# Q Learning and Deep Q Network

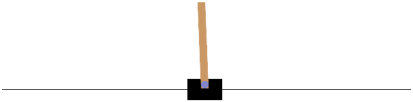
Ling Fei Zhang
260985358

April 9, 2023

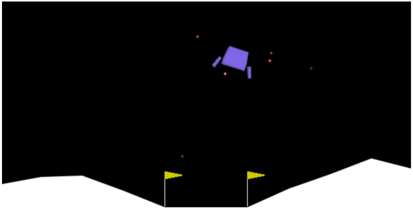
# Table of Contents

# Games

CartPole



Lunar Lander



# Agents

- Q Learning
  - Model-free reinforcement learning algorithm
  - Uses a table to store Q values for each state-action pair
  - Effective in simple environments
  - Struggles in more complexe environments since it is impractical to store and update Q values for all the state-action pairs
- Deep Q Network (DQN)
  - Uses neural networks to learn policies to map states to Q values
  - Neural networks can handle large state spaces and continuous action spaces

# Agents

- Q Learning
  - Model-free reinforcement learning algorithm
  - Uses a table to store Q values for each state-action pair
  - Effective in simple environments
  - Struggles in more complexe environments since it is impractical to store and update Q values for all the state-action pairs
- Deep Q Network (DQN)
  - Uses neural networks to learn policies to map states to Q values
  - Neural networks can handle large state spaces and continuous action spaces

# Agents

- Q Learning
  - Model-free reinforcement learning algorithm
  - Uses a table to store Q values for each state-action pair
  - Effective in simple environments
  - Struggles in more complexe environments since it is impractical to store and update Q values for all the state-action pairs
- Deep Q Network (DQN)
  - Uses neural networks to learn policies to map states to Q values
  - Neural networks can handle large state spaces and continuous action spaces

# Agents

- Q Learning
  - Model-free reinforcement learning algorithm
  - Uses a table to store Q values for each state-action pair
  - Effective in simple environments
  - Struggles in more complexe environments since it is impractical to store and update Q values for all the state-action pairs
- Deep Q Network (DQN)
  - Uses neural networks to learn policies to map states to Q values
  - Neural networks can handle large state spaces and continuous action spaces

# Agents

- Q Learning
  - Model-free reinforcement learning algorithm
  - Uses a table to store Q values for each state-action pair
  - Effective in simple environments
  - Struggles in more complexe environments since it is impractical to store and update Q values for all the state-action pairs
- Deep Q Network (DQN)
  - Uses neural networks to learn policies to map states to Q values
  - Neural networks can handle large state spaces and continuous action spaces

# Agents

- Q Learning
  - Model-free reinforcement learning algorithm
  - Uses a table to store Q values for each state-action pair
  - Effective in simple environments
  - Struggles in more complexe environments since it is impractical to store and update Q values for all the state-action pairs
- Deep Q Network (DQN)
  - Uses neural networks to learn policies to map states to Q values
  - Neural networks can handle large state spaces and continuous action spaces

# Agents

- Q Learning
  - Model-free reinforcement learning algorithm
  - Uses a table to store Q values for each state-action pair
  - Effective in simple environments
  - Struggles in more complexe environments since it is impractical to store and update Q values for all the state-action pairs
- Deep Q Network (DQN)
  - Uses neural networks to learn policies to map states to Q values
  - Neural networks can handle large state spaces and continuous action spaces

# Agents

- Q Learning
  - Model-free reinforcement learning algorithm
  - Uses a table to store Q values for each state-action pair
  - Effective in simple environments
  - Struggles in more complexe environments since it is impractical to store and update Q values for all the state-action pairs
- Deep Q Network (DQN)
  - Uses neural networks to learn policies to map states to Q values
  - Neural networks can handle large state spaces and continuous action spaces

# Background

- Future discounted return at time $t$ : $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$
- The optimal action-value function $Q^*(s, a) = \max_\pi \mathbb{E}(R_t | s_t = s, a_t = a, \pi)$
- Optimal stragery is to maximize the expected value of $r + \gamma Q^*(s', a')$

# Background

- Future discounted return at time $t$ : $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$
- The optimal action-value function $Q^*(s,a) = \max_\pi \mathbb{E}(R_t | s_t = s, a_t = a, \pi)$
- Optimal stragery is to maximize the expected value of $r + \gamma Q^*(s', a')$

# Background

- Future discounted return at time $t$ : $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$
- The optimal action-value function $Q^*(s,a) = \max_\pi \mathbb{E}(R_t | s_t = s, a_t = a, \pi)$
- Optimal stragery is to maximize the expected value of $r + \gamma Q^*(s', a')$

# Q Learning Algorithm

**Algorithm** Q Learning(episodes, $\alpha, \epsilon, \gamma$)

Initialize $Q(s, a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$ arbitrarily
Set $Q(terminal, \cdot) = 0$ for all terminal states
**for each** episode in episodes **do**
    Initialize $s$
    $done \leftarrow$ False
    **while** not $done$ **do**
        Choose $a \in \mathcal{A}$ from $s$ using policy derived from $Q$
        Take action $a$ and observe reward $r$ and next state $s'$
        $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_a Q(s', a) - Q(s, a)\right]$
        $s \leftarrow s'$
    **end while**
**end for**

# Q Learning

- Q Learning uses a table to store Q values for each state-action pair
- Impractical to store all Q values for a continuous observation space
- Discretization needed
- Hyperparameters used for Q Learning
  - Discount factor $\gamma = 0.99$
  - Start epsilon of 1
  - End epsilon of 0.001
  - Epsilon decay of $5^{-4}$
  - 10 bins
  - learning rate of 0.25 and 0.005 for CartPole and Lunar Lander, respectively

# Q Learning

- Q Learning uses a table to store Q values for each state-action pair
- Impractical to store all Q values for a continuous observation space
- Discretization needed
- Hyperparameters used for Q Learning
  - Discount factor $\gamma = 0.99$
  - Start epsilon of 1
  - End epsilon of 0.001
  - Epsilon decay of $5^{-4}$
  - 10 bins
  - learning rate of 0.25 and 0.005 for CartPole and Lunar Lander, respectively

# Q Learning

- Q Learning uses a table to store Q values for each state-action pair
- Impractical to store all Q values for a continuous observation space
- Discretization needed
- Hyperparameters used for Q Learning
  - Discount factor $\gamma = 0.99$
  - Start epsilon of 1
  - End epsilon of 0.001
  - Epsilon decay of $5^{-4}$
  - 10 bins
  - learning rate of 0.25 and 0.005 for CartPole and Lunar Lander, respectively

# Q Learning

- Q Learning uses a table to store Q values for each state-action pair
- Impractical to store all Q values for a continuous observation space
- Discretization needed
- Hyperparameters used for Q Learning
  - Discount factor $\gamma = 0.99$
  - Start epsilon of 1
  - End epsilon of 0.001
  - Epsilon decay of $5^{-4}$
  - 10 bins
  - learning rate of 0.25 and 0.005 for CartPole and Lunar Lander, respectively

# Q Learning

- Q Learning uses a table to store Q values for each state-action pair
- Impractical to store all Q values for a continuous observation space
- Discretization needed
- Hyperparameters used for Q Learning
  - Discount factor $\gamma = 0.99$
  - Start epsilon of 1
  - End epsilon of 0.001
  - Epsilon decay of $5^{-4}$
  - 10 bins
  - learning rate of 0.25 and 0.005 for CartPole and Lunar Lander, respectively

# DNQ Algorithm Setup

- Implementation of `ReplayMemory`, which acts as a replay buffer
- Implementation of deep neural network
  - Two hidden layers, each with 128 units
  - ReLU activation function
  - Adam's optimizer
  - Huber loss

# DNQ Algorithm Setup

- Implementation of `ReplayMemory`, which acts as a replay buffer
- Implementation of deep neural network
  - Two hidden layers, each with 128 units
  - ReLU activation function
  - Adam's optimizer
  - Huber loss

# DNQ Algorithm Setup

- Implementation of `ReplayMemory`, which acts as a replay buffer
- Implementation of deep neural network
  - Two hidden layers, each with 128 units
  - ReLU activation function
  - Adam's optimizer
  - Huber loss

# DQN Algorithm

**Algorithm** DQN(episodes, $\alpha, \epsilon, \gamma, C$)

Initialize replay buffer $\mathcal{D}$ with maximum capacity 100000
Initialize policy network $Q$ with random weights $\theta$
Initialize target network $\tilde{Q}$ with random weights $\tilde{\theta}$
**for each** episode in episodes **do**
    Initialize $s$
    Set $t \leftarrow 0$
    $done \leftarrow$ False
    **while** not $done$ **do**
        Choose $a \in \mathcal{A}$ from $s$ using policy network $Q$
        Take action $a$ and observe reward $r$, next state $s'$ and $done$
        Store transition $(s, a, r, s', done)$
        Sample a minibatch of random transitions $(s, a, r, s', done)$ from $\mathcal{D}$
        Set $\hat{y} = \begin{cases} r & \text{if } s \text{ is a terminal state} \\ r + \gamma \max_a \tilde{Q}(s', a; \tilde{\theta}) & \text{otherwise} \end{cases}$
        Perform gradient descent on $(\hat{y} - Q(s, a; \theta))^2$ w.r.t. the policy network parameters $\theta$
        **if** $t \bmod C = 0$ **then**
            $\tilde{Q} \leftarrow Q$
        **end if**
    **end while**
**end for**

- Discount factor $\gamma = 0.99$
- Learning rate 0.003
- Start epsilon of 1
- End epsilon of 0.01
- Epsilon decay of $5^{-4}$
- $\tau = 0.005$
- Default batch size of 64

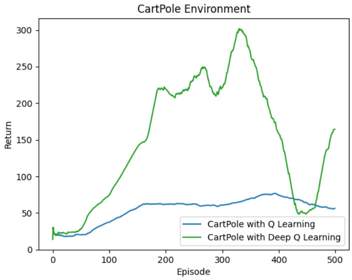
# DQN Algorithm

**Algorithm** DQN(episodes, $\alpha, \epsilon, \gamma, C$)

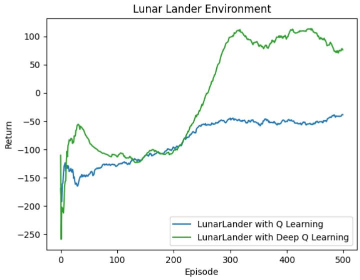
Initialize replay buffer $\mathcal{D}$ with maximum capacity 100000
Initialize policy network $Q$ with random weights $\theta$
Initialize target network $\tilde{Q}$ with random weights $\tilde{\theta}$
**for each** episode in episodes **do**
  Initialize $s$
  Set $t \leftarrow 0$
  *done* $\leftarrow$ False
  **while** not *done* **do**
    Choose $a \in \mathcal{A}$ from $s$ using policy network $Q$
    Take action $a$ and observe reward $r$, next state $s'$ and *done*
    Store transition $(s, a, r, s', done)$
    Sample a minibatch of random transitions $(s, a, r, s', done)$ from $\mathcal{D}$
    Set $\hat{y} = \begin{cases} r & \text{if } s \text{ is a terminal state} \\ r + \gamma \max_a \tilde{Q}(s', a; \tilde{\theta}) & \text{otherwise} \end{cases}$
    Perform gradient descent on $(\hat{y} - Q(s, a; \theta))^2$ w.r.t. the policy network parameters $\theta$
    **if** $t \bmod C = 0$ **then**
      $\tilde{Q} \leftarrow Q$
    **end if**
  **end while**
**end for**

- Discount factor $\gamma = 0.99$
- Learning rate 0.003
- Start epsilon of 1
- End epsilon of 0.01
- Epsilon decay of $5^{-4}$
- $\tau = 0.005$
- Default batch size of 64

# Plots




(a) CartPole




(b) Lunar Lander

Figure: 500 training episodes for Q Learning and DQN. Returns are averaged over the last 100 episodes.

# CartPole

DQN

- Learns much quicker than Q Learning
- Reached an average return of 300 at around 300 episodes
- Unstable learning process: significant drop at episode 350
- Agent is possibly trying to escape a local minimum

Q Learning

- Learns much slower compared to DQN
- Not great returns, even after 500 training episodes
- Agent might be stuck in a local minima

## CartPole

DQN

- Learns much quicker than Q Learning
- Reached an average return of 300 at around 300 episodes
- Unstable learning process: significant drop at episode 350
- Agent is possibly trying to escape a local minimum

Q Learning

- Learns much slower compared to DQN
- Not great returns, even after 500 training episodes
- Agent might be stuck in a local minima

# CartPole

DQN

- Learns much quicker than Q Learning
- Reached an average return of 300 at around 300 episodes
- Unstable learning process: significant drop at episode 350
- Agent is possibly trying to escape a local minimum

Q Learning

- Learns much slower compared to DQN
- Not great returns, even after 500 training episodes
- Agent might be stuck in a local minima

# CartPole

DQN

- Learns much quicker than Q Learning
- Reached an average return of 300 at around 300 episodes
- Unstable learning process: significant drop at episode 350
- Agent is possibly trying to escape a local minimum

Q Learning

- Learns much slower compared to DQN
- Not great returns, even after 500 training episodes
- Agent might be stuck in a local minima

## CartPole

DQN

- Learns much quicker than Q Learning
- Reached an average return of 300 at around 300 episodes
- Unstable learning process: significant drop at episode 350
- Agent is possibly trying to escape a local minimum

Q Learning

- Learns much slower compared to DQN
- Not great returns, even after 500 training episodes
- Agent might be stuck in a local minima

## CartPole

DQN

- Learns much quicker than Q Learning
- Reached an average return of 300 at around 300 episodes
- Unstable learning process: significant drop at episode 350
- Agent is possibly trying to escape a local minimum

Q Learning

- Learns much slower compared to DQN
- Not great returns, even after 500 training episodes
- Agent might be stuck in a local minima

## CartPole

DQN

- Learns much quicker than Q Learning
- Reached an average return of 300 at around 300 episodes
- Unstable learning process: significant drop at episode 350
- Agent is possibly trying to escape a local minimum

Q Learning

- Learns much slower compared to DQN
- Not great returns, even after 500 training episodes
- Agent might be stuck in a local minima

## CartPole

DQN

- Learns much quicker than Q Learning
- Reached an average return of 300 at around 300 episodes
- Unstable learning process: significant drop at episode 350
- Agent is possibly trying to escape a local minimum

Q Learning

- Learns much slower compared to DQN
- Not great returns, even after 500 training episodes
- Agent might be stuck in a local minima

# CartPole

DQN

- Learns much quicker than Q Learning
- Reached an average return of 300 at around 300 episodes
- Unstable learning process: significant drop at episode 350
- Agent is possibly trying to escape a local minimum

Q Learning

- Learns much slower compared to DQN
- Not great returns, even after 500 training episodes
- Agent might be stuck in a local minima

# Lunar Lander

DQN

- DQN outperforms Q Learning
- Unstable learning process

Q Learning

- Slow but stable learning
- No drastic change in average return. Instead, the average return slowly increases

# Lunar Lander

DQN

- DQN outperforms Q Learning
- Unstable learning process

Q Learning

- Slow but stable learning
- No drastic change in average return. Instead, the average return slowly increases

# Lunar Lander

DQN

- DQN outperforms Q Learning
- Unstable learning process

Q Learning

- Slow but stable learning
- No drastic change in average return. Instead, the average return slowly increases

# Lunar Lander

DQN

- DQN outperforms Q Learning
- Unstable learning process

Q Learning

- Slow but stable learning
- No drastic change in average return. Instead, the average return slowly increases

# Lunar Lander

DQN

- DQN outperforms Q Learning
- Unstable learning process

Q Learning

- Slow but stable learning
- No drastic change in average return. Instead, the average return slowly increases

# Lunar Lander

DQN

- DQN outperforms Q Learning
- Unstable learning process

Q Learning

- Slow but stable learning
- No drastic change in average return. Instead, the average return slowly increases

## Conclusion

DQN

- Learns much faster in both games, but the agent exhibited some instability during the learning process

Q Learning

- Learns much slower, but more steadily

**The choice of algorithm depends on the specific application and trade-offs between speed and stability.**

# Conclusion

DQN

- Learns much faster in both games, but the agent exhibited some instability during the learning process

Q Learning

- Learns much slower, but more steadily

The choice of algorithm depends on the specific application and trade-offs between speed and stability.

## Conclusion

DQN

- Learns much faster in both games, but the agent exhibited some instability during the learning process

Q Learning

- Learns much slower, but more steadily

The choice of algorithm depends on the specific application and trade-offs between speed and stability.

# Conclusion

DQN

- Learns much faster in both games, but the agent exhibited some instability during the learning process

Q Learning

- Learns much slower, but more steadily

The choice of algorithm depends on the specific application and trade-offs between speed and stability.

# Conclusion

DQN

- Learns much faster in both games, but the agent exhibited some instability during the learning process

Q Learning

- Learns much slower, but more steadily

**The choice of algorithm depends on the specific application and trade-offs between speed and stability.**

## Future Work

Deep deterministic policy gradient (DDPG)

- Combines Q Learning with policy gradients to learn a deterministic policy directly
- This has shown to be effective in continuous action spaces and could address some of the instability issues observed in DQN.

## Future Work

Deep deterministic policy gradient (DDPG)

- Combines Q Learning with policy gradients to learn a deterministic policy directly
- This has shown to be effective in continuous action spaces and could address some of the instability issues observed in DQN.

# Future Work

Deep deterministic policy gradient (DDPG)

- Combines Q Learning with policy gradients to learn a deterministic policy directly
- This has shown to be effective in continuous action spaces and could address some of the instability issues observed in DQN.