A Comparative Study of Reinforcement Learning Algorithms for playing ATARI games

CS2IS2 Final Project (2018/2019)

Baolei Chen, Sourojit Das, Qingqin Li, Xin Chen

bachen@tcd.ie, dasso@tcd.ie, qingqinl@tcd.ie, chenx4@tcd.ie

Abstract. The report sets forth a study which compares the performance of AI agents using the Deep-Q Network Learning (DQN), the Double DQN algorithm and Dueling DQN Reinforcement Learning algorithms while playing the ATARI games Seaquest and Kangaroo on the Arcade Learning Environment provided by OpenAI. This was in line with our study of research papers which showed the implementation of deep RL agents to play Atari games to beyond-human levels of accuracy. While conducting our own experiments we found it difficult to factor in the human-player-score baseline into our experiments due to difficulties in verifying the source of and gaining access to such result. This prompted us to use the AI agent's own scores for the comparative analysis. Our results were consistent with the accepted results in the fact that both the Double Deep-Q Network Learning and the Dueling Deep-Q Network Learning-based agents outperformed the Deep-Q Network Learning which was used as the baseline.

1 Introduction

The aim of our project is to present a comparative analysis of the performance of three popular Reinforcement Learning algorithms, viz. the Deep Q-Network, double Deep Q-Network and the Dueling Deep-Q Network. Our motivation came from notable research (Mnih et al., 2013; Van Hasselt, Guez, & Silver, 2016)has already been performed in the domain of training AI agents to play ATARI games by using standard Reinforcement Learning techniques like Q-learning in combination with artificial Neural Networks. This combination of techniques has been referred to as Deep RL. The success of such methods in acquiring beyond-human capabilities in playing ATARI games was the source of our motivation to conduct studies in a similar field.

However, for the most part the evaluation of these algorithms was done in comparison to pre-defined baselines which were either based on scores obtained by human players or an AI agent making random move choices in the same environment (Van Hasselt et al., 2016). Our approach proposes to use a baseline score of the Deep Q-Network algorithm and compare the performance of the other algorithms with respect to it. This was done because it was a breakthrough in RL-based agent performance and many other researchers tried to create new methods and algorithms on its basis. Before venturing into the details of the implementation and evaluations carried out, it is useful to understand the nature of the environment in which these experiments have been performed and the algorithms themselves.

Reinforcement Learning can be simply described as an approach in which an agent interacts with an environment by conducting actions that help it to maximize an aggregated reward. A simple representation based on the work of Sutton and Barto (Sutton & Barto, 1998) has been shown below which best describes this process.

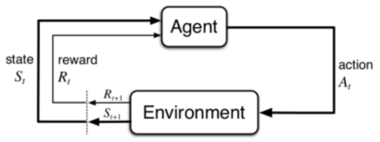


Figure 1. Basic Reinforcement Learning Process

The Agent takes an action (denoted by A_t) while in the present state (S_t). The environment responds to this and places the agent in a new state (S_{t+1}) and offers a Reward (R_{t+1}). The agent then proceeds to commit a new action based on this updated state and reward and this continues until the challenge posed by the environment is solved or some other termination condition is met.

The Open AI gym is a set of environments which have been tuned to help in evaluating and optimizing Reinforcement Learning algorithms in a range of domains like robot simulations, or in our case, Atari Games. This carries on the work initiated by the Arcade Learning Environment project created by Bellemare et.al (Bellemare, Veness, & Bowling, 2013). OpenAI has wrapped these environments to offer a more standardized environment and supports a total of 59 Atari games from the Atarti 2600 console as environment options. In our project, we're using OpenAI gym's Atari environments to train and test our RL methods.

The first algorithm implemented and subsequently used as a baseline to test the performance of the other algorithms is the Deep-Q-Network (Mnih et al., 2015). DQN is a Q-learning network combined with a flexible deep neural network. It showed human level performance in some of the games it was evaluated on. The advantage of the technique was that it augmented the Q-learning approach with a flexible deep neural network that provides flexible function approximation with potentially low approximation error and the deterministic character of the neural network can prevents the harmful effect of noise from normal Q-learning algorithm.

The Double Deep Q-learning algorithm was implemented by van Hasselt et al (Van Hasselt et al., 2016). which is an improved version of Deep Q-learning algorithm. It yields more accurate value estimates by helping remove overestimation from the Deep Q-learning. It's uses of dual estimator functions, with each function being updated with a value from the other Q function for successive states leads to both estimators using different set of experience samples. This leads to unbiased estimation and much better reward-scores as seen in our implementation.

The Dueling Q-learning (Wang et al., 2015)explicitly separates the representation of state values and action advantages. The Dueling network utilizes two streams to represent the value and advantage function but a single shared convolutional layer. An aggregation layer combines both to estimate the state-action value function Q. It can be thought of as a single Q network with two streams in contrast to the popular single-stream Q network in Deep Q-Network proposed by Mnih (Mnih et al., 2015) et al.

2 Related Work

No review of reinforcement learning applications can be complete without the mention of the success of TD-Gammon. Created by Gerald Tesauro (Tesauro, 1994) for playing backgammon. TD-Gammon stemmed from an ambitious idea to let a machine learning model keep on playing itself to 'master' the game. Such attempts at making computers proficient at certain domains by means of self-play originated as early as the checker-playing agent created by Samuel (Samuel, 2000) and the MENACE tic-tac-toe learner (Michie, 1961). However, these early solutions suffered from issues in scalability. Also, they were found to have developed brittle strategies (Pollack & Blair, 1997)which allowed them to only draw similar systems, yet loose to both humans and other programs. TD-Gammon provided the first real glimpse of hope in this domain when it achieved a near-impeccable style of play, bordering on the superhuman (Tesauro, 1995). A model-independent reinforcement learning algorithm, similar to Q-learning was implemented in TD-gammon, which calculated the value function by using a multi-layered perceptron containing a single hidden layer (Mnih et al., 2013).

As stated by Sutton and Barto (Sutton & Barto, 1998), reinforcement learning seeks to instruct an agent in choosing 'good policies' for solving sequential decision problems. It tends to do so by trying to reach an optimal future reward. This behaviour is based on concepts in behavioural psychology as outlined by Sutton (Sutton, 1988). The core idea hinges on the AI agent interacting with an environment in a pattern consistent with the observed interactions of biological agents. The artificial agent seeks to gather experience through exploration and exploitation in order to optimally satisfy certain objectives (François-Lavet, Fonteneau, & Ernst, 2015). The acceptance case for the optimal satisfaction of these objectives are represented as the accumulated rewards of performing a set of actions

Q learning, proposed by Watkins (Watkins, 1989) has remained a popular option for reinforcement learning-based agents, however it is known to suffer in performance due to learning improbably high action values. This happens as result of the maximation step in calculating action values which favours overestimation (Van Hasselt et al., 2016)to underestimation.

The synthesis of deep learning and reinforcement learning methods, primarily involving Q learning, was brought forward in a sequence of papers (Mnih et al., 2015; Van Hasselt et al., 2016). Whereas, previously implemented RL methods faced design difficulties due to feature-selection shortcomings, the deep RL approach was found to have successfully handle complex tasks, with only a smattering of prior knowledge, as it is capable of learning from data at different levels of abstraction. This was successfully shown by Minh (Mnih et al., 2015), in the utilization of such a combined approach in the training of a deep RL agent from visual inputs consisting of thousands of pixels. This approach enabled it to reach beyond-human capabilities in playing Atari games (Mnih et al., 2013), Go (Silver et al., 2016) and even Poker (Rupeneite, 2014). For the purpose of this research paper, we will focus on the implementation and evaluation of Deep RL algorithms in improving AI agent performance with regards to Atari games.

With the synthesis of the Arcade Learning Environment (ALE), (Bellemare et al., 2013)dozens of popular Atari games have been made available for training and evaluating the performance of AI agents, especially those concerned with Reinforcement Learning. Machado et.al. (Machado, Srinivasan, & Bowling, 2015) argue that ALE makes an ideal platform for AI agent evaluations as they provide a range of tasks which require general competence, both interest and challenge humans, and do not suffer from experimental bias because they are a third-party platform.

One of the state-of the art implementations of the use of Deep Q-learning in playing Atari games, which has been implemented in our project was the deep-Q network agent synthesized by Mnih (Mnih et al., 2015). This implementation achieved human-like performance in playing Atari games by using artificial neural networks to process sensory data, in this case frames from the game containing thousands of pixels. In subsequent work van Hasselt (Van Hasselt et al., 2016)improved on this algorithm by implementing the Double Deep Q-Learning which helped in yielding more accurate estimates by eliminating overestimations. The third algorithm taken into consideration was the Duelling Deep-Q Network (Wang et al., 2015)which made use of a two-stream Q network to obtain faster estimates of more valuable states without calculating the effect of actions at every state.

However, during our research, we found that most of the evaluations of these algorithms were done based on pre-defined human-player performance baselines which were difficult to obtain and replicate. Thus, we decided to make our evaluations based on comparing the agent's performance after training it through different algorithms as it would be easily replicable and have no third-party dependencies.

3 Problem Definition and Algorithm

3.1 The Problem Statement

This project addresses the problem of training and performance evaluation of AI agents, by using Reinforcement Learning techniques, in particular Deep RL techniques, to play ATARI games.

3.2 The Game Environments and Game Logic

ATARI Seaquest is an underwater shooter game which allows the player to control a submarine and shoot missiles to defeat sharks and enemy submarines while trying to save divers in the water. The sub has limited oxygen and must resurface to replenish it. Also, resurfacing without saving divers will result in a loss of life. Each instance of surfacing increases the difficulty level of the game.

ATARI Kangaroo game deals with a mother Kangaroo trying to save her Joey help captive on the top floor of every level. The player must avoid getting hit by the apples thrown by monkeys on the floors and knock them out with a boxing glove. Jumping to get fruits will give points to the player.

The OpenAI environment provides a wrapper of the Arcade Learning environment for ATARI is used to provide a standardized environment for the AI agent to train and evaluate itself using the RL algorithms mentioned in the subsequent section. In this domain the agent considers a video s_t having M image frames: $s_{t\,=}\,(x_{t\text{-}M=1}\ldots,\ x_t) \in S$ at time step t. The agent chooses an action from a discrete set of actions $a_t \in A = \{1\ldots, |A|\}$ and perceives a reward signal r_t synthesized by the emulator.

3.3 The Models Used

Deep Q-Network, double Deep Q-Network and the Dueling Deep-Q Network models were used to train the AI agent to play the ATARI 2600-compatible games Seaquest and Kangaroo. The following subsections give details on the mathematical bases on which these models were created and their algorithmic representations which were implemented in our project.

Deep Q Network – The network is a multi-layered deep neural network that for a given state S will provide as output an action value vector Q (s; ·; ***θ***), where θ represents the network parameters. The main features of DQN are its utilization of a target network and experience replay (Mnih et al., 2015). The target network used by DQN is represented by **(1)** where the parameters θ are copied for every step from the online network and kept fixed for the others –

$$\left(r_t + \gamma_t\left(\max_{a'} Q_t(s,a';\theta_i^-)\right) - Q_t(s,a)\theta_i\right)^2, \text{(1)}$$

The system is fed a given state, which is in a form of stack of raw pixel frames. At each step, the agent will select an action based on greedy policy with respect to the action values, and add the transition into the replay memory buffer, which can hold millions of transitions in memory. The parameters of the neural network are optimized by using stochastic gradient of decent to minimize the loss. The algorithm for the same is given in **appendix A**.

Double Deep Q Network – The problem in Deep Q-network is overestimation. It can attribute to insufficient flexible function approximation (Thrun & Schwartz, 1995) and noise (Van Hasselt et al., 2016). To solve this problem, van Hasselt et.al. (Van Hasselt et al., 2016) introduced Double Q-learning with two estimators. They decoupled the selection from the evaluation. The two value functions are cross learning by assigning each experience randomly to update one of the two value functions. There are two sets of weight and the estimator will randomly pick one of the weights to evaluate the value. By using cross validation, the two functions are less likely to have biased or overestimated values for the output.

$$Y_t^{\text{DoubleDQN}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta}_t), \boldsymbol{\theta}_t^-)\,.$$

In contrast to the Deep Q-Network formula, the weight of the target network θ−t is used in place of the weight of the second network θ′t. the algorithm pseudocode is given in **Appendix B.**

Dueling Deep Q Network – The dueling architecture was proposed by (van Hasselt, 2015). The architecture deliberately separates the representation of state and action values. The dueling architecture uses two streams for estimation function. The two channels use a single convolutional layer and are combined via a special aggregation layer to produce an estimate of the Q value. Each stream has the capability to provide separate estimates of the Q values. Finally, the two streams combined to produce a single output Q function. The estimators update the function as –

$$\boldsymbol{Q(s,a;\theta,\alpha,\beta) = R(s;\theta,\beta) + \left(Q(s,\alpha;\theta,\alpha) - \frac{1}{Q}\sum_{a'} Q(s,\alpha';\theta,\alpha)\right)}$$

The Dueling network has the advantage to only update action as fast as the mean, instead of compensating all the change to the optimal action. The drawback with this method is that the state and action will lose their original semantics because they are now off-target by a constant.

4 Experimental Results

4.1 Evaluation Methodology

This section describes the methods followed to evaluate the AI agent's performance in playing the game on being trained using different algorithms. Some pre-requisites are: -

1. In order to evaluate the performance of the AI agent in playing the game, the agent appropriate environment inside the OpenAI Atari module is invoked. In our case it is either the KangarooNoFrameskip-v4 environment or the SeaquestNoFrameskip-v4 environment. This specifies to the AI Agent game it must master.
2. The agent accepts the environment details and a set of parameters and begins to negotiate the environment.
3. For the sake of creating a stable ground for evaluation the same set of parameters like discount factor(gamma); learning rate, exploration ratio etc are kept constant.

The overall structure of the evaluation procedure can be explained through the pseudocode shown below –

```
start
Create Environment env
Create DQN Agent using env and Parameters
        (env_name='KangarooNoFrameskip-v4', cuda=False, display_interval=10)
Train the agent using env:
For timestep in 10^7
Calculate reward and loss for each 100 episodes
```

```
Save the ***reward*** and ***loss*** data into files
Update the target ***network***
**End**
```

The following metrics are used for evaluating agent performance –

- Rewards (with respect to timesteps) – A Reward is defined the score of every action strategy adopted by the agent in the game. A timestep is the total number of steps (or frames of the game) taken/processed from the beginning to the current point. We adopted this formula as Reinforcement Learning approaches to ATARI game playing have sought to hypothetically maximize rewards. This means that all targets can be described as maximizing the expected rewards. Higher the rewards, per timestamp the better the performance of the algorithm.
- Training Loss (with respect to timesteps) – Loss is the mean square deviation of actual q value and predicted q value, and an episode is one complete play of the agent interacting with the environment. In reinforcement learning, our objective loss function is to get the q value in the shortest training episodes. So, the lower the training loss is, the better performance the reinforcement learning algorithm has.
- Episodes processed (with respect to timesteps) – An Episode is considered to be a sequence of states, actions and rewards that end with a terminal state. This terminal state can be a state of win/loss/draw for the player. This corresponds to one session of playing the game for a human user. We try to evaluate how quickly a method enables the agent to complete one instance of the game (in this case one episode).

4.2 Results

Figures 2 – 7 shows the results obtained upon evaluation. **Figures 2** and **3** describes the accumulation of rewards with increasing timesteps for the 3 algorithms in the two games. **Figures 4** and **5** describe the training loss calculated with number of episodes for the same. The **Figures 6** and **7** finally describe the number of timesteps taken for the algorithms to complete the episodes –

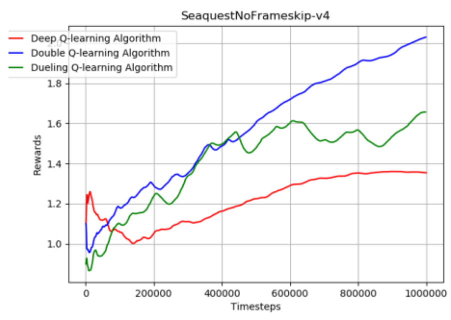


Figure 2 Comparing Reward with Timesteps for ATARI Seaquest

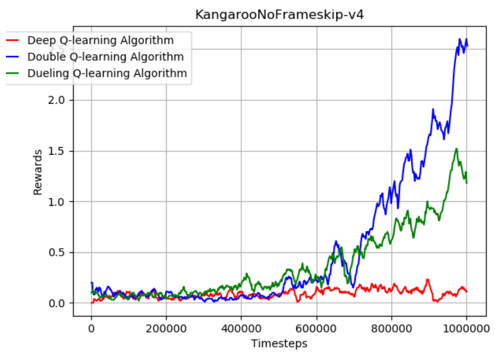


Figure 3 Comparing Rewards with Timesteps for ATARI Kangaroo

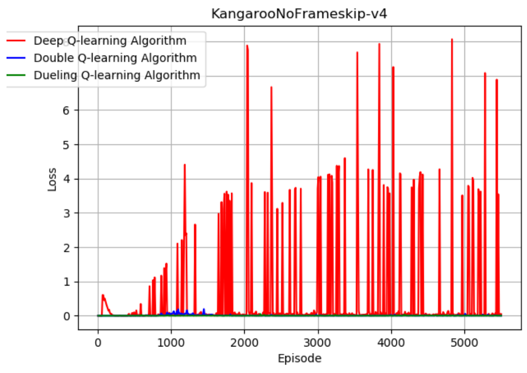


Figure 4 Comparing Training loss with Episodes of Atari Kangaroo Game

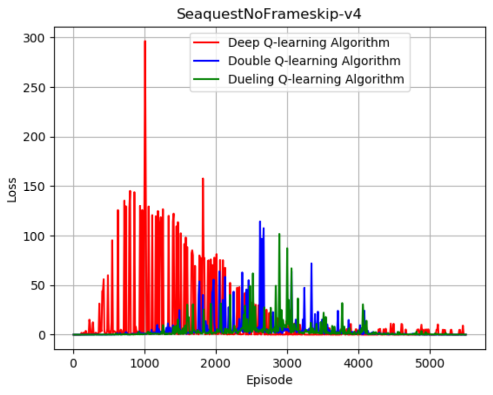


Figure 5 Comparing Training loss with episodes of Atari Seaquest Game

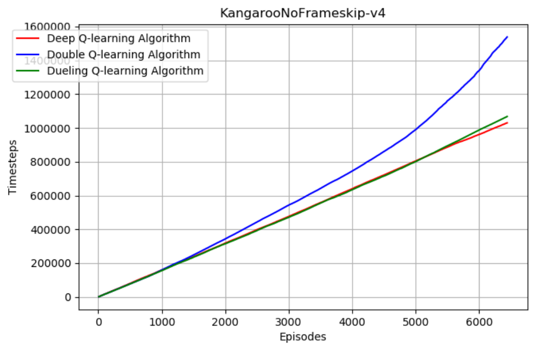


Figure 6 Comparing Timesteps take to complete episodes for different RL Methods in the ATARI Kangaroo Game

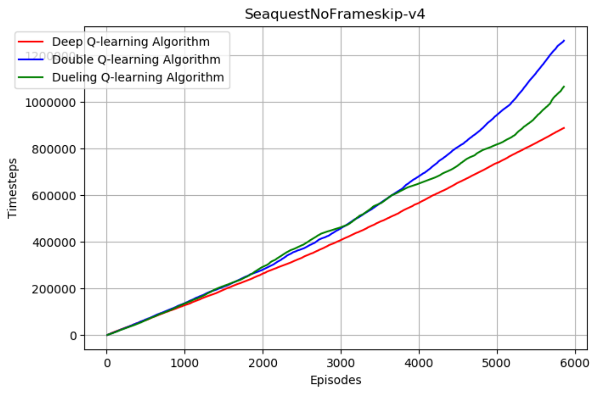


Figure 7 Comparing Timesteps take to complete episodes for different RL Methods in the ATARI Seaquest Game

4.3 Discussion

Figure 1 and **Figure 2** describe the change in accumulated rewards with increasing timesteps for the 2 games. A general pattern in shown with the agent utilizing Double Deep-Q Network having the best performance of the three, followed by the one with Dueling Deep-Q Network. Both outperform the agent trained on the baseline Deep-Q network algorithm, which is consistent with the results outlined in the papers. However, Double Deep-Q network performs better than the overtime which is inconsistent with the results of the paper by Hessel et.al (Hessel et al., 2018). However even in their results had noted that Dueling Architecture had performed worse than the Double Deep-Q network for certain Atari games (though not for Seaquest or Kangaroo). This maybe because of the fact that the parameters were not optimized for the Dueling Deep-Q Network and favoured a single stream approach.

From **Figure 3** and **Figure 4**, we see the results of training loss with episodes for the 2 games. We can see that Deep Q-learning performs the worst because it lacks a well-formed estimator function to prevent overestimations. It also oscillates a lot due to the overestimated output from the deep neural network. On the other hand, the Double Q-learning performs much better benefitted by the unbiased second estimator. The result is much smoother and contains less oscillations. Similarly, the Dueling Q-learning performs better than the baseline. This is consistent with the results described in Hessel et.al (Hessel et al., 2018).

From **Figure 5** and **Figure 6**, we observe that Double Deep-Q takes around 40%-50% more timesteps (or frames) to process the same number of episodes as the Deep-Q Learning Network and the Dueling Deep Q-learning networks. It was expected that Dueling Deep-Q Learning network would take less timesteps to process the same episodes as compared Double Deep-Q Learning. However, it was surprising that Dueling Deep-Q learning would perform poorly compared to traditional Deep-Q learning with regard to the timesteps(frames) taken to process an episode. It may be due to the fact that the games tested contained few or no redundant or valueless states and thus Dueling Deep Q-learning was unable to leverage on the fact that its agent could obtain faster estimates of more valuable states without calculating the effect of actions at every state.

5 Conclusions

Our project focussed on the task of evaluating the performance of AI agents which were trained using Deep Q-Learning Network, Double Deep Q-Learning Network and Dueling Deep Q-Learning Network algorithms. In the absence of human-player scores to use as a baseline like state-of-the-art methods have used, we decided to take the performance of Deep Q-Learning Network as a baseline as the other two approaches have been built to address its shortcomings. This would help us avoid biased scores in case of improperly evaluated human-player scores and make our results readily replicable. Our results were consistent with the existing research in that both Double Deep Q-Learning Network and Dueling Deep Q-Learning Network agents performed better than the baseline. The relatively low performance of the Dueling Deep-Q network could be explained by the fact that the parameters were not optimized for the double stream calculations and that the games themselves contained little or no sub-optimal frames for the Dueling Deep-Q network to leverage its capability of producing faster approximations of the Q value without observing the actions at all states. The evaluation methodology also bore out clearly that the lack of a human-referenced baseline did not hinder the comparison of results.

The authors would like to mention that the team learnt a lot about using Reinforcement Learning methods in conjunction with Deep Learning techniques. Though this process is very time consuming it offers a chance to train agents to play games to almost human-like levels of expertise and even beyond.

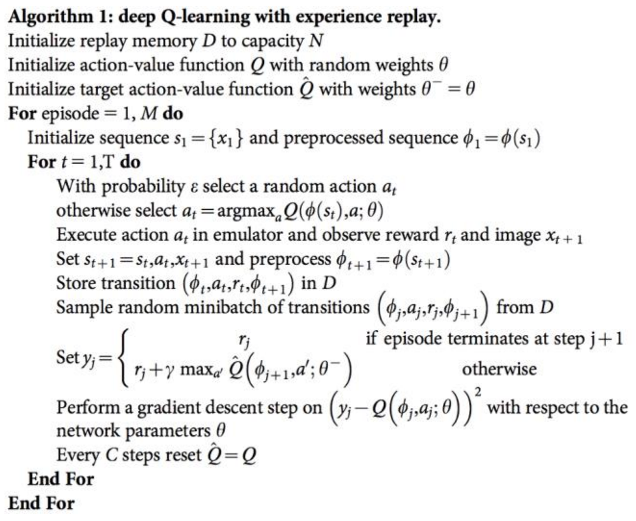
Though we were short of time to perform the experiments on more ATARI games we would like to do so given a chance in the future and also acquire suitable human baselines to see whether our agents perform satisfactorily as compared to humans.

References

1. Bellemare, M., Veness, J., & Bowling, M. Bayesian learning of recursively factored environments. Paper presented at the International Conference on Machine Learning. (2013).
2. François-Lavet, V., Fonteneau, R., & Ernst, D. How to discount deep reinforcement learning: Towards new dynamic strategies. arXiv preprint arXiv:1512.02011. (2015).
3. Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Silver, D. Rainbow: Combining improvements in deep reinforcement learning. Paper presented at the Thirty-Second AAAI Conference on Artificial Intelligence. (2018).
4. Machado, M. C., Srinivasan, S., & Bowling, M. Domain-independent optimistic initialization for reinforcement learning. Paper presented at the Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence. (2015).
5. Michie, D. Trial and error. Science Survey, Part, 2, (1961). 129-145.
6. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602. (2013).
7. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Ostrovski, G. Human-level control through deep reinforcement learning. Nature, (2015). 518(7540), 529.
8. Pollack, J. B., & Blair, A. D. Why did TD-gammon work? Paper presented at the Advances in Neural Information Processing Systems. (1997).
9. Rupeneite, A. Building Poker Agent Using Reinforcement Learning with Neural Networks. (2014).
10. Samuel, A. L. Some studies in machine learning using the game of checkers. IBM Journal of research and development, 44(1.2), (2000). 206-226.
11. Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G.,Lanctot, M. Mastering the game of Go with deep neural networks and tree search. nature, 529(7587), (2016). 484.
12. Sutton, R. S. Learning to predict by the methods of temporal differences. Machine learning, 3(1), (1988). 9-44.
13. Sutton, R. S., & Barto, A. G. Introduction to reinforcement learning (Vol. 135): MIT press Cambridge. (1998).
14. Tesauro, G. TD-Gammon, a self-teaching backgammon program, achieves master-level play. Neural computation, 6(2), (1994). 215-219.
15. Tesauro, G. Temporal difference learning and TD-Gammon. Communications of the ACM, 38(3), (1995). 58-68.
16. Thrun, S., & Schwartz, A. Finding structure in reinforcement learning. Paper presented at the Advances in neural information processing systems. (1995).
17. Van Hasselt, H., Guez, A., & Silver, D. Deep reinforcement learning with double q-learning. Paper presented at the Thirtieth AAAI Conference on Artificial Intelligence. (2016).
18. Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., & De Freitas, N. Dueling network architectures for deep reinforcement learning. arXiv preprint arXiv:1511.06581. (2015).
19. Watkins, C. J. C. H. Learning from delayed rewards. (1989).

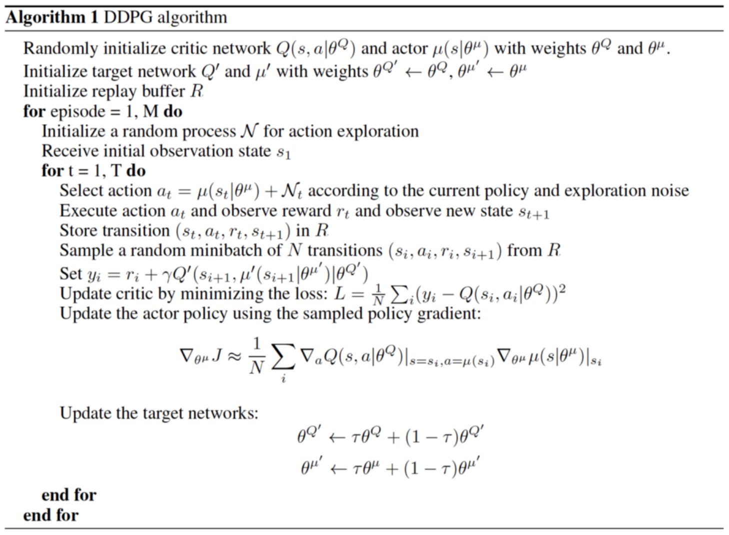
Appendices

Appendix A



```
**Algorithm 1: deep Q-learning with experience replay.**
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
**For** episode = 1, $M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  **For** $t$ = 1,T **do**
    With probability $\varepsilon$ select a random action $a_t$
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the
    network parameters $\theta$
    Every $C$ steps reset $\hat{Q} = Q$
  **End For**
**End For**
```

Appendix B



```
**Algorithm 1** DDPG algorithm
  Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
  Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
  Initialize replay buffer $R$
  **for** episode = 1, M **do**
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state $s_1$
    **for** t = 1, T **do**
      Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
      Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
      Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
      Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
      Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
      Update critic by minimizing the loss: $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
      Update the actor policy using the sampled policy gradient:

          $\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$

      Update the target networks:
          $\theta^{Q'} \leftarrow \tau\theta^Q + (1 - \tau)\theta^{Q'}$
          $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1 - \tau)\theta^{\mu'}$
    **end for**
  **end for**
```