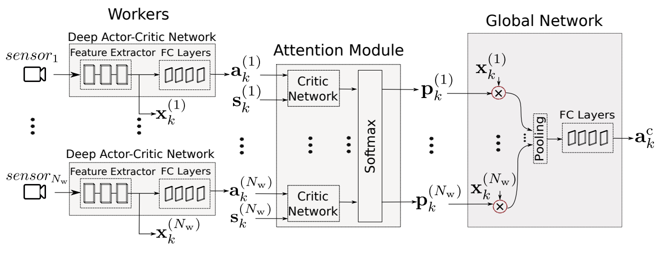


Figure 1: The architecture of the deep network that leverages the attention mechanism in its global network. $\mathbf{p}_k^{(w)}$ is the weight of worker w obtained from the attention module according to the importance of its view.

3 Method

We propose a method to train policy networks by leveraging multiple observations of the same environment. We adopt multiple sensory inputs that each provides a distinct observation of the RL environment. In the first step of our learning process, we train multiple workers that each selects an action given a single view of the environment, while in the second step, a single agent learns a policy given all the views of the environment in the global network to make the final decision.

3.1 Attention-based RL Framework

Our model (depicted in Figure 1) is trained in two stages and contains the following two components: (1) the workers' networks that are separately trained at the first stage and retrained at the second stage and (2) the global network that is trained at the second stage by using the feature representations obtained from the workers' networks. The decisions made by the workers and their corresponding states are leveraged in an attention module to measure the importance of their associated views. The workers' networks, the global network, and the attention mechanism rely on an actor-critic architecture.

By utilizing an attention weighted representation as introduced in [Bahdanau *et al.*, 2015], our method incorporates the importance of the views in computing a unit representation of the environment ($\mathbf{x}_k$). The attention module has inputs from the critic networks of the workers and gives the highest weight to a view that its corresponding critic network computes the highest Q-value. We use a softmax gate function to compute the attention weights ($\mathbf{p}_k^{(w)}$) as:

$$\mathbf{x}_k = \sum_{w=1}^{N_\mathrm{w}} \mathbf{p}_k^{(w)} \odot \mathbf{x}_k^{(w)} = \sum_{w=1}^{N_\mathrm{w}} \frac{\exp(g_w f_w)}{\sum_l \exp(g_l f_l)} \odot \mathbf{x}_k^{(w)} \ , \quad (1)$$

where $\odot$ stands for Hadamard product, $\mathbf{x}_k^{(w)}$ is the feature representation obtained by worker w, and g_w is a parameter of the model. In the above equation, f_w is obtained from the output of critic network learned for each worker w separately, i.e.,

$$f_w = Q(s^{(w)}, a^{(w)}) \ . \quad (2)$$

At each time step k, the attention mechanism generates a positive weight $\mathbf{p}_k^{(w)}$ for each worker which can be interpreted as the probability that the worker w makes a correct decision. The attention weight $\mathbf{p}_k^{(w)}$ determines the relative importance of worker w in blending the feature vectors $\{\mathbf{x}_k^{(w)} | w = 1, 2, \ldots, N_\mathrm{w}\}$ together.

We aim to promote the behavior of each worker by comparing its selected action with the actions selected by all the other workers. To do so, we modify the reward at the second stage of training by introducing a penalty term that depends on the actions selected by all the workers (A_k) as:

$$\begin{aligned} r_k^\mathrm{c} &= r_k - \gamma_\mathrm{r} \delta(A_k) \\ &= r_k - \gamma_\mathrm{r} \frac{1}{N_\mathrm{w}} \sum_{w=1}^{N_\mathrm{w}} \delta^{(w)}(A_k) \end{aligned} \quad (3)$$

where r_k is the original reward, γ_r is a constant that provides a trade-off between the deviation of actions and the original reward, and A_k is the action matrix with columns $\mathbf{a}_k^{(w)}$. In (3), $\delta(A_k)$ is a deviation function that depends on the variation of action $\mathbf{a}_k^{(w)}$ of worker w from the average of actions selected by the other workers in the network ($\delta^{(w)}(A_k)$), i.e.,

$$\delta^{(w)}(A_k) = \left|\left| \mathbf{a}_k^{(w)} - \frac{1}{N_\mathrm{w} - 1} \sum_{\substack{v=1 \\ v \neq w}}^{N_\mathrm{w}} \mathbf{a}_k^{(v)} \right|\right|^2 \ . \quad (4)$$

In (4), the first term is the action of worker w while the second term is the average action of other workers. This assumption is valid if the majority of views provide enough information to make the right decision. Some of the views can have lower quality and if we train them individually their corresponding workers might not be trained properly. In the case that all workers have trained sufficiently, the deviation $\delta(A_k)$ becomes close to zero, while by experiencing weak training performance from a worker, we get a higher $\delta(A_k)$ and a lower reward r_k^c. Therefore, the penalty term in (3) enforces improvement in the training of the workers that are yet to be trained sufficiently. We provide more details regarding the training of the workers in Section 4.2. We consider γ_r in (3) to be 0.1 in our experiments. While our method utilizes the actions of the workers in computing the modified reward, these actions do not determine the final action. As described earlier, the final action is determined by the global network from an aggregation of feature representations obtained from the workers.

Given the state $\mathbf{s}_k^\mathrm{c}$ that we set to be $\mathbf{x}_k$ from (1) and the modified reward r_k^c from (3), the parameters of the network including weights of fully-connected (FC) layers of global network (θ^c) and workers ($\{\theta^{(w)}\}_{w=1}^{N_\mathrm{w}}$) and the weights of attention module are trained by using an actor-critic algorithm [Lillicrap *et al.*, 2015]. At each time step k, we stack the modified reward r_k^c on the replay buffer, forming a tuple $\prec \mathbf{s}_k^\mathrm{c}, \mathbf{a}_k^\mathrm{c}, r_k^\mathrm{c}, \mathbf{s}_{k+1}^\mathrm{c} \succ$. Then, we sample a random mini-batch from the replay buffer to train critic network by minimizing the following loss function:

$$\mathcal{L}(\theta_k^{Q^\mathrm{c}}) = \mathbb{E}\Bigg(r_k^\mathrm{c} + \gamma \overline{Q}(\mathbf{s}_{k+1}^\mathrm{c}, \overline{\mu}^\mathrm{c}(\mathbf{s}_{k+1}^\mathrm{c}; \theta^{\overline{\mu}^\mathrm{c}}); \theta^{\overline{Q}^\mathrm{c}}) - Q(\mathbf{s}_k^\mathrm{c}, \mathbf{a}_k^\mathrm{c}; \theta_k^{Q^\mathrm{c}}) \Bigg)^2 \ . \quad (5)$$

We update the actor network at each time step with respect to the set of parameters $\theta_k^{\mu^\mathrm{c}}$ by using sampled policy gradient:

$$\nabla_{\theta^{\mu_k^\mathrm{c}}} J = \mathbb{E}\Bigg(\nabla_\mathbf{a} Q(\mathbf{s}, \mathbf{a}; \theta_k^{Q^\mathrm{c}})|_{\mathbf{s}=\mathbf{s}_k^\mathrm{c}, \mathbf{a}=\mu^\mathrm{c}(\mathbf{s}_k^\mathrm{c})} \nabla_{\theta_\mu^\mathrm{c}} \mu^\mathrm{c}(\mathbf{s}; \theta_k^{\mu^\mathrm{c}})|_{\mathbf{s}=\mathbf{s}_k^\mathrm{c}} \Bigg) \ . \quad (6)$$

We construct the exploration policy by adding a Gaussian noise to the selected action through the policy $\pi(\mathbf{x})$ of the workers and the global network as $\mathbf{a} = \pi(\mathbf{x}) + \epsilon \mathcal{N}(\mathbf{0}, \mathbf{1})$. To promote the behavior of each worker, we control the value of ϵ for that worker based on the deviation of its actions from the average of actions selected by the other workers. To do so, we choose the value of $\epsilon^{(w)}$ for worker w to be linearly proportional to the average of $\delta^{(w)}(A_k)$ in T_w instances. We choose $T_\mathrm{w} = 1000$ in the experiments. We also compute ϵ^c for exploration during the second stage of training from the average $\epsilon^{(w)}$ of all workers.

We train the network in M iterations. Each of these iterations contains M_w episodes of training workers (first stage of training) and M_c episodes of training the global network and retraining the FC layers of workers (second stage of training). M, M_w, and M_c are hyper-parameters of the model. Before beginning the training of the global network, the workers are trained separately. M_w should be large enough such that the decision made by the majority of the workers can be used in the global network to improve the behavior of faulty workers. Following this strategy, we train the task-dependent layers of workers (fully-connected layers) by taking all workers into account; However, we train the view-dependent layers (convolutional layers) of a worker independent from the other workers.

4 Experiments

4.1 Baselines

We compare the performance of our method, called Attention-based Deep RL (`ADRL`), with the performances of state-of-the-art actor-critic based methods that are designed to work on continuous action spaces. `DDPG` [Lillicrap *et al.*, 2015], `D3PG` [Barth-Maron *et al.*, 2018], `PPO` [Schulman *et* *al.*, 2017], and A3C [Mnih *et al.*, 2016] are four of the baselines in this work. For D3PG and A3C, we dedicated the same number of workers as ADRL (4 workers) to obtain a stabilized training of the network. However, in D3PG and A3C, we fed the copies of a single view of an environment into their multiple workers. In other words, each worker interacts with its copy of the environment. However, all the workers use the same camera (front view camera) to obtain observations from their own copy of the environment. We also propose four extensions to DDPG called ACT-AVG, ACT-CNT, ACT-MJV, and FT-COMB and utilize them to provide ablation studies to verify the design choices in the proposed framework. These four extensions of DDPG adopt multiple workers, each with a different view of an environment, trained by using DDPG. Inspired from [Wiering and Van Hasselt, 2008], during the training of ACT-AVG, ACT-CNT, and ACT-MJV, the final action is obtained from a combination of actions that the workers take. In ACT-AVG, we take an average of decisions made by workers as the final decision. In ACT-CNT, we find the action vector with the least Euclidean distance from other action vectors. In ACT-MJV, we divide every dimension of the action space into equally distant ten bins and find the bin that contains the maximum number of actions taken by the workers. However, FT-COMB is different from others as its states are obtained from the combined features of all views provided by different workers. The final representation is the concatenation of all views features. Unlike DDPG, D3PG, PPO, and A3C which work with only a single view of the environment, ACT-AVG, ACT-CNT, ACT-MJV, FT-COMB, and ADRL make use of multiple views in which each view is represented by high-dimensional raw pixels.

4.2 Experimental Setup

TORCS Environment. We verified our method for autonomous driving on an open-source platform for car racing called TORCS [Wymann *et al.*, 2000]. We used six different tracks: five tracks for training named in TORCS version 1.3.4 as: (1) *CG Speedway number 1*, (2) *CG track 2*, (3) *E-Track 1*, (4) *E-Track 2*, and (5) *Brondehach*, and one track (*Wheel 2*) for testing. We modified road surface textures in TORCS to have the same lane types for all training and testing tracks. In our experiments, we consider the following actions: acceleration, braking, and steering. Our objective is to maximize the velocity of the car on the longitudinal axis of the road, minimize the velocity of the car on the lateral axis, and minimize the distance of the car from the road center. Similar to [Lillicrap *et al.*, 2015], we define the reward function r_k for an agent that takes the action $\mathbf{a}_k$ at time-step k to move from a state to another.

MuJoCo Environment. We also compare the performance of our method with the baselines on three challenging tasks with the continuous action space Ant-Maze, Hopper-Stairs, and Walker-Wall developed in the MuJoCo physics simulator [Todorov *et al.*, 2012]. In Ant-Maze task, a 3D maze contains five walls, and a MuJoCo ant must navigate from start to the goal state. Hopper-Stairs task includes three upward and three downward stairs in a 3D environment where the MuJoCo hopper agent must hop over them. In Walker-Wall task, the 3D environment contains three walls that the MuJoCo walker2d must jump or slide over. Since we train ADRL and its baselines, given high-dimensional row pixels, it is computationally expensive to achieve the converge to the stable state for all the methods; Therefore, we simplified the texture by modifying the color of the background, walls and body parts of agents. The action dimension of the Ant-Maze, Hopper-Stairs, and Walker-Wall are, respectively, 8, 3, and 6. The reward is chosen to encourage the agent to reduce the distance from the goal state and increase the forward velocity. In the Ant-Maze environment, the reward also encourages the agent to keep a distance from the walls. In the Hopper-Stairs, the agent is also rewarded based on the torso height.

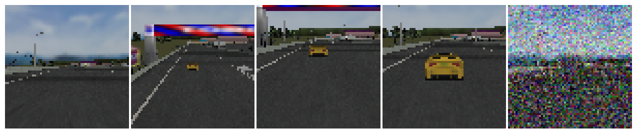


Figure 2: Four camera views used in this paper plus one with perturbed image sample. In TORCS version 1.3.4, these four camera views (from left to right) are named: (1) *inside car (front view)*, (2) *behind far*, (3) *behind near*, and (4) *car behind* (distance= 30). The last camera view belongs to Camera 1 but perturbed with random Gaussian noise with variance 0.02.

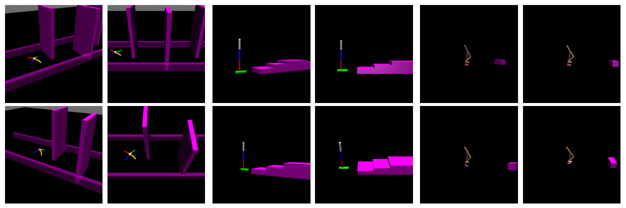
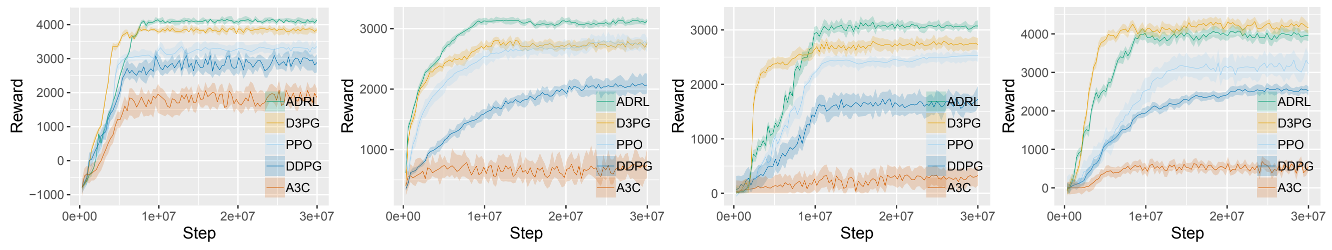


Figure 3: Four camera views: *left view*, *front view*, *right view*, and *top view* (shown in clockwise order) obtained from three MuJoCo based environments: Ant-Maze, Hopper-Stairs, and Walker-Wall. We obtain the left, right and top views of the environment by 30° changes in the camera angle.

Multiple Views of Environment

To provide multiple views of environments described by high-dimensional raw pixels, we obtained four different camera views from each environment (depicted in figures 2 and 3). Particularly, in MuJoCo environment, we obtained these camera views through 30° changes in the camera angle of the front view, while in TORCS environment, we use the pre-defined camera views shown in Figure 2. In the TORCS environment, the front view camera, unlike the other cameras, does not capture the car itself. The Ant-Maze task provides the most diverse camera views because of the complexity of the environment, while the Walker-Wall task provides the least diverse camera views.

In all the experiments, for the baselines that take a single view of the environment, i.e., DDPG, D3PG, PPO, and A3C, we adopted the *front view* camera. Because we observed that this view provides more useful information that the other views. We used the four camera views in figures 2 and 3 for ADRL and the baselines that take multiple views of the environment, i.e., ACT-AVG, ACT-CNT, ACT-MJV, and FT-COMB. We also performed experiments in the cases



(a) TORCS (b) Ant-Maze (c) Hopper-Stairs (d) Walker-Wall

Figure 4: Average reward vs. training step for the methods DDPG, D3PG, PPO, A3C, and ADRL. We obtain the rewards in these figures by averaging rewards obtained from 5 runs.

of (1) having a random Gaussian noise $\mathcal{N}(0, \sigma^2_{E,n})$ on all of the camera views, (2) having one of the cameras perturbed by the noise $\mathcal{N}(0, \sigma^2_{P,n})$ with variance $\sigma^2_{P,n}$ higher than $\sigma^2_{E,n}$ (see the last view in Figure 2), and (3) having one or more irrelevant views.

Setup. We down-sampled the RGB frames into 84×84 pixels and normalized them to the range of $[0, 1]$. We used three convolutional layers, each with 32 filters as the feature extractor module shown in Figure 1 followed by two fully connected layers with 600 and 400 units. All hidden layers are followed by ReLUs. We employed the same architecture for the fully-connected layers in the policy networks of workers and the global network. We set the size of replay memory to be 10^5, the value of discount factor $\gamma = 0.99$, and the learning rates of actor and critic networks to be 10^{-4} and 10^{-3}. To implement and train the model, we used Tensorflow [Abadi *et al.*, 2016] distributed machine learning system and applied Adam optimization [Kingma and Ba, 2015] for learning the neural network parameters with the batch size of 32. For ADRL, we performed $M = 100$ iterations of training workers and the global network. At the initial training iteration, we dedicated more episodes to train the workers than the global network (i.e., $M_{\text{w}} = 10 \times M_{\text{c}}$), and gradually modified the portion of episodes in favor of training the global network such that at the last training iteration $M_{\text{w}} = 0.1 \times M_{\text{c}}$.

4.3 Results and Discussion

Training Speed Comparison. We present the training speed of ADRL and four of its baselines in terms of average reward per step in Figure 4. At the first stage of training ADRL, multiple workers are trained separately given multiple views of the environment, and the reward of ADRL shown in Figure 4 is the average reward of all these workers. Since D3PG uses all its workers to train the network given a single view of the environment, at the initial steps of training, the reward of ADRL is less than D3PG for all the tasks other than Ant-Maze. However, after convergence, the reward of ADRL is either higher or comparable to D3PG in all the tasks, which is an indication of the advantage of attending to multiple various views of the environment. In Ant-Maze task, different camera views (shown in Figure 3) provide significantly diverse information about the environment, and leveraging these views via the attention mechanism in ADRL leads to a significantly higher reward for ADRL in comparison to its baselines. In Walker-Wall task, the reward of ADRL is slightly less than D3PG, which is due to having a low diversity between the views.

Comparison with Baselines. We report the performance of ADRL in comparison to its baselines in Table 1. After learning the parameters of all methods in 30 million training steps, we included background noise $\mathcal{N}(0, \sigma^2_{E,n})$ with $\sigma^2_{E,n} = 0.01$ to all the camera views and capture the time and distance that the agent moves. In Table 1, we present the mean μ and standard deviation σ of the time and distance that the agent moves, computed over 100 runs. In this table, the values of time and distance are normalized with the maximum time and distance that the agent moves forward before termination. ADRL surpasses all the baselines with respect to mean and standard deviation of time and distance on all tasks except Walker-Wall. In comparison with the other tasks, in Walker-Wall, different views have the least amount of diversity; therefore, the performance of ADRL is comparable to its baselines in this task. Since FT-COMB does not utilize an attention mechanism, it has its best performance in the Walker-Wall task where there is a small difference in the importance of different views. ADRL and the baselines ACT-AVG, ACT-CNT, ACT-MJV, and FT-COMB employ multiple workers fed with different views of environment; however, unlike ADRL, these four baselines can only beat DDPG and A3C. This is mainly an indication of improvement due to the attention mechanism and the policy network used in ADRL. As a result of averaging actions made by multiple workers trained on multiple views, we observe that ACT-AVG, ACT-CNT, and ACT-MJV have lower variances than DDPG; nevertheless, its variance is still higher than ADRL. ADRL has the highest improvement over D3PG (>20%) in the Ant-Maze task, which has the highest diversity in its views. This suggests that ADRL has higher improvement when there is higher diversity among views.

Irrelevant views Impact. We examine the stability of ADRL in the case of existing a number of irrelevant views. We trained ADRL for the case of having four cameras that are located in the same way described in subsection 4.2. After 30 million steps of training, we made camera views irrelevant by randomly changing the camera angles to ones that have no view of the agent. In the TORCS environment, we captured the irrelevant view by a side camera along the road that provides a view of the car only in a very limited period of time.

Method	Time (sec)	Distance (m)
	μ / σ	μ / σ
DDPG	0.71 / 0.09	0.64 / 0.10
ACT-AVG	0.82 / 0.06	0.68 / 0.05
ACT-CNT	0.80 / 0.07	0.68 / 0.07
ACT-MJV	0.74 / 0.07	0.66 / 0.08
FT-COMB	0.73 / 0.12	0.65 / 0.13
A3C	0.45 / 0.11	0.48 / 0.14
PPO	0.83 / 0.04	0.71 / 0.05
D3PG	0.89 / 0.03	0.82 / 0.03
ADRL	0.92 / 0.03	0.90 / 0.03

(a) TORCS

Method	Time (sec)	Distance (m)
	μ / σ	μ / σ
DDPG	0.41 / 0.12	0.43 / 0.14
ACT-AVG	0.63 / 0.10	0.57 / 0.11
ACT-CNT	0.58 / 0.11	0.55 / 0.11
ACT-MJV	0.57 / 0.11	0.53 / 0.09
FT-COMB	0.47 / 0.17	0.46 / 0.16
A3C	0.18 / 0.12	0.19 / 0.10
PPO	0.59 / 0.06	0.60 / 0.08
D3PG	0.61 / 0.07	0.58 / 0.08
ADRL	0.76 / 0.06	0.70 / 0.05

(b) Ant-Maze

Method	Time (sec)	Distance (m)
	μ / σ	μ / σ
DDPG	0.54 / 0.08	0.52 / 0.11
ACT-AVG	0.64 / 0.07	0.60 / 0.07
ACT-CNT	0.60 / 0.07	0.57 / 0.06
ACT-MJV	0.61 / 0.08	0.58 / 0.08
FT-COMB	0.56 / 0.18	0.56 / 0.13
A3C	0.21 / 0.12	0.18 / 0.09
PPO	0.59 / 0.04	0.60 / 0.05
D3PG	0.73 / 0.09	0.76 / 0.07
ADRL	0.82 / 0.04	0.80 / 0.03

(c) Hopper-Stairs

Method	Time (sec)	Distance (m)
	μ / σ	μ / σ
DDPG	0.40 / 0.14	0.40 / 0.15
ACT-AVG	0.48 / 0.12	0.42 / 0.14
ACT-CNT	0.43 / 0.13	0.38 / 0.14
ACT-MJV	0.42 / 0.12	0.38 / 0.13
FT-COMB	0.44 / 0.15	0.39 / 0.15
A3C	0.15 / 0.07	0.14 / 0.06
PPO	0.49 / 0.15	0.47 / 0.16
D3PG	0.71 / 0.10	0.70 / 0.11
ADRL	0.70 / 0.12	0.68 / 0.12

(d) Walker-Wall

Table 1: Performance comparison of ADRL with its baselines on TORCS and three MuJoCo environments. μ and σ are the mean and standard deviation of the time/distance that an agent moves before termination computed over 100 runs. The values of time and distance in this table are normalized with the maximum time and distance the agent travels in all of the runs.

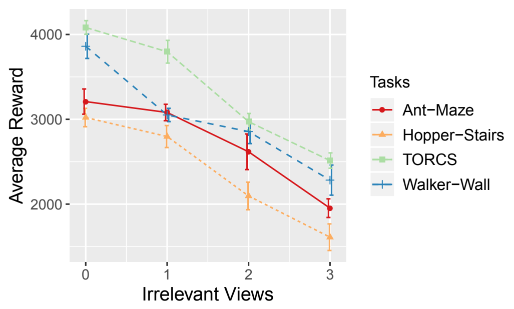


Figure 5: Impact of irrelevant views in average reward.

Method/ Feature Level	$\sigma^2_{E,n}$	$\sigma^2_{P,n}$	N_w	Time (sec)	Distance (m)
				μ / σ	μ / σ
ADRL	0.01	—	1	0.73 / 0.08	0.67 / 0.10
ADRL	0.05	—	1	0.14 / 0.04	0.15 / 0.05
ADRL	0.01	—	4	0.92 / 0.03	0.90 / 0.03
ACT-AVG	0.01	—	4	0.82 / 0.06	0.78 / 0.05
ADRL	0.05	—	4	0.29 / 0.12	0.32 / 0.14
ACT-AVG	0.05	—	4	0.18 / 0.07	0.18 / 0.08
ADRL	0.01	0.05	4	0.84 / 0.08	0.80 / 0.07
ACT-AVG	0.01	0.05	4	0.77 / 0.19	0.74 / 0.15

Table 2: Noise perturbation impact on the mean (μ) and standard deviation (σ) of time and distance a car travels without leaving the road by using ADRL and ACT-AVG. We normalize the values of time and distance in this table. $\sigma^2_{E,n}$ and $\sigma^2_{P,n}$ are the environment noise and noise on a perturbed sensor.

Figure 5 demonstrates the average testing rewards over 5 runs for the case of having 0, 1, 2, and 3 irrelevant camera views. We observed that for the cases of having at least one relevant camera, the testing rewards were satisfactory. Specifically, by making three out of four cameras irrelevant, the average rewards decrease around 37% in all the tasks. This observation indicates that the attention module is able to adjust the weight of the irrelevant camera to reduce deterioration in the average reward. From Figure 5 we observe that by having one irrelevant camera, there is a slight decrease in the average testing reward (around 7%). From Figure 5 and Figure 4, we can conclude that ADRL works better than DDPG in the case of having only one relevant camera. In other words, due to the penalty term considered in the training reward of ADRL, each of the workers gets better training than the case of training them individually.

Noise Perturbation Impact. We examine the performance of ADRL in the TORCS environment under a scenario in which one out of four views of the environment comes with less quality due to having a randomly chosen camera perturbed by noise with higher variance than the others. We used the first four camera views listed in Figure 2 to obtain the high-dimensional views. In this scenario, we adopted single and multiple views (shown in Table 2) to compare ADRL with the best performing baseline in Table 1 that takes multiple different views (i.e., ACT-AVG). Using only a single view, ADRL does not need any attention mechanism over the views, which makes its performance equivalent to the performance of DDPG. Table 2 summarizes the mean and the standard deviation of the time and distance that a car travels until it leaves the road. We added zero-mean Gaussian noise with the variances 0.01 and 0.05 to represent high- and low-quality views, respectively. We observe the superior performance of ADRL over ACT-AVG in all the cases in Table 2. We observe that the worst performance occurs when all the views contain high-variance noise. We can conclude that ADRL is more successful than ACT-AVG in dealing with the case of having a low-quality view.

5 Conclusion

We proposed an attention-based deep reinforcement learning method for multi-view environments. Our method takes advantage of multiple views of the environment with a different exploration strategy for each view to obtain a stabilized training policy. This method dynamically attends to views of the environment according to their importance in the final decision-making process where we use a critic network designated for each view to measure the importance of the view. Through the experiments, we observed that our method outperforms its baselines that use single or multiple views of the environment. The experiments also reveal the superior performance of our method in the face of a degraded view of the environment. As future work, we will investigate incorporating memory components in the attention module to model temporal changes in the quality of views of the environment.

References

[Abadi *et al.*, 2016] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[Bahdanau *et al.*, 2015] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

[Bansal *et al.*, 2018] Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent complexity via multi-agent competition. In *ICLR*, 2018.

[Barati *et al.*, 2019] Elaheh Barati, Xuewen Chen, and Zichun Zhong. Attention-based deep reinforcement learning for multi-view environments. In *AAMAS*, pages 1805–1807, 2019.

[Barbalios and Tzionas, 2014] Nikos Barbalios and Panagiotis Tzionas. A robust approach for multi-agent natural resource allocation based on stochastic optimization algorithms. *Applied Soft Computing*, 18:12–24, 2014.

[Barth-Maron *et al.*, 2018] Gabriel Barth-Maron, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients. In *ICLR*, 2018.

[Heess *et al.*, 2017] Nicolas Heess, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, Ali Eslami, Martin Riedmiller, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[Horgan *et al.*, 2018] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. In *ICLR*, 2018.

[Kingma and Ba, 2015] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[Lauer and Riedmiller, 2000] Martin Lauer and Martin Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *ICML*, pages 535–542, 2000.

[Lillicrap *et al.*, 2015] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2015.

[Lowe *et al.*, 2017] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *NIPS*, pages 6379–6390, 2017.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[Mnih *et al.*, 2016] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, pages 1928–1937, 2016.

[Nguyen *et al.*, 2018] Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. Deep reinforcement learning for multi-agent systems: A review of challenges, solutions and applications. *arXiv preprint arXiv:1812.11794*, 2018.

[Omidshafiei *et al.*, 2017] Shayegan Omidshafiei, Jason Pazis, Christopher Amato, Jonathan P How, and John Vian. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *ICML*, pages 2681–2690, 2017.

[Palmer *et al.*, 2018] Gregory Palmer, Karl Tuyls, Daan Bloembergen, and Rahul Savani. Lenient multi-agent deep reinforcement learning. In *AAMAS*, pages 443–451, 2018.

[Peng *et al.*, 2017] Peng Peng, Ying Wen, Yaodong Yang, Quan Yuan, Zhenkun Tang, Haitao Long, and Jun Wang. Multiagent bidirectionally-coordinated nets: Emergence of human-level coordination in learning to play starcraft combat games. *arXiv preprint arXiv:1703.10069*, 2017.

[Schulman *et al.*, 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[Silver *et al.*, 2014] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, pages 387–395, 2014.

[Tassa *et al.*, 2018] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[Todorov *et al.*, 2012] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pages 5026–5033. IEEE, 2012.

[Wiering and Van Hasselt, 2008] Marco A Wiering and Hado Van Hasselt. Ensemble algorithms in reinforcement learning. *SMC*, 38(4):930–936, 2008.

[Wymann *et al.*, 2000] Bernhard Wymann, Eric Espié, Christophe Guionneau, Christos Dimitrakakis, Rémi Coulom, and Andrew Sumner. Torcs, the open racing car simulator. *Software available at http://torcs. sourceforge. net*, 2000.