Playing the game of Catch with DQN, DDQN, Dueling DQN, DQV and DQV-Max

Kyriakos Antoniou s5715881[1] **Abi Raveenthiran s4010132**[1]

Abstract

In this report, we explore and implement several Deep Reinforcement Learning algorithms, namely, Deep Q-Learning, Double Deep Q-Learning, Deep Quality-Value Learning, Deep Quality-Value Max Learning and the Dueling Architecture. The environment that used to evaluate these algorithm is a simple game called the game of Catch. The results of our experiment show that all algorithms are able to reach a 90% accuracy at the end of training.

1. Introduction

Reinforcement Learning (RL) is a subfield of Machine Learning that uses value functions to estimate how good the current state is or how good an action given the current state is for an agent. Deep Reinforcement Learning (DRL) combines RL with Deep Learning (DL), where deep neural networks are used to approximate the value functions used in RL.

In this project, we aim to explore and implement the following DRL algorithms: Deep Q-Learning (DQN) (Mnih et al., 2015), Double Deep Q-Learning (DDQN) (Van Hasselt et al., 2016), The Dueling Architecture (Wang et al., 2016), Deep Quality-Value Learning (DQV) (Sabatelli et al., 2020), and Deep Quality-Value Max Learning (DQV-Max) (Sabatelli et al., 2020). The performance of these DRL algorithms will be evaluated using the game of Catch.

The remainder of the report will be structured as follows. In section 2, we describe the environment that the algorithms will be evaluated with, the game of Catch. In section 3, we explain the DRL algorithms that we have implemented. Section 5 presents the results of the DRL algorithms and in section **??** we give a conclusion of this project.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

2. The game of Catch

The game of "Catch", introduced by Mnih et al. (2014), is a simple game in which an agent has to catch a falling ball using a paddle that can be moved horizontally. The environment of the game is represented by a 21×21 grid, where the ball falls from top to bottom and can bounce of the borders (except the bottom) of the grid. The ball falls with a vertical speed of $v_y = -1\ cell/s$. At the start of each episode the horizontal position of the ball is set randomly within the grid and the horizontal speed of the ball v_x is a random value $\in \{-2, -1, 0, 1, 2\}$. The action space of this environment is three, the agent is able to move the paddle one pixel to the left, one pixel to the right, or keep the paddle in the same position in the grid. The agent takes an action at each time step until the end of an episode. An episode ends when the agent catches the paddle or when it fails to catch the ball. The reward given at the end of each episode is determined by whether the agent caught the ball or not, resulting in a reward of 1 when the ball is caught and a reward of 0 if the ball was not caught.

3. Background Knowledge

3.1. Reinforcement Learning

A RL environment is formally defined as a Markov Decision Process (MDP). An MDP consists of the following components:

- A finite state space S
- A finite action space $\mathcal{A}_t$
- A state transition probability distribution $p(s_{t+1}|s_t, a_t)$
- A reward function $\Re(s_t, a_t, s_{t+1})$
- A time step counter t

In a RL environment a RL agent finds itself in a state $s_t \in S$ given a time step t. In s_t, the agent then takes an action $a_t \in \mathcal{A}$ based on its policy π. After taking the action a_t, the agent will move to the next state based on the state transition probability distribution $p(s_{t+1}|s_t, a_t)$. The Reward function $\Re(s_t, a_t, s_{t+1})$ then sends a reward signal r_t to the agent,

which indicates how good the action was that the agent took. The state-value function $V^{\pi}(s)$ is defined as follows:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty}\gamma^{k} r_{t+k} \,\middle|\, s_t = s, \pi\right] \tag{1}$$

where π is the agent's policy, s is the current state, r_t is the reward signal at t, and γ is the discount factor. The state-value function returns the expected cumulative discounted reward given that an agent follows its policy π in state s.
The state-action value function Q^{π} is defined as follows:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty}\gamma^{k} r_{t+k} \,\middle|\, s_t = s, a_t = a, \pi\right] \tag{2}$$

where π is the agent's policy, s is the current state, a is the taken action, r_t is the reward signal at t, and γ is the discount factor. The state-action value function returns the expected cumulative discounted reward of an agent taking an action a in state s based on its policy π.
Lastly, the advantage function subtracts the Q-value from the value function. This is used to measure how good a specific action is compared to the average of all possible actions in a state. It is defined as follows:

$$A^{\pi}(s,a) = V^{\pi}(s) - Q^{\pi}(s,a) \tag{3}$$

The goal of the DRL algorithms covered in this project will be to learn an approximation of the optimal value functions stated above.

3.2. Deep Q-Learning

The Deep Q-Learning (DQN) algorithm is based on the Q-Learning algorithm introduced by Watkins & Dayan (1992). In Q-Learning the Q-values are updated using the Bellman equation, where α is the learning rate:

$$Q(s_t,a_t) := Q(s_t,a_t) + \alpha\left[r_t + \gamma \max_{a\in\mathcal{A}} Q(s_{t+1},a) - Q(s_t,a_t)\right] \tag{4}$$

In the DQN algorithm, the update rule is the update rule of the Q-Learning algorithm is changed into the following loss function:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r+\gamma \max_{a\in\mathcal{A}} Q(s_{t+1},a;\theta^{-}) - Q(s_t,a_t;\theta)\right)^2\right] \tag{5}$$

where D is the Experience-Replay memory buffer, θ represents the online network's parameters and θ^{-} represents the target network's parameters. Two of the main components of a DQN introduced by Mnih et al. (2015) are the use of a target network and the Experience-Replay memory buffer. In DQN, rather than an online network, a target network with fixed parameters is used to estimate the Q-values. The target network, with parameters θ^{-}, is the same as the online network. However, unlike the online network's parameters θ, the target network's parameters θ^{-} are temporarily frozen. θ is updated every time-step, but θ^{-} is only updated periodically where the values of θ are copied to θ^{-}.
The Experience-Replay memory buffer is used to store the experiences of a RL agent in a queue at each time-step. An experience represents a transition from one state to another state $\langle s_t, a_t, r_t, s_{t+1}$. These experiences are used to update the network.

3.3. Double Deep Q-Learning

Double Deep Q-Learning (DDQN) as the name already suggests is based on Double Q-Learning (Hasselt, 2010). This algorithm works similarly to standard Q-Learning, however rather than using the same value to estimate and select the Q-value, Double Q-Learning separates the selection and the evaluation.
In DDQN this results in updating DQN's loss function to the following loss function:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r+\gamma Q(s_{t+1}, \max_{a\in A} Q(s_{t+1},a;\theta);\theta^{-} - Q(s_t,a_t;\theta)\right)^2\right] \tag{6}$$

The action a is now selected using the online network's parameters θ, but is evaluated using the target network's parameters θ^{-}.

3.4. The Dueling Architecture

The Dueling Architecture, introduced by Wang et al. (2016), is a Q-network that instead of using a single stream, uses two separate streams to estimate the state-value and the advantage value for each action. These values are then aggregated in the last module of the network using the following formula:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \max_{a'\in|A|} A(s,a';\theta,\alpha)\right) \tag{7}$$

The training procedure of the Dueling Architecture network is similar to the standard DQN network, using the same loss function.

3.5. Deep Quality-Value Learning

The Deep Quality-Value (DQV) Learning algorithm, introduced by Sabatelli et al. (2020), is a DRL algorithm based

on the QV(λ) algorithm by Wiering (2005). The DQV algorithm utilises two separate neural networks to learn an approximation of both the V function and the Q function. The loss function for the state-value function V is defined as follows:

$$L(\Phi) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim U(D)} \left[(r_t + \gamma V(s_{t+1};\Phi^-) - V(s_t;\theta))^2 \right] \quad (8)$$

where Φ are the network parameters of one of the neural networks and Φ^- the parameters of the target network.

The loss function for the Q function is defined as follows:

$$L(\theta) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim U(D)} \left[(r_t + \gamma Q(s_t,a_t;\Phi^-)^2 \right] \quad (9)$$

3.6. Deep Quality-Value Max Learning

Lastly, the Deep Quality-Value Max (DQV-Max) Learning algorithm, introduced by Sabatelli et al. (2020) and based on the QV-Max Learning algorithm by Wiering & Van Hasselt (2009), is similar to the DQV algorithm mentioned above, however the main difference is that the DQV-Max algorithm makes use of the maximum Q-value in the loss function for learning the approximation of the V function. This makes it so that the algorithm is learning off-policy rather than on-policy as done in the DQV algorithm. This changes the DQV loss function of the V function in to the following loss function for DQV-Max:

$$L(\Phi) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim U(D)} \left[(r_t + \gamma \max_{a\in\mathcal{A}} Q(s_{t+1};\theta^-) - V(s_t;\Phi))^2 \right] \quad (10)$$

The loss function for the Q function of DQV-Max is the same as in DQV 9, however instead of using the target network Φ^-, the online network Φ is used.

4. Experimental setup

4.1. Hyper parameters

In the below results the following hyper parameters were used in order to train each of the appropriate neural network.

- Discount factor of 0.99, used in combination with rewards gained from the networks and the actual rewards.
- Bath size of 32 was used in the sampling of memories from the buffer to be used to train the neural networks.
- Learning rate of 1e-4 was used as the learning rate for the optimizer
- Initial epsilon of 1 , minimum epsilon of 0.01 and epsilon decay of 0.99. This values were used in order to decide if a random action should be selected or use the networks to select an action for the step of the enviroment.
- Memory size of 1000 was used, this is where the current state, the results of an action that was used to step in the envrioment as well as the action itself were stored.

4.2. Loss function

The loss function used was

4.3. Optimizer

In the experiments the optimizer used was RMSProp. The optimizer firstly calculates the gradient $g_t = \nabla_\theta J(\theta_t)$, where $J(\theta)$ is the loss function. It then calculates the moving average of the squared gradients

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta) g_t^2 \quad (11)$$

with a decay rate usually set at 0.9. It then updates the parameters as follows :

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \quad (12)$$

where η is the learning rate and ϵ is a small constant preventing divisions by zero.

The algorithm was chosen due to its ability to handle sparse rewards that would make the gradients sparse and noisy (Saripalli et al., 2020).

5. Results and Conclusion

It can be seen from the 1 that dueling DQN and DQV-Max perform the best in our current environment with the normal DQN only falling slightly behind. As for DQV and DDQN even though they are able to reach a solved state of being able to catch the ball 90% of the times they are much slower to learn. Furthermore we can see that all algorithms were learning approximately the same amount up until the first 1000 episodes reaching to about 40% accuracy. More specifically we can see that DQV-Max was able to reach a solved condition at about 2000 episodes while both Dueling DQN and normal DQN were only slithly behind they were only able to reach a solved condition only after about 700 more episodes. As for DDQN and DQV even though DQV learned more evenly compared to DDQN they both managed to reach a solved case only after about 4500 episodes.

From the results it can be seen that in this specific case it is really hard to choose which algorithm from the dqn family

is best. Due to time restrictions and the high running time required to run the algorithms they were only run once as such further testing is required to provide a more concrete conclusion, more runs through the algorithms are required in order to average the learning rates out and get a smoother gradient as well as get a cloud of the maximum and minimum values of each of the algorithms. Thus the results are not as interpret-able as they could be but specific algorithms can be seen reaching above the rest.

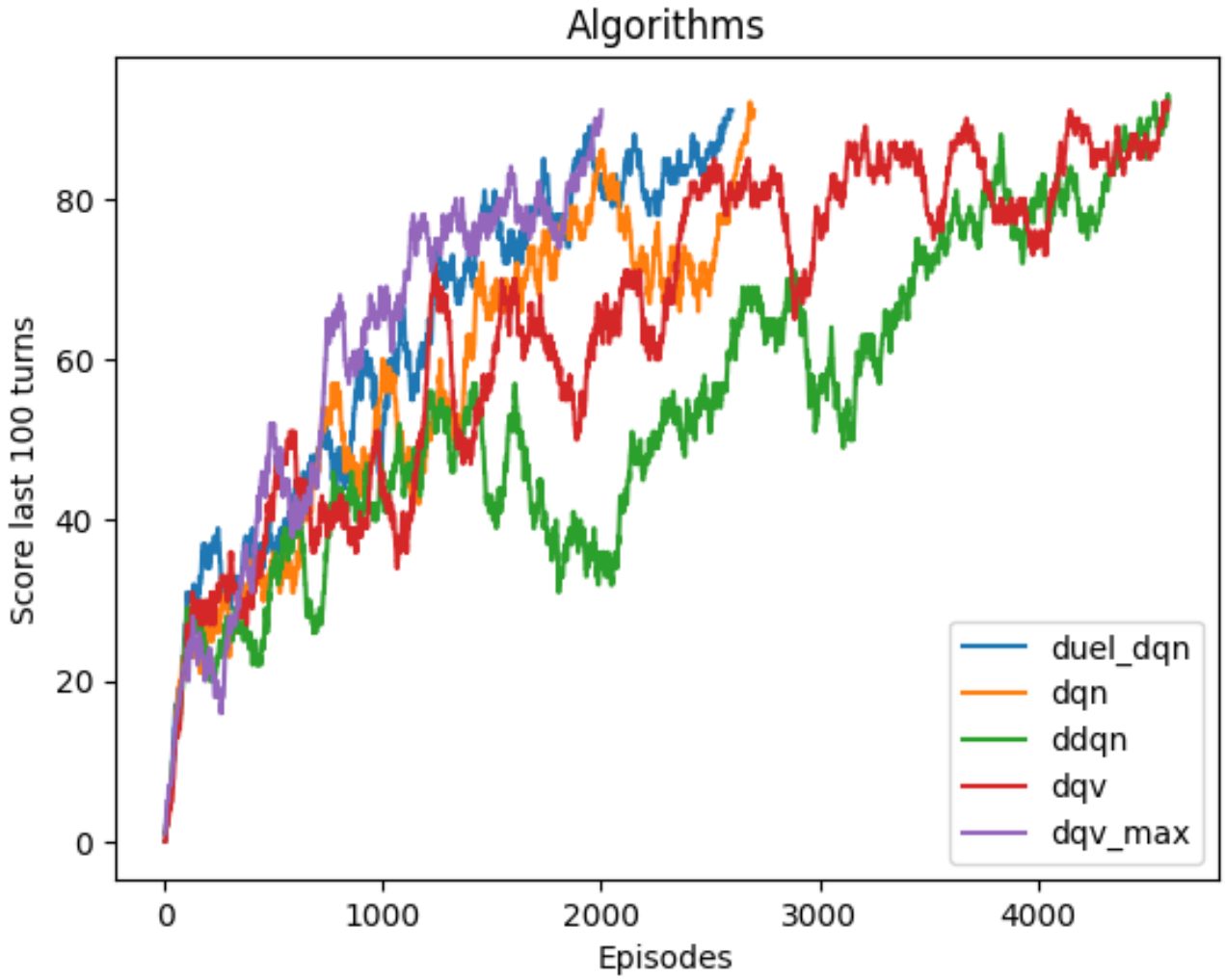

Figure 1. Last 100 turn rewards vs Episodes

A. Appendix

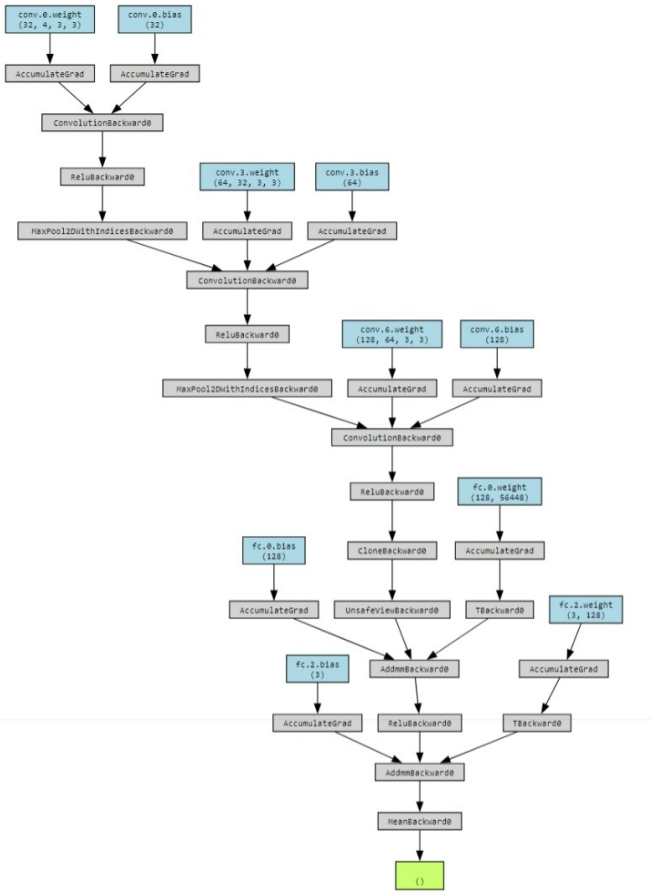

Figure 2. CNN + FC layers used as the main model for DQN, DDQN, DQV and DQV-Max and as the target model for DQN, DDQN and DQV-Max

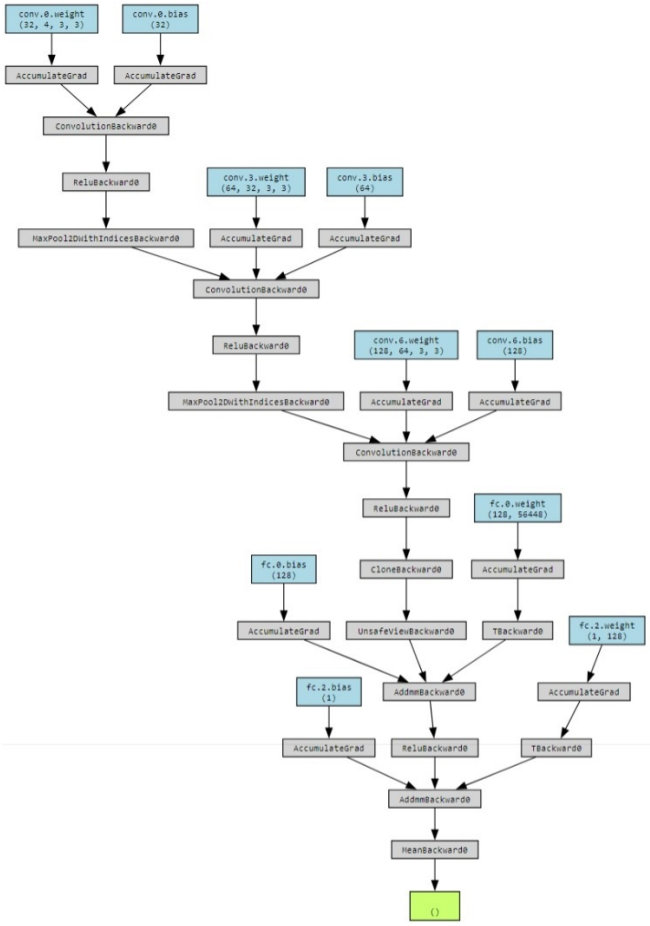

Figure 3. CNN + FC layers used as the main value function model for the DQV and DQV-Max as well as the target value function model for DQV

References

Hasselt, H. Double q-learning. *Advances in neural information processing systems*, 23, 2010.

Mnih, V., Heess, N., Graves, A., et al. Recurrent models of visual attention. *Advances in neural information processing systems*, 27, 2014.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533, 2015.

Sabatelli, M., Louppe, G., Geurts, P., and Wiering, M. A. The deep quality-value family of deep reinforcement learning algorithms. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2020.

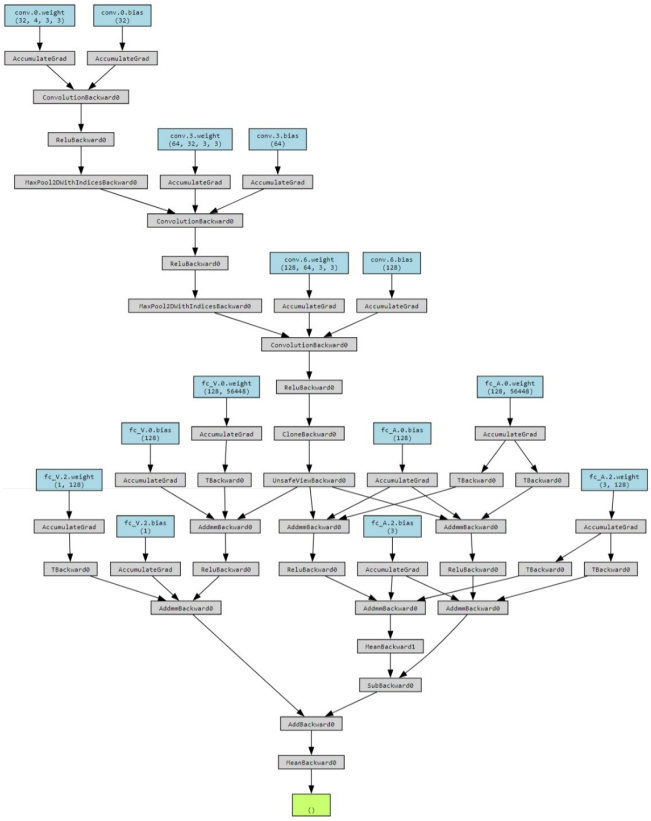

Figure 4. CNN + FC layers used as the main model and target model for the Dueling DQV

Saripalli, V. R., Pati, D., Potter, M., Avinash, G., and Anderson, C. W. Ai-assisted annotator using reinforcement learning. *SN Computer Science*, 1(6):327, 2020.

Van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pp. 1995–2003. PMLR, 2016.

Watkins, C. J. and Dayan, P. Q-learning. *Machine learning*, 8:279–292, 1992.

Wiering, M. A. Qv (λ)-learning: A new on-policy reinforcement learning algorithm. In *Proceedings of the 7th european workshop on reinforcement learning*, volume 7. Univ. Naples Dep. Math. Stat. Naples, Italy, 2005.

Wiering, M. A. and Van Hasselt, H. The qv family compared to other reinforcement learning algorithms. In *2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pp. 101–108. IEEE, 2009.