Deep Q-Learning To Solve Complex Grid Environments

Ratik Dubey
MS AI, University at Buffalo
ratikdub@buffalo.edu

Denver Dsouza
MS CS, University at Buffalo
denverel@buffalo.edu

Tyler Banks
MS AI, University at Buffalo
tylerban@buffalo.edu

Abstract

We present a deep learning model to successfully learn how to solve complex grid environments. The model is a convolutional neural network trained using a variant of Q-learning, whose output is a value function that estimates future rewards. We apply our method to different OpenAI gym environments, with no adjustment to the learning algorithm or architecture. We implement a DQN with prioritized experience replay and benchmark its performance against a DQN with uniform replay.

Introduction

The goal of reinforcement learning [1] is to learn good policies for sequential decision problems, by optimizing a cumulative future reward signal. Q-learning [2] is one of the most popular reinforcement learning algorithms, but it is known to sometimes learn un- realistically high action values because it includes a maximization step over estimated action values, which tends to prefer overestimated to underestimated values. In this paper we investigate the performance of the recent Deep Q Network algorithm [3]. DQN with Q-learning with a flexible deep neural network was evaluated on a diverse and wide number of deterministic Atari 2600 games, with human-level performance on several of them.

There have been many improvements developed to scale reinforcement learning to solace sequential complex decision making problems. We will implement one such improved algorithm called Prioritized Experience Replay. Experience replay lets online reinforcement learning agents remember and reuse experiences from the past. Experience transitions were previously sampled uniformly from a replay memory. However, regardless of their relevance, this strategy just reruns transitions at the same frequency as they were originally experienced. We use prioritized experience replay in our Deep Q-Networks (DQN), to outperform a DQN with uniform replay and achieve new state-of-the-art performance.

This paper demonstrates a convolutional neural network that learns how to solve complex grid environments. We train the network using a variant of Q learning. To reduce the problems of correlated data we use an experience replay mechanism which randomly samples previous transitions and smoothens the training behavior.

Possible Outcomes

1. Implement a Deep Q Network (DQN) to solve complex grid environments
2. Use Prioritized Experience Replay and Double-DQN to modify our DQN and see how it affects our results
3. Test our DQN algorithms (DQN and Improved DQN) on complex environments from OpenAI's gym module (environments will include CartPole, LunarLander)
4. Discuss the evaluation results after applying the two algorithms implementation on the environment
5. Provide our interpretation of the results and show real-world scenarios that can possibly be solved using this methodology (like path planning for ships, airplanes etc.)

Timeline

Table 1: Timeline

Task	Due Date
Implement the Deep Q Network	3-31-22
Use Prioritized Experience Replay to modify our DQN	4-30-22
Test our DQN algorithms	5-15-22
Discuss the evaluation results after applying the two algorithms	5-15-22
Provide our interpretation of the results	5-15-22
Finalize Report	5-22-22

Environments

1. Grid environment

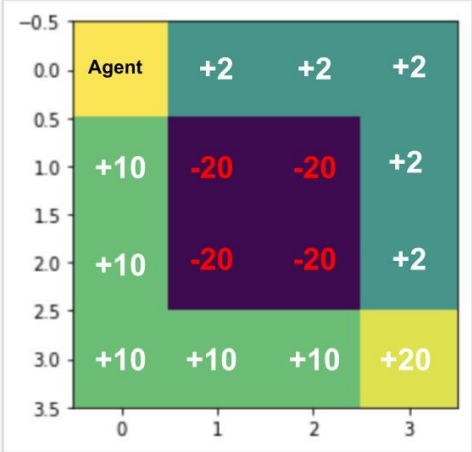

Figure 1: Grid Environment
Yellow is the location of the agent. All other coloured grids correspond to the possible rewards. Goal is the bottom right position.

- *Actions*: Up, Down, Up, Right, Left (Encoding is 0,1,2, 3)
- *States*: 4*4 grid representation of the 16 states
- *Rewards*: 0, 2, 10, -20, 20
- *Main objective*: Reach the goal (last row, last column) in the grid
  Start position is the top row, first column the grid

2. OpenAI - CartPole (Version 1)

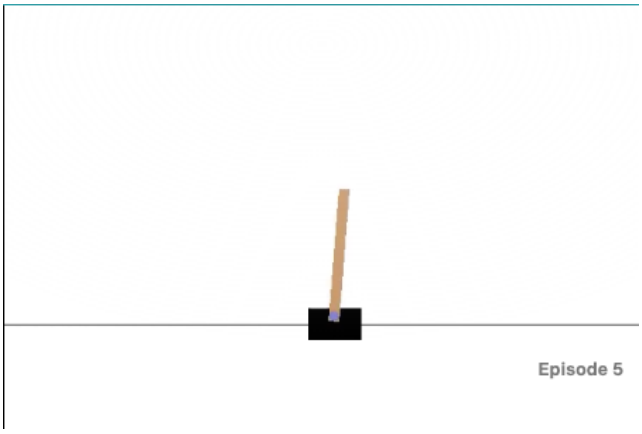

Figure 2: CartPolev1

- *Actions*: apply force to move the cart (+1 for left or -1 for right)
- *Reward*: +1 per timestep the pole is upright
- *Goal*: prevent the cart from falling
- Episode ends when the cart is 15 degrees or more from vertical and/or 2.4 degrees or more away from center

3. OpenAI - LunarLander (Version 2)

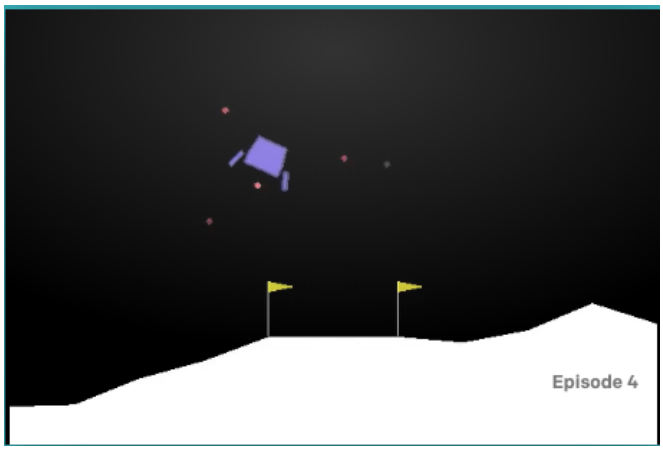

Figure 3: LunarLanderv2

- *Actions:* do nothing, fire left orientation engine, fire main engine, fire right orientation engine
- *Reward:* +10 for each leg of the lander with ground contact, 0.3 each frame for firing the main engine, +200 for solving the environment
- *Goal:* Moving from the top of the screen to landing pad with 0 speed
- Episode ends with -100 if the lander crashes with +100 if it comes to rest

Implementation

DQN is similar to Q-Learning, except that instead of keeping the random agent's learnt state-action values in a table, we use a neural network with inputs that represent states and outputs that represent actions. We still apply the bellman equation when appropriate, but we use the neural network instead of the Q-Table to get the target and predict Q-values.

We adopt DQN because neural networks have demonstrated to be effective at modeling complex functions. The function obtains the Q-values of all possible actions in a state and maps the state to these values. It learns the weights, which are the network's parameters, and then utilizes them to discover the best Q-values, which are the model's ultimate output. A DQN's approach is fundamentally similar to classic Q-learning in that both employ previous and present Q-values of an action to enhance the Q-value estimate. A DQN is more likely to produce the best state-action value pairings.

Traditional RL agents incrementally change their parameters (of the policy) as they proceed through the episodes, then discard the data after processing and updating the parameters. RL agents can re-enact previous trials using experience replay. In DQN, a sliding window (known as buffer size) is utilized to sample replay memory at random. This aids in the stabilization of value function training and minimizes the number of episodes required to learn. The optimal buffer size is determined by tweaking our network (typically between 1000 and 10000). Using the correct buffer size allows the network to converge more quickly. When the buffer size is too little, the data becomes strongly correlated, and when it is too large, the data becomes outdated. To make a compromise between the quality of the data and its correlation, a balance must be found.

Experience replay helps us tackle two critical challenges in RL: it helps break temporal correlations by updating parameters with less recent experience. It also remembers unusual events that may be relevant in the future, and this information is utilized to update the settings many times. Although experience replay requires more computation and memory, it is often less expensive than the agent's interactions with the environment.

A Deep Neural Network version of Double Q-Learning is the Double Deep Q Network (Double DQN). Double DQN employs two identical neural network models - Q and Q'. One learns via a repetition of the event, similar to DQN, and the other is a clone of the first model's final episode. The action from the second model is calculated using the index of the main model's highest Q-value.

The fundamental Q-learning method is notorious for inflating some action values in specific instances. In classical Q-learning, the max operator, which we employed in our DQN implementation, picks and evaluates actions based on the same values. As a result, overly optimistic value judgments are more likely to be picked. We use Double Q-learning to prevent this problem; the key principle underlying Double Q-learning is to isolate the selection from the evaluation. Using these independent estimators, we can obtain unbiased Q-value estimates for the actions undertaken. We can avoid maximizing bias by separating our updates from biased estimations - this is how the Q-value is calculated in Double DQN.

Results

1. Grid Environment

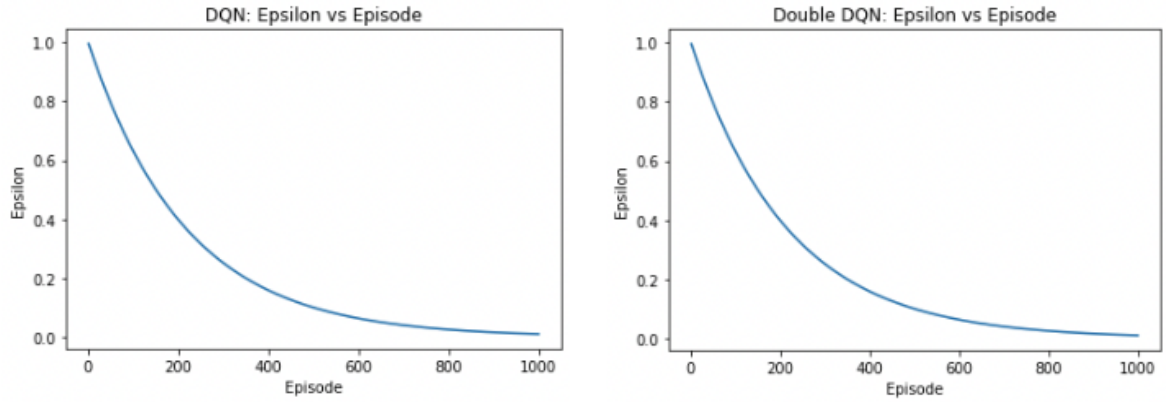

Figure 4: Epsilon Decay for DQN and Double-DQN on Grid Env

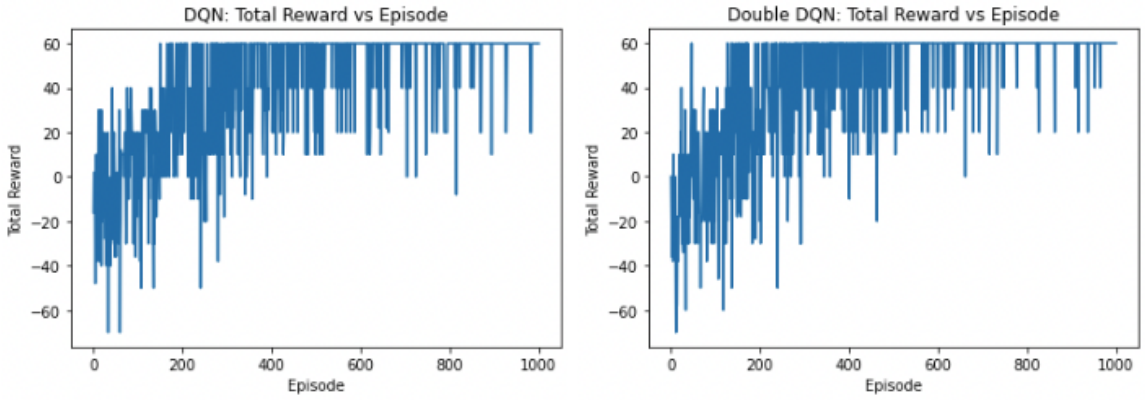

Figure 5: Total Reward per Episode for DQN and Double-DQN on Grid Env

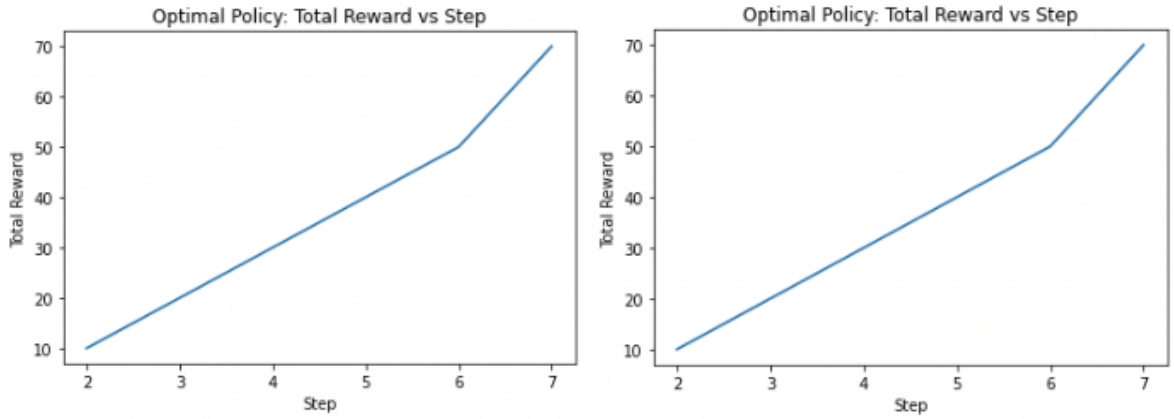

Figure 6: Optimal Policy Reward for DQN and Double-DQN on Grid Env

2. OpenAI CartPole

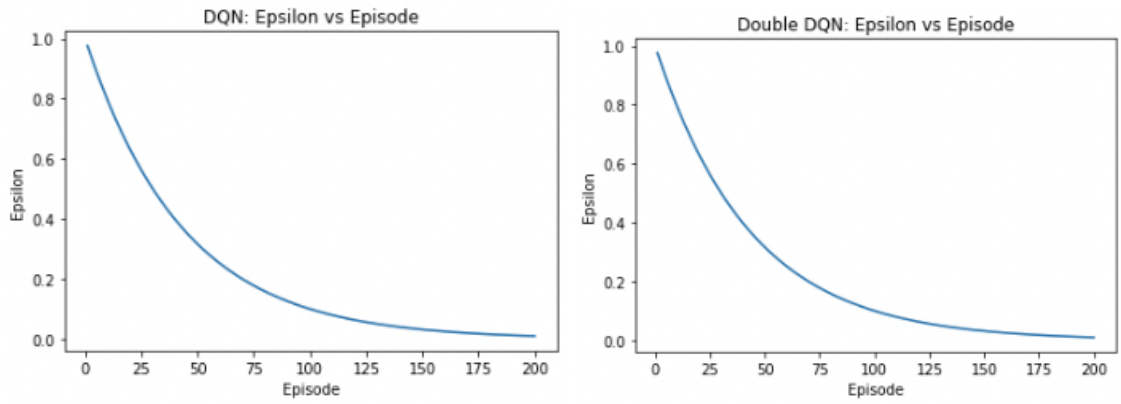

Figure 7: Epsilon Decay for DQN and Double-DQN on CartPolev1

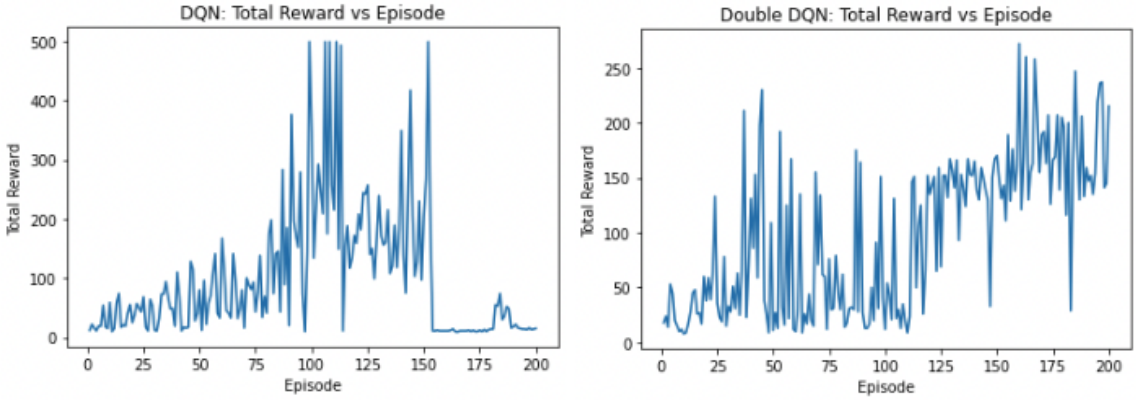

Figure 8: Total Reward per Episode for DQN and Double-DQN on CartPolev1

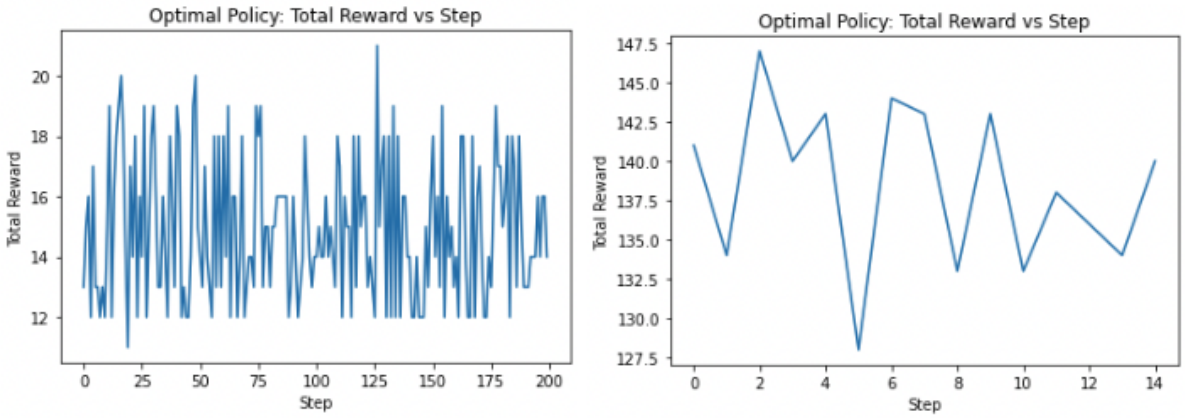

Figure 9: Optimal Policy Reward for DQN and Double-DQN on CartPolev1

3. OpenAI LunarLander

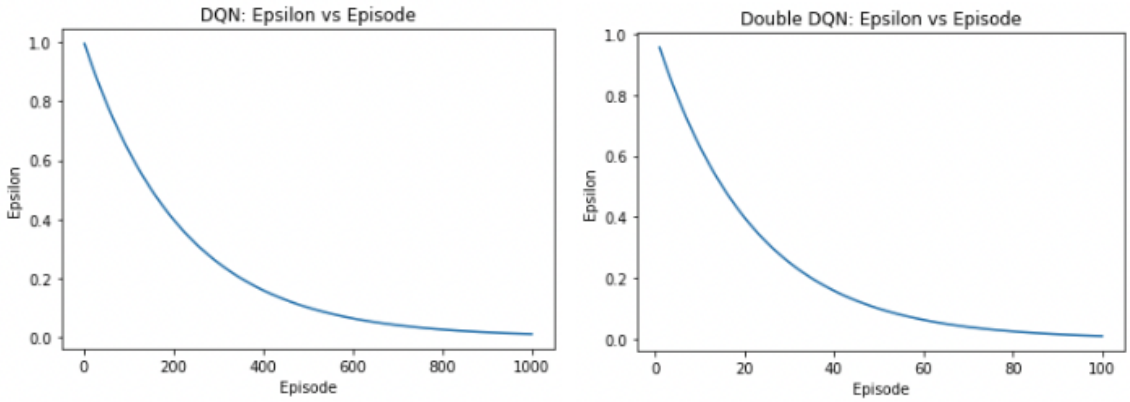

Figure 10: Epsilon Decay for DQN and Double-DQN on LunarLanderv2

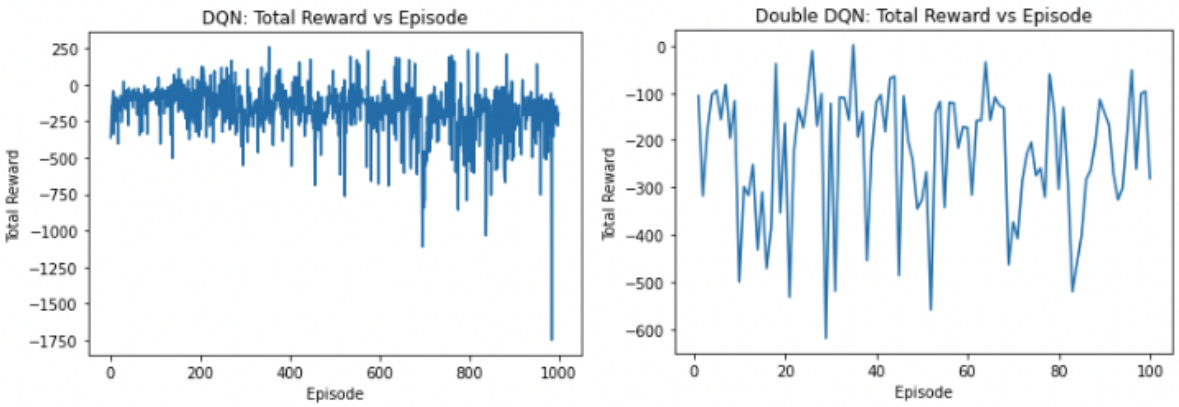

Figure 11: Total Reward per Episode for DQN and Double-DQN on LunarLanderv2

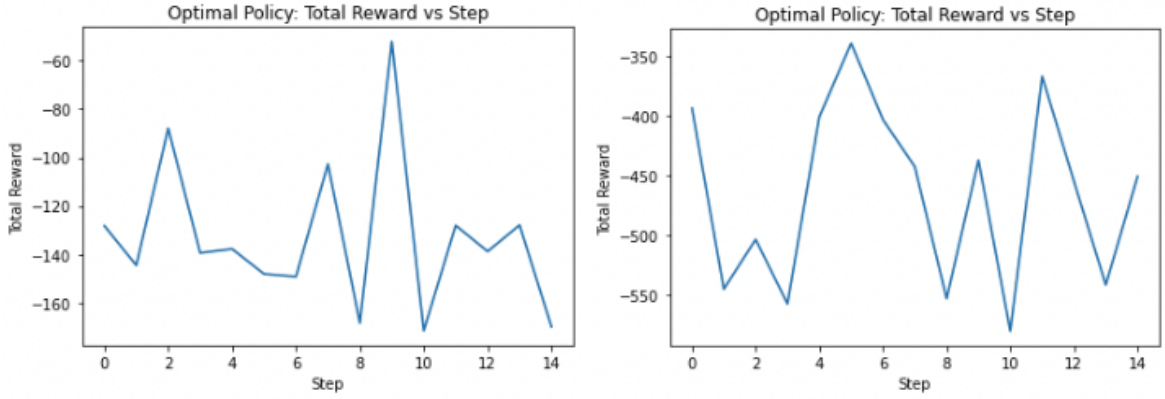

Figure 12: Optimal Policy Reward for DQN and Double-DQN on LunarLanderv2

Conclusion

- The plots show that Double-DQN converges quicker than DQN, as expected
- In addition, unlike a standard DQN, Double-DQN does not overstate action values in some circumstances
- Average score:
    - CartPole
        - DQN = 14.89
        - Double-DQN = 138.46
    - LunarLander
        - DQN = -132.97
        - Double-DQN = -464.47
- During the training phase, we discovered that employing Double-DQN for the same environment was more computationally expensive than using DQN
- DQN and Double-DQN produced similar results for CartPole and Grid Environment
- When we used Double-DQN instead of DQN for LunarLander, we saw considerably better convergence and policy - this was reinforced when we changed the hyperparameters and even tested a different environment, in both cases the results from Double-DQN were far better than DQN.

Code

Github:

References

[1] Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learn- ing: An Introduction*. The MIT press, Cambridge MA.

[2] C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge England, 1989.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostro- vski, 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*, 518 (7540):529–533, 2015.