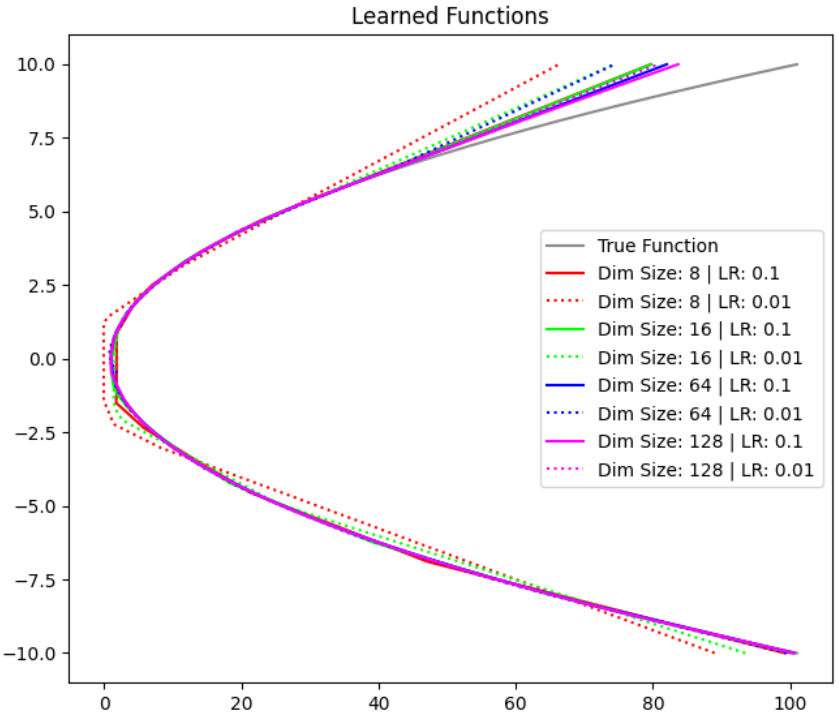
Exercise 8
Gregory Attra
04.05.2022
CS 7180 - Prof. Amato
Code Setup:

1. Unzipped the ‘ex7.zip‘ file
2. ‘cd‘ into the ‘code‘ directory
3. run ‘source ./init‘ to setup the pyenv

Question 1:

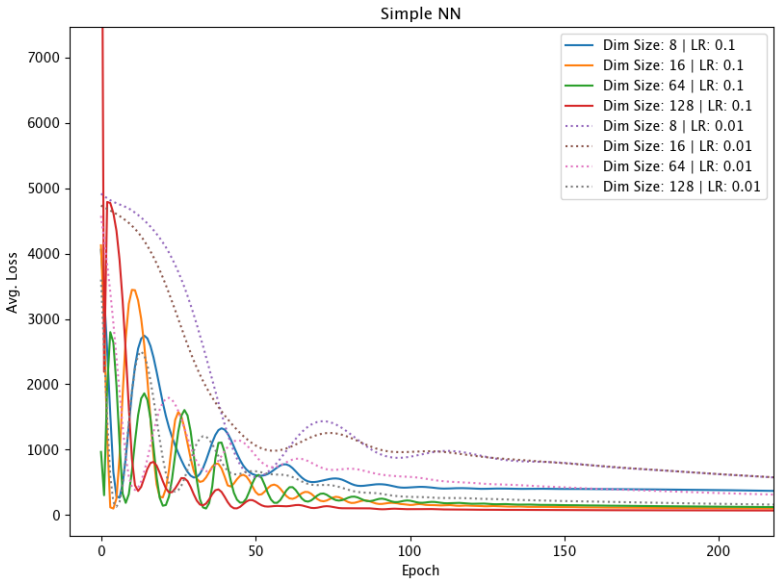
- To run the neural network program, run: ‘python src/dqn/run_nn.py‘

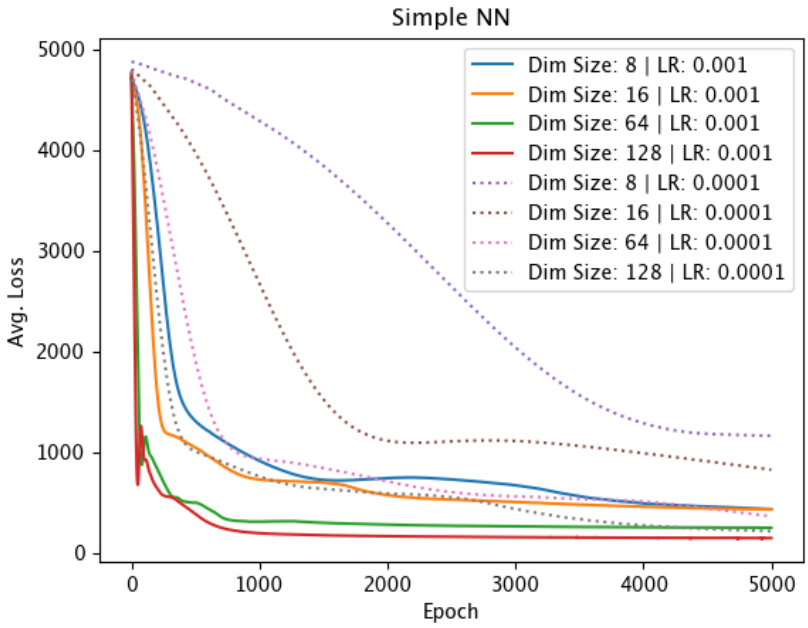
Below is a plot of the learned function compared against the true function (plotted in black). As expected, the bigger the network, the more closely the learned function matches the true function. In most ML use-cases, however, a bigger network runs the risk of overfitting to the data and failing to generalize.



Below are plots of the training loss across various learning rates. We see that with a high learning rate, the loss oscilates as the weights make big jumps

around the optima. With smaller learning rates, we see less oscilation but slow training times.





Question 2: Four Rooms

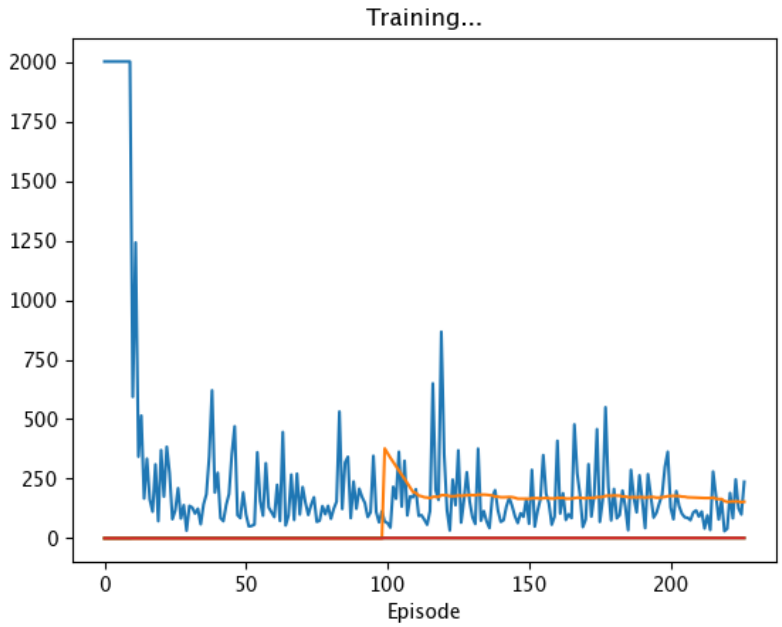
Despite successfully learning policies for Cart Pole and Lunar Lander using DQN, I struggled to learn an optimal policy for Four Rooms. Often the learned policy would result in the agent getting stuck in an infinite loop of bad action selection, which would perform far worse than a random policy.

I tried tuning many different hyperparameters and tested on the following state representations:

- Single integer representing the "id" of the current cell occupied by the agent
- (x, y) coordinates of the agent
- One-hot encoding (tabular) state representation
- Passing the entire flatted map, -1 for walls, 1 for the goal, and 2 for the agent's current position

Finally, with one small trick, I was able to learn a near-optimal policy with DQN: add stochasticity to the domain. I added some "slip" to the action selection, where 20% of the time, the agent executed an action it did not select. This solved the issue of getting stuck in an infinite loop of suboptimal action-selection by increasing the probability that the agent would move to the goal state, which would increase the Q-values for nearby states as well.

The results are plotted below. In blue, I plot the episode durations, along with the running average (orange). I also plot the rewards per episode but because the living reward is 0, the reward is flat.



The key metric here is episode duration. We see that for the first several episodes, the agent fails to find the goal state before the max number of timesteps is reached. But once it manages to find the goal state, learning improves dramatically and we see a significant drop in the average episode duration. Eventually performance converges at 150-200 steps per episode.

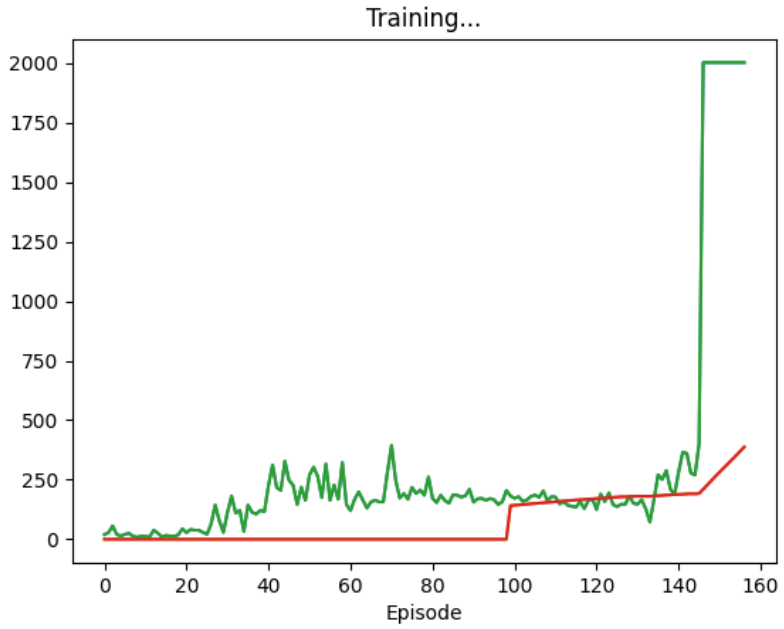
This is still less optimal than the tabular function approximation solution.

Question 3:

CartPole

- To run the CartPole DQN program, run 'python src/dqn/run_cartpole.py'
- For my implementation, I used an epsilon decay factor of 0.995 to anneal the epsilon for action selection from 1. to 0.05

The plot below plots the reward over time of the CartPole DQN (green) along with the running average (red).



We see that for the first 25 episodes, the agent is rapidly exploring and quickly failing the task. Once enough data is generated, we see a slight increase in the reward.

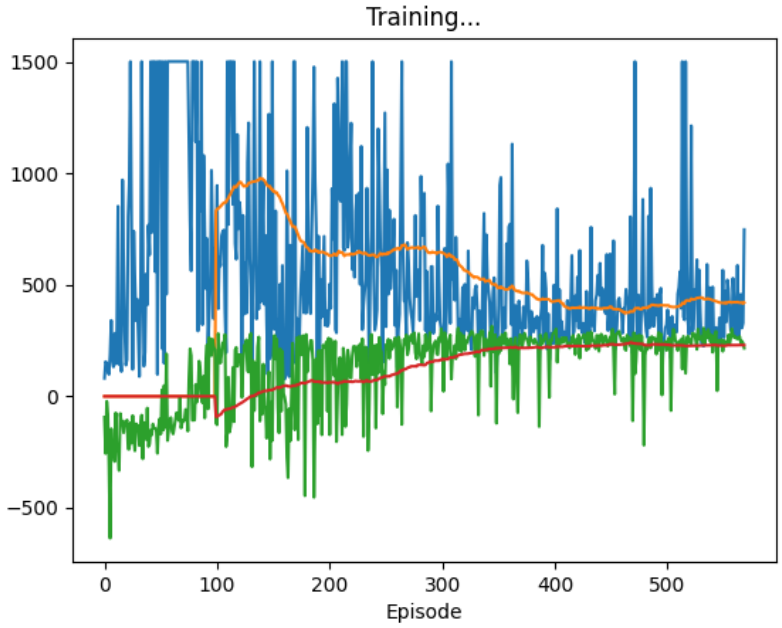
Watching this live, I noticed that, by episode 40, the agent seemed to have learned not to let the pole fall. However, it's strategy was to move the cart swiftly in the direction the pole was falling, causing the cart to move out of the screen bounds. The value function for this behavior placed high value on actions moving the cart in the direction the pole was falling.

My intuition says that, as more experience was collected, the value of moving in the direction the pole was falling decreased the closer the agent got to the edge of the screen. Over time, as the cart approached the edge of the screen, it began to try moving in the opposite direction. At this stage, the value of moving away from the edge of the screen was greater than the value of moving towards the edge.

From there it took many iterations of this behavior before the agent learned to oscillate left and right rapidly to keep the pole balanced without exiting the screen. Once this behavior was learned, the agent successfully balanced the pole for 2000 timesteps in each episode thereafter. By now it had learned to avoid getting close to the edge of the screen. My intuition says that the value function would be like a Gaussian over the middle of the track on which the cart ran, with the center being the most ideal place for the cart, and state values decreasing closer to the edge of the screen.

Lunar Lander

Below is a plot of the training process of the DQN on the Lunar Lander domain. In blue, I plot the duration of each episode, along with the running average in orange. In green I plot the reward earned for each episode, along with the running average (red).



- To run the Lunar Lander DQN program, run 'python src/dqn/run_lunar_lander.py'
- For my implementation, I used an epsilon decay factor of 0.995 to anneal the epsilon for action selection from 1. to 0.1

Watching this live, I observed the learning process was as follows:

- Episodes 1 - 50: Random action selection / exploration
- Epsidoes 50 - 150: Avoid the ground. Because initial experiences close to the ground ended with crashing (low negative rewrad), the agent first learned to avoid the ground altogether. This would result in very long episodes. Because there is a living negative reward when not touching the ground, the agent would eventually learn low Q-values for these state-actions.
- Episodes 150-275: Touch legs to the ground. There is a small reward when the lander's legs touch the ground. This was the first step in learning to overcome its initial "fear" of the ground. This breakthrough resulted in a dramatic decrease in episode length as the agent would race to reach the ground. However, it would spend a sub-optimal amount of time trying to land gently, so there was still room for improvement / efficiency.
- Episodes 275 - 400: Come to rest. The agent would learn to land softly and stop firing its engines. This would yield a high positive reward. The agent This is the second best behavior, with the best behavior being to land within the goal posts.
- Epsiodes 400 - 600: By this point the agent had learned that the goal was to land softly between the goal posts. Through experience it would learn to perform this task with increasing efficiency, but due to the high epsilon value of .1, exploration would prevent the agent from being "perfect".