HW2: DQN

Emma Clark
Department of Aerospace Engineering
UIUC
emmac4

I. Introduction

A. Environment

The environment being used as a simple pendulum with a discrete action space and continuous state space; where the transition model describing the transition dynamics of the system is not available. The two states of the environment ($\theta, \dot{\theta}$ are the angle of the pendulum and the angular velocity, respectively. The goal task of this environment is for the agent to learn to hold the pendulum upright, within a specified range of θ values, without violating a maximum angular velocity of 15. Every environment step for which the pendulum is considered upright, a reward of $+1$ is given, and for every step in which the maximum $\dot{\theta}$ is violated, a reward of -100 is given; for any other state there is no reward given. There is a maximum of 100 steps for this environment; meaning the maximum possible reward would be 100 - though this may be unrealistic in practice, given the random state initialization, the pendulum may have an initial state not already upright, so steps will need to be taken to get to that position.

B. DQN

Deep Q-Learning (DQN) is off policy and model free method. DQN is a sort of extension of traditional Q-Learning for continuous state spaces. This is done by replacing the tabular learned value function with a neural network. This nueral network, or Q-network, is a function approximator parameterized by weights θ, $Q(s, a, \theta)$; [Note that θ used in this section and the following is **not** the same as the θ state in the environment discussed previously] allowing for the optimal state-value function

$$Q^*(s,a) = \max_{\pi} \mathbf{E}\Big[\sum_{t=1}^{T} \gamma^{t-1} r_t | s_t = s, a_t = a\Big] \quad (1)$$

to be estimated

$$Q^*(s,a,\theta) \approx Q^*(s,a) \quad (2)$$

The Q-network is then trained iteratively to reduce the mean squared error of the Bellman Equation:

$$\big(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big)^2 \quad (3)$$

In practice, this learning process is represented as the minimization of the loss function:

$$L_i = \mathbf{E}_{(s,a,r,s'}\Big[\big(r + \gamma \max_{a'} Q(s',a',\theta_i) - Q(s,a,\theta_i)\big)^2\Big] \quad (4)$$

using stochastic gradient descent, where the gradient is given by

$$\nabla L_i = \mathbf{E}_{(s,a,r,s'}\Big[\big(r+\gamma \max_{a'} Q(s',a',\theta_i) - Q(s,a,\theta_i)\big)\nabla_{\theta_i} Q(s,a,\theta_i)\Big] \quad (5)$$

There are three main causes of potential instability with this approach. The first being correlation between state-action sequences. The second is the fact that small changes in Q can significantly alter the policy and thus the data distribution. Third, the correlation between action values and the target action values. These issues are addressed by Mnih et al. [1] by using an experience replay and target Q-network. The experience replay allows for single transitions in an episode to be stored and then randomly sampled from. This random sampling of transitions breaks the correlation between successive steps in an environment and prevents destabilization with the changing data distribution. The target network is a separate Q-network, parameterized by θ^-, that is used to calculate the target action values; meaning the loss function described in 4, is now:

$$L_i = \mathbf{E}_{(s,a,r,s'}\Big[\big(r + \gamma \max_{a'} Q(s',a',\theta_i^-) - Q(s,a,\theta_i)\big)^2\Big] \quad (6)$$

This network is periodically updated such that the weights of the Q-network learning the policy are copied to the target network. This prevents correlation between the action and target values.

II. Implementation

The algorithm for training the DQN is as follows: Begin by initializing networks, policy network $Q(s, a, \theta)$, and target network $Q(s, a, \theta^-)$ with $\theta^- = \theta$; and replay buffer D with a specified max length. Then start the training loop for some specified number of iterations. Within this loop, we initialize a state s_t from the environment. Then find an action a_t using an $\epsilon - greedy$ action selection method. Then use this action to take a step in the environment, thus getting the next state s_{t+1} and reward r_t. Each of the environment steps are saved to the replay buffer D. After each step a batch size number amount of transitions (s_j, a_j, r_j, s_{j+1}) are randomly drawn from the replay buffer, these samples are used to find $y_j = r_j + \gamma * Q(s_{j+1}, a_j, \theta^-)$. Then this is used to perform the gradient descent step on $(y_j - Q(s_j, a_j, \theta))^2$. Every C steps the weights of the policy network are copied to the target network; and at every iteration ϵ used in the action selection is decayed linearly - starting from 1.0 representing completely random actions, to 0.0 representing policy only actions.

The policy and target neural networks use the same architecture consisting of two hidden layers with *tanh* activation functions. The specifications of this architecture are shown in table I. The input size is that of the state-space dimensions, and the output size is that of the action-space dimensions of the environment; therefore the ouput of the network is the function approximation of Q, so we can input the states and get the "optimal" action is described by the policy by taking the argmax over the network output. The hyperparemeters

TABLE I
BASIC NEURAL NETWORK ARCHITECTURE USED

Layer	Type	Input Dimension	Output Dimension
1	Linear	state-space size	64
2	TanH	-	-
3	Linear	-64	64
4	TanH	-	-
5	Linear	64	action-space size

used during training are shown in table II. Training over 100 iterations using batch sizes of 128 represents a total of $12,800$ steps in the environment used to train the network. The mean squared mean squared error loss function is used because it is representaitve of the loss function described in 6. The Adam optimizer is chosen, as it is a common optimizer used for gradient descent methods, and demonstrated a better performance than the RMSProp optimizer used by Mnih et al. The replay buffer size is chosen such that a sufficient amount of samples from previous episodes, acquired with previous versions of the policy, are available for learning. The value of ϵ used for the $\epsilon - greedy$ action selection is decayed linearly from 1.0, representing randomly sampled actions, until it reaches 0.0 at the 95^{th} iteration, so that the last several iterations are trained only on the policy-chosen actions.

TABLE II
TRAINING HYPERPARAMETERS SUMMARY

Hyperparameter	Value
Batch Size	128
Iterations	100
Target Update Frequency	10
Replay Buffer Size	10,000
Optimizer	Adam
Learn Rate	0.001
Loss Function	Mean Squared Error
Discount Factor	0.95
Epsilon Decay	Linear

III. RESULTS

Figure 1 shows the training learning curve of the algorithm described in the previous section, this curve is averaged over five different training iterations; the shadow area shows this average $\pm$ the standard deviation of these five trainings. You can see that there is a steady increase over the training loop, eventually reaching near 100 reward, representing the pendulu, being held in the upright position for nearly the entire episode. From these training iterations, the policy that gave the highest return in the final half of training was selected for further visualizations. A sample trajectory of this policy is shown in Figure 2, it can be clearly seen that after the first few steps the pendulum stabilizes in the upright position, with only slight oscillations in the angle and angular velocity. Figures 3 and 4 show the learned policy and learned state value functions for the state space (within the bounds of $\dot{\theta}$ and θ given in the environment), respectively.

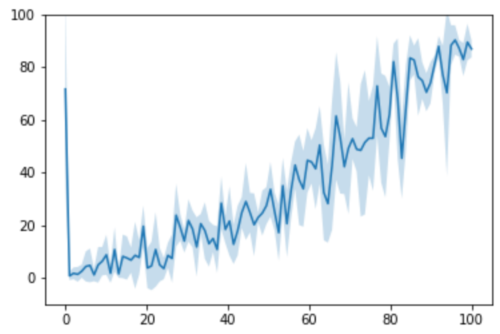

Fig. 1. Learning curve, plotting the return for iteration of the DQN during training

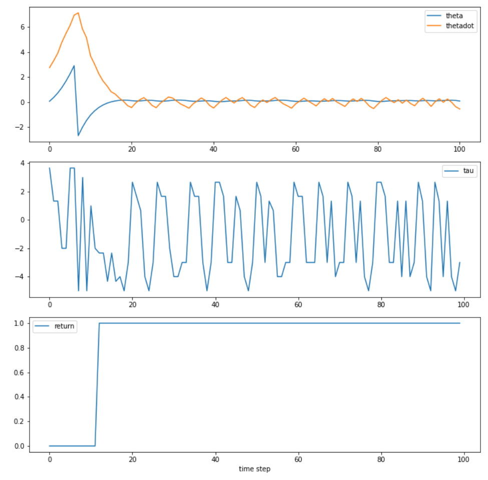

Fig. 2. Example trajectory of a learned policy

Next we perform an ablation study, where we seek to examine the effects of utilizing the experience replay and target network. We have the DQN discussed previously that uses both replay and target network. Then we have a DQN that uses a replay but no target, functionally we implement this by updating the target network weights at every step rather than every C steps, so that the two networks always have the same weights. Then we have a DQN that has a target network but no replay, we do this by setting the replay buffer maximum size to be that of batch size, meaning the DQN is only being trained on transitions from the current policy. Then we have a DQN trained with neither the target network nor the experience replay. Figure 5 shows the results of this ablation study.

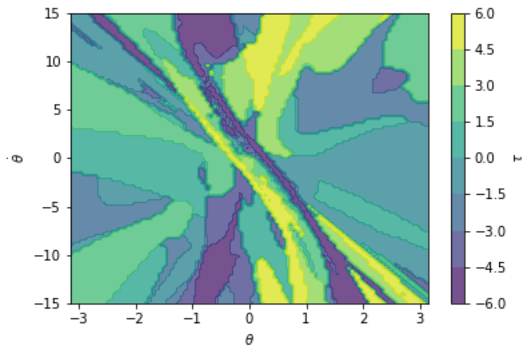

Fig. 3. Example plot of the learned policy

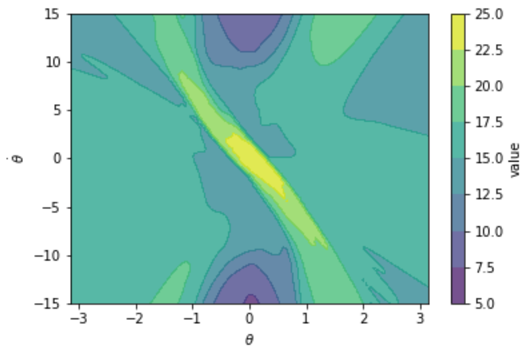

Fig. 4. Example plot of the learned state values

We can see that the DQN's trained without the experience replay perform much worse. Without the experience replay, the network only learns from transitions created from the current policy. This tells us that changes in the data distribution has a large effect on the efficiency of learning. It is slightly more difficult to draw a conclusion on the correlation between environment steps, given that in this implementation that batch is still randomly sampled, albeit from the same episode. This lowered performance, and the large negative spikes seen in the learning curve, could also be indicative of a phenomenon in reinforcement learning sometimes referred to as catastrophic forgetting; where, as the policy begins to learn more optimal actions to take, without the experience buffer having previous less optimal transitions, the policy may lose the capability to recover from these sub-optimal state-action pairs in the future, thus hindering the learning progress. We can also see that the network without the target network does just as well as the network trained on both the replay and target network. This tells us that, at least in this environment, the correlation between the action and target values in the loss minimization actually has little effect on the learning.

IV. Conclusions

We were able to train a DQN to hold a pendulum in an upright position within some specified angle range using around

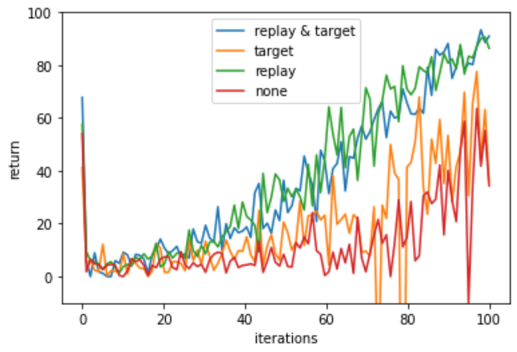

Fig. 5. Ablation study: learning curves for DQN networks trained with and without the presence of target network and experience replay

$12,800$ environment steps, as well as visualize the trained policy and value function. We also performed an ablation study on the effects of using experience replay and target network during train and found that while the presence of the experience replay greatly effects the learning, the presence of the target network is less important in this environment.

References

[1] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... Hassabis, D. "Human-level control through deep reinforcement learning". Nature, vol. 518 (7540), pp. 529-533. 2015.