# Q-learning

Gaurav and Tia
Data Science Learning Group
March 25, 2020

# Overview

---

- Basics
- Bellman Equation
- Temporal Differencing
- Q-table
- Parameters
- Learning Procedure
- Issues
- Summary

# Q-learning - Basics

- Model-free
  - Model: predicts what the environment will do next (i.e. state change due to agent's action, reward from each state)
- "Q": function that returns the reinforcement reward
  - The "quality" of an action taken in a given state
- Learns a policy
  - Policy: what we do in any particular state
  - Maximizes expected value of total reward over successive steps, starting from current state
  - For any finite Markov Decision Process (FMDP), finds optimal policy (with infinite time and partly-random policy)

# The Optimization Problem

---

- How do we get to the Q-function?
- This is where Bellman's principle of optimality comes in:

> "Whatever the initial state and initial decision, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision"

- This breaks down a complex problem into a series of simpler sub-problems
  - Instead of optimizing global reward all at once, optimize local reward

# Bellman Equations

---

- Suppose the state at time t is given by $x_t$, and an action $a_t$ is taken from that state. The reward from taking the action is $R(x_t,a_t)$.
- We can repeat this process for each successive state, while adding in a discount factor, $\gamma$, to prioritize earlier rewards.
- This defines the value function for the initial state $x_0$:

$$V(x_0) = \max_{a_t} \sum_{t=0}^{T_{fi}} \gamma^t R(x_t, a_t)$$

# Bellman Equations

---

- We can separate out the terms as follows:

$$V(x_0) = \max_{a_0}[R(x_0, a_0) + \gamma(\max_{a_t} \sum_{t=1}^{T_{fi}} \gamma^{t-1} R(x_t, a_t))]$$

- This gives us the Bellman Equation (suppressing subscripts):

$$V(x) = \max_{a}[R(x, a) + \gamma V(T(x, a))]$$

- This is a functional equation that can be solved for the value function - in our case, the Q-function

# Temporal Differencing

---

- Solve for the Q-function iteratively - initialize Q-values randomly and update them at every time step.

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Big(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}}\Big)}^{\text{temporal difference}}$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{new value (temporal difference target)}}$$

- The prediction at any given time step is updated to bring it closer to the prediction of the same quantity at the next time step.

# Q-table

---

- In practice: Q-function represented with a table
- Q-table maps states and actions to q-values

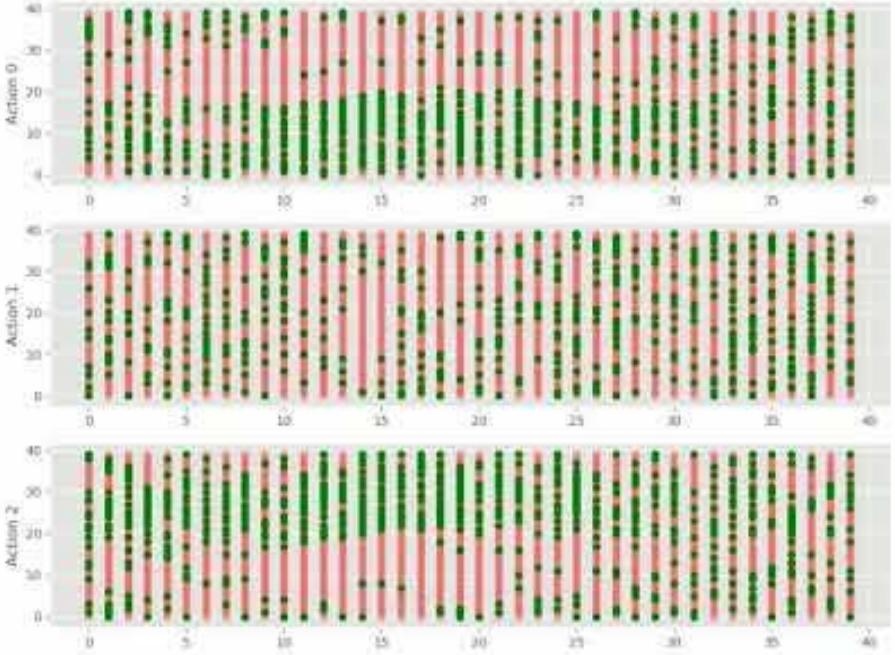
Initialized ---> Trained

| Q-Table | | Actions | | | | | |
|---|---|---|---|---|---|---|---|
| | | South (0) | North (1) | East (2) | West (3) | Pickup (4) | Dropoff (5) |
| States | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | . | . | . | . | . | . | . |
| | . | . | . | . | . | . | . |
| | . | . | . | . | . | . | . |
| | 327 | 0 | 0 | 0 | 0 | 0 | 0 |
| | . | . | . | . | . | . | . |
| | . | . | . | . | . | . | . |
| | . | . | . | . | . | . | . |
| | 499 | 0 | 0 | 0 | 0 | 0 | 0 |

| Q-Table | | Actions | | | | | |
|---|---|---|---|---|---|---|---|
| | | South (0) | North (1) | East (2) | West (3) | Pickup (4) | Dropoff (5) |
| States | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | . | . | . | . | . | . | . |
| | . | . | . | . | . | . | . |
| | . | . | . | . | . | . | . |
| | 328 | -2.30108105 | -1.97092096 | -2.30357004 | -2.20591839 | -10.3607344 | -8.5583017 |
| | . | . | . | . | . | . | . |
| | . | . | . | . | . | . | . |
| | . | . | . | . | . | . | . |
| | 499 | 9.96984239 | 4.02706992 | 12.96022777 | 29 | 3.32877873 | 3.38230603 |

- Policy selects action with highest value for state

# How Q-learning Learns - Q-table



# Q-learning - Parameters

- Learning rate
  - Affects how strongly the current and future rewards influence the updated q-value
    - i.e. how quickly it learns
- Discount parameter
  - tracks how much future reward should affect current decision making
  - Value of 1: future reward is of equal value as current reward
  - Value of 0: future reward not considered
  - Rewards received earlier valued higher than those received later

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left( \underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)}^{\text{temporal difference}}$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{new value (temporal difference target)}}$$

# Q-learning - Parameters (Cont'd)

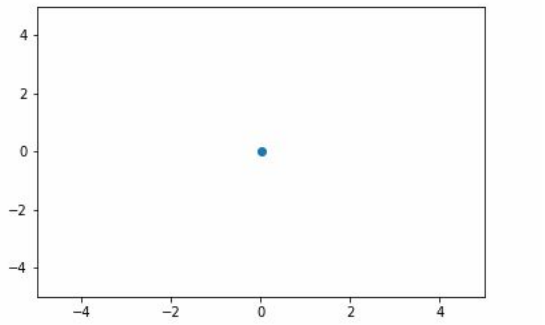
- Epsilon parameter
  - determines to what extent newly acquired information overrides old information
    - 0 - agent solely exploits prior knowledge (exploitation)
    - 1 - ignores prior knowledge to explore possibilities (exploration)
  - slowly decays to escape false minima
  - Epsilon-greedy strategy:
    - Picks the current best option ("greedy") most of the time, but pick a random option with a small (epsilon) probability sometimes

# Q-learning Procedure

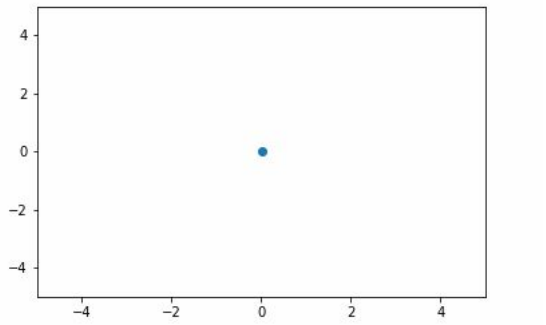
- Big picture:
  - Select training data (and testing data)
  - Iterate over time
    - We have our state
      - consult our policy to select our action (exploitation)
      - Select random action (exploration)
    - Get next state and reward
  - Use experience tuple (s,a; s’,r) to update q-table
  - Test policy
  - Repeat until it converges

# Example

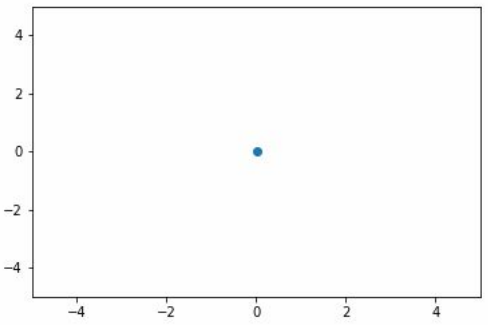
Some training….




More training...




Even more training




Goal: Stay in center of box

# Issues with Q-learning

- Q-table grows very quickly with complexity
  - System dimensionality/finer discretization scheme dramatically increase q-table size
  - Larger action spaces increase size and also the time required to learn the table
- Very easy to get stuck in local minima due to the size of Q-tables (even with random exploration)
  - There can be “unexplored” parts of the Q-table which result in a greater reward.

# Summary

---

**Q-learning:** a model-free RL technique to **develop an optimal policy** for an agent in some environment

- Q-table values optimized at each time step
- Training uses a combination of exploration and exploitation

# Deep Q-learning

---

- (Very) High level: replace q-table and update mechanics with deep neural network
  - Input: observation space
  - Output: q-values
- Major issue: overfitting
  - Since net is updated every step, high tendency to overfit to current observation space
- Solution: 2 copies of the neural network
    - Update one of the nets at every step, but the fitting is done w.r.t past steps (so called ‘replay memory’)
    - The other is used to predict the q’s at each step and is updated periodically after a set number of steps (hyperparameter to be tuned)