Assignment: Deep Q Learning
Course: Reinforcement Learning, Leiden University

Sam de Regt [1] **Lily Lin** [1] **Matthijs van Groeningen** [1]

Abstract

We present a study of the cartpole environment where we employ a Deep Q-learning algorithm to balance the pole. Our utilised Deep Q-Network (DQN) consists of a Multi-Layer Perceptron implemented in TensorFlow. The optimal hyperparameters of the DQN are explored with a grid-search of different configurations. We find that the best results are achieved with an ε-greedy policy with constant $\varepsilon = 0.05$, a replay buffer size of $bs = 800$, a target network update step of $tus = 50$, a learning rate of $\alpha = 10^{-4}$, and a single hidden layer with 64 nodes. Moreover, we find that the performance of our DQN model improves by a factor ~ 6.8 when samples are drawn from a replay buffer. The addition of a temporarily frozen target network does not improve the performance significantly, but convergence is reached $\sim 2\,500$ steps sooner. The final mean accumulated reward over 32 repetitions is measured as ~ 170, meaning that the pole could be balanced for ~ 340 steps[2]. We conclude that deep Q-learning is an effective algorithm to handle the cartpole environment.

1. Introduction

In this report, we apply reinforcement learning to a continuous environment consisting of a cart which moves horizontally and a pole which rotates around its attachment point to the cart. The pole is pulled down by gravity towards a stable state where it is hanging from the cart. The cartpole environment is described in Barto et al. (1983) and we utilise the OpenAI implementation in Gym (Brockman et al., 2016). An example is shown in figure 1. Initially, the pole

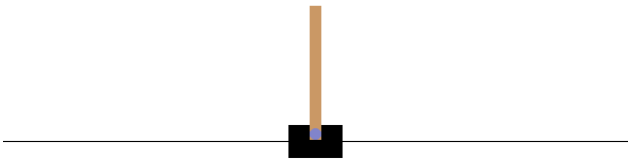

Figure 1. Illustration of the cartpole environment as implemented in Gym.

is approximately upright and the objective is to balance the pole by moving the cart left or right. Hence, there are two actions: push the cart left, or push the cart right. From the Gym environment, we retrieve four observations to describe the current state: cart position, cart velocity, pole angle, and pole angular velocity. The episode terminates if one of the following conditions occurs:

1. Pole angle is greater than $\pm 12°$
2. Cart position is greater than ± 2.4
3. More than 500 steps are simulated.

At each simulated timestep, a reward of $+1$ is returned. Therefore, the total reward for an individual episode will be higher if the pole is well-balanced.

Since the state space of the utilised environment is very large, we cannot utilise tabular reinforcement learning methods. Instead, we use a deep neural network to approximate the Q-value function which takes the state s as input and outputs the expected reward $Q(s, a)$ for each action a. By using a Deep Q-Network (DQN), we can store the function in memory and similar states can be seen as correlated.

[1]Leiden University, Leiden, The Netherlands. Correspondence to: Sam de Regt <regt@strw.leidenuniv.nl>, Lily Lin <llin@strw.leidenuniv.nl>, Matthijs van Groeningen <mvgroeningen@strw.leidenuniv.nl>.

[2]See section 3 for an explanation about the relation between the mean accumulated reward and the expected number of steps for which the pole can be balanced.

The architecture of the DQN is outlined in section 2. In section 4, we describe the results of a grid-search of hyperparameters for the neural network and the training process. Section 5 discusses the results of an ablation study where parts of the model are removed. The discussion and the conclusion of this study are found in section 6.

2. Deep Q-learning

We implemented the Deep Q-learning algorithm using TensorFlow (Abadi et al., 2015). Our employed network is a Multi-Layer Perceptron (MLP) consisting of an input layer of size 4 (cart position, cart velocity, pole angle, pole angular velocity), one or two densely-connected layers with 32 or 64 nodes using the ReLu activation function (Glorot et al., 2010), and an output layer with two nodes utilising a linear activation function. Convolutional layers are not utilised, because the cartpole environment is low-dimensional and the observations are largely unrelated. For instance, a Convolutional Neural Network (CNN) is unlikely to recognize a pattern between the cart position and the pole angle. For the hidden layers, we have experimented with different numbers of nodes and layers. As is listed in table 1, we employ either one hidden layer with 32 or 64 nodes, or we utilise two hidden layers with 32 nodes each. The DQN is trained by comparing the target G_t at timestep t:

$$G_t = r_t + \gamma \cdot \max_{a'} Q(s_{t+1}, a'), \tag{1}$$

with the actual Q-value obtained by following action a_t from state s_t. A discount of $\gamma = 0.9$ is utilised and we optimize the Mean-Squared Error loss function:

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (Q(s_i, a_i) - G_i)^2, \tag{2}$$

where we sum over a batch of observations when using a replay buffer, or we sum over only the current observation. The Adam optimizer applies the computed gradients to update the network weights (Kingma & Ba, 2014). We evaluate three different hyperparameters for the learning rate $\alpha = [0.01, 0.001, 0.0001]$ as described in table 1.

2.1. Exploration Strategy

In order to learn which actions have beneficial outcomes, the agent should not always follow the greedy policy. Instead, the state space must be explored. We have implemented several exploration strategies with four different policies:

1. An ε-greedy policy. A random action is chosen with probability ε, and otherwise the greedy policy chooses the optimal action $a = \arg\max_{b \in \mathcal{A}} Q(s, b)$, where $\mathcal{A}$ is the set of possible actions.

2. A Boltzmann policy. A softmax function calculates the probability of choosing an action:

$$\pi(a|s) = \frac{e^{Q(s,a)/\tau}}{\sum_{b \in \mathcal{A}} e^{Q(s,b)/\tau}}, \tag{3}$$

   where the temperature parameter τ determines the amount of exploration.

3. An ε-greedy policy with a decaying exploration parameter ε. By annealing ε, the algorithm initially explores the state space, but converts to exploiting the learned information of the environment.

4. A Boltzmann policy with a decaying temperature τ. Similar to the annealing ε-greedy, the algorithm transforms from exploration towards exploitation.

Our implementation of the decaying exploration parameters uses $\tau_{\text{initial}} = 1.0$, $\tau_{\text{final}} = 0.1$ and $\varepsilon_{\text{initial}} = 1.0$, $\varepsilon_{\text{final}} = 0.1$. The final exploration parameter is reached after 25% of the total timesteps. The evaluated ε and τ parameters in the grid-search are found in table 1.

2.2. Experience Replay

In supervised deep learning, the neural network is trained on a random sample of the training data. Contrarily, end-to-end reinforcement learning learns from a sequence of strongly correlated training states. For example, a pole angle of $4°$ is more likely to be preceded by a state with pole angle of $3°$ rather than $-10°$. As a result, the neural network experiences a type of overfitting to recent, similar states and does not generalize well to other states. By combining reinforcement learning with supervised learning, we can break the correlations. During training, the observations and actions are stored in a replay buffer with a certain size. If we train the network with only one sample each time, the resulting gradients for each training will have a huge variance, which stops the network from converging (Doshi, 2021). Therefore, a mini-batch of 50 samples are randomly drawn from the replay buffer to train the DQN network. Table 1 lists the different replay buffer sizes (bs) explored in the grid-search. Figure 2 shows the workflow of the experience replay method.

2.3. Target Network

For tabular Q-learning, a single state-action value $Q(s_t, a_t)$ is updated at each timestep. In the case of deep Q-learning, the weights of the entire network are updated, thus influencing the state-action values of future steps. To improve the stability of the learning process, we can temporarily freeze a copy of the DQN. This target network is used to compute the target estimate G_t and the DQN is optimized compared to a past version. After a certain number of timesteps, the

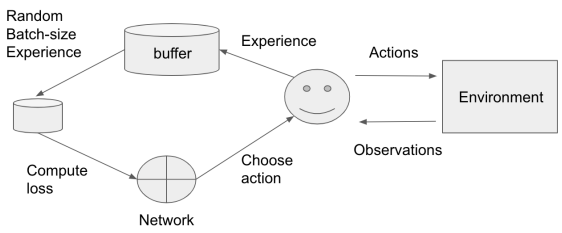

Figure 2. Experience replay workflow. Adjustment has been made based on (Choudhary, 2020).

weights of the DQN are copied to the target network weights to update the target network.

To understand why the target network stabilizes the learning, we can compare the equations of the mean-squared error. By combining equations 1 and 2, we find that the algorithm without a target network has:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left[Q(s_i, a_i) - \left(r_i + \gamma \cdot \max_{a'} Q(s_i', a')\right)\right]^2, \tag{4}$$

where s_i' denotes the next state following state s_i for sample i. For example, an update of the DQN weights for state-action (s_j, a_j) can result in an over-estimation of $Q(s_j', a_j')$ at the following state. If the observation $(s_j,\ a_j)$ is re-visited, the MSE between $Q(s_j, a_j)$ and $G_j = (r_j + \gamma \cdot \max_{a'} Q(s_j', a'))$ will be larger. When a target network Q_{target} is used, we have a different target G_i:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left[Q(s_i, a_i) - \left(r_i + \gamma \cdot \max_{a'} Q_{\text{target}}(s_i', a')\right)\right]^2 \tag{5}$$

The target G_i is independent of the updating network Q. Hence, the agent can evaluate its recent actions more effectively as the target G_i is not continuously updated. As a result, oscillations of the Q-value are avoided since the target network is not updated at each timestep. The number of steps between updating the target network (tus) provides another hyperparameter which we analyse in the grid-search (see table 1).

3. Performance metric

In Fig. 3, 4, 5 and 6, the results of our experiments are presented. These experiments constitute an exploration of hyperparameters, as explained in section 4, and an ablation study, explained in section 5. In order to evaluate the performance of our model for different configurations of hyperparameters and model features, we use the following metric. For every step taken by the agent, we store the accumulated reward of the current episode. As the reward for each step is $+1$, the accumulated rewards of an episode consisting of just three steps would be $\{1, 2, 3\}$. When an episode has terminated, we reset the accumulated reward to 0, so the accumulated rewards of two subsequent three-step episodes are $\{1, 2, 3, 1, 2, 3\}$. In practice, due to the termination conditions, the minimum number of steps per episode is 8. For each experiment, we train the model for a fixed number of steps and repeat this for a fixed number of repetitions. Then we take the mean value over the repetitions. For the plots, we smooth the mean accumulated reward with a Savitzky-Golay filter. Note that, on average, an episode lasts almost twice as many steps as the mean accumulated reward. From the repetitions, we obtain a sample of accumulated rewards for each step. The mean of a sample of accumulated rewards between 1 and N will tend to $(N + 1)/2$, where N is the average episode length for a certain step. So a mean accumulated reward of ~ 100 means that an episode lasts for approximately 200 steps on average.

4. Exploration of hyperparameters

The performance of any reinforcement learning algorithm depends on the choice of hyperparameters. In this section, we present our exploration of hyperparameters to determine the optimal configuration. The tested hyperparameters are listed in table 1.

4.1. Exploration, buffer size & target network update

First, we investigate which combination of exploration strategy, target network update step (tus), and buffer size (bs) gave the best results. In total, we tested $(2 \text{ policies}) \cdot (4\ \varepsilon/\tau) \cdot (3\ bs) \cdot (3\ tus) = 72$ hyperparameter configurations. In these experiments, the learning rate is fixed to 0.01 and the neural network uses a single hidden layer with 32 nodes. The total budget of the experiments is set to 10 000 timesteps and each experiments is repeated 8 times to retrieve consistent results.

4.1.1. ε-GREEDY POLICY

The learning curves resulting from the ε-greedy policy are presented in the subfigures of figure 3. The 8 repetitions are averaged and the curves are smoothed with a window of 1 001 steps. For any adaptation of exploration, we find that the model learns faster with the smallest target network update steps ($tus = 50$, blue curves) than with the less frequent updates ($tus = 800$, green curves). If the target network is updated more frequently, it adopts the new network weights quicker, therefore allowing the model to learn quicker. The choice of a replay buffer size plays a less im-

Hyperparameters	Setting 1	Setting 2	Setting 3	Setting 4
Network architecture	32	64	32, 32	-
Learning rate α (lr)	0.01	0.001	0.0001	-
ε (ε-greedy policy)	0.2	0.1	0.05	Linearly annealing ($\varepsilon_{\text{initial}} = 1.0$, $\varepsilon_{\text{final}} = 0.1$)
Temperature τ (Boltzmann policy)	10	1	0.1	Linearly annealing ($\tau_{\text{initial}} = 1.0$, $\tau_{\text{final}} = 0.1$)
Buffer size (bs)	50	200	800	-
Target network update step (tus)	50	200	800	-

Table 1. Hyperparameters explored in the grid-search presented in section 4.

portant role in the model performance. In general, however, the models learn somewhat faster with the largest buffer size ($bs = 800$, dash-dotted curves). A large replay buffer allows the algorithm to sample more uncorrelated observations from previous episodes and the algorithm therefore avoids getting stuck at local optima. Comparing the different exploration strategies, we find minor differences in the overall results. As expected, the models learn faster with the largest exploration parameters, that is $\varepsilon = 0.2$ and the annealing $\varepsilon_{\text{initial}} = 1.0$. On the other hand, the smaller $\varepsilon = 0.05$ and $\varepsilon = 0.1$ reach higher mean rewards. In conclusion, we find that a configuration with replay buffer size equal to 800, a target update step equal to 50 and an exploration parameter $\varepsilon = 0.05$ attain the best results. These hyperparameters are passed on for the future hyperparameter tuning of the learning rate and the network architecture.

4.1.2. Boltzmann policy

The learning curves resulting from the Boltzmann policy are presented in the subfigures of figure 4. The 8 repetitions are averaged and the curves are smoothed with a window of $1\,001$ steps. The temperature parameter (τ) for Boltzmann policy is less intuitive for tuning than the ε in ε-greedy policy. The models with $\tau = 1.0$ and 10.0 perform extremely poorly with final rewards of ~ 80 and ~ 20, respectively. Under these two configurations, the agent is mostly exploring the environment, taking semi-random actions. Hence, the learned Q-values are barely used and the mean reward shows very little progress. For $\tau = 0.1$ and when τ is annealing from 1.0 to 0.1, the models perform similar to the ε-greedy policy. As with the ε-greedy results, we find that the network obtains better results when the target network is updated more frequently and with a larger replay buffer size.

4.2. Learning rate & network architecture

To further explore the best choice of the learning rate and network architecture, we employ the hyperparameter configuration found in section 4.1.1 ($bs = 800$, $tus = 50$, constant $\varepsilon = 0.05$). In this section, we evaluate 9 different models with learning rates $\alpha = [10^{-2}, 10^{-3}, 10^{-4}]$ and network architectures 32, 64 and [32,32]. Each configuration is repeated 16 times with a total budget of $30\,000$ steps. The resulting mean reward is shown in figure 5 where the curves are smoothed with a window of $3\,001$ steps. For any architecture, we find that a learning rate of 0.001 learns fastest and reaches its first local maximum around $\sim 5\,000$ steps. It takes the configurations with a learning rate of 0.01 around $\sim 3\,000$ more steps to reach their local maxima. Here it appears that the lower learning rate stabilizes the learning and the minima of the loss function are occasionally overshot with the higher learning rate. If we continue to decrease the learning rate to a value of 0.0001, the networks learn slower, but reach higher mean rewards at the final epoch. The evaluated network architectures do not affect the performance as much as the learning rate does. For learning rates 0.001 and 0.0001, the architecture with a single layer of 64 nodes reaches the highest rewards. The two-layer network with 32 nodes at each layer performs best with the highest learning rate 0.01.

5. Ablation study

We performed an ablation study to determine how the addition of an experience replay (ER) buffer and a target network (TN) affect the DQN's performance. We trained 4 different models:

1. DQN: complete network,
2. DQN-ER: network without experience replay,
3. DQN-TN: network without target network,
4. DQN-ER-TN: network without experience replay and without the target network.

The hyperparameters were fixed to the optimal result found in the grid search of section 4. A learning rate of $\alpha = 10^{-4}$ was used to train neural networks with a single hidden layer consisting of 64 nodes. The ε-greedy policy was utilised with a constant exploration parameter of $\varepsilon = 0.05$. When using a replay buffer, the buffer size was set to $bs = 800$ samples and a target update step of $tus = 50$ was employed when a separate target network was utilised. Each

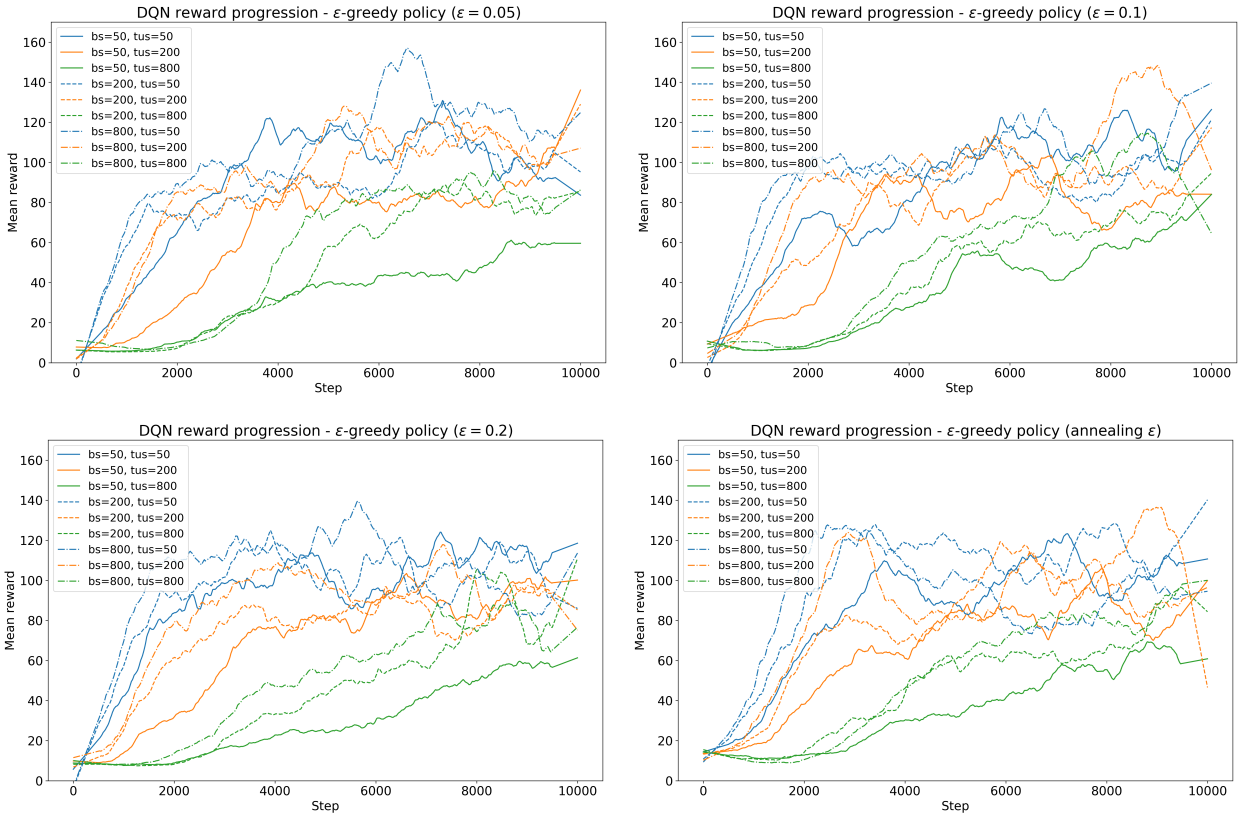

Figure 3. Learning curves for hyperparameters explored in the grid-search (figure 1) with ε-greedy policy. Each of the curves here is the smoothed average over 8 repetitions. All of the results in this figure are produced by a neural network with a single hidden dense layer of 32 nodes and a learning rate of 0.01. In each sub-plot, the replay buffer size (bs) and target update step (tus) are varied, while ε remains an unchanged variable.

network was trained for $50\,000$ steps and for 32 repetitions. The learning curves of the repetitions were averaged and subsequently smoothed with a window of $5\,001$ steps to obtain the results shown in figure 6.

In figure 6, we notice a clear distinction between models which implemented a replay buffer and models without experience replay. The models with experience replay (DQN-TN and DQN) reach mean rewards of ~ 170 whereas models without replay (DQN-TN-ER and DQN-ER) attain mean rewards of ~ 25. The DQNs with replay can keep the pole upright for $170/25 \sim 6.8$ times more steps. This result was expected because the models with experience replay can generalize better to other states. These models do not suffer from overfitting to the most recent observation. For the first $10\,000$ steps, a decreasing mean reward is observed for the models without replay as a result of the correlated observations. An upward increasing trend is seen from $\sim 30\,000$ to $50\,000$ steps, but it is unclear if this trend continues beyond our chosen budget.

In contrast with experience replay, there is not a clear difference between models which use a separate target network and models which use the same network to calculate the targets. We expected that freezing the weights of the target network would stabilize the learning. However, it appears that our choice of hyperparameters already stabilized the learning curves. A minor difference that we observe is that the complete DQN model converges to its optimal reward roughly $\sim 2\,500$ steps sooner than the model which does not freeze the weights of the target network.

6. Discussion & conclusion

We have applied deep Q-learning to the cartpole environment and examined which combination of hyperparameters was most effective in balancing the pole. Moreover, we assessed how the removal of a replay buffer and a separate target network affected the model's performance, learning speed and stability.

Through our grid search of hyperparameters, we found that an ϵ-greedy policy with a constant $\epsilon = 0.05$ worked best for

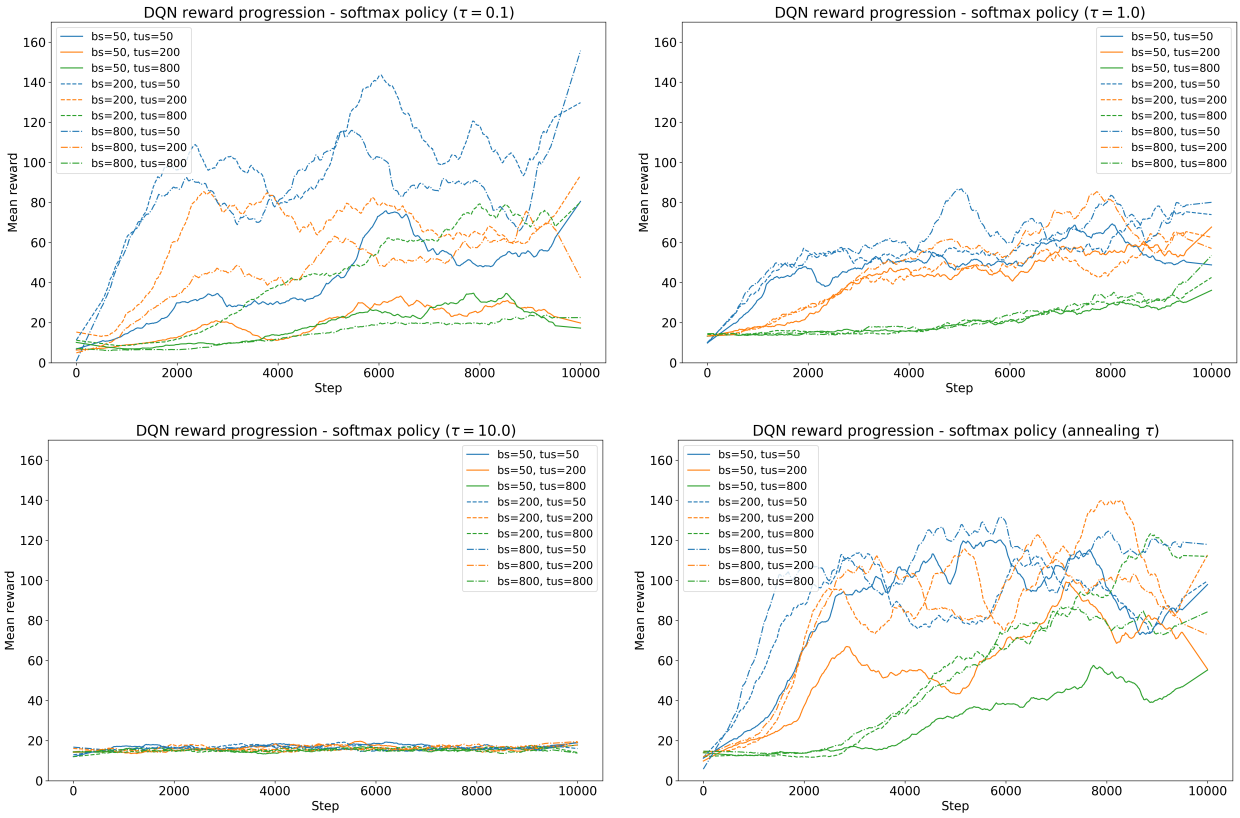

Figure 4. Learning curves for hyperparameters explored in the grid-search (figure 1) with the softmax policy. Each of the curves here is the smoothed average over 8 repetitions. All of the results in this figure are produced by a neural network with a single hidden dense layer of 32 nodes and a learning rate of 0.01. In each sub-plot, the replay buffer size (bs) and target update step (tus) are varied, while τ remains an unchanged variable.

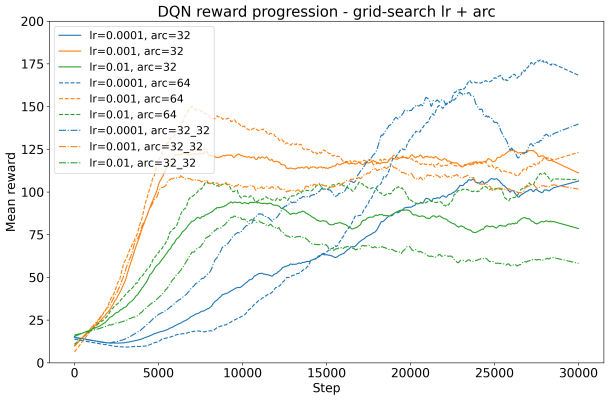

Figure 5. Performance of the DQN model for different learning rates and neural network architectures as given in Table 1. The other hyperparameters are fixed at a policy of ε-greedy with $\varepsilon = 0.05$, a replay buffer size of 800 and a target update step size of 50. Each curve is the result of the smoothed average of 16 repetitions.

this task. Additionally, a replay buffer size of $bs = 800$ and target network update step of $tus = 50$ worked allowed the DQN to be more stable and generalize to varying observations. A small learning rate of $\alpha = 10^{-4}$ was utilised for the Adam optimizer as it obtained the highest final reward in our tests. Furthermore, a simple network architecture of a single hidden layer with 64 nodes was sufficient to learn this cartpole problem.

In the ablation study, we found that the addition of a replay buffer was crucial to the performance of the DQN model. Without a replay buffer, the networks would achieve mean rewards of $\sim$ 20, whereas a mean reward of $\sim$ 170 was obtained by sampling from a replay buffer which stored previous observations. The implementation of a target network did not result in very contrasting results, but appeared to speed up the learning by $\sim 2\,500$ steps.

After training for $50\,000$ steps, our implementation of the complete DQN converged to a mean reward of 170 per episode. Hence, on average, the pole could be balanced for 340 timesteps by moving the cart left or right. In conclusion,

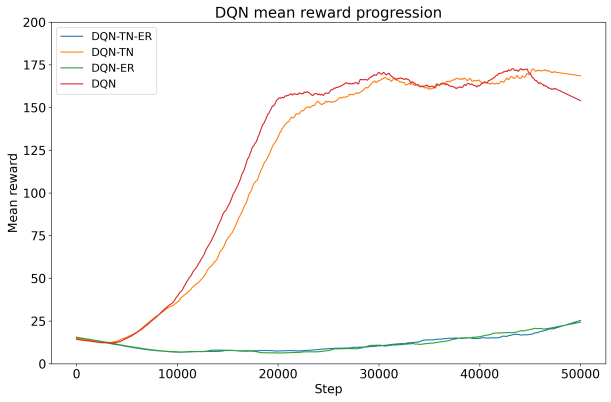

Figure 6. Results of the ablation study, which evaluates the effect of experience replay and a target network on the performance of the DQN model. Each configuration is evaluated with a learning rate of 0.0001, a neural network architecture of one hidden layer with 64 nodes and an ε-greedy policy with $\varepsilon = 0.05$. If experience replay was used, the replay buffer size was 800 and for a configuration with a target network, a target update step size of 50 was used.

deep Q-learning appears to be an effective method to deal with this cartpole environment.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Barto, A. G., Sutton, R. S., and Anderson, C. W. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983. doi: 10.1109/TSMC.1983.6313077.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016. URL https://arxiv.org/abs/1606.01540.

Choudhary, A. Deep q-learning: An introduction to deep reinforcement learning, Apr 2020. URL https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/#:~:text=In%20deep%20Q%2Dlearning%2C%20we,is%20generated%20as%20the%20output.

Doshi, K. Reinforcement learning explained visually (part 5): Deep q networks, step-by-step, Feb 2021. URL https://towardsdatascience.com/reinforcement-learning-explained-\visually-part-5-deep-q-networks\-step-by-step-5a5317197f4b.

Glorot, X., Bordes, A., and Bengio, Y. Deep sparse rectifier neural networks. volume 15, 01 2010.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization, 2014. URL https://arxiv.org/abs/1412.6980.