Improving Q-Learning with Neural Networks

Henry Feinstein '19 and Eric Wang '18

December 18, 2017

Abstract

There have recently been major advancements in applying deep learning to the traditional reinforcement learning technique Q-learning. The progress we've seen is largely due the deep neural network's strength in developing compact representations of high dimensional input. While Q-learning has success in some domains, the challenge of assigning an action policy for each state grows rapidly as the state space size for the problem expands, rendering it unable to effectively solve many real-world problems. While this problem can be solved by hand-picking features, that typically requires a high amount of domain knowledge of the problem that the learning agent designer may not have. Neural networks can automatically produce condensed input representations while achieving the learning task. In this paper, we closely examine the Deep Q-learning Network's (DQN) efficacy in dealing with large state spaces by comparing its performance with that of a traditional implementation of Q-learning on a robot crawling task. We incrementally increase the complexity of the task by artificially expanding the size of the state space, successfully showing that standard Q-learning fails as the task complexifies while the DQN displays no change in performance.

1 Introduction

The task of reinforcement learning revolves around training a responsive agent to behave optimally in an environment with only occasional feedback. The field has recently been revolutionized by deep learning - the use of neural networks in learning agents - an approach that hugely widens the possibilities of reinforcement learning application. For example, a team of researchers recently demonstrated deep Q-learning's ability to achieve human-expert level performance on a range of Atari 2600 video games with only pixel values as input [2]. Deep Q-learning is an improvement on previous reinforcement learning techniques through the introduction of deep neural networks. In this paper we seek to demonstrate those improvements by comparing a DQN with its traditional counterpart on an increasingly difficult task.

1.1 Q-Learning

Q-learning is a value function technique, meaning that it solves a problem by estimating the value or expected return for each possible action given a particular state [3]. Assuming the success in learning those values, an agent may greedily take an action of high value to maximize reward.

In the case of Q-learning, the state-action values are approximated iteratively and stored in a table. As an agent wanders around its world, it observes the state transitions caused by its actions and updates its values using the following equation:

$$Q(s,a) += \alpha[R(s) + \gamma[\max_{a'} Q(s',a')] - Q(s,a)] \tag{1}$$

The $R(s)$ term represents the intrinsic reward of the resulting state s'. We evaluate the max Q-value of s' to represent expected future return, and discount it by multiplying by discount γ, a tuneable parameter between 0 and 1. We subtract the current Q-value because this is an iterative update which is dampened by the learning rate α to preserve previous learning. We call values approximated by this technique Q-values [4].

1.2 Feature Selection and Deep Q-Learning

While the Q-learning technique is effective on some problems and proven to converge upon a successful policy over time [1], the time it takes to converge grows rapidly as the state space grows larger [4].

We solve this issue by extracting features from the state space for complex tasks, creating a condensed representation of the full state space. The learner instead learns using this feature space.

Rather than storing the Q-values in a tabular form, the DQN approach seeks to approximate the Q-function using a neural network. As before, Q-value updates are calculated with the Q-value update function, but are instead stored in a memory queue at each step. After each episode, we draw a random training set from the memory to train the neural network. The neural network has one output per action, trained to output the estimated Q-value. Observations are drawn randomly from the memory to avoid catastrophic forgetting. The Q-value's recursive nature may cause oscillations in the training progress; this is alleviated by holding an old version of the neural network constant, which is used for update calculations. The old neural network is only updated occasionally [2].

Because deep neural networks effectively perform feature selection in their early layers, a DQN essentially learns features to solve the problems presented by large state spaces. This is convenient because it does not require the experimenter to hand pick features, a highly labor-intensive task that may not even be possible for very complex problems.

1.3 Hypothesis

Considering the well-documented ineffectiveness of standard Q-learning on problems with large search spaces and neural networks' feature selection capacities, we propose the following hypothesis:

As the sensorimotor space grows, the performance of the standard Q-learning approach will degrade, while deep Q-learning will remain successful.

2 Experiment

2.1 Learning Task & Environment

To test our hypothesis, we needed to create a learning task using an agent and environment for which the sensorimotor space can be expanded in a controlled manner without rendering the task impossible to learn. We settled on an experiment in which a simple simulated robot attempts to learn to crawl. The environment is a simulated plane that is infinite and flat. It tracks the position of the robot, as well as its velocity. The robot has a single two jointed arm actuator and no sensors other than knowledge of its state, which consists of an "arm" position and "hand" position, corresponding to the two portions of the actuator (Figure 1). The state space can be gradually expanded by discretizing the actuator's movement space to a greater and greater extent; this increases the possible number of arm and hand positions the robot can occupy without actually extending the possibilities of movement. At each step it can take one of 4 possible actions: "arm-up," "arm-down," "hand-up," and "hand-down." At each step the robot is rewarded for its velocity: the distance traveled in that step.

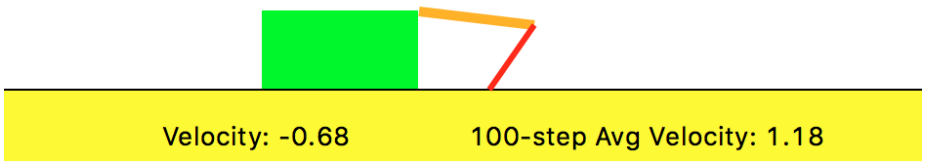

Figure 1: Environment and Robot. The environment is a flat, infinite plane (in yellow); the robot's actuator is split into a longer "arm" portion (in orange) and a shorter "hand" portion (in red).

2.2 Learning Agents

2.2.1 Standard Learning Agent

Our standard learning agent follows the traditional form for a Q-learning algorithm; the agent constructs a table the exact dimensions of the state space for a given trial. It then fills in that table with Q-values as it explores and moves through the world using the Q-value update function (Equation 1). Every value in the table is initialized to zero. The hyperparameters may be found in the Appendix.

To allow necessary policy exploration, the agent is initialized with an ϵ value of 0.5. At each step, the agent has probability ϵ of taking a random action instead of its policy. The ϵ value decays linearly starting at step 5000 and reaches zero by step 12000.

2.2.2 Deep Learning Agent

To determine the best action at each step, our deep learning agent uses a dense feed-forward neural network. Its inputs are the current state and its outputs are the Q-values for each of the four possible actions. The weights of the neural network are initialized to random values. The network topology is depicted in Figure 2 and the hyperparameters may be found in the Appendix. For fairness, the DQN agent uses the same epsilon decay as the Q-learner.

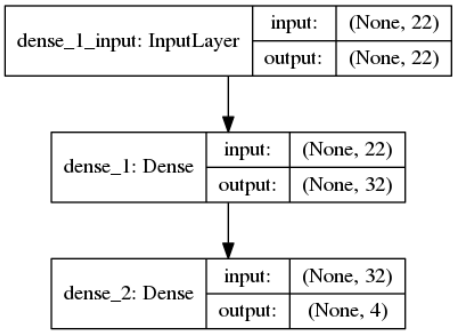

Figure 2: Our DQN neural network topology: one fully connected ReLU 32-unit hidden layer and a linear 4 unit output layer. The input dimension is variable because we one-hot encode arm and hand position.

2.3 Experimental Setup

To test our hypothesis, we executed each learning agent on a series of trials, increasing the number of possible hand positions by five at each round. Each trial was 15,000 steps long. Although both the number of arm and hand positions could be altered in the simulator, we opted to only experiment with hand position granularity for two reasons. Firstly, changing only the hand positions

proved to be enough of a state space expansion to show compelling results; secondly, if we had increased both arm and hand positions simultaneously, it would have expanded the state space in multiple dimensions. We believe that it is easier and more meaningful to analyze the results when the space was expanded in only one dimension. Our results present the mean of ten trials at each of 15 hand position levels, beginning at 15 possible hand positions and ending with 85.

3 Results

Our results show a significant difference in performance between the standard and deep Q-learning agents. Figures 3 and 4 chart the adjusted total distance traveled and the average final velocities at each number of hand positions, respectively. Adjusted total distance and final velocity were calculated using the following formulae:

$$ATD = \textit{total distance} * (\#\ \textit{hand positions}/15) \tag{2}$$

$$AFV = \textit{final velocity} * (\#\ \textit{hand positions}/15) \tag{3}$$

The reasoning behind this transformation of the data is that as the granularity of the possible hand positions increases, it will take more steps to achieve the same forward progress regardless of the quality of the crawling policy. For example, the reward gained from one hand movement when there are only 15 hand positions will require two movements at 30 hand positions. Plotting the raw total distance data gives the illusion that the deep Q-learning agent was severely impaired by state space expansion, when in reality it was learning an equally good policy and instead being slowed by the phenomenon just described. Thus, to represent the true "quality" of the policy learned by each agent, we transform the data using Equations 2 and 3.

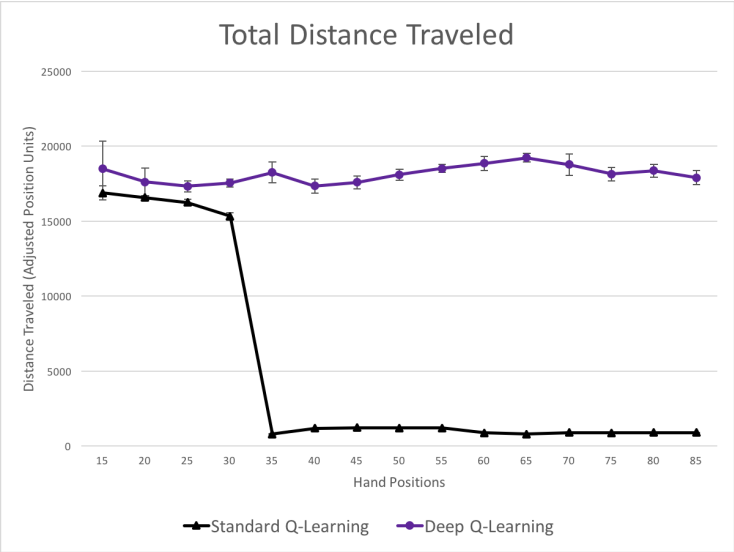

Figure 3: Adjusted Total Distance Traveled. Error bars represent the standard deviations for each agent at each number of hand positions (10 trials each).

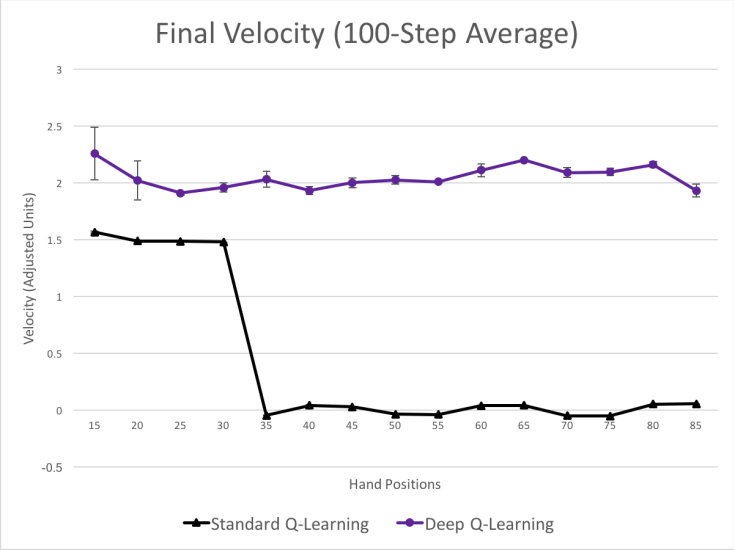

Figure 4: Adjusted Final Velocity. Velocities shown here are averaged over the final 100 steps of each run. Error bars represent the standard deviations for each agent at each number of hand positions (10 trials each).

Figure 3 shows a striking difference between the performances of our stan-

dard and deep Q-learning implementations. The deep learning agent performs consistently across the entire range of possible hand positions, showing no deterioration as the state space expands. The standard Q-learning agent, on the other hand, hits an insurmountable wall at 35 hand positions, and never recovers; from 35 all the way to 85 hand positions, it is completely unable to learn a forward-moving policy once randomness decays.

Surprisingly, the deep agent's performance was significantly better than the standard agent's at *every* number of hand steps, even 15 - as can be seen in Figure 4, the deep agent's final velocity was consistently greater than the standard agent's at high statistical significance. Additionally, the differences in total distance traveled are statistically significant at every hand position number. This implies that at all hand position numbers tested, the deep agent was able to learn a more effective crawling policy. Links to videos of 85 hand position trials for both agents can be found in the Appendix.

4 Discussion

In Figures 3 and 4, we can clearly see a stark drop-off in standard Q-learning's performance between 30 and 35 hand positions for the robot. This phenomenon, while initially surprising, makes sense because of the nature of our learning task. By formulating arm and hand angles as the robot's state and learning input, we have restricted the policy to only be able to assign one optimal action response to each body position regardless of action history. Thus we can imagine a policy to necessarily be some cycle in the state space, given the cyclical nature of a typical crawling pattern. We can also imagine "traps" in the state space, accidentally-learned mini-cycles (oscillating back and forth between two body positions, for example) in the space that the agent cannot escape without stochasticism. Indeed, from our qualitative observation, we do see the traditional Q-learning repeatedly move one actuator back and forth, producing no net progress. For larger state spaces, not only is it much more challenging to fill out a full cycle in the Q-table, but it is also much easier to fall into mini-cycles that cannot be escaped after 12,000 steps when exploration is completely removed.

Likewise, the performance of the DQN is also unsurprising. If we reason that the neural network learns to condense the input space, and see that the DQN successfully solves the problem with 15 hand positions, it should be achievable that the network simply learns to "divide" input data to return the problem back to the original 15 hand position task. Why the DQN learns a higher velocity than standard Q-learning even at 15 hand positions (2.258 units/step vs. 1.567 units/step) is a question that eludes us and deserves further investigation.

4.1 Neural Network Design

In the process of implementing our DQN, we experimented with hyperparameters through many failed iterations before settling on our final topology and

hyperparameter set. Network topology turned out to be a particularly sensitive parameter. When initially using a topology with two 12-node hidden layers, we struggled to make the agent learn at all.

We suspect that the necessity of a small topology is due to the high frequency of network training and small batch size. Each time we train the network we are susceptible to some catastrophic forgetting, constraining the potential for learning. Learning may not be able to extend very far beyond the information gleaned from training on 1 minibatch, which we only train for 1 epoch. Thus, a small network, which learns fast, may be able to capture more from the epoch than a larger network.

Our decision to train so frequently emerged from our biased experience in designing the standard Q-learner. Its learning speed biased us to believe the DQN learner should also be able to finish learning within a few hundred steps and thus necessitated frequent training. In general, expecting a neural network to converge so quickly is an unrealistically high expectation. We suspect that if we extended minibatch size and trained less frequently, a network with a larger topology could also succeed, albeit over more steps.

5 Conclusion

Our results strongly confirm our initial hypothesis: our DQN implementation performs consistently regardless of state space size, while our standard agent fails completely when the state space grows too much, and is unable to recover throughout the remainder of the experiment. Our results are convincing, if not surprising; it is well-established in the field that the standard approach to Q-learning is highly susceptible to large state spaces. Our experiment does give some insight, however, into exactly what "large" means - the steep dropoff of our standard agent's learning capability at a mere 35 hand positions implies that a state space need not be extremely complex to thwart the standard approach. Considering how few real-world problems are composed of state spaces as simple as the 35 hand position trial, our paper clarifies and reinforces the power and necessity of using neural networks in the field of reinforcement learning.

Acknowledgements

We would like to thank Professor Lisa Meeden for her guidance and support over the course of this project. We would also like to thank Professor Bryce Wiedenbeck and Samuel Sakota for their consultation in designing and implementing our DQN agent.

References

[1] Peter Dayan Christopher J.C.H. Watkins. Technical note: Q-learning. *Machine Learning*, 8, 1992.

[2] Volodymyr Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 518, 2015.

[3] J. Andrew Bagnell Jens Kober and Jan Peters. Reinforcement learning in robotics: A survey. *International Journal of Robotics Research*, 32(11):1243–1244, 2013.

[4] Peter Norvig Stuart Russell. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010.

6 Appendix

6.1 Learning Agent Hyperparameters

Table 1: Q-learning Hyperparameters

Hyperparameter	Value
discount	0.8
learning rate	0.8
initial exploration	0.5
final exploration	0
exploration time frame	5000-12000

Table 2: DQN Hyperparameters

Hyperparameter	Value
minibatch size	128
replay memory size	2000
target network update frequency	64
discount	0.95
update frequency	64
learning rate	0.01
initial exploration	0.5
final exploration	0
exploration time frame	5000-12000

6.2 Trial Videos

Standard Q-Learner Video:
`https://youtu.be/kEa41v9qR8o`

DQN Video:
`https://youtu.be/UHXBdPaXlOg`

6.3 Consulted Source Code

Keon's DQN implementation:
`https://github.com/keon/deep-q-learning`

n1try's DQN CartPole implementation:
`https://gym.openai.com/evaluations/eval_EIcM1ZBnQW2LBaFN6FY65g/`

Source code for environment and robot simulator taken from CS63 (Artificial Intelligence), taught by Prof. Bryce Widenbeck in Spring 2016.