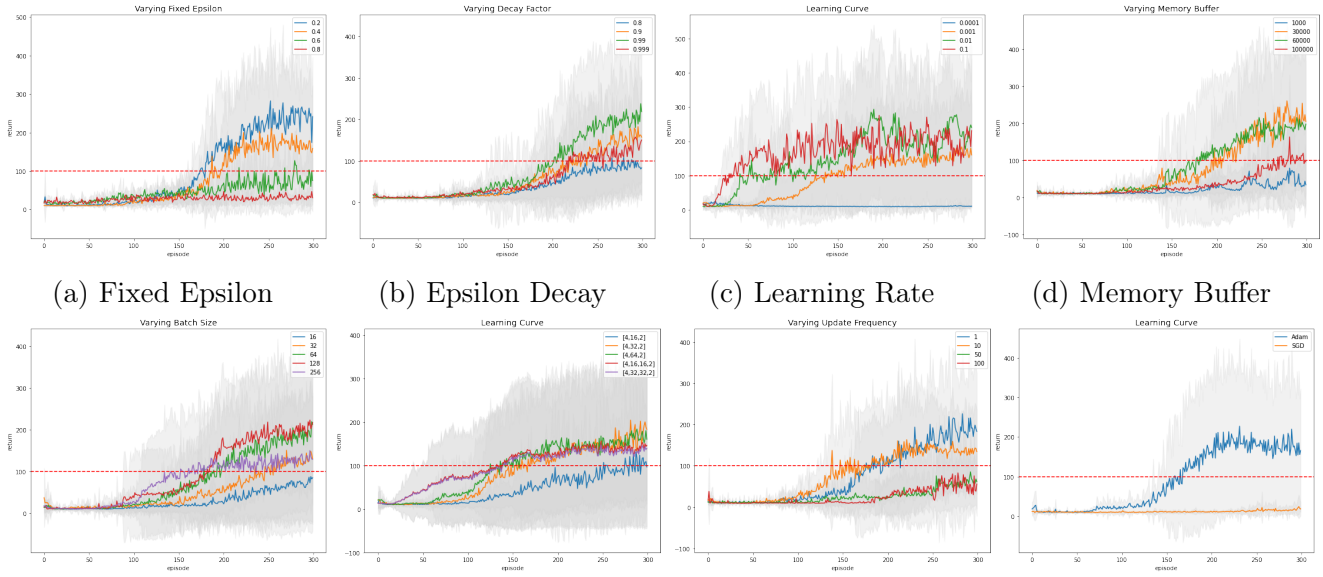
Reinforcement Learning Coursework 2

Alexander Mikheev
02346908
Computing Department
MSc Computing (Artificial Intelligence & Machine Learning)

November 2022

1 Tuning the DQN



(a) Fixed Epsilon (b) Epsilon Decay (c) Learning Rate (d) Memory Buffer

(e) Batch Size (f) Layers (g) Update Frequency (h) Optimizer

Figure 1: Plots for various manual hyperparameter experiments (averaged across 10 runs)

1.1 Hyperparameters

Hyperparameter	Value	Justification
Layer size	[4,32,2]	Figure 1(f) illustrates that increasing to 2 hidden layers results in no performance increase in comparison to single hidden layer models. This is because the additional model complexity is unnecessary for this small-scale problem where states are represented by only 4 features. Architectures [4,32,2] and [4,64,2] perform similarly, thus the simplest high-performing architecture is favoured.
Memory Buffer	60,000	From Figure 1(d) we can see that buffers of size 30,000 and 60,000 converge to the highest return values, however 60,000 has a lower variance and yields more stable results
Batch Size	128	Figure 1(e) illustrates that a batch size set to 128 converges to the highest average returns. There is a general trend of higher batch sizes increasing the performance during training.
Epsilon	1	Initially fixed epsilon was tested (Figure 1(a)) showing that a low epsilon of 0.2 yields highest returns. Fixing epsilon results in varied returns with a lot of 'spikes' in the learning curve, thus epsilon decay is favoured.
Epsilon decay	0.99	Figure 1(b) illustrates that a decay factor set to 0.99, with initial epsilon set to 1 yields the best exploration/exploitation trade-off, resulting in the highest returns.

Update Frequency	1	Figure 1(g) illustrates that an update frequency set to 1 achieves the best return values. Theoretically incorporating an update frequency such that the target policy is updated after n steps increases training stability. However increasing n has a negative impact on performance because in this setting where there are only 2 actions, the neighbouring states actually benefit from being assigned similar state-actions.
Learning Rate	0.001	Figure 1(c) illustrates the learning curves when varying the learning rate parameter. Although learning rates 0.01 and 0.1 converge faster than the selected 0.001, the variance is so extreme that when running for a single run (to learn model weights) the values are too unpredictable.
Optimizer	Adam	Figure 1(h) demonstrates that Adam performs significantly better than Stochastic Gradient Descent. .

Due to the fact that tuning a DQN is very problem specific, the decision of each particular hyperparameter value was primarily based on empirical data. The plots shown in Figure 1, vary the current hyperparameter being tested, but fix the rest of the hyperparameters to the found optima.

1.2 Exploration vs Exploitation

When training the DQN, the exploitation versus exploration trade-off is handled by the epsilon parameter. A higher epsilon essentially results in more exploration of the environment, with less exploitation of previous actions that were found effective. To obtain a high reward the agent must select the most effective actions, but to find the most effective actions that have not been discovered, the agent must explore the environment. From the fixed epsilon experiments shown in Figure 1(a), we can see that lower epsilon values yield higher returns, informing us that in this environment, a relatively small amount of exploration is sufficient to approximate the Q-function accurately. The limitation of using fixed epsilon is that later in the training curve when the Q-function is more accurate, there is no preference given to the better actions that would maximise reward. For this reason, the architecture utilizes epsilon decay after each episode; allowing high exploration initially to learn the optimal Q-values, and high exploitation with more certainty of the Q-function, resulting in a smoother learning curve with less varied returns. Figure 2 illustrates the exploration schedule of the various decay factors tested; visually the plot suggests that a decay factor set to 0.99 finds the best balance of exploration (high epsilon) and exploitation (low epsilon) over 300 episodes. This is reflected in the learning curve (Figure 1(b)), demonstrating that the best results are achieved with this hyperparameter setting.

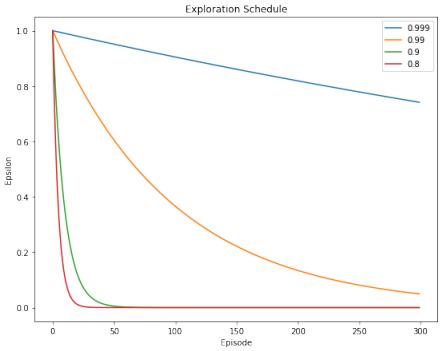


Figure 2: Exploration Schedules

1.3 Learning Curve

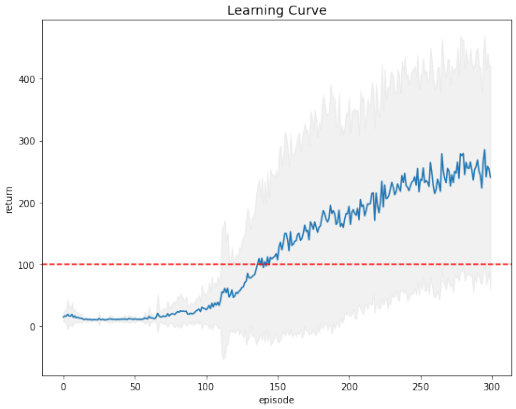


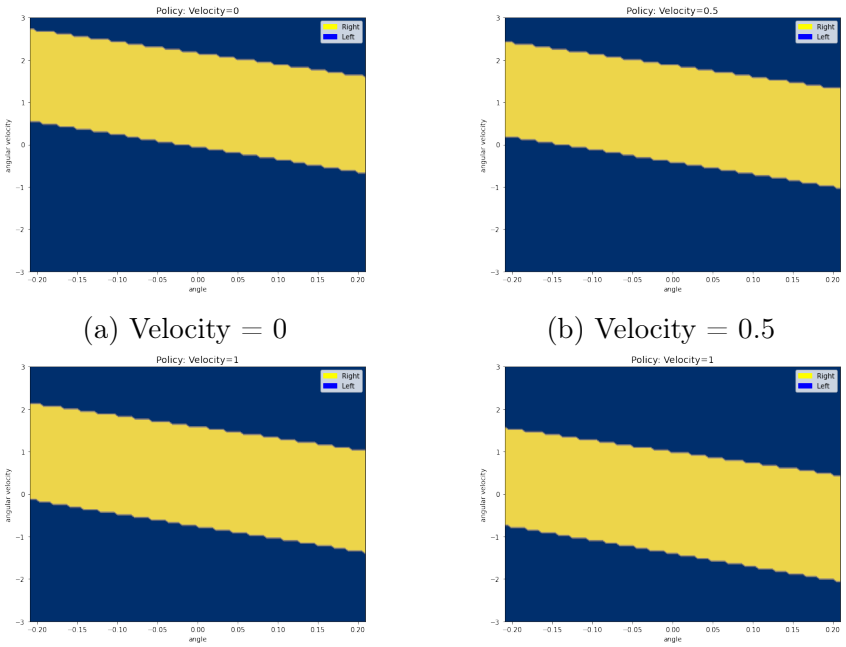
Figure 3: Average DQN learning curve over 10 runs

2 Visualising the DQN

2.1 Slices of the Greedy Policy Action

If we first consider the greedy policy plot where the cart velocity is set to 0 (Figure 4a), we can see the basic intuition that when the angle of the pole is tilted towards a particular direction (positive = right, negative = left) the agent learns to push the cart in the same direction to fight it from toppling over. The agent learns the same intuition for the angular velocity; when the pole has a velocity moving in a particular direction to each other, the model learns to balance the pole by moving the cart in the same direction. The boundary is diagonal because of the case where the two components (velocity and angle) are in the opposite direction and balance each other out, thus the model is equally happy to move the cart in either direction.

The blue triangle in the top right initially seemed counter-intuitive, but with further analysis has been identified as the states of no hope. In this case, the states are such that no action prevents the pole from falling over, therefore the model sets a default as going left. This is evidenced by observing the Q-function (Figure 5), where the top right corner states have a value of 0, because no action prevents the pole from toppling, thus collecting 0 reward. The Q-function demonstrates that these 'hopeless' cases are in the corners where there are extreme values.



(a) Velocity = 0 (b) Velocity = 0.5

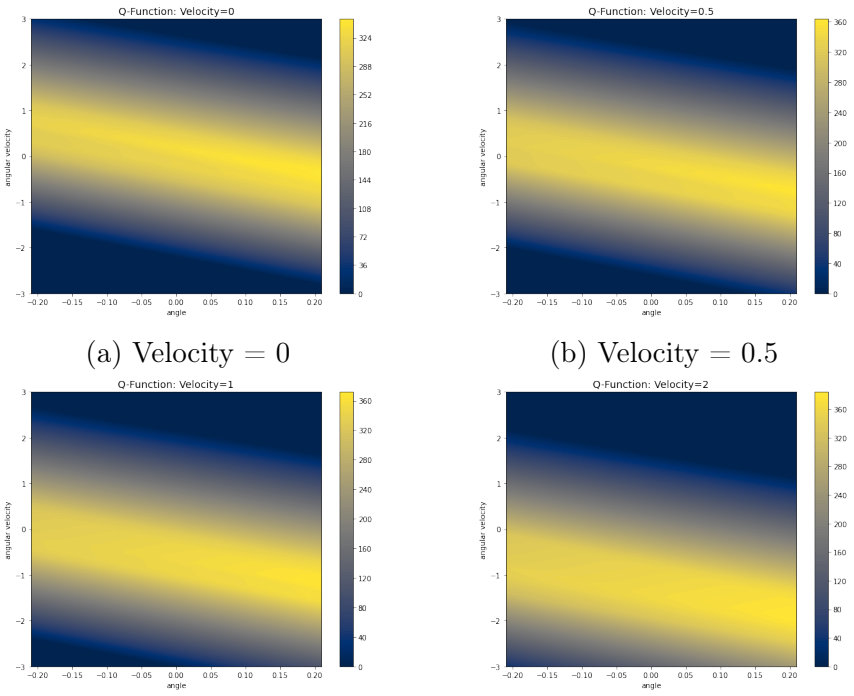
(c) Velocity = 1 (d) Velocity = 2

Figure 4: Slices of the Greedy Policy Action

Across the plots, we can see a trend that with higher cart velocity moving to the right, the decision boundary shifts down (meaning there are states with a previous greedy action assigned left are now assigned right, and the hopeless states are shifted down). We can now observe counter intuitive cases where the pole is tilted to the left, but the greedy action is to move the cart to the right (which would accelerate the toppling of the pole). To investigate this peculiarity, movie clips of training episodes were generated using the gym cartpole environment; it was observed that during training there are no cases where the cart is moving with velocity 2 at the centre point, therefore the greedy action gets assigned by extrapolating similar cases, and is not learned through experience. The decision boundary shifting down is a product of this extrapolation, and not the strategy explicitly learned by the model.

2.2 Slices of the Q-function

The Q-function is closely related to the greedy policy; the area with the highest Q-values (brightest line within the yellow diagonal plane in Figure 5) correspond to the decision boundary of the greedy policy plot (Figure 4). This outcome is intuitive because the decision boundary is where the model is equally happy moving in either direction, because the pole is balanced from the fact that the momentum and angle of the pole balance each other out, thus collecting the most reward during training. The regions that are close to the decision boundary collect high reward, as these are the non-extreme states where the pole can be easily balanced by taking the best action. Further distance from the boundary corresponds to more extreme states, where there is a higher magnitude of the angle of tilt, or the angular velocity of the pole. In these cases the pole is much closer to collapsing, therefore collects less reward during training, hence assigned a lower Q-value. By observing the colour-bar it is evident that the dark blue regions in the corners correspond to the hopeless states, where no action can prevent the pole from collapsing, thus assigned a Q-value of 0.



(a) Velocity = 0 (b) Velocity = 0.5

(c) Velocity = 1 (d) Velocity = 2

Figure 5: Slices of the Q function

As already mentioned, the Q-function is closely related to the greedy-policy, thus we can observe the same trend of the yellow plane shifting down with increasing velocity. In this case, this occurs because the decision boundary has shifted down, due to extrapolation.

3 Transform DQN into a DDQN

3.1

A problem with DQNs is that the same network is used to decide the greedy action at a particular state, and the associated Q-value of taking that action, therefore this action-value is subject to overestimation. The DDQN is an extension of the original DQN, aiming to counter this problem by decomposing this step into action selection and action estimation [1]. In the case of the DQN, the estimate for the Q-function is determined by equation (1).

$$Y_t^{DQN} = R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_t) \tag{1}$$

with the corresponding line of code within the **loss** function of the **utils.py** file below:

```
bellman_targets = (~dones).reshape(-1)*(target_dqn(next_states)).max(1).values + rewards.reshape(-1)
```

To convert to a DDQN, within the **utils.py** file, a new **DDQN_loss** function is created, where the policy network selects the greedy action, and the target network estimates the corresponding Q-value for that particular, according to equation (2).

$$Y_t^{DDQN} = R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t)\theta_t^-) \tag{2}$$

where θ_t^- is the policy network parameters, and θ_t is the target network parameters.

The corresponding line of code within the **DDQN_loss** function of the **utils.py** file is given below:

```
greedy_actions = policy_dqn(next_states).argmax(1).reshape(-1,1)
greedy_q = (target_dqn(next_states).gather(1, greedy_actions)).reshape(-1)
bellman_targets = (~dones).reshape(-1)*greedy_q + rewards.reshape(-1)
```

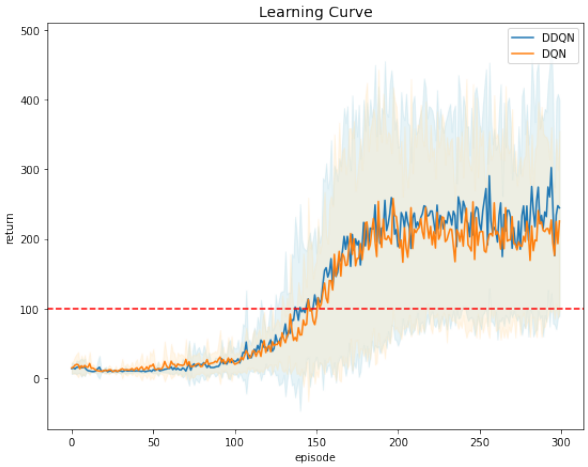


Figure 6: DDQN vs DQN learning curve

From a theoretical standpoint, DDQN should counter the overestimation of values drawback of DQN. Observing Figure 6 it is clear that the performance of DDQN and DQN with the same hyperparameters outlined in 1.1, is nearly identical, both in terms of the speed of convergence, the average returns, and variance. This outcome suggests there is little overestimation when using DQN for this particular problem, hence the target and policy networks end up selecting similar greedy actions. An important thing to consider is that the DDQN learning curve plotted in Figure 6 is not tuned

References

[1] Hado van Hasselt , Arthur Guez, and David Silver (2015). *"Deep Reinforcement Learning with Double Q-Learning"*. Google Deepmind. Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)