Reinforcement Learning Homework 2

David M. Stanley (DavidMS4)*
University of Illinois, Urbana-Champaign, IL 61801

I. Nomenclature

ϵ	=	Exploration rate $\in [0, 1]$
ϵ_0	=	Initial exploration rate
ϵ_f	=	Final exploration rate
α	=	Neural net optimizer learning rate
γ	=	Successor state discount
π	=	Policy
$V^\pi(s)$	=	Value function under a given policy
s	=	State
s'	=	Successor state
a	=	Action
r	=	Reward
$Q^\pi(s, a)$	=	State action pair value under a given policy
$\mathcal{A}$	=	Action space for each state s
$\mathcal{S}$	=	State space
C	=	Steps between target network resets

II. Introduction

A. Discrete Action Pendulum

The discrete action pendulum model is a dynamic model of a pendulum with a length of 1 m, a mass of 1 kg, a coefficient of viscous drag of 0.1 with an acceleration of gravity of 9.8. The details of this model may not be accessed by the DQN algorithm. DQN must learn the continuous state-action value function by only observing the reward when transitioning between states. For each action an integrator is used to predict what the next state for the pendulum will be. Because DQN can accept a continuous state for the input the angle and speed of the pendulum have legal ranges of [$-\pi$, π] rads and [-15, 15] rads/s respectively. The default action space includes 31 actions. These actions correspond to discrete torque values between [-5, 5] Nm. If the pendulum is near the unstable equilibrium point, then the reward is 1 regardless of the pendulum's speed. If the magnitude of the pendulum's angular rate exceeds the limits of the legal range, then a large negative reward of -100 is given.

1. Deep Q Network

TD Q Learning was used in the last homework to learn the state-action value function in two discrete environments: Gridworld and the discrete pendulum environments. While the gridworld value function was easily estimated, Q Learning struggled to quickly learn the value function in the pendulum environment and only by using a limited descretized state. As the state and action spaces increase, the computationally and storage load on Q Learning increases dramatically, making TD Q Learning impractical for many environments and unusable for environments with state spaces that cannot be easily descretized. Deep Q Networks as described by Minh et al [1] overcomes these limitations by using neural networks to learn the Q Values. Since neural network are universal function estimators they are well suited to estimating multi-variable functions. The neural network is setup to learn state-action values for each of the possible actions within any of the continuous state. Taking the maximum of theses values in any state provides the policy. To train the network a bellman update equation is used to estimate the new Q values using the observed reward

*PhD Student, Department of Aerospace Engineering

and the Q value for the successive state and action according to:

$$Q^{\pi}(s, a) = r + \gamma Q'^{\pi}(s', a') \tag{1}$$

This value will differ from the network's estimate. This difference is used to calculate a loss which is then back propagated through the network updating the weights to minimize this loss and more accurately estimate Q. However, directly learning the Q function with a neural network can be unstable as the target weights are changing as they are being used to estimate the new Q values. This can cause the neural network's weights to rapidly fluctuate. To limit this, DQN uses a target network that has an identical structure to the main neural network but is only used to estimate Q' for the successive state and action. After a certain amount of steps, the target network's weights are set to be the same as the main network to ensure Q' is accurately updated over time. Alternatively, a soft update may be used with a bellman like equation to set the target network's weights towards (but not equal to) the main network's weights at every step. Unlike Q Learning, the weights for DQN are not updated with the reward and successive state at the time the transition is observed. Instead the transition tuple defined as (state, action, successive-state, reward) is stored in a memory cache which is sampled randomly in batches to train the neural network. This has the advantage of removing causality in successive actions and also allows rarely observed transitions to be used to train the neural network multiple times.

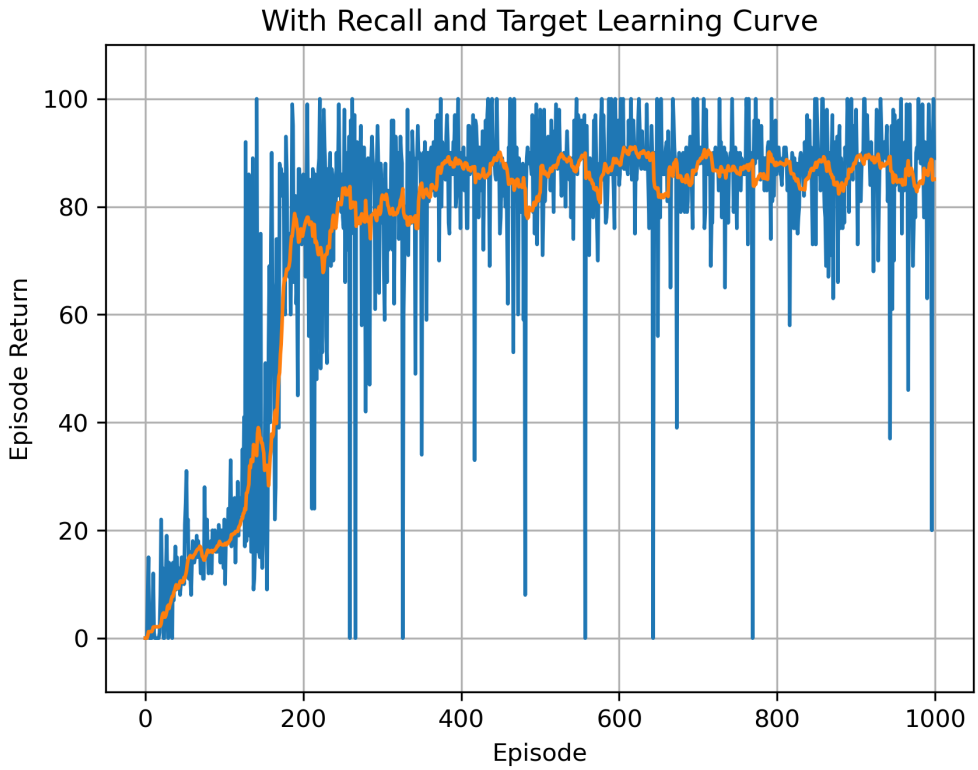

Fig. 1 Example of a typical learning curve

III. Results

The following hyper parameters were chosen for training the networks.

ϵ_0	=	0.90
ϵ_f	=	0.05
Exploration decay rate	=	1000 (larger is slower)
γ	=	0.95
Memory batch size	=	128
α	=	1e-4
C	=	200 (1 for no target network scenario)
Memory buffer size	=	10000 (128 for no recall scenario)

Compared to Q Learning and SARSA, DQN was able to more quickly find an optimal policy for balancing the pendulum. Within 500 episodes, the DQN network can usually be seen to average more than 80 seconds inverted per episode. In figure 1 the blue line shows the return for each episode and the orange line is the moving average over the last 20 episodes. There are some episodes with much lower performance even after 1000 episodes, indicating that agent is not fully trained. Some, but not all, of the lower values can be attributed to poor initial states. For example, a resting uninverted pendulum would require the agent to spend many steps inverting the pendulum before collecting any reward. Figures 2 & 3 show the learned policy and state values for the above trained agent. Because DQN takes a continuous input the policy and value functions can be render much finer than for TD SARSA and Q Learning. Lastly, for this trained agent, a trajectory plot is shown in figure 4. The agent takes a very direct path towards the reward inversion state within 20 seconds and remains there for the remainder of the episode. The agent does not violate the angular rate restriction. To better understand how the individual components of the DQN effect its performance an ablation study

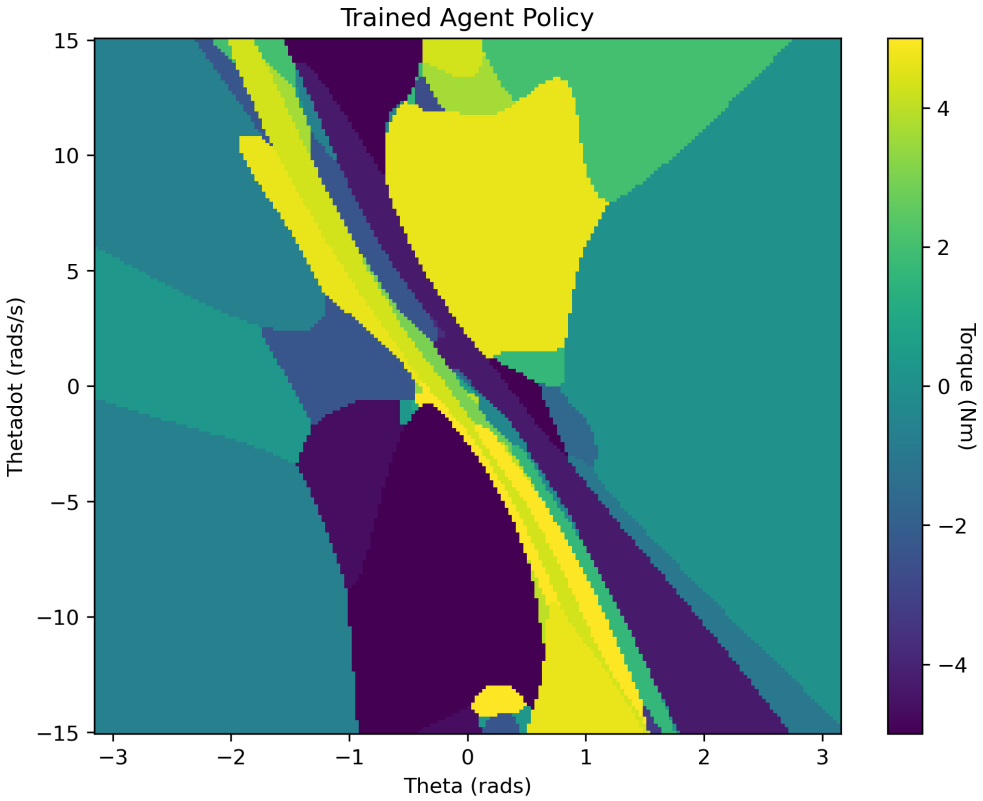

Fig. 2 Policy for trained agent

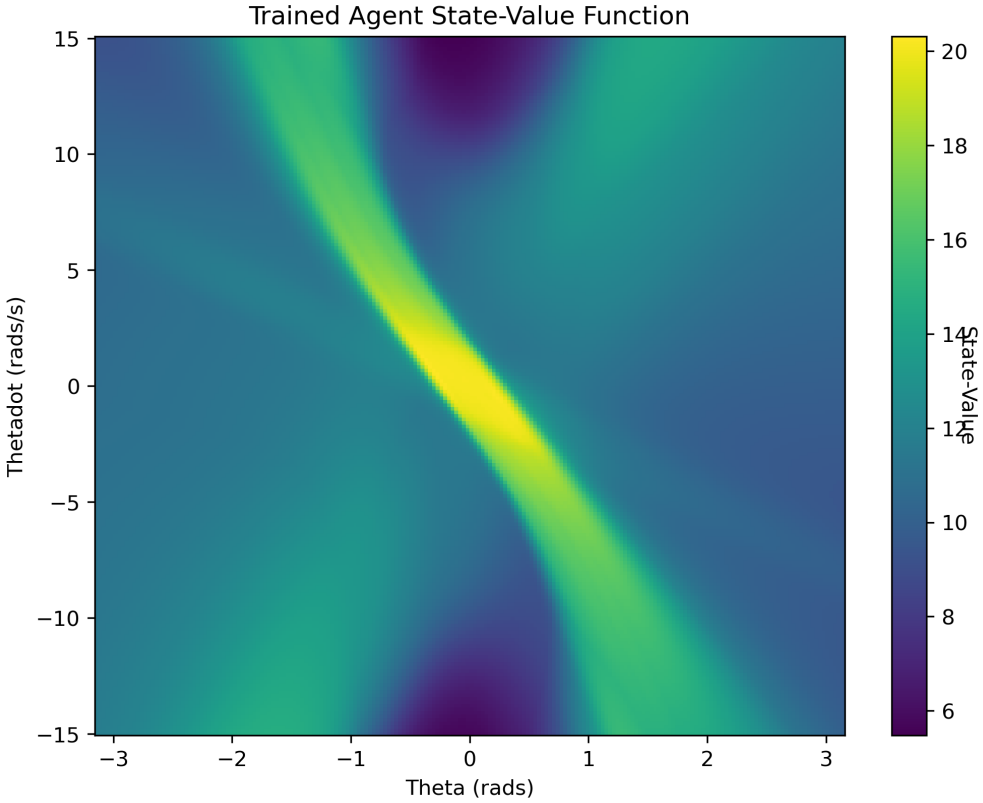

Fig. 3 State value function for trained agent

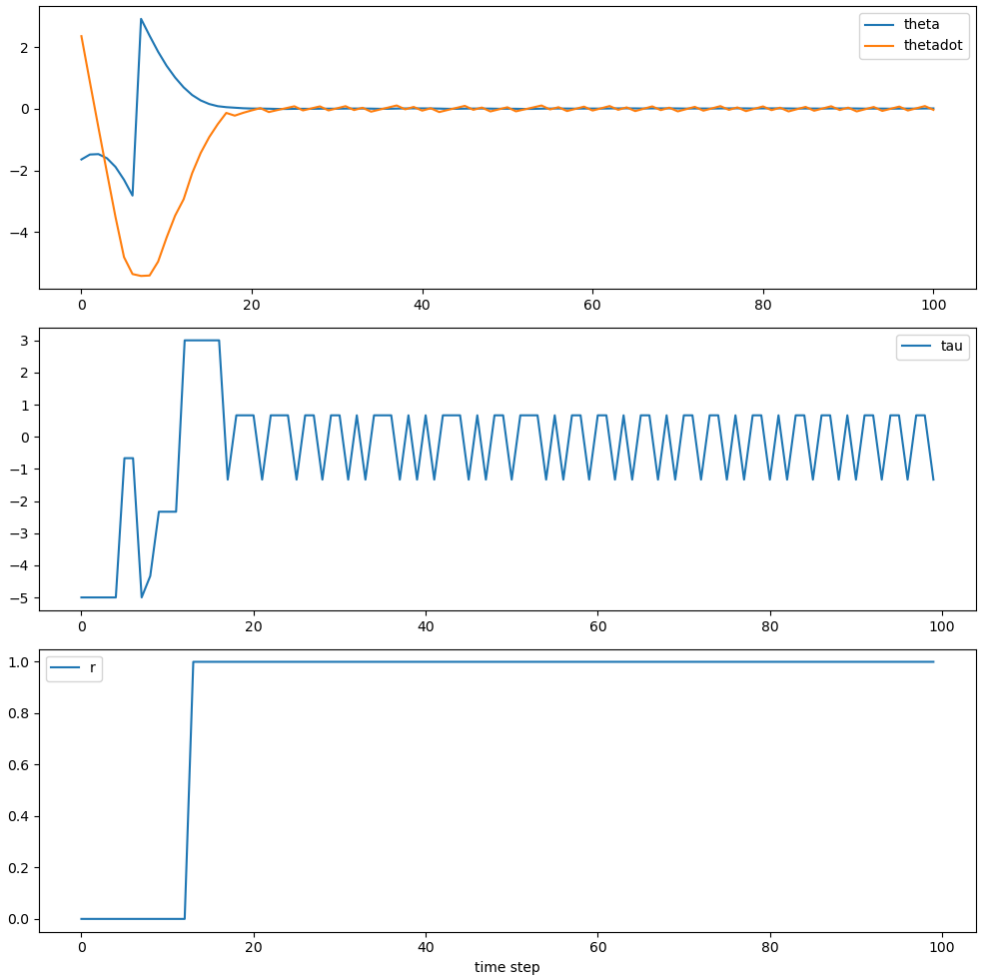

Fig. 4 Trained agent trajectory

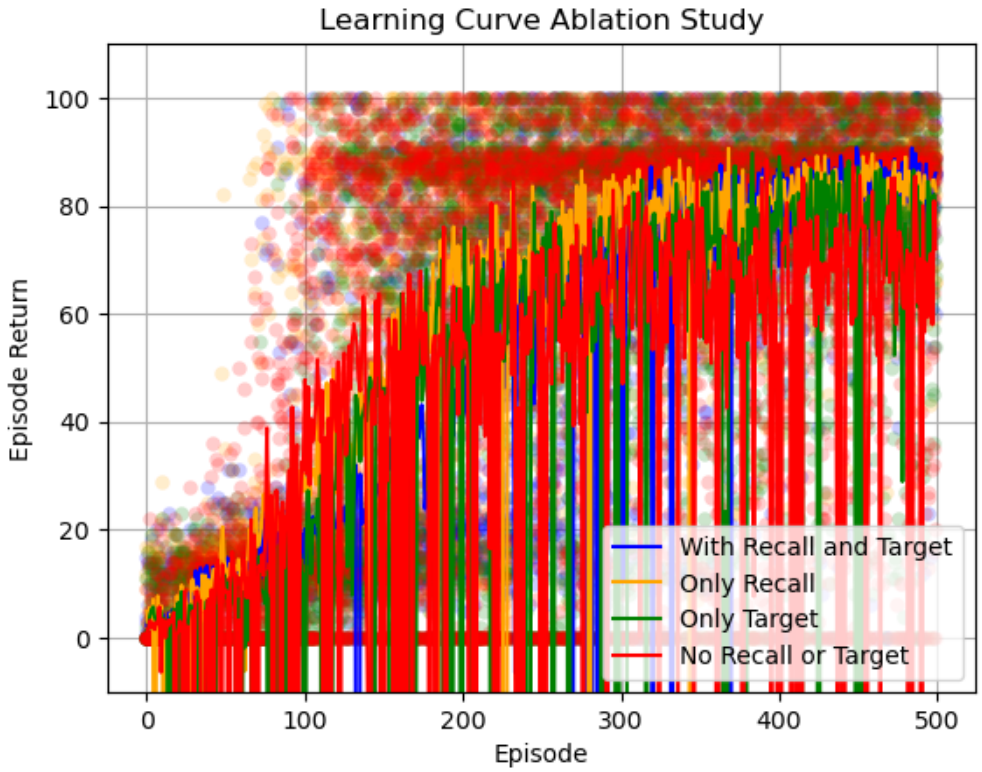

Fig. 5 Trained agent trajectory

was performed. The full DQN makes used of memory cache for recalling previous transitions and a target network to improve learning stability. These were removed from the model in turn and the networks were trained for 500 episodes over 10 runs per scenario and the learning curves were averaged over the runs. In total there are 4 scenarios.

- With recall and target network
- With recall but no target network
- No recall but with target network
- No recall and no target network

The results are show in figure 5. It can be seen that the full model in blue edges out the other models in both total average return over time and consistency of the return. The model with recall but without a target network in orange has slightly lower final average returns but appears to start learning faster than the full model. This makes sense because the main network is allowed to directly learn from the loss, which is faster in the short term but unstable in the long term. The network without recall but with a target network in green appears to be about as stable as the full model, but is learning at a slower rate. THis shows how randomly sampling the transitions for learning improves the learning rate. Finally the model without recall and without a target network is shown in red. This model performs the worst averaging a return of around 70 per episode and with a large amount of variance in the return but rarely reaching the return of the full model. This clearly shows how the target network and recall improce the learning process.

IV. Conclusion

Deep Q Networks are a powerful tool for learning policies in continuous state spaces and overcome some of the shortcomings of tabular methods. However they also introduce their own difficulties that must be overcome with techniques like random recall and using target networks.

References

[1] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. DOI: 10.1038/nature14236. URL: https://doi.org/10.1038/nature14236.