Reinforcement Learning: coursework 2

Eamon Dutta Gupta
CID: 01722077
Dept. of Computing
MSc Computing (AI and machine learning)

December 2, 2022

1 Question 1

1.1 Hyperparameters

Hyperparameter	Value	Justification
Optimizer algorithm	Adam	Adam performed better than stochastic gradient descent and gave more consistent results.
Loss function	Huber	Huber loss combines advantages of L1 loss and mean squared error (MSE) loss, while being less sensitive to outliers than MSE, and being a smooth function around 0 (unlike L1 loss).
Huber loss parameter	0.2	Empirically derived from the environment setup.
Activation function	Leaky ReLU	This is as efficient as ReLU but circumvents the problem of dead neurons [5].
Buffer size	4500	Larger and smaller buffer sizes were found not to reach the required success threshold in the allotted training time of 300 episodes, even when maintaining the ratio $\frac{\text{buffer size}}{\text{batch size}}$.
Batch size	70	Larger batch sizes significantly increased the computational cost of the training loop without increasing the performance. Smaller batch sizes are not desired as they produce a less accurate loss value.
Update time period	25	High frequency updates of the target DQN gives rise to unstable learning. Lower frequency updates is not likely to result in 'beneficial' learning.
Learning rate	0.002	Empirically derived where the learning rate was tested between 1×10^{-4} and 1×10^{-1}.
Learning rate decay	0.99	Linear decay was used since exponential was too fast and resulted in bad performance.
Learning rate decay interval	20	Empirically derived to optimise the reward in 300 episodes.
Exploration factor	10	A sufficient balance of exploration and exploitation is required as explained in the next question.
δ	0.5	A sufficient balance of exploration and exploitation is required as explained in the next question.
σ	150	A sufficient balance of exploration and exploitation is required as explained in the next question.
Number of episodes	300	This was sufficient for the DQN agent to learn to a satisfactory standard (achieve average reward of 100 over 50 consecutive episodes).
Architecture	(128,128)	Empirically derived as the optimal number of layers and the number of nodes in each layer. Increments were tested as powers of 2.

Note that the leaky ReLU is used instead of standard ReLU in order to avoid the problem of **dead neurons**. The problem with ReLU activation is that they cannot learn on inputs where their activation is 0. This can happen due to an incoming large gradient, hence the neuron updates with a large negative weight and bias. The neuron will thus produce 0 during forward propagation and the gradient will always be 0 subsequently. This neuron is no longer capable

of learning and is **dead**. The Huber loss is given as follows

$$L(x,y) = \begin{cases} \frac{1}{2}(x-y)^2 & \text{if } |x-y| < \delta \\ \delta(|x-y| - \frac{\delta}{2}) & \text{else} \end{cases}$$

This loss function is the same as MSE in interval $(-\delta, \delta)$ where $\delta = 0.2$ was empirically chosen. Outside this interval it reverts to an L1-like loss to reduce sensitivity to outliers.

1.2 Exploration vs exploitation

In RL there is an exploration-exploitation tradeoff where the agent must choose whether develop a better understanding of the environment or act according to what it knows best. In the initial phases of training, exploration is particularly important as the agent has no prior knowledge. In the later phases the agent has sufficient knowledge of the environment is encouraged to exploit its knowledge. The first exploration scheme was used to decay the epsilon in the epsilon greedy policy used in a fashion that encouraged the agent to explore for the initial episodes.

$$\epsilon = min(1, \frac{\text{total episodes}}{\beta(\text{episode}+1)}) \tag{1}$$

$$\pi(s,a) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A(s)|} & \text{if } a = argmax_a(Q(s,a)) \\ \frac{\epsilon}{|A(s)|} & \text{else} \end{cases} \tag{2}$$

In this way the epsilon decays linearly with respect to the episode number. Increasing the β parameter results in fewer episodes where the epsilon is set to 1 as is seen in Figure 1, i.e. the policy is uniform for maximum exploration. Despite the predefined exploration periods, the learning curves do not reflect this difference until halfway through the run, where a higher mean is reached by the agents using a larger beta, shorter exploration phase. Smaller beta values also represent a slower decay rate; Figure 2 illustrates the worse performance by setting beta = 10, as the epsilon values are much higher than the other 2 beta values, even in later episodes. Setting beta=150 in this setup of 300 total episodes is setting epsilon=1 only for the first 2 episodes. This implies the agent does not require an epsilon value of 1 to explore, but a decaying epsilon is sufficient for exploration. Nevertheless, all the values of beta in shown in Figure 1 resulted in a successful DQN agent learning off policy. In order to experiment, other sources of error had to be minimized. The sources of randomness included the initialization of neural network weights, the initialization of state in the Cartpole environment and the non deterministic policy leading to random behaviour. The latter 2 are necessary for the the agent to explore and generate

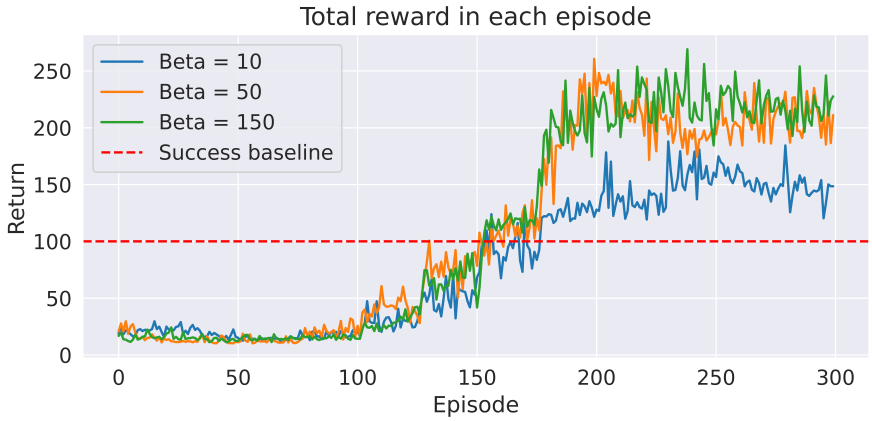

Figure 1: The effect of exploration parameter beta on the mean reward per episode.

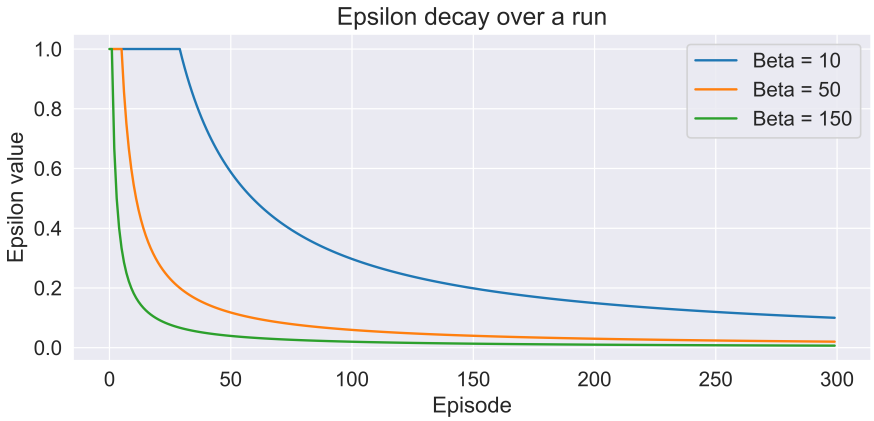

Figure 2: This is the graph showing the epsilon decay for the corresponding values of beta.

accurate values for the Q function, otherwise, the agent would produce the same trace every episode. The neural network weights were initialized with a consistent distribution, that was more 'optimistic' with respect to a trained DQN. It was found that the He initialization was optimal for the leaky ReLU activation function [4], instead of the Xavier normal or uniform initialization. The default initialization performed particularly poorly and the stochasticity limits the strength of the results [1]. Using the He (Kaiming normal) initialization produced more consistent results, as to be expected since the weights initialization has a significant impact on the performance of deep neural network [6]. The weights of each layer is initialized separately with a mean of 0 and they are scaled by a factor $\sqrt{\frac{2}{D_{in}}}$ where D_{in} is the dimension of the input layer. This ensures that the activations of the leaky ReLU for the latter layers do not reach 0, unlike with the Xavier initialization.

This implementation required the use of a learning rate schedule since the network weights initialization is much closer to a trained one, but a small constant learning rate is too slow. A decaying learning rate resulted in more consistent success in the latter episodes. The learning rate was decayed by 0.99 every 20 steps that the agent took.

Another exploration scheme used was the Value difference based exploration (VDBE) scheme, inspired by the work of M. Tokic [2]. This is an **adaptive** epsilon greedy approach where the epsilon changes depending on the Bellman error. Although the mini-batch sample may not

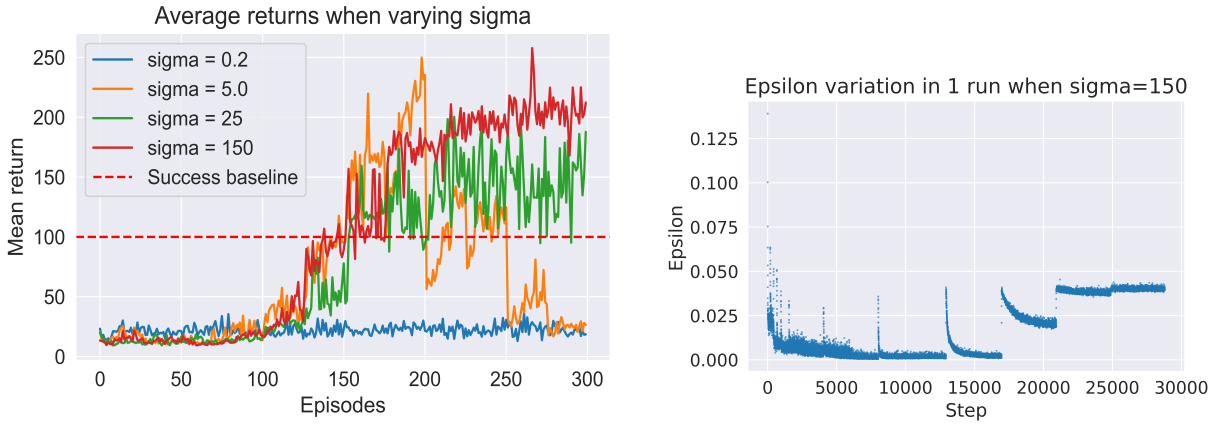

Figure 3: The graph on the left shows the average returns when varying sigma using the VDBE exploration scheme, and the graph on the right shows the epsilons for the best performing sigma value.

contain the Bellman error of the state action pair, the error does bear some correlation with the need to explore. Higher Bellman errors indicate the rate of exploration should be lowered, and lower Bellman errors indicate more exploration can be done. This is because accurate Q values imply that the state action pair has been visited multiple times. The scheme updates epsilon according to a softmax distribution as follows

$$f(\sigma, B) = \frac{1 - e^{-\frac{B}{\sigma}}}{1 + e^{-\frac{B}{\sigma}}} \tag{3}$$

$$\epsilon_{t+1} = \delta f(\sigma, B) + (1 - \delta)\epsilon_t \tag{4}$$

where σ is the **inverse sensitivity** to adjust the influence of the error on the exploration rate, $\delta \in [0, 1)$ determines the influence of the function of the error on the new epsilon value, and B is Bellman error of the mini-batch. Higher values of σ causes the agent to explore only at high values of Bellman error, lower values of inverse sensitivity cause exploration even with low Bellman errors. A recommended value for δ in the Tokic paper was $\frac{1}{|\mathcal{A}(s)|}$ which is 0.5 in this case, where the action space is push left or right.

This scheme changes epsilon every step rather than every episode, and also does not monotonically decrease as the first scheme as can be seen in Figure 3. The initialization of epsilon had no bearing on the final result as the first step would reduce the epsilon significantly. It can be seen in Figure 3 that higher inverse sensitivity, i.e. lower sensitivity produced more favourable results. There is no upper bound on sigma, but experimentation was stopped at 150 as higher values would given unreasonable results for the epsilon value. Comparing Figures 1 and 3, it can be seen that the first exploration schedule works better and will therefore be used in the further questions. In the latter episodes the VDBE scheme suffers from a lack of stability and is very volatile due to a fluctuating epsilon. The initial exploration scheme however uses a monotonically decreasing epsilon and therefore is much more stable in the latter episodes. The first decaying epsilon scheme is therefore used to train the final DQN agent. The learning curve in Figure 4 shows that the agent achieves a mean reward of 200 and it is one standard deviation above the success threshold.

1.3 Learning curve

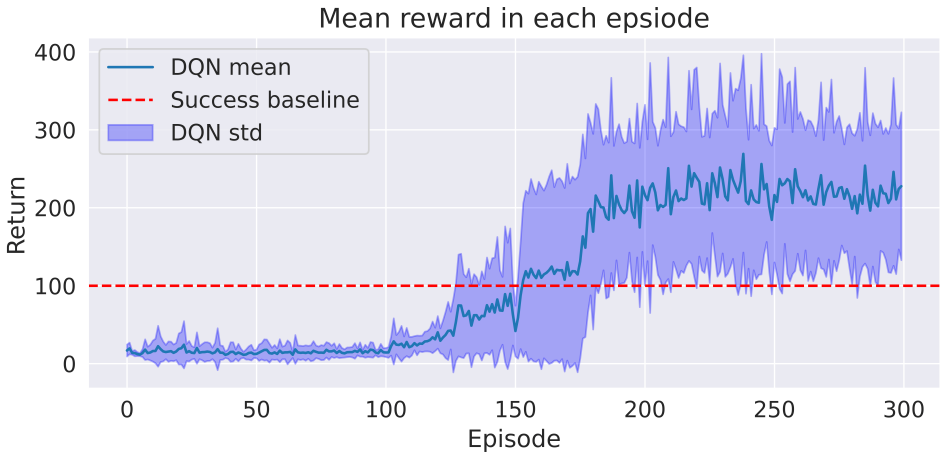

Figure 4: Learning curve for the DQN agent over 10 runs showing mean reward and standard deviation using beta = 150.

2 Question 2

2.1 Policies

The plot for v=0 shows the decision boundary crossing through the equilibrium point (0,0) where the pole is stationary and upright while the cart is also stationary. In this state, the reward obtained by taking either action is the same. The decision boundary is straight and slanted, due to the symmetry of the environment as well as the state. In the state where the pole is pointing right, the greedy policy is to push the cart to the right unless the pole has is turning anti clockwise quickly enough. This is the optimal action as it allows the cart to 'catch' the pole before it falls. But if the cart is already turning anti-clockwise then it will reach the upright position without a rightward impulse from the cart. Similarly, when the pole is pointing left, the optimal action is push left unless the pole already has a high enough clockwise angular velocity. As the instantaneous rightward velocity of the cart increases, the decision boundary moves downwards. In each of these plots the decision boundary represents the states of equilibrium, where the values of the next state will be the same no matter which action is taken. For a given pole angle, increasing the rightward velocity of the cart shifts the equilibrium position to a pole with a more negative angular velocity. For a cart moving faster to the right, the policy is to push right more often; the threshold for the leftward angular velocity is more negative for the pole to be pushed left. Interestingly, for a pole pointing left and an anti-clockwise angular velocity, the policy is not always to push left for high enough cart velocities to the right (the lower left quadrant is not always fully blue). The larger the angle to the left, the more negative the angular velocity threshold is for the policy to push it left. This is because a force applied in the direction of motion of the cart results in a smaller change in pole angle than if the force was in the opposite direction. That is, if the cart is moving right, a push to the left causes a more significant change in the pole angle, thus always pushing to the left when the pole angle and angular velocity is negative may cause the pole to overshoot and fall.

2.2 Q values

When the cart is stationary, the optimal state is for the pole to be upright and stationary as expected. The 'band' of good states lines up with the decision boundary in the policy plot; these are the states where either action may be taken. With higher rightward cart velocity, the optimal state shifts to one with a pole more tilted to the right and with a more negative angular velocity. This is particularly evident in the last plot of Figure 5 where v=2. This is the case since gravity is inducing a clockwise moment on the pole when it is pointing right, however, the rightward velocity of the cart induces an anti-clockwise moment on the pole to keep it in equilibrium. The magnitude of the velocity decides the magnitude of the moment, hence at higher rightward velocities, the optimal states are when the pole has more negative angular velocity. Unfortunately the optimal state may be impossible to obtain since the episode terminates when the pole angle moves outside the range of $(-0.2095, 0.2095)$.The least favourable states are ones where the pole is heavily tilted in one direction and the angular velocity is in the same direction, i.e. when the pole is about to fall. The Q values for the optimal states are around 12 which in this environment, indicates an expected survival time of 12 steps. This is low compared to the final rewards obtained by the agent (around 200) and accentuates the sensitivity of the environment where small changes in the state may cause a large difference in reward..

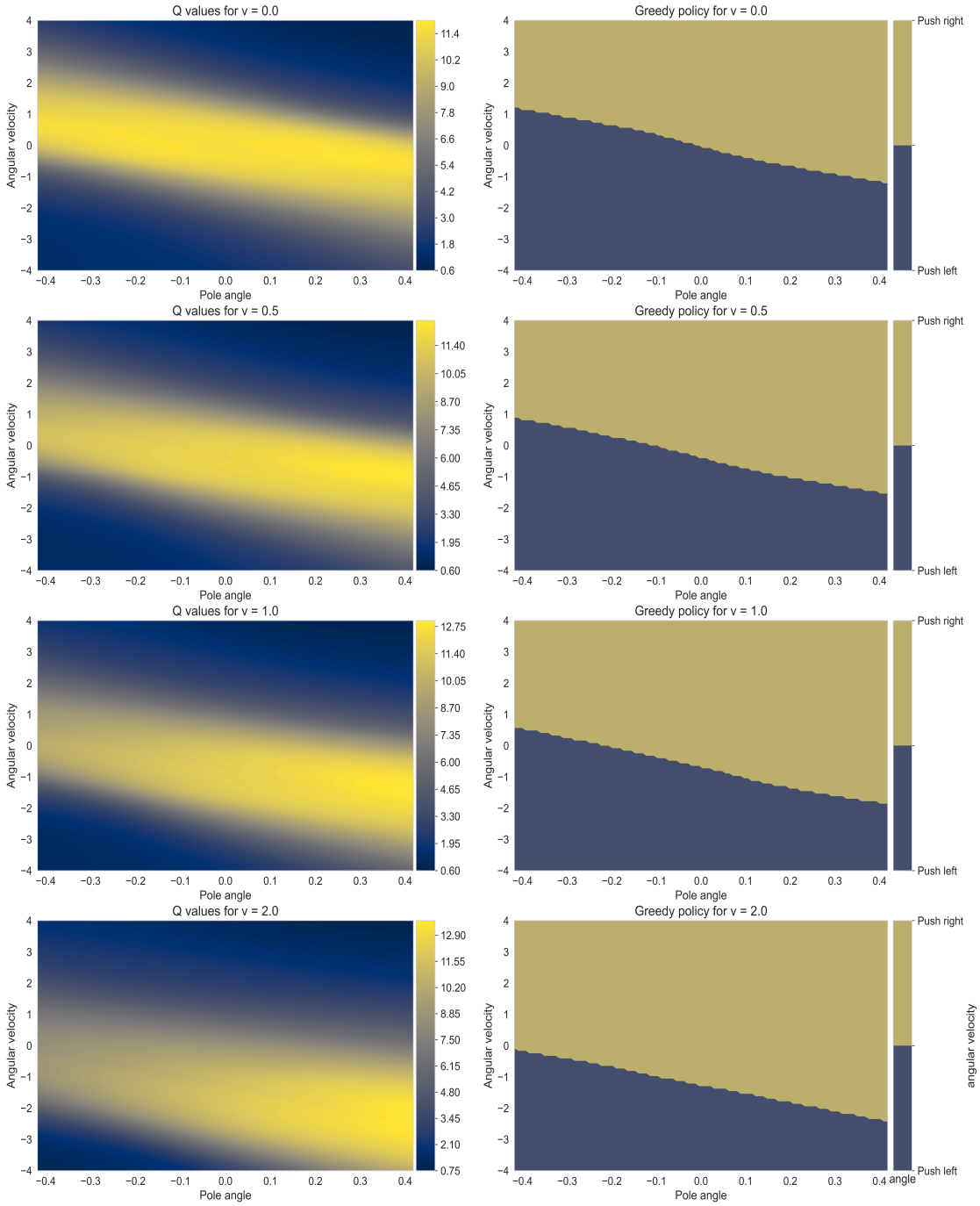

Figure 5: Q value and policy plots, averaged over 60 runs of the DQN agent using the first exploration schedule, fixing cart position at 0 and varying velocity. Units of angular velocity is rad/s and angle is radians. If the pole angle is outside the range of (-0.2095, 0.2095), the episode terminates. Greedy policy takes the state action pair with highest Q value.

3 Question 3

DDQN requires 2 networks, the policy network selects the best action for the next state and the target network evaluates the action. This has the effect of reducing the maximisation bias that arises from the bias/variance problem in standard DQN. DQN agents are more likely to overestimate the values of states unless the agent is run over a large number of episodes. The issue arises since the DQN uses bootstrapping, and hence will use the overestimated Q values to update the other Q values [3]. It was implemented by adding the following lines of code.

```
model_actions = policy_dqn(next_states).argmax(1).unsqueeze(-1)
targets = (~dones).reshape(-1)*target_dqn(next_states).gather(1, model_actions)
                                .reshape(-1) + rewards.reshape(-1)
q_values = policy_dqn(states).gather(1, actions).reshape(-1)
criterion = nn.HuberLoss(reduction='sum', delta=0.2)
return criterion(q_values, targets.squeeze(-1))
```

The code above shows the actions being selected by the policy network in one tensor operation; the unsqueeze function adjusts the dimension so that it can be used as a filter in the gather function below. The target network then evaluates the actions taken from the next states which are non terminal.

The notable difference between the agents is in the earlier episodes where the DQN agent achieves consistently low rewards with very small variance. However the DDQN agent learns more quickly than the DQN agent. This can be expected as the DDQN is using 2 network in a parallel (with a slight delay) to learn the Q values, whereas the DQN only uses the policy network to choose an action and evaluate it which is a noisy process. The DDQN agent thus has faster and more stable learning. The proximity of final results of the 2 agents suggests that in this setup, there was little or no overestimation of Q values. A further extension to the DDQN would be to introduce a soft update [7] to increase the stability.

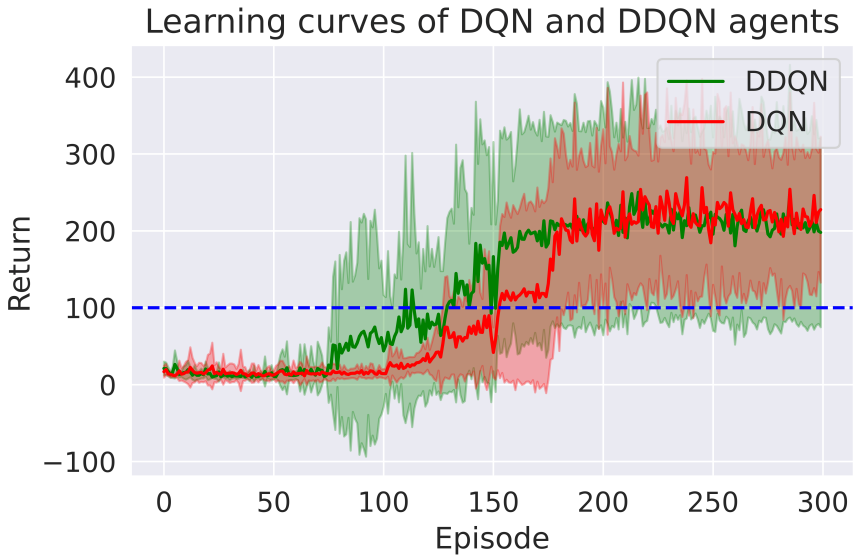

Figure 6: Comparison of learning curves for the DQN and DDQN agents. Both agents were evaluated over 10 runs using the same hyperparameters, with shaded regions representing the standard deviations.

Bibliography

[1] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. URL: https://proceedings.mlr.press/v9/glorot10a.html.

[2] Michel Tokic. "Adaptive ϵ-Greedy Exploration in Reinforcement Learning Based on Value Differences". In: *KI 2010: Advances in Artificial Intelligence*. Ed. by Rüdiger Dillmann et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 203–210. ISBN: 978-3-642-16111-7.

[3] Hado van Hasselt, Arthur Guez, and David Silver. *Deep Reinforcement Learning with Double Q-learning*. 2015. eprint: arXiv:1509.06461.

[4] Kaiming He et al. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. 2015. eprint: arXiv:1502.01852.

[5] Lu Lu et al. "Dying ReLU and Initialization: Theory and Numerical Examples". In: (2019). DOI: 10.4208/cicp.OA-2020-0165. eprint: arXiv:1903.06733.

[6] Marcin Andrychowicz et al. *What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study*. 2020. eprint: arXiv:2006.05990.

[7] Taisuke Kobayashi and Wendyam Eric Lionel Ilboudo. "t-Soft Update of Target Network for Deep Reinforcement Learning". In: (2020). DOI: 10.1016/j.neunet.2020.12.023. eprint: arXiv:2008.10861.