Project 1 Writeup

Lukas Dannull

February 2021

1

To explain my project I first have to explain Reinforcement Learning. In Reinforcement Learning your goal is to train an agent which will map states to actions that maximize reward. A state, denoted by s, is some representation of the environment (the game being played), an action, denoted by a, is a choice the agent makes that alters the environment somehow, and a reward, denoted by r, is a metric that indicates to the agent how well it's doing. My agent of choice was the Deep Q-Network (DQN), which aims to learn a network $Q(s, a)$ that learns to predict the expected discounted return of a state and action. In simpler terms, $Q(s, a)$ is estimating the total reward if you take action a in state s and play perfectly after that.

At a given time t, the DQN observes a state s_t, takes an action a_t observes the resultant state and reward s_{t+1} and r_{t+1} as well as an indication if the game has completed, d_{t+1}. The DQN then stores this experience $(s_t, a_t, s_{t+1}, r_{t+1}, d_{t+1})$. In order to learn, the DQN samples from its database of experiences and for each experience sampled j, it performs a step of gradient descent on the loss function

$$L = \begin{cases} (Q(s_j, a_j) - r_{j+1})^2 & if \;\; d_{j+1} \\ (Q(s_j, a_j) - (r_{j+1} + \gamma \max_a Q(s_{j+1}, a)))^2 & otherwise \end{cases}$$

where $\gamma \in [0, 1]$ is the discount factor, which makes future rewards worth slightly less than immediate rewards.

In addition to the DQN above, I tried out six different improvements on it:

Dueling DQN - This is a change to the DQN model architecture that separates the learning of how good the current state is and how good each of the actions is relative to each other.

Double DQN - The goal of the loss function is to get the first term as close to the second term as possible. However, the Q function is used in both the first and second term. This means that as the first term changes, so to does

the second term, which the first is trying to match. This can cause difficulties, so the double DQN aims to combat this by introducing a second network Q_{target}, which has the same architecture as the Q network, but it's parameters are frozen, and periodically copied from the Q, so that the goal of the Q network does not change as frequently. The loss of Double DQN is then

$$L = \begin{cases} (Q(s_j, a_j) - r_{j+1})^2 & if \;\; d_{j+1} \\ (Q(s_j, a_j) - (r_{j+1} + \gamma \max_a Q_{target}(s_{j+1}, a)))^2 & otherwise \end{cases}$$

Distributional DQN - This operates under the assumption that reward is partially random and non-deterministic, and because of this assumption, the output of $Q(s, a)$ is not a single value, but rather a discrete probability distribution.

Noisy Networks - When the DQN starts off, it knows nothing about the environment and so it would be pointless to use the network to make actions. Instead, we typically ignore the output of the network and instead take random actions in the beginning, doing this less frequently as the network learns. Noisy Networks incorporate this randomness into the architecture of the network. Consider a normal fully-connected layer with input x: $z = wx + b$, where the parameters being learned are w and b. A noisy fully-connected layer would instead sample w and b from Gaussian Distributions with means and standard distributions μ_w and σ_w and μ_b and σ_b, respectively, where μ_w, σ_w, μ_b, σ_b are the parameters being learned. This way, the network itself learns when to be random and when to be precise.

N-Step Returns - The idea of this is that instead of learning from experiences that only span over 2 time steps, the network learns from experiences that span over N time steps. Experiences are then given by

$$(s_t, a_t, s_{t+N-1}, \sum_{i=t}^{t+N-1} \gamma^{i-t} r_t, d_{t+N-1})$$

Prioritized Experience Replay - The idea here is that some experiences have more to be learned from them than others. Instead of experiences being sampled uniformly to learn from, they are sampled with probability proportionally to their resultant loss the last time they were used.

I had planned to create one final agent, called the Rainbow DQN, that combines all six of these, but I ran out of time.

2

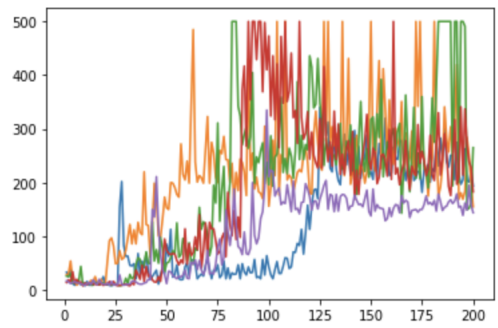
The "data set" in this case would be the game I was playing, which is CartPole. I chose it because it is learned very quickly, and so it was quick to debug.

It is a very simple game and doesn't really have any implications other than demonstrating that the agents are learning proberly, so there isn't much to say here.

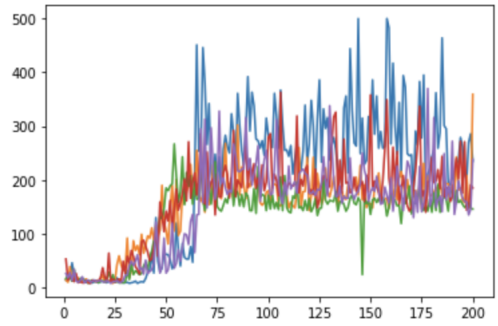
3

Each agent was trained for 200 games and reset 5 times, for a total of 1000 games. The x axis is the game number and the y axis is the total reward achieved during that game. Let's take a look at the plots:

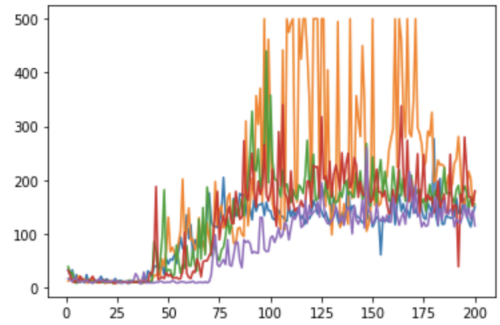
DQN:



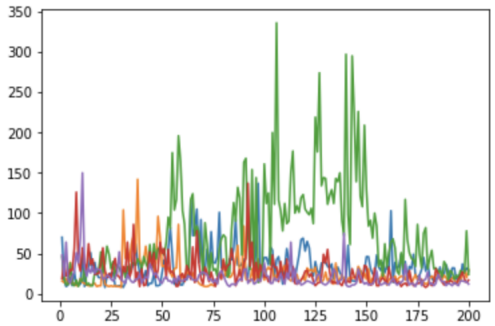
Dueling DQN:



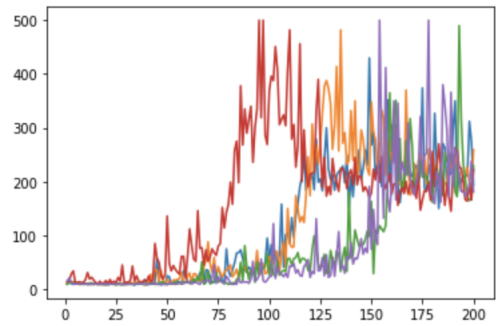
Double DQN:

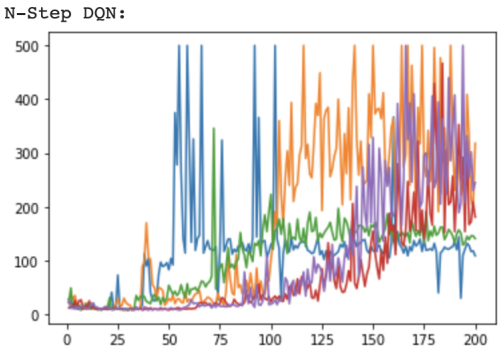


Distributional DQN:



Noisy DQN (unfactorised noise):





Here are my thoughts on them: Dueling DQN - Much more stable and consistent than the DQN, but did not achieve very high rewards often. I think this one would have performed best if I had trained them all for longer. Double DQN - Surprisingly similar to the vanilla DQN. Distributional DQN - CartPole should not be very hard to learn; There is probably a bug here, but I couldn't find it. Noisy - You can see that it differs a lot when rewards really start to pick up, which is expected because of the different exploration scheme and inherent randomness. N-Step DQN - A lot of variance in how this performed; my guess would be because this model does not see the immediate next frame, only the frame N steps in the future, so it could miss some things. I was unable to get a graph for the Prioritized Experience Replay, but from singular, shorter training runs, it appeared to be one of if not the best performers out of all of them.

4

I'm not exactly sure what this question means, so I'll use this section to explain a little about the reward scheme of CartPole. The agent receives a reward of 1 for every step it keeps the pole upright and the cart near the center, and then the game ends when it fails to do either of those. I think Distributional DQN is poorly suited to this environment because rewards are pretty close to deterministic, but also that Prioritized Experience Replay is well suited to it because there are going to be be a lot of frames that don't mean much, but the frames near the ends of the games are probably going to be more important and should receive more attention.