CCE Reinforcement Learning Final Project

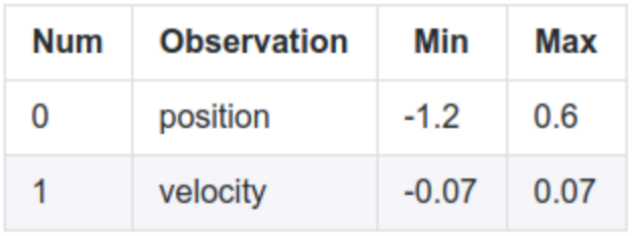
Avinash Shanker
avinash270394@gmail.com

Solving Mountain Car with Q-Learning

Abstract:
Mountain Car is a classic reinforcement learning problem where the objective is to create an algorithm that learns to climb a steep hill to reach the goal marked by a flag. The car's engine is not powerful enough to drive up the hill without a head start so the car must drive up the left hill to obtain enough momentum to scale the steeper hill to the right and reach the goal.

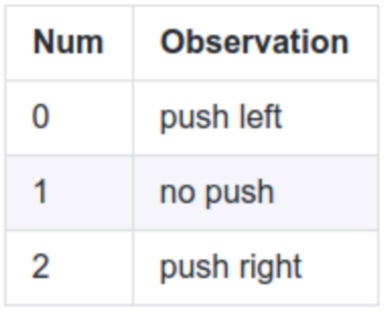
Q-Learning:
We will use a reinforcement learning technique called Q-Learning to solve this problem. Q-Learning is an algorithm that attempts to learn a function or *policy* which takes an observation of the environment as input and returns an action as output. Q-Learning does this by determining which action is best in the current state as well as all future states. We call this function the action value function or ***Q(a,s),*** where ***Q*** is the value of taking action ***a*** in state ***s.***

Mountain Car:
Returning to Mountain Car for a moment, when the problem begins the car is dropped into the valley and given an initial position and velocity as a vector. This is the car's ***state***. Our agent must then tell the car to take one of three ***actions***: drive left, do nothing, or drive right. This action is sent to the Mountain Car environment algorithm which returns a new state (position and velocity) as well as a reward. For each step that the car does not reach the goal, located at position 0.5, the environment returns a ***reward*** of -1. We will use these rewards in our Q-Learning algorithm to solve the Mountain Car problem.



Num	Observation	Min	Max
0	position	-1.2	0.6
1	velocity	-0.07	0.07

Mountain Car Range of Possible Positions and Velocities — State Space



Num	Observation
0	push left
1	no push
2	push right

Set of Possible Actions — Action Space

Updating Q:
We will use a PyTorch neural network with one hidden layer of 200 units to learn ***Q.*** **Our goal is to figure out a way to turn this problem into a supervised learning problem** so we can use a feed-forward neural net to solve. Here is the equation we will use to solve Mountain Car. Let's break it down.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big[r_t + \gamma \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\big]$$

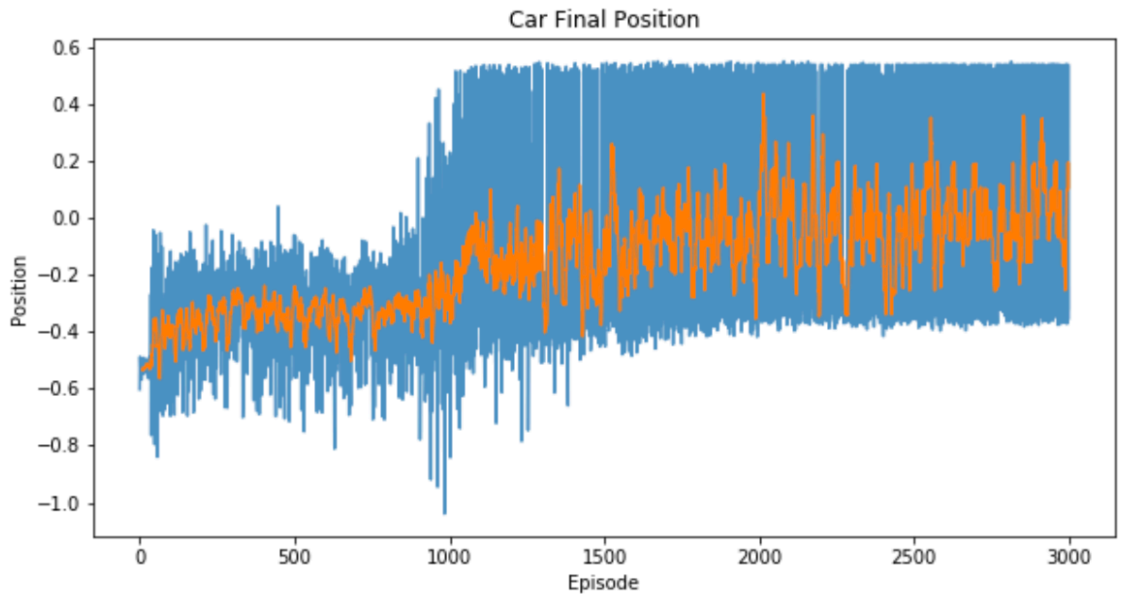
Q-Learning Update Equation

Every time we step forward in the environment by driving left, right, or doing nothing we will update ***Q*** based on the reward for the agent's action and the maximum future action value function one step in the future. The portion inside the brackets becomes the **loss function** for our neural network where ***Q(st,at)*** is the output of our network and rt + γ max Q(st+1,at+1) is the **target Q** value as well as the label for our neural net turning the problem into a supervised learning problem solvable using gradient descent where α is our learning rate.

Pseudocode

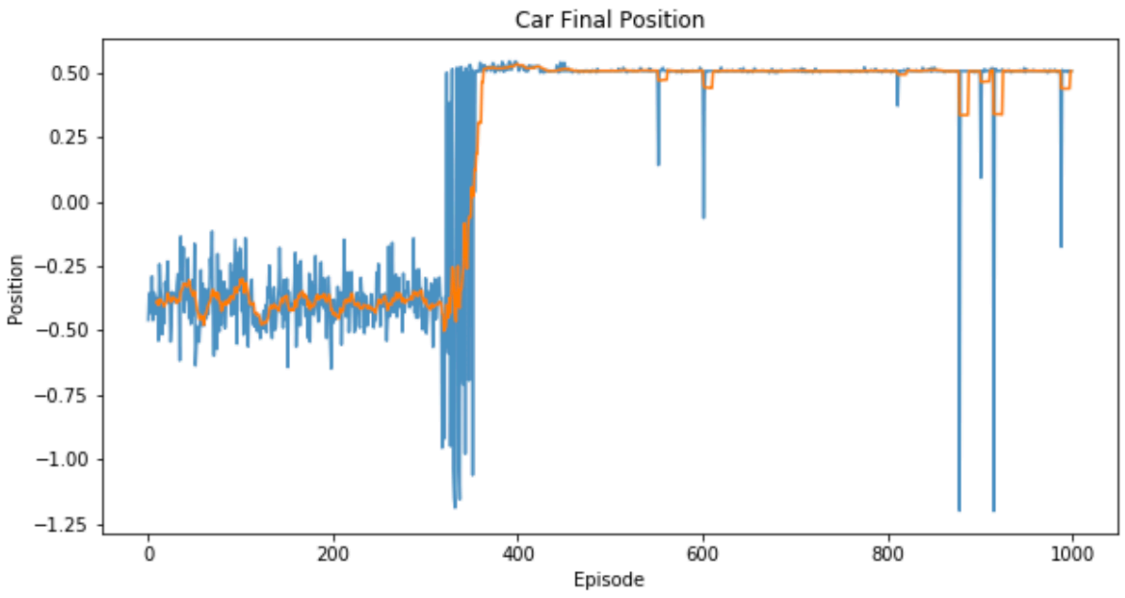
```
Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Repeat (for each step of episode):
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until $S$ is terminal
```

When we run the code for 3000 episodes we get the following results for the final position of the car. The car moves randomly at first and around episode 1000 finally reaches the top of the hill and is able to learn a policy which reaches the top of the hill in around 1/3 episodes from that point forward.



Car Final Position — Unmodified Reward

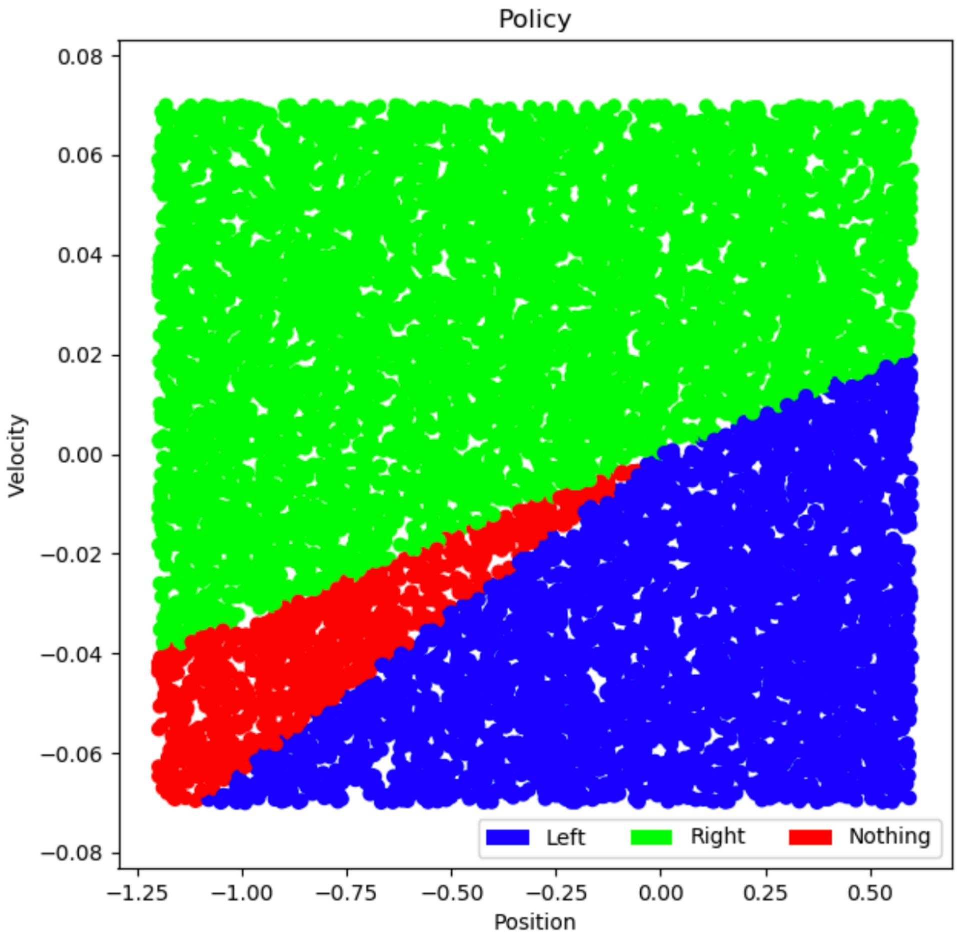
If we adjust the reward parameter as per the code below and use a hidden layer of 50 neurons, the agent learns to reach the goal around episode 300 and achieves its objective in almost every episode thereafter.



Car Final Position — Modified Reward

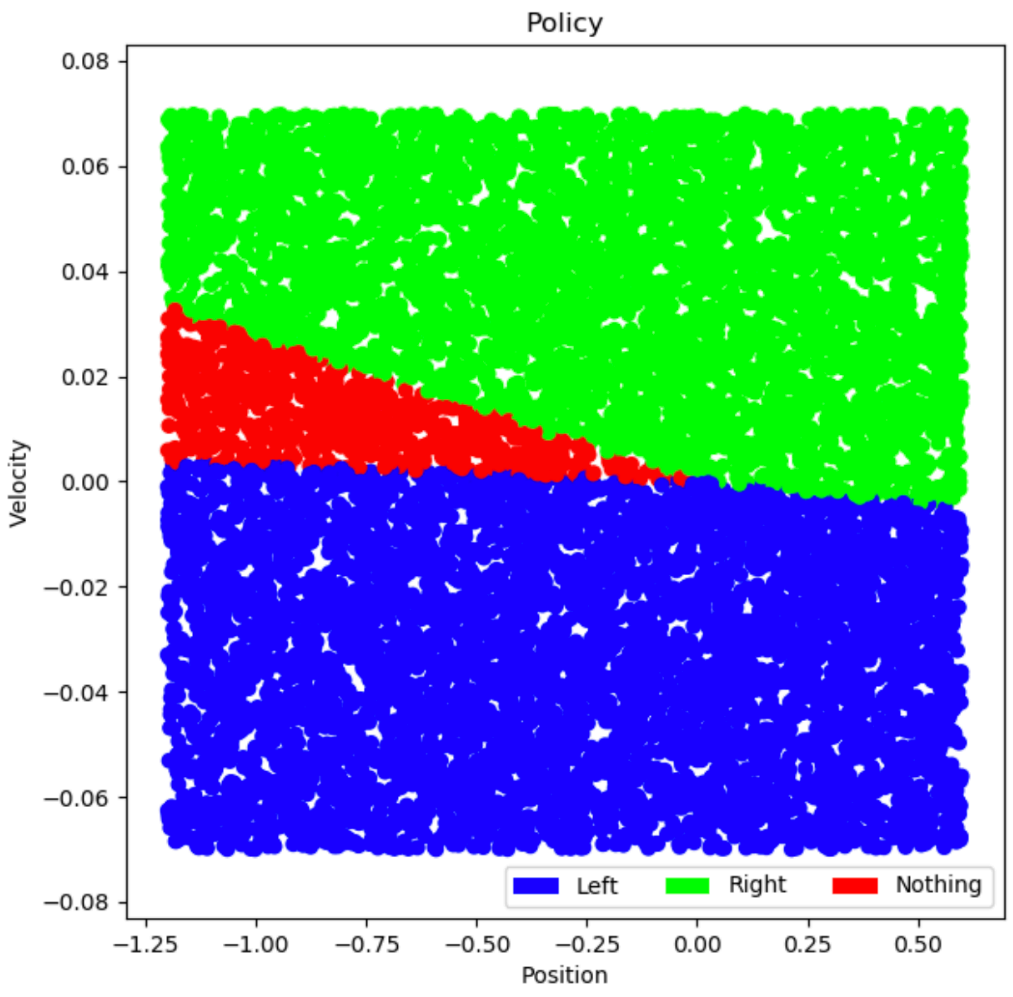
Policy Visualization

We can see the policy by plotting the agent’s choice over a combination of positions and velocities. You can see that the agent learns to, *usually*, move left when the car’s velocity is negative and then switch directions when the car’s velocity becomes positive with a few position and velocity combinations on the left side of the environment where the agent will do nothing.



Policy — Unmodified

The agent with a modified reward function learns a different policy that is entirely based on velocity. The agent always moves the car left when velocity is negative, sometimes does nothing, and almost always moves right when velocity is positive. Based on the results we can tell that this policy is much more effective.



Policy — Modified Reward