Multi-Agent Deep Reinforcement Learning

Zahra Parham* Mohammad Akbari

Amirkabir University Of Technology, Computer Science Department, Iran

ABSTRACT

Keywords: Deep Reinforcement Learning, Multi-agent, Deep Learning.

1 INTRODUCTION

Reinforcement learning is a computational approach to understanding and automating goal-directed learning and decision-making. It is distinguished from other computational approaches by its emphasis on learning by an agent from direct interaction with its environment, without relying on exemplary super- vision or complete models of the environment[8].
Reinforcement learning uses a formal framework defining the interaction between a learning agent and its environment in terms of states, actions, and rewards. This framework is intended to be a simple way of representing essential features of the artificial intelligence problem. These features include a sense of cause and effect, a sense of uncertainty and nondeterminism, and the existence of explicit goals[8].

Due to the astronomic number of states of any realistic scenario, for a long time algorithms implementing reinforcement learning were either limited to simple environments or needed to be assisted by additional information about the dynamics of the environment.
Recently, however, the Swiss AI Lab IDSIA [2] and Google DeepMind [6] have produced spectacular results in applying reinforcement learning to very high-dimensional and complex environments such as video games.
In particular, DeepMind demonstrated that AI agents can achieve superhuman performance in a diverse range of Atari video games.
Remarkably, the learning agent only used raw sensory input (screen images) and the reward signal (increase in game score). The proposed methodology, so called Deep Q-Network, combines a convolutional neural network for feature representation with Q-learning training [4].
It represents the state-of-the-art approach for model free reinforcement learning problems in complex environments. The fact that the same algorithm was used for learning very different games suggests its potential for general purpose applications.[11]

The present article builds on the work of Agorov[1] and explores how multiple agents controlled by autonomous Deep Q-Networks interact when sharing a complex environment.
Multiagent systems appear naturally in most of social, economical and political scenarios. Indeed, most of game theory problems deal with multiple agents taking decisions to maximize their individual returns in a static environment [5]. Phenomena such as cooperation, communication, and competition may emerge in reinforced multiagent systems[11].
A multi-agent system consists of a number of intelligent agents that interact with other agents in a multi-agent environment. An agent is an autonomous entity that observes the environment and takes an action to satisfy its own objective based on its knowledge. The agents in a multi-agent system can be software agents or physical agents such as robots [3].
Unlike a stationary single-agent environment, the multi-agent environment can be complex and dynamic. The agents in a multi-agent environment may not have a priori knowledge of the correct actions or the desired policies to achieve their goals. In a multi-agent environment, each agent may have independent goals. The agents need to learn to take actions based on their interaction with other agents. Learning is the essential way of obtaining the desired behavior for an agent in a dynamic environment[5].

The purpose of the work ahead is to study emergent competitive strategies between multiple agents controlled by autonomous Deep Q-Networks. we used the pursuit-evasion game in a two-dimensional environment. to multi-agent systems with this state representation.
The reformulation involves decomposing the global system state into an image like representation with information encoded in separate channels. This reformulation allows us to use convolutional neural networks to efficiently extract important features from the image-like state. We extend the state-of-the-art approach for solving DRL problems[10].
In particular, our algorithm uses the image-like state representation of the multi-agent system as an input, and outputs the estimated Q-values for the agent in question. We describe a number of implementation contributions that make training efficient and allow agents to learn directly from the behavior of other agents in the system.[1]

2 RELATED WORK

Combining neural network and reinforcement learning has become a new approach to solve problems in this field. One of the researches that were done in this field that attracted everyone's attention, was defeat of Go champion by artificial intelligence and other trained agents that reached a superhuman performance in Atari video games like Pong.

In 2015, Tampuu et al[11]. introduced a neural network-based method for playing pong. They extend the Deep Q-Learning Network architecture to multiagent environments and show that how we can manipulate the classical rewarding scheme of Pong to bring out competitive and collaborative behaviors of agents.
In fact, in competitive mode, the agent learns to score the most points. In cooperative mode, they learn to keep the game going on and keep the ball in play as long as possible.
In this work, they use an autonomous Q-Learning algorithm for each agent in the environment because of its simplicity, decentralized nature, computational speed, and consistent result.
The agents receive only the raw screen images and respective reward signals as input.
In the Pong game, we have natural competitive mode but by changing the reward function its changes the competitive mode to cooperative mode. In the traditional rewarding scheme of Pong, the player who last touches the outgoing ball gets a plus point and the player losing the ball a minus point. This makes it essentially a zero-sum

*e-mail: mina.parham766@gmail.com

game, where a positive reward for the left player means negative reward for the right player and vice versa.
They call this fully competitive mode as the aim is to simply try to put the ball behind your opponent. but for cooperative mode, they penalize both of the players whenever the ball goes out of play. Which of the players let the ball pass does not matter and no positive rewards are given.[11].
In this work agents learn for 50 epochs, 250000 time steps each. The exploration rate decrease from an initial 1.0 to 0.05 in million time steps and stays fixed at that value.
Finally, in both competitive and cooperative mode, great outcomes are obtained and the results show that in competitive mode, the ball bounces between two players nearly 8 times before one of them scores a point, and in cooperative mode, this number reaches more than 600. In this game, the agents also reached interesting results, including realizing that the more the ball hits the walls, the more likely their opponent is to score points, so over time they tried to hit the ball much less to the walls, and this number was significantly reduced.
on the other hand in the competitive mode as soon as the serving player has relaunched the ball its reward expectation increases slightly. This immediate increase in Q-value makes the agents choose to serve the ball as soon as possible and thus explains the decrease in serving time but in the case of cooperation, the opposite was true and the agents learn to postpone putting the ball into play.

in 2017, silver et al.[10] introduced AlphaGo Zero program that achieved superhuman performance in the game of Go by reinforcement learning from self-play and again in 2018, silver et at.[9] generalized new approach into a single AlphaZero algorithm that can achieve superhuman performance in many challenging games. Starting from random play and given no domain knowledge except the game rules, AlphaZero convincingly defeated a world champion program in the games of chess and shogi (Japanese chess), as well as Go.
This achievement shows that reinforcement learning in games that scientists have been looking for a solution for years has made significant progress and has the ability to compete with humans.

Another paper that demonstrates the application of deep reinforcement learning is presented by Marco Wiering.[12] This paper describes using multi-agent reinforcement learning algorithms to control traffic light and reduce the amount of time that cars wait.
In their modeling, they considered cars and traffic lights to be one agent each. They used model-based RL to learn to estimate waiting times of cars given particular inputs states, a transition model is estimated, and dynamic programming-like methods are used to compute the value function mapping agent states to expected long term reward.

in 2015, Mnih et al.[6] used deep neural networks to develop an agent that can learn policies directly from high-dimensional input using reinforcement learning.
In this study, they showed that by using the deep convolutional network that receives pixels and game score as input, it can achieve remarkable performance. They were able to create a single algorithm that can develope competencies in different games with different policies.
They tested this agent on the challenging domain of classic Atari 2600 games and DQN agent performed at a level that was comparable to that of a professional human games tester across the set of 49 games, achieving more than 75 percentage of the human score on more than half of the games.

And finally, my work is based on a paper that Maxim Egorov published in 2017,[1] , a new approach to solving multi-agent reinforcement learning problems is introduced. They pick the pursuit-evasion problem as their evaluation domain.
In pursuit-evasion, a set of agents (the pursuers) are attempting to chase another set of agents (the evaders). one of the challenges that attempt to solve it is present the problem in a way that you do not need to change the architecture of the neural network with the arbitrary number of agents.
On the other hand, because the input of the neural network is image-like, it is used convolutional neural networks. with each channel of the image containing agent and environment specific information while the out- put layer of the network represents the Q-values for all the actions the agent can take from that state and each agent has their own neural network controller. This work aims to implement a part of it.

3 METHODS

3.1 Reinforcement Learning

RL formalizes the interaction of an agent with an environment using a Markov decision process (MDP). An MDP is defined by the tuple (S, A, R, T, γ) where S represents a finite set of states. A represents a finite set of actions. The transition function $T : S \times A \times S \rightarrow [0, 1]$ determines the probability of a transition from any state $s \in S$ to any state s$\prime \in S$ given any possible action $a \in A$. The reward function $R : S \times A \times S \rightarrow R$ defines the immediate and possibly stochastic reward that an agent would receive given that the agent executes action a while in state s and it is transitioned to state s$\prime$, $\gamma \in [0, 1]$ represents the discount factor that balances the trade-off between immediate rewards and future rewards.[7]

Q-learning. One of the most well known algorithms for RL is Q-learning. It has been devised for stationary, single-agent, fully observable environments with discrete actions. A Q-learning agent keeps the estimate of its expected payoff starting in state s, taking action a as $\hat{Q}$(s,a). Each tabular entry $\hat{Q}(s, a)$ is an estimate of the corresponding optimal $Q\star$ function that maps state-action pairs to the discounted sum of future rewards starting with action a at state s and following the optimal policy thereafter. Each time the agent transitions from a state s to a state $s\prime$ via action a receiving payoff r, the Q table is updated as follows[7]:
[Q(s,a) = Q(s, a) + α(r + γmaxQ(s$\prime$,a$\prime$)$-$Q(s, a))].

3.2 Deep Reinforcement Learning

While tabular RL methods such as Q-learning are successful in domains that do not suffer from the curse of dimensionality, there are many limitations: learning in large state spaces can be prohibitively slow, methods do not generalize (across the state space), and state representations need to be hand-specified. Function approximators tried to address those limitations, using for example, decision trees, tile coding, radial basis functions, and locally weighted regression to approximate the value function. [7]

Similarly, these challenges can be addressed by using deep learning, i.e., neural networks as function approximators. For example, Q(s,a;θ) can be used to approximate the state-action values with θ representing the neural network weights. This has two advantages, first, deep learning helps to generalize across states improving the sample efficiency for large state-space RL problems. Second, deep learning can be used to reduce (or eliminate) the need for manually designing features to represent state information.[7]

The major breakthrough work combining deep learning with Q-learning was the Deep Q-Network (DQN). DQN uses a deep neural network for function approximation and maintains an experience replay (ER) buffer to store interactions (s, a, r, s$\prime$). DQN keeps an additional copy of neural network parameters, $\hat{\theta}$, for the target network in addition to the θ parameters to stabilize the learning, i.e., to alleviate the non-stationary data distribution. For each training iteration i, DQN minimizes the mean-squared error (MSE) between the Q-network and its target network.[7]

where target network parameters $\hat{\theta}$ are set to Q-network parameters θ periodically and mini-batches of (s, a, r, s′) tuples are sampled from the ER buffer, as depicted in Figure 1.

3.3 Multiagent Deep Reinforcement Learning (MDRL)

In a multi-agent environment, each agent may have independent goals. The agents need to learn to take actions based on their interaction with other agents. Learning is the essential way of obtaining the desired behavior for an agent in a dynamic environment. Different from supervised learning, there is no external supervisor to guide the agent's learning process. The agents have to acquire the knowledge of their desired actions themselves by interacting with the environment.[5]

There exist several possible adaptations of the Q-learning algorithm for the multiagent case. In practice the simplest method consists of using an autonomous Q-learning algorithm for each agent in the environment, thereby using the environment as the sole source of interaction between agents. In this work we use this method due to its simplicity, decentralized nature, computational speed, and ability to produce consistent results for the range of tasks we report. Therefore, in our tests each agent is controlled by an independent Deep Q-Network with architecture and parameters as reported in.[11]

4 PROBLEM FORMULATION AND NETWORK ARCHITECTURE

Layer (type)	Output Shape
conv2d_1 (Conv2D)	(None, 8, 8, 256)
activation_1 (Activation)	(None, 8, 8, 256)
max_pooling2d_1 (MaxPooling2	(None, 4, 4, 256)
dropout_1 (Dropout)	(None, 4, 4, 256)
conv2d_2 (Conv2D)	(None, 2, 2, 256)
activation_2 (Activation)	(None, 2, 2, 256)
max_pooling2d_2 (MaxPooling2	(None, 1, 1, 256)
dropout_2 (Dropout)	(None, 1, 1, 256)
flatten_1 (Flatten)	(None, 256)
dense_1 (Dense)	(None, 64)
dense_2 (Dense)	(None, 9)

Total params: 614,281
Trainable params: 614,281
Non-trainable params: 0

Figure 2: The multi-agent deep Q-Network architecture.

Our main goal in this work is to choose a simple game environment so that we can evaluate the performance of multi-agent reinforcement learning on it.
For this purpose, we picked the pursuit-evasion problem. We will refer to the two types of agents as thief and police. The police are attempting to catch the thief and the thief is trying to escape.
Since we limit the problem to a discrete state space, the police receive a reward anytime they occupy the same grid cell as a thief. Each of the agents can go to the 8 houses around them or stay in place and do not move.
If the police capture the thief, it gets 25 points and the thief loses 25 points. On the other hand, for each step that the police take, one point is lost and the thief gets one point.
This makes it essentially a zero-sum game, where a positive reward for the police player implies a negative reward of the same size for the thief player and vice versa.[11]

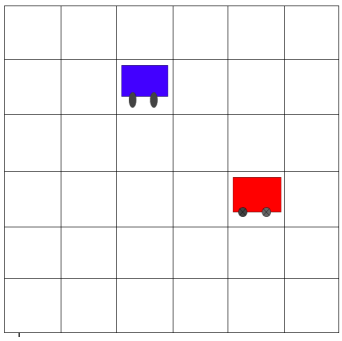

Figure 3: Where the blue robot as a police is trying to catch the red robot that we considered as a thief. Each of the agents can go to the 8 houses around them or stay in place and do not move.

In this work, we consider each player as an agent which decides what action to take in each state to receive the maximum reward. we use a deep convolutional neural network to approximate the optimal action-value function.
each agent has their own convolutional neural network controller that takes the pixels of an image as input and as output, represents the Q-values for all the actions the agent can take from that state. The network architecture used in this work is shown in Figure 2.
In practice the simplest method consists of using an autonomous Q-learning algorithm for each agent in the environment, thereby using the environment as the sole source of interaction between agents.
In this work we use this method due to its simplicity, decentralized nature, computational speed, and ability to produce consistent results for the range of tasks we report. Therefore, in our tests each agent is controlled by an independent Deep Q-Network with architecture and parameters as reported in.[6] we use two insights that significantly improve convergence and training rates are the use of experience real dataset and the use of a target Q-network for computing the loss.
The experience replay dataset contains a fixed number of transition tuples in it that contain(s, a, r, s′) where r is the reward obtained by performing action a from state s, and s′ is the state the agent transitions to after performing that action.
The experience tuples are sampled uniformly from the experience replay dataset in mini-batches and are used to update the network. The addition of experience replay helps prevent the correlation between training samples, which improves convergence.The use of a target network in the loss function calculation helps convergence as well. The target network in the loss function is kept fixed for a certain number of iterations, and is updated periodically.[1]
we focused on a $10*10$ instance of the pursuit-evasion problem and let the agents learn for 20000 episodes, 200 step each time.
During the training time, the exploration rate (percentage of actions chosen randomly) decreases from an initial 1.0 to 0.001 in million time steps and stays fixed at that value.

Figure 1: Deep Q-Network (DQN): Inputs are four stacked frames; the network is composed of several layers: Convolutional layers employ filters to learn features from high-dimensional data with a much smaller number of neurons and Dense layers are fully-connected layers. The last layer represents the actions the agent can take (in this case, 9 possible actions).[7]

Here, selecting a random action simply means that we draw a command randomly from amongst all the possible actions in the game instead of using the prediction that would have been made by the Deep Q-Network.[11]

5 RESULT

The purpose of the work ahead is to study emergent competitive strategies between multiple agents controlled by autonomous Deep Q-Networks. we used the pursuit-evasion game in a two-dimensional environment.
For the majority of the evaluation, we focused on a 10∗10 instance of the pursuit-evasion problem. The image-like states were generated by simulating the motion of the agents, and converting the state into a three dimensional tensor. In this section we provide two experimental results that test the capabilities of the network controller.
Figure 4 shows the accuracy of the model and Figure 5 shows the average reward of two agents
One of the criteria we have developed to evaluate our model is the average reward. Which in a certain number of iterations has determined the average score of each agents. As shown in figure 5, the average score of a thief agent has decreased over time and the average score of a police agent has increased, indicating that a police agent has trained over the iterations and has been able to catch the thief agent.

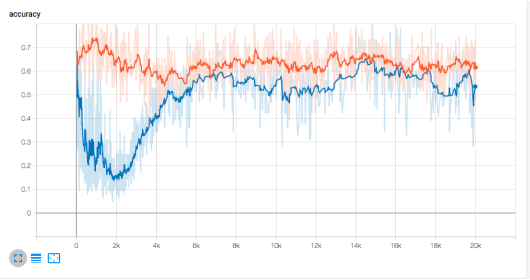

Figure 4: The accuracy of the model, blue chart is for thief agent and the orange is for police agent .

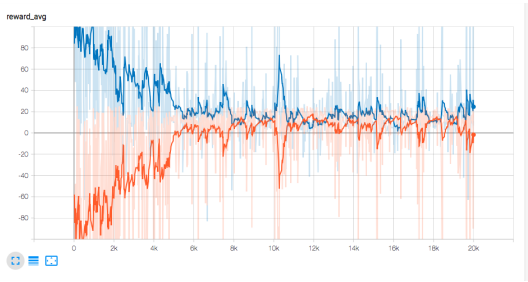

Figure 5: The average reward of the model, blue chart is for theif agent and the orange is for police.

6 CONCLUSION

Despite many of the recent advances in optimal control and automated planning, multi-agent systems remain an open research challenge. The difficulties arise from the curse of dimensionality that makes systems with large numbers of agents intractable. In this work, we demonstrated that a deep reinforcement learning approach can be used to solve decision making problems that are intractable for classical algorithms such as value iteration.[1]

There are a number of potential directions for future work.
One of the things that can be done in the future is to increase the number of agents and also we can use a partially observed environment here each agent is only able to see a few houses around it and the whole game environment is not obvious for them, in which case agent can use other associate agents to find the thief.

Another area of future work would involve extending this approach to more complicated domains. One potential application is robotic soccer. Specifically, selective co- ordination approaches have been shown to work well for robot soccer.[1]

REFERENCES

[1] M. Egorov. Multi-agent deep reinforcement learning. 2017.
[2] Jan Koutnik, Giuseppe Cuccu, Jurgen Schmidhuber, and Faustino Gomez. Evolving large-scale neural networks for vision-based reinforcement learning. *In Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 1061–1068, 2013.
[3] G. A. Kaminka. Robots are agents,too! *AgentLink News*, pages 16–17, 2004.
[4] L.-J. Lin. Reinforcement learning for robots using neural networks. *Technical report, DTIC Document*, 1993.

Hyperparameter	Value	Decription
Max train iterations	20000	The maximum number of training samples generated
Minimbatch size	64	Number of training samples per update
Replay size	50000	Size of the experience replay dataset
Learning rate	0.99	Rate used by the optimizer
Update rule	ADAM	The parameter update rule used by the optimizer
Initial exploration	1.0	Initial ε value in ε greedy exploration policy
ε decay	0.99975	Rate at which ε decreases
Target network update	5	Frequency of updating the target network

Table 1: Deep Q-Network hyperparameters

[5] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *Systems, Man, and Cybernetics, Part C: Applica- tions and Reviews, IEEE Transactions on*, pages 56–172, 2008.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King1, D. Kumaran1, D. Wierstra1, S. Legg1, and D. Hassabis1. Human-level control through deep reinforcement learning. *NATURE*, 518:529–533, 2015.

[7] M. E. T. Pablo Hernandez Leal, Bilal Kartal. Is multiagent deep reinforcement learning the answer or the question? a brief survey. 2018.

[8] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, second edition, 1998.

[9] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 1140–1144:362, 2018.

[10] D. Silver, J. Schrittwieser, K. Simonyan, ioannis Antonoglou, A. Huang, A. Guez1, T. Hubert1, L. baker1, M. Lai1, A. bolton1, Y. chen1, T. Lillicrap1, F. Hui1, L. Sifre1, G. van den Driessche1, T. Graepel1, and D. Hassabis1. Mastering the game of go without human knowledge. *NATURE*, 550:354–359, 2017.

[11] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, , and R. Vicente. Multiagent cooperation and competition with deep reinforcement learning. *arXiv preprint arXiv:1511.08779v1*, 2015.

[12] M. Wiering. Multiagent reinforcement learning for traffic light control. 2000.