03: Reinforcement Learning

Antorweep Chakravorty

Topics

- Markov Decision Process
- Policy and Value Functions
- Q Learning
- Deep Q-Learning
- GridWorld
- Q Learning with Experience Replay
- Q-Learning with Experience Replay and a Target Network

Markov Decision Process (MDP)

- Mathematical framework for modeling decision-making problems
- Outcomes are partly random and partly controllable

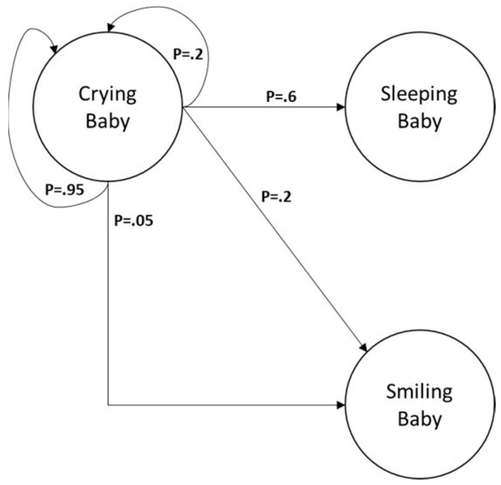
The Markov Property

- The *Markov property* states that to execute decisions, all necessary information is captured in the current state without dependence on history

$$\mathrm{P}[S_{t+1} \mid S_t] = \mathrm{P}[S_{t+1} \mid S_1, \dots, S_t]$$

Where **P** is the probability of the next state $\mathbf{S_{t+1}}$ given state $\mathbf{S_t}$

Markov Process / Chain

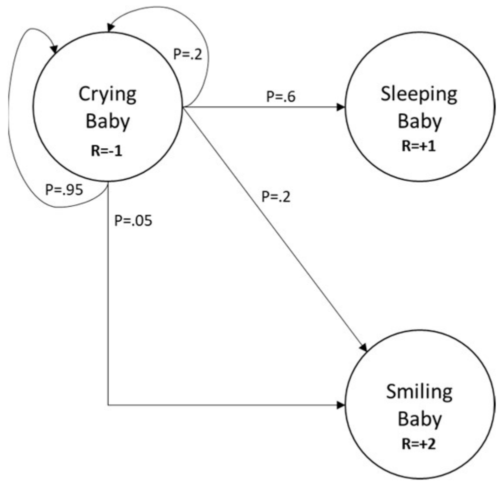


- Describes the state transition probabilities.

$$P_{ss'} = P[S_{t+1} = s' \mid S_t = s]$$

Where $\mathbf{P_{ss'}}$ is the probability of transitioning to state **s'** given state **s**

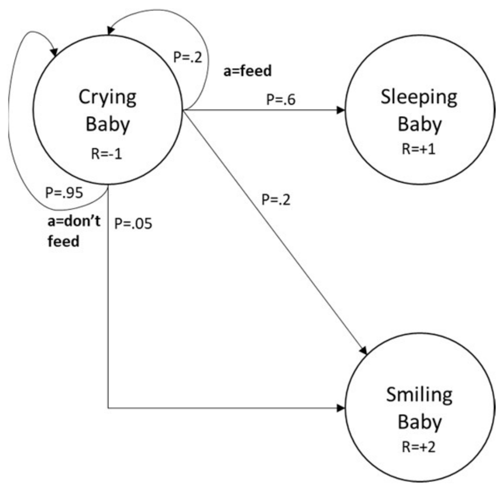
Markov Reward Process (MRP)



$$\mathrm{R}_s = \mathrm{E}[R_{t+1} \mid S_t = s]$$

Where $\mathbf{R_s}$ is the expected reward for being at state **s**

Markov Decision Process (MDP)



- Introduces a set of actions **A**
- Allows control over the MRP

$P_s^a = P[S_{t+1} = s' \mid S_t = s, A_t = a]$

$\mathrm{R}_s^a = \mathrm{E}[R_{t+1} \mid S_t = s, A_t = a]$

The objective of MDP is to choose the correct sequence of actions from a state **s** that leads toward maximizing the total discounted reward from time-step **t**

The total discounted reward is denoted by $\mathbf{G_t}$ (called **return**)

The rewards are discounted to address uncertainty of future rewards. Higher weights are given to actions in near future.

The discount factor is denoted by $\gamma \in [0, 1]$

$G_t = \gamma^0 R_{t+1} + \gamma^1 R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

Policy Functions

- Policy (**π**) functions describe the probability distributions over all actions A for a given state. In other words, it maps a state s to the probabilities of each action
- Based on the given policy function an action that maximizes the return (**G_t**) could be chosen

$$\pi(a|s) = \mathrm{P}[A = a \,|S_t = s]$$

Value Functions

- Value (**v**) functions describe the long-term value or **return** of a state based on its available choices of actions.

Bellman Expectation Equation

$$v(s) = \mathrm{E}[G_t |\, S_t = s]$$

$$= \mathrm{E}[\gamma^0 R_{t+1} + \gamma^1 R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots |S_t = s]$$

$$= \mathrm{E}[R_{t+1}\gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots |S_t = s]$$

$$= \mathrm{E}[R_{t+1} + G_{t+1}|S_t = s]$$

$$= \mathrm{E}[R_{t+1} + \gamma v(S_{t+1} = s)|S_t = s] \longrightarrow \text{Equation for MRP}$$

Value Functions

- In RL the choice of transitioning to a state or choosing an action depends on a policy (not the same as policy functions).
  - In the bandit problems state that produced the maximum average payout was chosen as the policy for transitioning to the next state
  - Alternatively, the epsilon greedy or softmax strategies were also chosen to choose appropriate actions
- Agents in RL act based on certain policy it is following
- Therefore, value functions are defined with respect to policies

Value Functions

- State Value Functions

$v_\pi(s) = \mathrm{E}_\pi[G_t | S_t = s]$

$= E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s]$

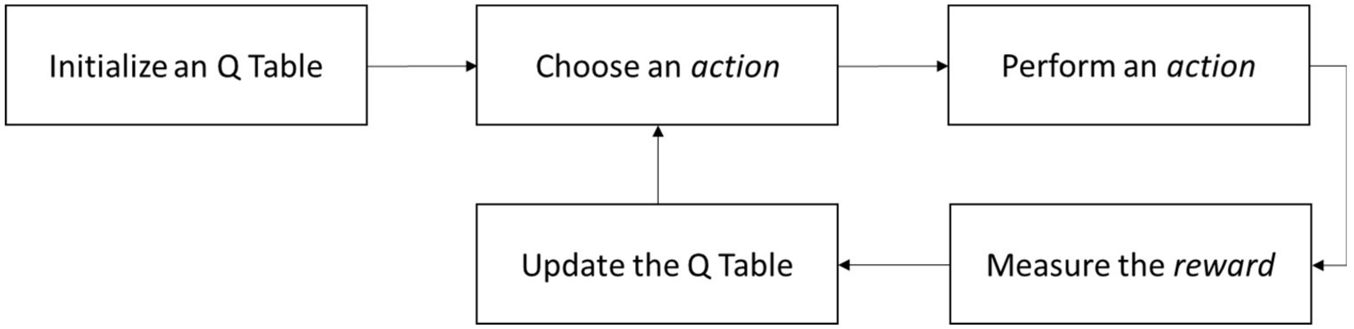
- Action Value Functions

$q_\pi(s, a) = \mathrm{E}_\pi[G_t | S_t = s, A_t = a]$

$= E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a]$

Q Learning

- Used for solving control task problems
- Learns a value function called Q-functions
- It accepts a state and predicts how valuable all the possible actions are
- These value predictions are called Q values
- Q-Learning is an **off-policy** algorithm. off-policy algorithms mean that the choice of policy does not affect the ability to learn accurate values. It determines the optimal policy regardless of the policy used for making action choices.
- Q-Learning is a **model-free** algorithm. It does not require a model of the environment to learn the optimal policy and uses trial and error to understand the working of the environment



Q Learning

$Q_\pi(s,a) = \mathrm{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s, A_t = a]$

Given

$G_t = R_{t+1} + \gamma^1 R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

$\Rightarrow Q_\pi(s,a) = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a]$

Q Learning

- WHITEBOARD

Deep Q-Learning

- A Deep Neural Network that can learn and approximate the *Q learning rule*
- Given: $Q_\pi(s, a) = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a]$
- An ANN that can update the Q values at each epoch without using a table

$$Q_\pi(s, a | S_t = s, A_t = a) = Q_\pi(s, a) + \alpha[R_{t+1} + \gamma \max Q(s, a | S_{t+1} = s,\ A_{t+1} = a) - Q_\pi(s, a)]$$

Here

$Q_\pi(s, a | S_t = s, A_t = a)$ is the current Q value for a given state and action

α is the learning rate

$R_{t+1}$is the reward for taking an action in the current state

γ is the discount factor

$Q(s, a | S_{t+1} = s,\ A_{t+1} = a)$ is the Q value for the next state and action

Grid World

- An **environment** to introduce Q Learning
- A **nXn** grid that contains
    - A **goal** that signifies win
    - A **pit** that signifies loss
    - A **wall** that limits traversal
    - **Open spaces** that allow traversal in the grid
- Objective: a **player** needs to *traverse through the grid* and *reach the goal using the shortest path*
- Three versions of the environment are available
    - *Static:* The positions of the goal, pit, wall, and player are fixed and do not change
    - *Player*: The positions of the goal, pit, and wall are fixed. The position of the player randomly changes each time the game is initialized
    - *Random*: The positions of the goal, pit, wall, and the player randomly changes each time the game is initialized.

GridWorld

- WHITEBOARD

GridWorld

- CODE

Agent

- A deep ANN
- Contains
  - One input layer <- *receives the state info from the environment*
  - Two hidden layers
  - One output layer <- *outputs the expected Q value for each action*
- Executed over multiple epochs
  - In each epoch the agent executes one play
  - A play consists of multiple moves
  - A play is over if the agent can reach the goal, pit, or undertakes the maximum number of allowed moves
  - After each move the agent updates the Q values

Agent

- WHITEBOARD

Agent

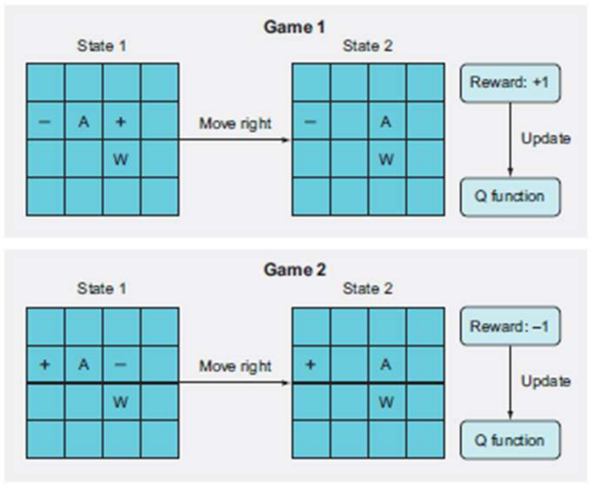
- CODE

Catastrophic Forgetting

- An important issue with gradient descent-based training methods in **online training**
- Online training is the mechanism of backpropagating after each move or feed-forward
- When two states are very similar and yet lead to very different outcomes

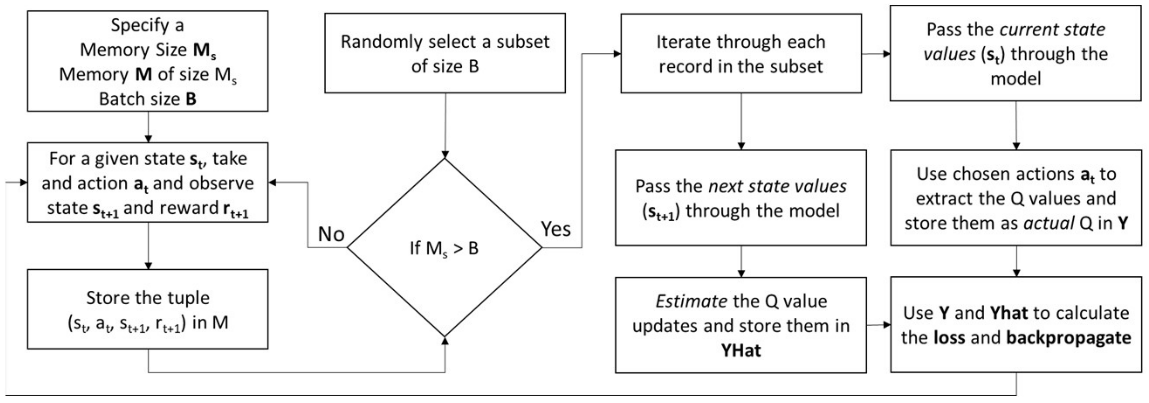
Catastrophic Forgetting

- WHITEBOARD



Experience Replay

- Addresses the issue of catastrophic forgetting
- Introduces **batch updating** in an online learning scheme



Specify a Memory Size $\mathbf{M_s}$ Memory **M** of size M_s Batch size **B**
For a given state $\mathbf{s_t}$, take and action $\mathbf{a_t}$ and observe state $\mathbf{s_{t+1}}$ and reward $\mathbf{r_{t+1}}$
Store the tuple (s_t, a_t, s_{t+1}, r_{t+1}) in M
Randomly select a subset of size B
If $M_s > B$
No
Yes
Iterate through each record in the subset
Pass the *next state values* ($\mathbf{s_{t+1}}$) through the model
Estimate the Q value updates and store them in **YHat**
Pass the *current state values* ($\mathbf{s_t}$) through the model
Use chosen actions $\mathbf{a_t}$ to extract the Q values and store them as *actual* Q in **Y**
Use **Y** and **Yhat** to calculate the **loss** and **backpropagate**

Experience Replay

Experience Replay

- WHITEBOARD

Experience Replay

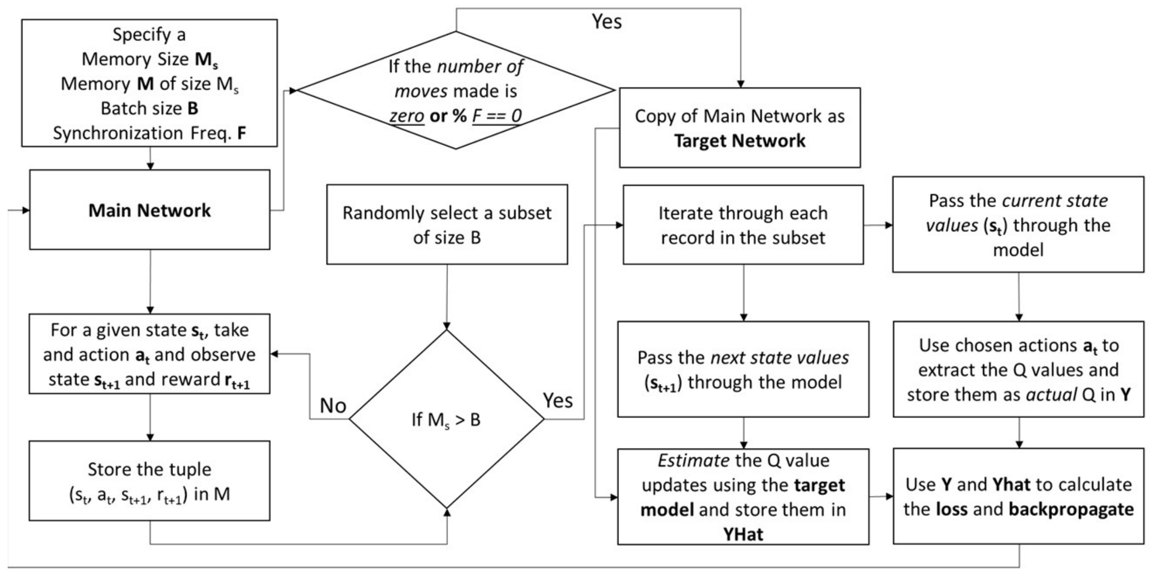
- CODE

Improving Stability

- Cases where rewards are **sparse**. Significant rewards are received upon winning or losing and in-between steps do not contribute significantly to the learning process.
- Might lead to oscillating behavior

Improving Stability

- Maintains a **second model** called the **target network**
- Initially the target network is a *copy of the **main model***
- However, during the training process the **weights and bias** in the target network **lags** behind the main network
- The algorithm is like *experience relay*, however, the estimated Q values used to backpropagate the *main model* are calculated based on the target model
- The *weights and bias* of the *target model* get synchronized with the *main model* after a period (thus the lag)


Specify a
Memory Size $\mathbf{M_s}$
Memory **M** of size M_s
Batch size **B**
Synchronization Freq. **F**
Main Network
If the *number of moves* made is *zero* **or %** *F == 0*
Yes
Copy of Main Network as **Target Network**
Randomly select a subset of size B
For a given state $\mathbf{s_t}$, take and action $\mathbf{a_t}$ and observe state $\mathbf{s_{t+1}}$ and reward $\mathbf{r_{t+1}}$
No
If $M_s > B$
Yes
Store the tuple (s_t, a_t, s_{t+1}, r_{t+1}) in M
Iterate through each record in the subset
Pass the *current state values* ($\mathbf{s_t}$) through the model
Pass the *next state values* ($\mathbf{s_{t+1}}$) through the model
Use chosen actions $\mathbf{a_t}$ to extract the Q values and store them as *actual* Q in **Y**
Estimate the Q value updates using the **target model** and store them in **YHat**
Use **Y** and **Yhat** to calculate the **loss** and **backpropagate**

Improving Stability

Improving Stability

- CODE