Balancing an Inverted Pendulum Using Deep Q Network

John Sunny George
Aerospace Engineering
University of Illinois at Urbana-Champaign
Champaign, USA

***Abstract*—The problem of balancing a simple pendulum in an upright position by controlling the input torque is solved using the Deep Q Network (DQN) algorithm for reinforcement learning.**

I. Introduction

A. Dynamics of a Simple Pendulum

The motion of a simple pendulum is governed by a second-order ordinary differential equation given as follows:

$$ml^2\ddot{\theta}(t) + b\dot{\theta(t)} + mgl\sin\theta(t) = u(t) \tag{1}$$

where the state variables are angular position θ and angular velocity $\dot{\theta}$, and the control input is torque $u(t)$. m, l, g, b are the mass, length, acceleration due to gravity, and coefficient of viscous friction of the pendulum, respectively. The advantage of Deep Q Network (DQN) over Q-learning is its ability to handle continuous state spaces. However, DQN still requires a finite action space. For this reason, in this experiment, the input torque, $u(t)$, is modeled as a discrete action space of dimension 31, with a maximum absolute value of $5\,\mathrm{Nm}$.

B. Reinforcement Learning Objective

The objective of the agent is to hold the pendulum in an upright position. In a reinforcement learning framework, an agent learns to achieve such a goal through a carefully engineered reward signal which is described as follows:

- If the absolute value of angular velocity exceeds a prescribed maximum value, the agent receives a sizeable negative reward of -100.
- If the absolute value of the angular position is less than a prescribed maximum value, the pendulum is said to be upright, and the agent receives a small positive reward of 1.
- In all other cases, the agent receives no reward.

The agent achieves the goal by interacting with the environment by selecting actions (adjusting the torque) to maximize future reward (maintaining the upright position).

II. Problem Approach

A. Neural Network Architecture

For the Deep Q Network (DQN), a fully connected neural network with two hidden layers, each having 64 units, was used. A $\tanh$ activation function was used at both the hidden layers, with a linear activation function at the output layer. The input vector is two-dimensional, each component referring to angular position (θ) and angular velocity ($\dot{\theta}$). The output vector's dimension depends upon the size of the discrete action space (31 for this experiment).

B. Training Details

All experiments were run for 1000 episodes, each containing 100 training steps with a discount factor (γ) of 0.95. The behavior policy during training was ϵ-greedy with ϵ linearly annealed from 0.9 to 0.05 over the first 10000 training steps. AdamW [1] algorithm with a learning rate of 0.0001 and mini-batches of size 128 was used to update the neural network. Other parameters of the AdamW algorithm were set to the default values in the Pytorch [2] implementation.

The loss function for training the neural network was chosen as Huber loss [3], which is less sensitive to outliers in data than the squared error loss. So the loss function in algorithm 1 described in [4] is reformulated as:

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \le \delta \\ \delta.\big(|a| - \frac{1}{2}\delta\big), & \text{otherwise} \end{cases} \tag{2}$$

where,

$$a = y_j - Q(s_j, a_j; \theta) \tag{3}$$

where y_j, Q_j, s_j are the target Q-value, Q-value from policy network and state, respectively, of the j-th transition from a random sample of minibatch, as described in [4]. θ is policy network parameters. The δ in Huber loss is set to 1. The gradient values required for updating the neural network parameters are clipped between -100 and 100 to improve the algorithm's stability. TABLE I shows the list of all hyperparameters used in the experiments.

C. Implementation

The code was written in python and PyTorch. The yml file required to re-create the environment is provided in the GitHub repository.

III. Results

A. DQN Agent

Target network and memory replay were used along with the parameters mentioned in TABLE I to obtain the results unless specified. Fig. 1 shows the learning curve of the DQN agent

TABLE I
HYPERPARAMETERS AND THEIR VALUES.

Hyperparameter	Value
Minibatch size	128
Replay memory size	10000
Target network update frequency	500
Discount factor	0.95
Learning rate	0.0001
Initial exploration	0.9
Final exploration	0.05
Final exploration steps	10000

for both discounted and undiscounted returns. The algorithm converges to the optimal policy roughly by 400 episodes.

Fig. 2 is a heatmap of the optimal policy for different combinations of angular position and velocity, and the policy is also physically intuitive. Consider a case when the pendulum is upright, i.e., $\theta = 0.03$ rad. An anti-clockwise torque should be expected to maintain the upright position for a clockwise angular velocity ($\dot{\theta}$). The policy seems to follow this physical intuition and is more clearly observable in Fig. 3.

The trajectory plots shown in Fig. 4 also show that the agent can bring the pendulum from its stable equilibrium to its unstable equilibrium (upright position), using a sufficiently large torque and maintain the pendulum upright by a torque that counterbalances the angular velocity.

Fig. 5 shows the heatmap of the state-value function (maximum action-value) for several combinations of angular position and velocity. The maximum value function is observed when the pendulum is upright with no angular velocity. Fig. 6 shows the trajectory plotted over the value function and how the agent traces a path that maximizes the value function.

B. Ablation Study

DQN agents were trained using the standard hyperparameters mentioned in TABLE I for all possible combinations of turning replay on and off, using or not using a separate target Q-network. Fig. 7 and Fig. 8 show the learning curve and the maximum mean returns for each combination. Experience replay breaks the correlation between consecutive updates. It prevents the learning process from being influenced too heavily by the sequence of experiences encountered during training. At the same time, a target network helps in reducing the correlation between the current estimates of Q-values and the target Q-values. Although Fig 7 shows that the agent without replay and target was less stable during training, it was not reproducible for other tests. So, the advantages of these techniques were not observable in the current experiment. This is likely due to using other techniques like Huber loss and AdamW optimizer, which are additional sources of stability.

However, experiments were conducted using mean squared error and RMSprop algorithm using the parameters described in [4]. In these experiments, the advantage of experience replay and target network was observable, although slightly, as shown

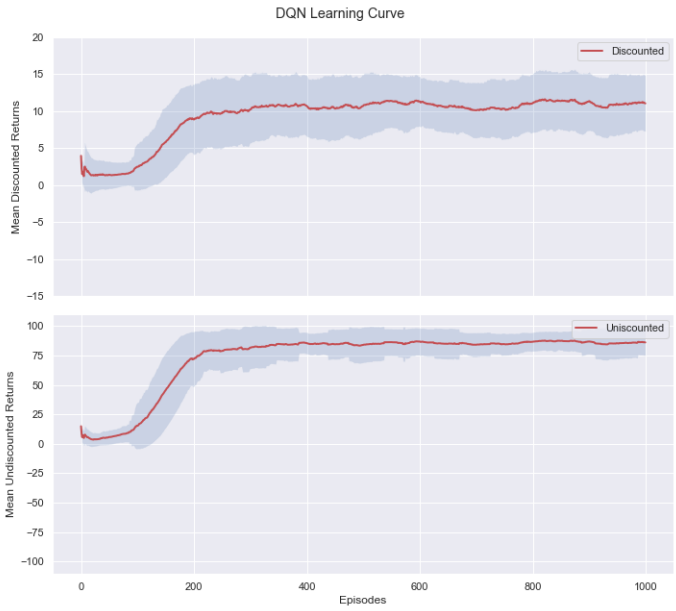

Fig. 1. DQN learning curve. The plot shows a running average of over 100 episodic returns. Shaded bands indicate one standard deviation over 100 episodes. For the first 100 episodes, the mean and standard deviation at n-th episode is computed from all episodes $\leq n$.

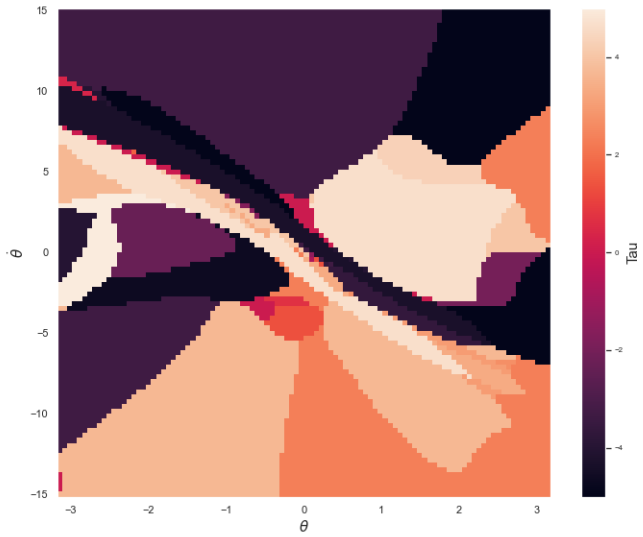

Fig. 2. Policy

in Fig 9 and Fig 10, where the agent trained without experience replay and a target has the lowest maximum mean returns.

IV. CONCLUSIONS

The DQN agent was successfully trained to balance an inverted pendulum, as shown by the trajectory plots and animation (in GitHub). However, the advantages of using a target network and experience replay were not pronounced. Plots similar to the one shown in the report are available on GitHub for all combinations used in the ablation study.

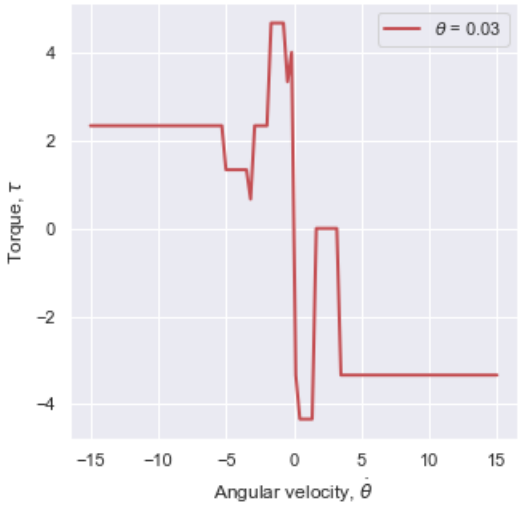

Fig. 3. Value of torque recommended by the agent for varying angular velocity in the upright position.

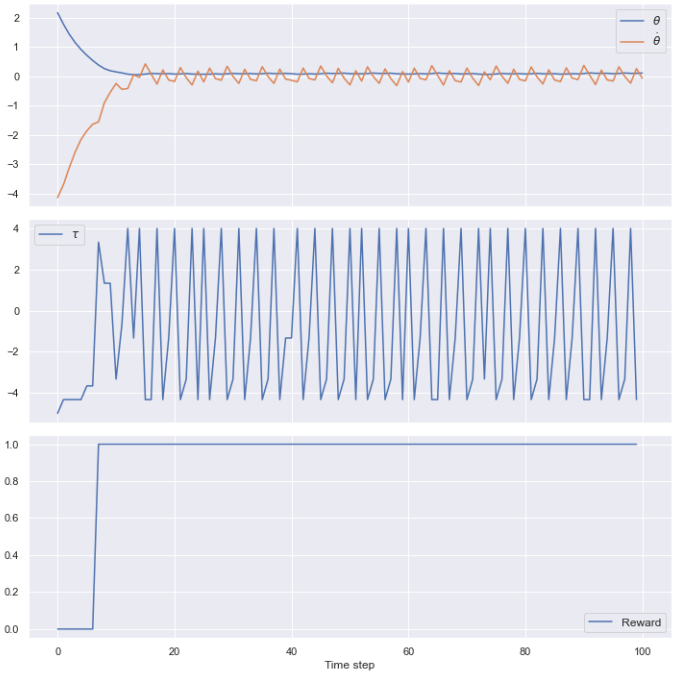

Fig. 4. Trajectory.

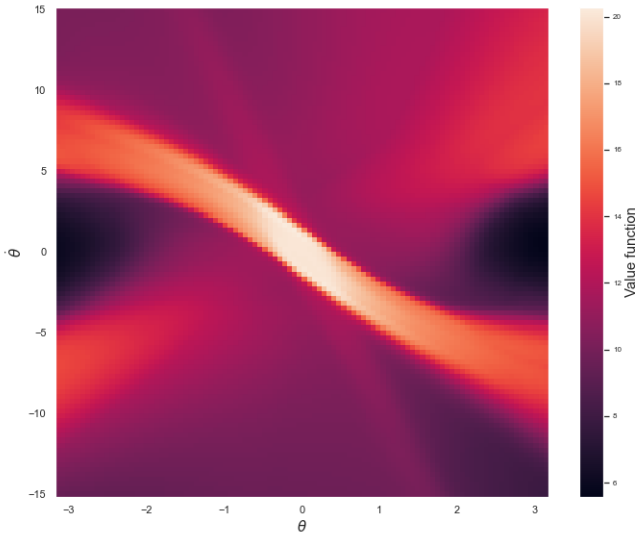

Fig. 5. Value function.

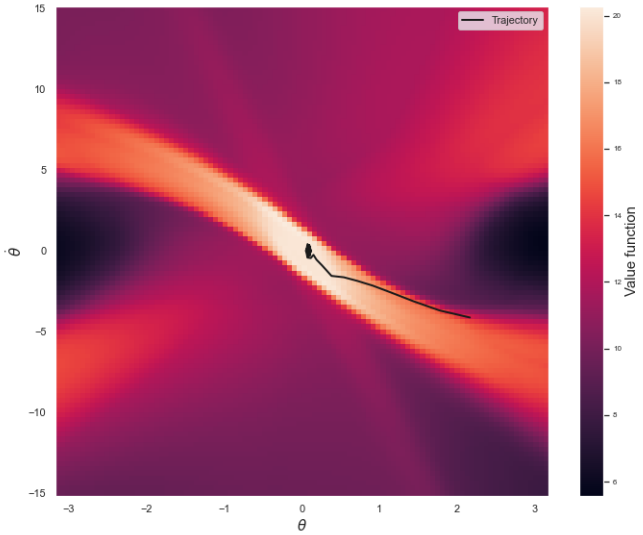

Fig. 6. Trajectory over value function.

Also, the ”test1” directory in the GitHub repository contains all the experiments repeated for the RMSprop algorithm and MSE loss. The training was generally more stable when using AdamW and Huber loss than RMSprop and MSE loss.

REFERENCES

[1] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” 2019.

[2] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

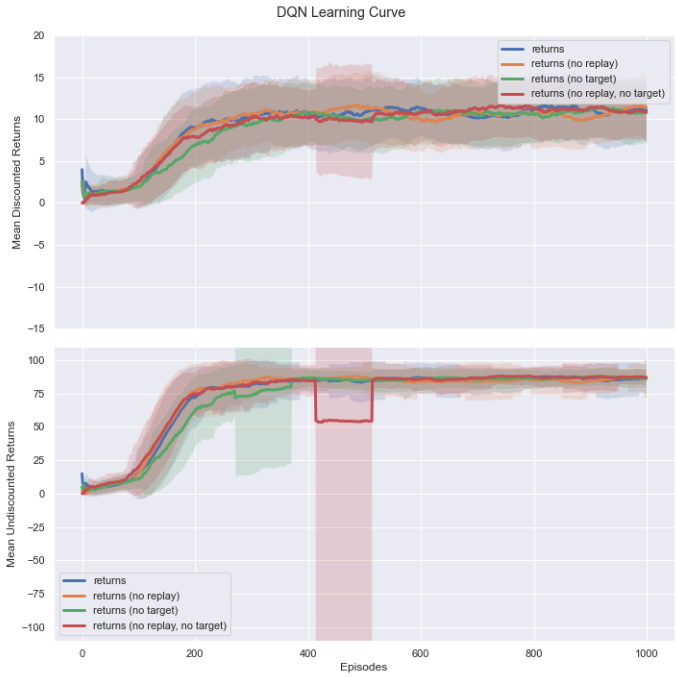

Fig. 7. DQN learning curve for all combinations using Huber loss and AdamW algorithm.

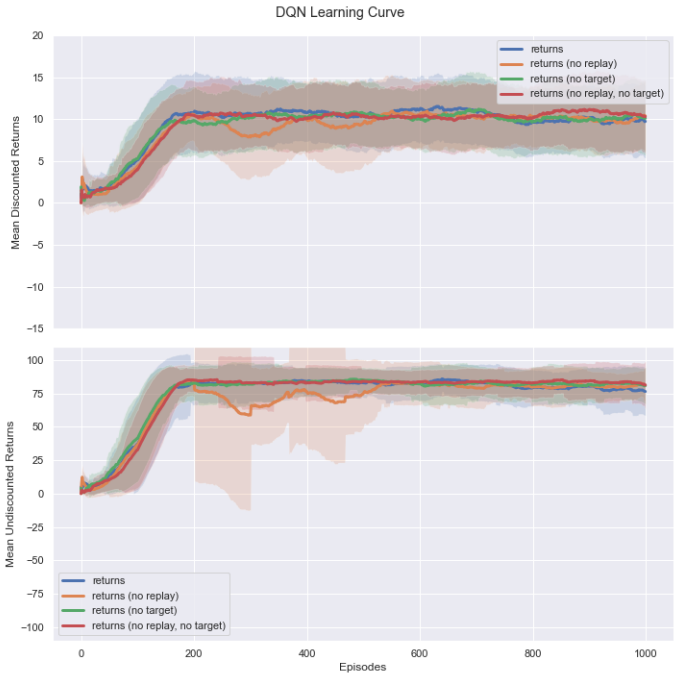

Fig. 9. DQN learning curve for all combinations using MSE loss and RMSprop.

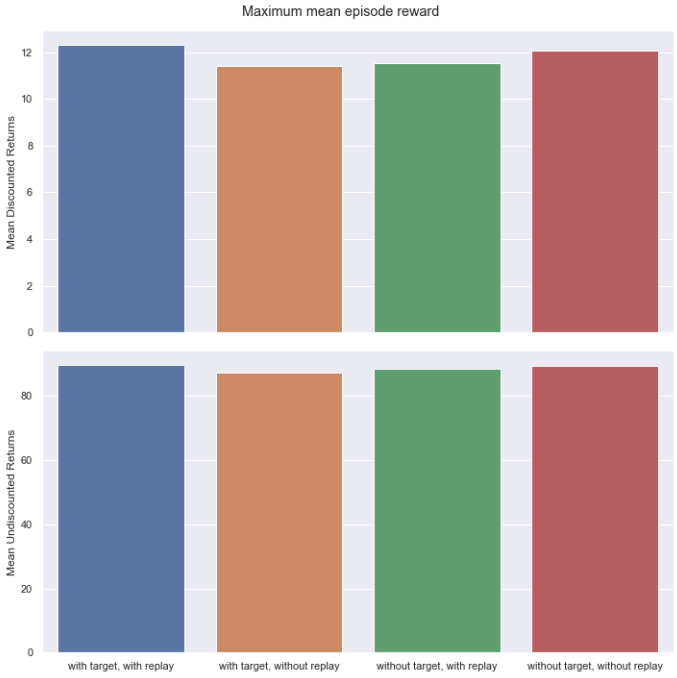

Fig. 8. Each agent was tested for 500 episodes using Huber loss and AdamW algorithm, and a maximum running average over 100 episodes is shown in the plots.

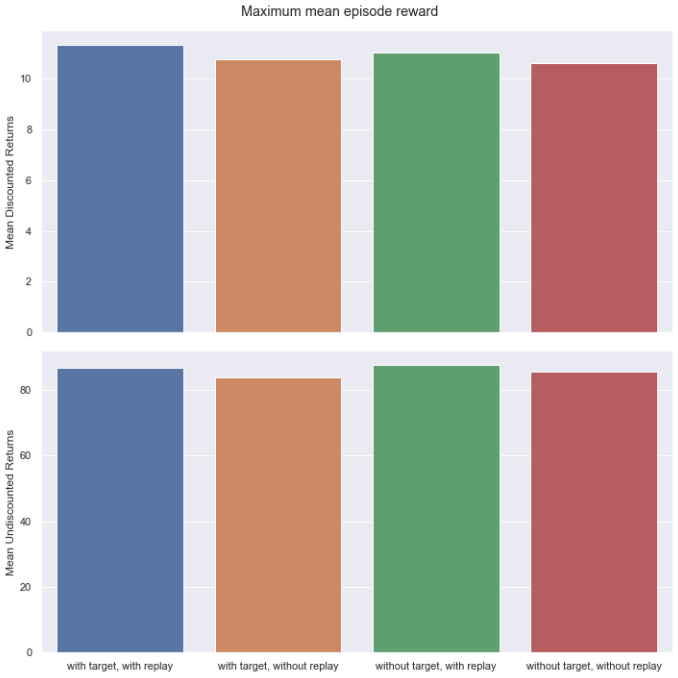

Fig. 10. Each agent was tested for 500 episodes using MSE loss and RMSprop algorithm, and a maximum running average over 100 episodes is shown in the plots.

[3] P. J. Huber, "Robust Estimation of a Location Parameter," *The Annals of Mathematical Statistics*, vol. 35, no. 1, pp. 73 – 101, 1964. [Online]. Available: https://doi.org/10.1214/aoms/1177703732

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236