Q-Learning: Deep Deterministic Policy Gradient

Zander Zemliak
CMSI 5350
December 6, 2022

Overview of Topics

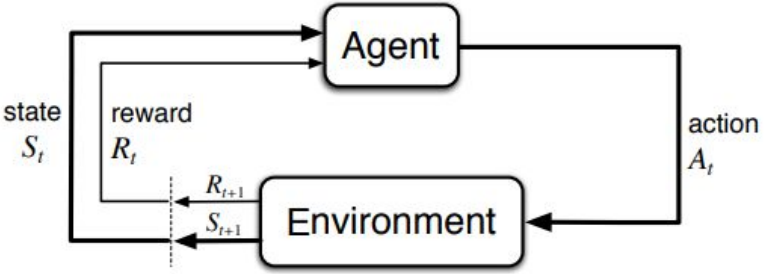
Reinforcement Learning

Introduction to RL

- RL is one of the main disciplines of machine learning.
- Differences between the three main ML disciplines:
    - Supervised learning is concerned with learning an approximate mapping between an input and an output using a labeled dataset.
    - Unsupervised learning is concerned with finding a pattern in unlabeled data.
    - RL agents learn by interacting with their environment.
- RL agents main objective is to take actions that maximize reward.
- A dog learns that when it sits on command it gets a treat.

Markov Decision Process (MDP)

- MDP is an essential framework for RL, it is a mathematical model of decision making where the outcomes can be partly arbitrary or under control.
- Four main components for MDP: states, actions, effects of actions on future states, effects of actions on future rewards.



- Agriculture ex: How much to plant given the current state of water and soil?

Figure: https://towardsdatascience.com/introduction-to-reinforcement-learning-markov-decision-process-44c533ebf8da

Brief MDP Math

- An MDP can be described as: MDP = (S, A, T, R, γ)
  - S = set of possible states
  - A = set of possible actions
  - T = set of transition probabilities where the probability of going from s to s' is described as P(s'|s, a).
  - R = set of rewards
  - γ = discount factor (determines how much agent cares about future rewards and is necessary for algorithm convergence)
- We will use the Bellman equation to evaluate the max current and future reward:

$$\mathbf{V(s) = \max_a(R(s, a) + \gamma V(s'))}$$

- The value for our current state, **V(s)**, is equal to the action which maximizes current reward in state ***s*** with action **a** plus the value of the next state **s'**.

Q-Learning

Q-Learning

- Q-learning is a *value based* RL algorithm that attempts to learn the quality (Q) of actions in particular states that are defined by future value.
- The mapping of action to state is known as the policy (π) and these values are stored in a Q-table with two columns: state-action pair and value.
- Agents choose actions in certain states that result in the highest reward Q*(s,a).
- Q-table values are updated using TD:

$$Q_{new}(s, a) = Q(s, a) + \alpha[R(s,a)+\gamma \max_a Q(s', a=a) - Q(s, a)]$$

- TD target looks very familiar to Bellman equation...
- This will converge towards π*

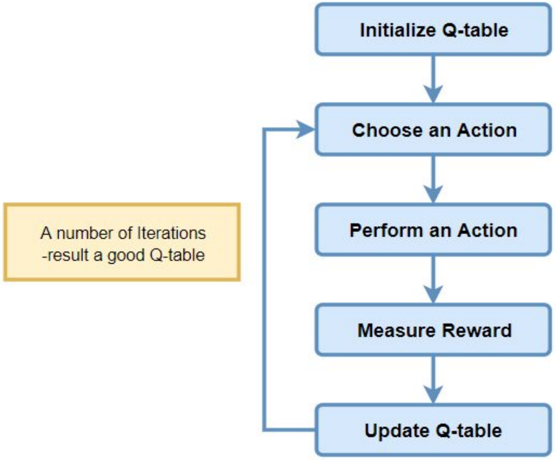


Figure: https://towardsdatascience.com/a-beginners-guide-to-q-learning-c3e2a30a653c

Deep Q-Learning

Deep Q-Learning

- Very similar idea to Q-Learning but instead of using a Q-table with (state, action) pairs mapped to a Q-value for choosing actions, we have a neural network that maps input states to (action, Q-value) pairs.
- Since in real life we could have many (state, action) pairs for a given environment, it is better to use a NN for a large state space because of its generalizability.

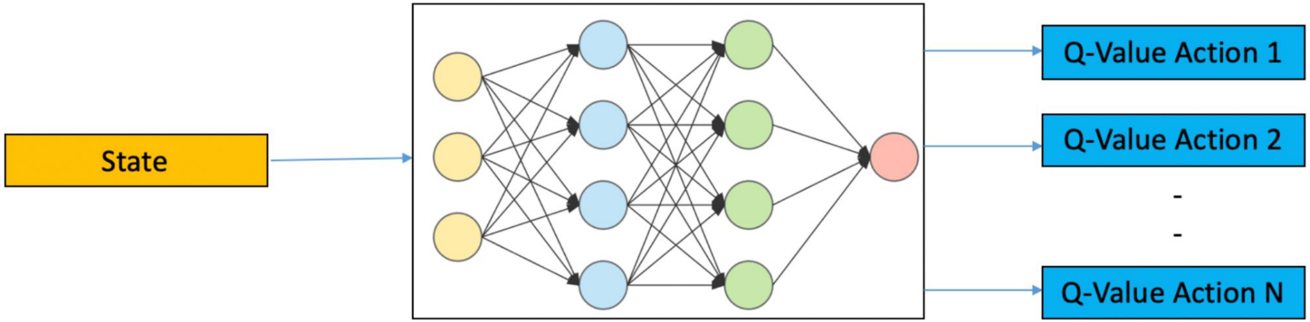


Figure: https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/

Training a DQN

- DQN training works by taking an action via epsilon-greedy, observing the episode of that action, storing it in memory, and then sampling from the replay memory to update main prediction NN.
- We use two networks: a main and target network
    - Main NN: Where our Q-values are derived
    - Target NN: Same parameters as the main NN however, this network is not trained. It is synchronized with the main NN after N steps. This allows us to train the main NN with a single target thus making training stable.
- Weights are updated in the main network via the Q-Learning Loss function:

$$L(\theta) = (\overbrace{(r + \gamma max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta^{target}))}^{\text{Target Q value}} - \overbrace{Q(s, a; \theta^{pred})}^{\text{Predicted Q value}})^2$$

Figure: https://medium.com/intro-to-artificial-intelligence/deep-q-network-dqn-applying-neural-network-as-a-functional-approximation-in-q-learning-6ffe3b0a9062

Training a DQN

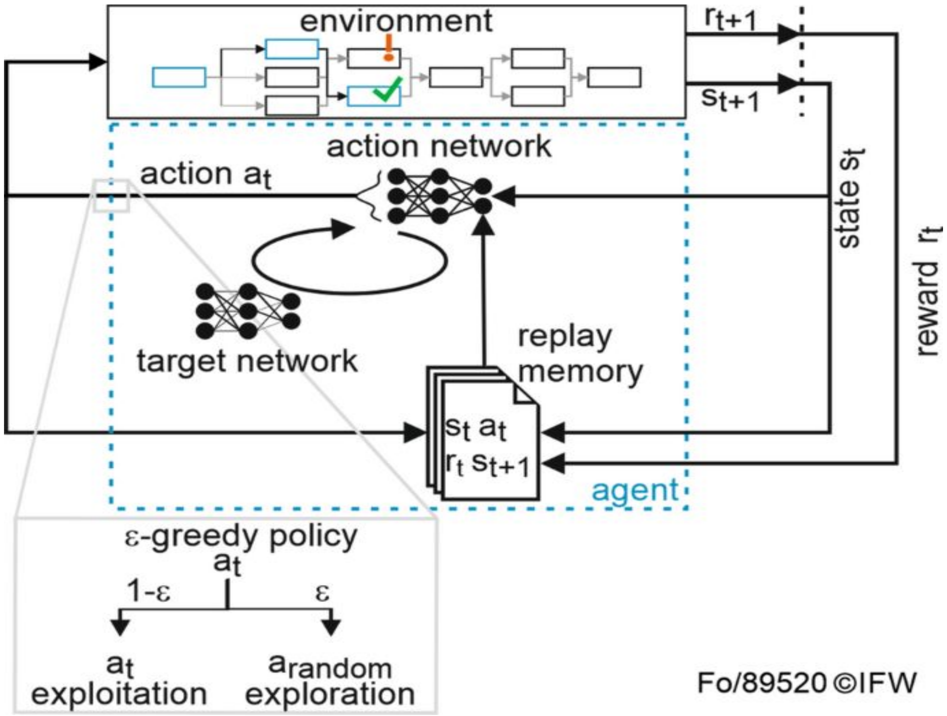


Figure: https://link.springer.com/article/10.1007/s11740-020-01000-8

DQN Algorithm: Deep Deterministic Policy Gradient (DDPG)

DDPG Overview

- Q-Learning and Deep Q-Learning are great for environments that have discrete action-state spaces. But, what if our action-state space is *continuous*?
- What if the amount of actions we can take in any given state is not finite?
- Remember, the ultimate goal of our agent is to choose the best action for any state:

$$\pi(s) = a^*(s)$$

$$\text{where; } a^*(s) = \text{argmax}_a Q(s, a)$$

- To solve this issue of having a continuous action-state space, DDPG uses an actor network and critic network.
  - Actor network: Responsible for mapping state to a^*
  - Critic network: Responsible for mapping (s, a^*) tuple to $Q(s, a)$

DDPG Training

- The actor network is trained to maximize the Q-value from the critic network so its update rule uses gradient **ascent**.
- The critic network is updated using the traditional Q-Learning Loss function with.
- Similar to DQN, each actor and critic network both have target networks to help with stabilizing learning.

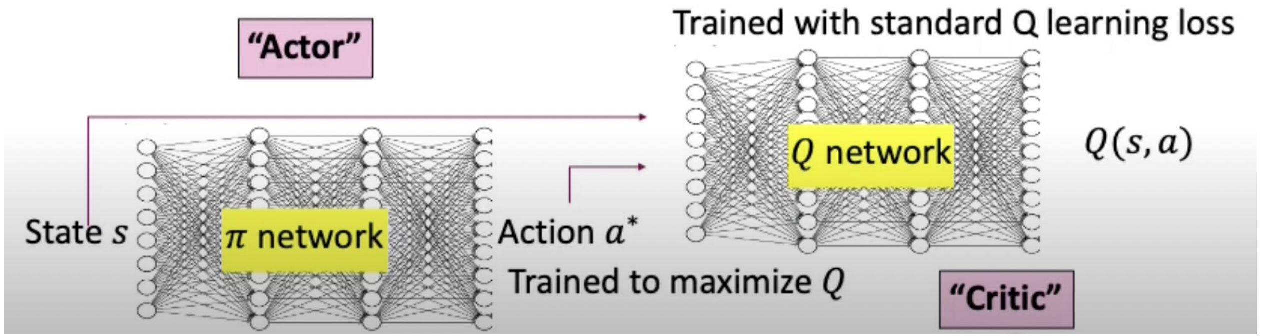


Figure: https://www.youtube.com/watch?v=ljHwbY9QrJU&ab_channel=CIS522-DeepLearning

DDPG Algorithm Steps

1. Initialize actor θ params (π net), critic ϕ params (Q net), and empty replay buffer B
2. Set target params equal to their respective main networks (just like DQ Learning)
3. Repeat until convergence:
    a. Observe state s and get a* from critic and execute a*
    b. Observe state s', get reward r, and set d = is in terminal state (0=no, 1=yes)
    c. Store episode (s, a, r, s', d) in replay buffer
    d. If d = 1 reset environment parameters otherwise continue
    e. If time to update:
        i. For however many updates:
            1. Get sample batch from B
            2. Compute target via Bellman: $\mathbf{y(r, s', d) = r + \gamma(1\text{-}d)Q_{\phi_{target}}(s', \pi_{\theta_{target}})}$
            3. Update critic net via one step of GD using $\mathbf{y(r, s', d)}$
            4. Update actor net via one step of GA using: $\mathbf{Q_\phi(s, \pi_\theta(s))}$
            5. Update target networks with their respective main network

Pseudocode: https://spinningup.openai.com/en/latest/algorithms/ddpg.html

DDPG Example:
Stock Trading Agent

Problem Justification

- Managing a portfolio of stocks to create a comfortable return can be very difficult:
    - Emotions can impact one's due diligence on a planned strategy.
    - Trading in live time can be stressful.
    - You don't know what you're doing.
- Because the stock market changes every second of everyday, the state space is **extremely** large. DDPG is intended for large state spaces.
- Using DDPG, we can potentially find a viable trading strategy within the vast complexities of the financial markets.

Problem Overview

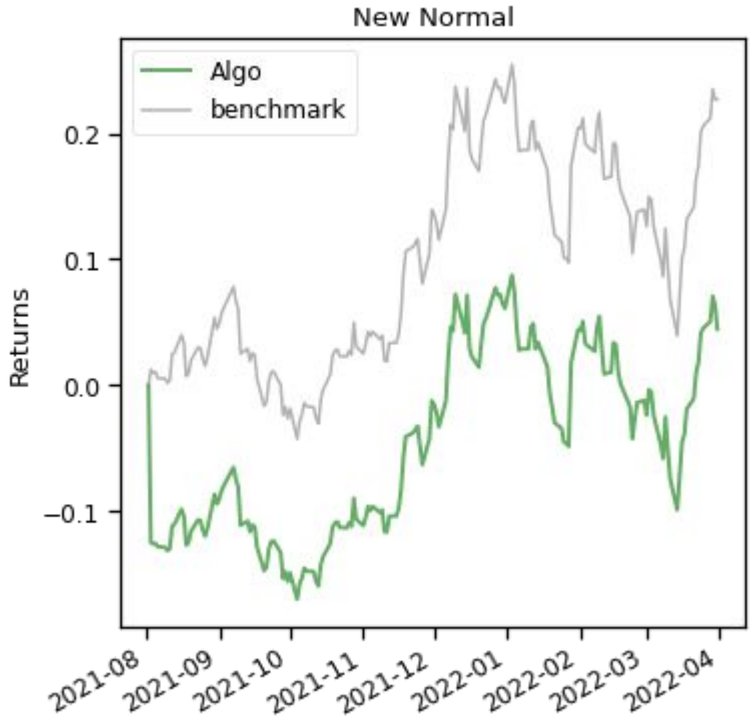
Some Stuff for Q-Learning:

- Actions: buy, sell, or hold a stock.
- State space: N stocks we are interested in that have features such as closing, high, low, volume, day of week, rsi, ema, sma, etc..
- Reward: Return of stock of trade.
- Therefore, overall goal of agent is to maximize trading returns

Data:

- We will have our agent only trade Apple stock (AAPL).
- We will compare the performance of our agent to how Apple stock performed on its own in the same time frame.
- For training, we will use 10 years of historical data from Apple.
- For testing, we will use 8 months of historical data of Apple.

Results



August 2021 to April 2022:

- DDPG result: +4.47%
- AAPL result: +22.7%

Results

- Model performed poorly with respect to the overall movement of Apple in the same time frame.
- An individual buying and holding the stock with X amount of starting cash would have outperformed the trading agent significantly.
- The model appeared to have adopted a FOMO strategy. Buying and selling with respect to Apple's price movement. Were the technical indicator features just noise?
- A positive return appears as a success. Should you let this model trade with real money? **NO!** A positive backtest result does not indicate any kind of model generality for trading.

Future Improvements

- Adjusting hyperparameters:
  - Changing learning rate
  - Lowering batch sample size
  - Lowering the number of steps.
- Using different features for training (deeper research into technical indicators).
- Using a different set of stocks that have a lot more volatility.
- Hardcoding functions that step into the agent's actions if a certain threshold of success/failure is met. E.g. stop trading if cumulative return is +/- 5%.

Thank you!
Questions?