Reinforcement Learning

Sudip Bhujel

University of Kentucky

October 4, 2024

Supervised Learning Vs Unsupervised Learning Vs RL

- Supervised Learning
  - Given: Labeled data, correct output is provided for each input
  - Goal: Minimize the difference between predicted o/p and actual o/p
  - Example: Cat and dog image classification
- Unsupervised Learning
  - Given: Unlabeled data
  - Goal: Identify the underlying structure, such as cluster and associations
  - Example: Grouping region by similar weather
- Reinforcement Learning
  - Given: Possible actions/interactions and Environment
  - Goal: Maximize cumulative reward by learning policy (Which action to take in current state)
  - Example: Training autonomous agent to fly helicopter

Terminology

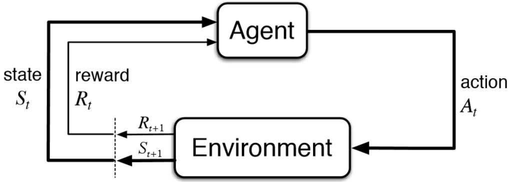


Figure 1: Basic RL in action (From the internet)

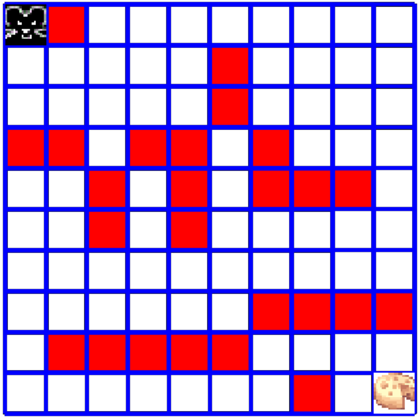


Figure 2: Maze Problem (From the internet)

Rewards

- Scalar Value: Numerical feedback (positive, negative, or zero)
- Immediate Feedback: Given immediately after an action is taken
- Objective: The agent aims to maximize cumulative rewards over time

Definition (Reward hypothesis)

All goals can be described by the maximisation of expected cumulative reward

Rewards (1)

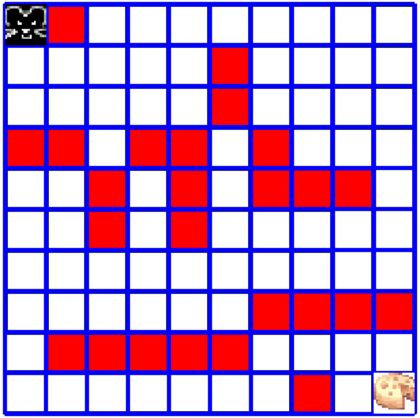


Figure 3: Maze Problem (From the internet)

Rewards

- Positive reward (**+10 points**) for the goal
- Negative reward (**-5 points**) for red blocks
- Neutral reward (**-1 points**) for non-goal standard movements, *discourage* taking too many steps unnecessarily

History and State

- The history of sequence of observations, actions, and rewards

$$H_t = O_1, R_1, A_1, ..., A_{t-1}, O_t, R_t$$

- It contains all the information the agent has accumulated and can be very complex
- State is a summary of the history and contains relevant information needed for decision-decision making
- Reduces the complexity of the entire history into a manageable form
- Formally, state is a function of history:

$$S_t = f(H_t)$$

State

- Environment state (S_t^e):
  - Whatever data the environment uses to pick the next observation/reward
  - Usually not visible to the agent
- Agent state (S_t^a):
  - Whatever information agent uses to pick next action
  - It can be any function of history:

$$S_t^a = f(H_t)$$

Information State

An information state (a.k.a. Markov state) contains all useful information from the history.

Definition

A state S_t is Markov if and only if

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

- The future is independent of the past given the present
- Once the state is known, the history may be thrown away
- i.e. The state is a sufficient statistic of the future

State (1)

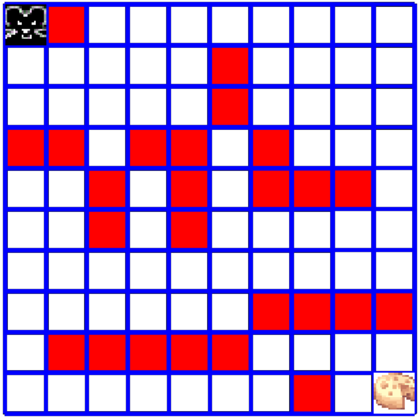


Figure 4: Maze Problem (From the internet)

State

- Each Square
- Start (Top left corner) state
- Termination (Bottom right corner) state

State Transition Matrix

For a Markov state s and successor state s', the *state transition probability* is defined by,

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

State Transition Matrix P defines transition probability from all state s to all successor state s'

$$P = \begin{array}{cc} & \mathit{to} \\ \mathit{from} & \begin{bmatrix} P_{11} & \cdots & P_{1n} \\ \vdots & & \\ P_{n1} & \cdots & P_{nn} \end{bmatrix} \end{array}$$

Where,

$$\sum_{s'} P_{ss'} = 1$$

This ensures that the process always transitions to some state.

State Transition Matrix (Intuition)

- We have three states A, B, and C
- State transition matrix,

$$P = \begin{bmatrix} 0.6 & 0.3 & 0.1 \\ 0.2 & 0.7 & 0.1 \\ 0.3 & 0.4 & 0.3 \end{bmatrix}$$

$$= \begin{bmatrix} P_{AA} & P_{AB} & P_{AC} \\ P_{BA} & P_{BB} & P_{BC} \\ P_{CA} & P_{CB} & P_{CC} \end{bmatrix}$$

Environments

Fully observability: agent directly observes the environment state

$$O_t = S_t^a = S_t^e$$

- Agent State = environment state = information state
- Formally, this is a Markov decision process (MDP)

Partial observability: agent indirectly observes environment:

- Robot navigation in a foggy environment
- Formally this is a Partially Observable Markov Decision Process (POMDP)

Return

- It is total accumulated reward that an agent receives from time step t onward
- Used to evaluate the total reward that the agent expects to receive starting from time step t

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} \ldots$$

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

where,

- R_{t+k+1} is the reward received at time step $t + k + 1$
- $\gamma \epsilon [0, 1]$ is a discount factor, helps to determine whether to take immediate reward or not
  - $\gamma = 0$, takes immediate rewards, ignoring all future rewards
  - $\gamma = 1$, future rewards are valued equally with immediate rewards

Components of an RL Agent

- An RL agent may include one or more of these components
  - Policy
  - Value function
  - Model

Policy

- A policy is a strategy that maps states to actions
- Using policy agent takes an action
- It can be deterministic or stochastic
- Deterministic policy: a specific action is chosen for each state

$$a = \pi(s)$$

- Stochastic policy: actions are chosen according to a probability distribution

$$\pi(a \mid s) = \mathbb{P}[A_t = a \mid S_t = s]$$

Policy (1)

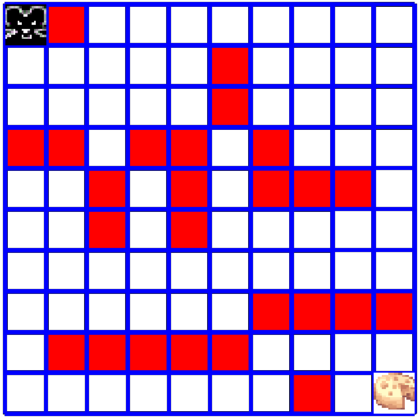


Figure 5: Maze Problem (From the internet)

Policy

- At start, agent takes **down** action to avoid obstacle

Value Function

- Estimates the expected return/reward
- Used to evaluate the goodness/badness of states

$$
\begin{aligned}
V_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \ldots\right) \mid S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma V_\pi(S_{t+1}) \mid S_t = s\right]
\end{aligned}
$$

Value Function (1)

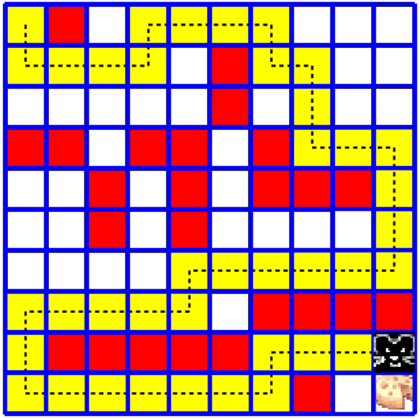


Figure 6: Maze Problem (From the internet)

Value Function

- Estimates expected rewards for each state (how valuable it is to be in a particular state)
- $V(s) = R_{t+1} + \gamma V(s\prime)$

Agent is one step away from the cookie

- Here,
  $R_{t+1} = +10$ (Immediate reward),
  $V(s\prime) = 0$ (No expected future reward) and
  $\gamma = 0.9$
- $\mathbf{V(s)} = 10 + 0.9 \times 0 = +\mathbf{10}$

Value Function (2)

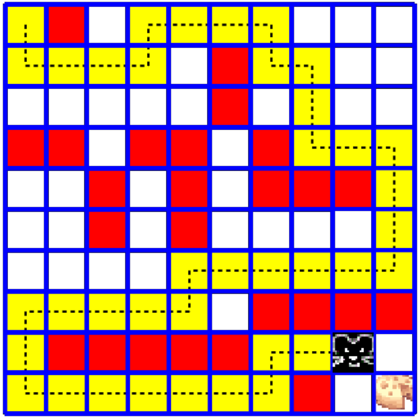


Figure 7: Maze Problem (From the internet)

Value Function

- Estimates expected rewards for each state (how valuable it is to be in a particular state)
- $V(s) = R_{t+1} + \gamma V(s\prime)$

Agent is two step away from the cookie

- Here,
  $R_{t+1} = -1$ (cost of the first step),
  $V(s\prime) = 10$ (the value of the next state, which is one step away from the cookie),
  $\gamma = 0.9$
- $V(s) = -1 + 0.9 \times 10 = -1 + 9 = \mathbf{+8}$

Model

- A model predicts what the environment will do next
- We have two model **Transition model** and **Reward model**
- Transition model predicts the next state P

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

Probability of transitioning to state s' from state s after taking action a

- Reward model predict the next reward R

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

Expected reward received after taking action a in state s

Markov Process

- A type of *stochastic process* where future state depends only on the present state and not on the sequence of events that preceded it (Markov Property)
- Process are **memoryless**

Definition

A Markov process (or Markov chain) is defined as a tuple $\langle S, P \rangle$, where

- S is a finite set of states $S_1, S_2, S_3, \ldots$
- P is a *state transition probability matrix*

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

Markov Reward Process

Markov Reward Process = Markov Process + Value

Definition

A Markov process (or Markov chain) is defined as a tuple $\langle S, P, R, \gamma \rangle$, where

- S is a finite set of states $S_1, S_2, S_3, \ldots$
- P is a *state transition probability matrix*

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

- R is Reward function, $R_s = \mathbb{E}[R_{t+1} \mid S_t = s]$
- γ is discount factor, $\gamma \epsilon [0, 1]$

Bellman Equation for MRPs

Value function can be decomposed into to parts:

- immediate reward R_{t+1}
- discounted value of successor state $\gamma v(S_{t+1})$

$$
\begin{aligned}
v(s) &= \mathbb{E}\left[G_t \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \ldots\right) \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right]
\end{aligned}
$$

And, $\mathbb{E}(x_i) = \sum P(x_i)x_i$

$$v(s) = R_s + \gamma \sum_{s' \epsilon S} P_{ss'} v(s')$$

Solving Bellman Equation

We can express Bellman Equation in matrix form as,

$$v = R + \gamma P v$$

where v is column vector with one entry per state

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} R_1 \\ \vdots \\ R_n \end{bmatrix} + \gamma \begin{bmatrix} P_{11} & \cdots & P_{1n} \\ \vdots & & \\ P_{1n} & \cdots & P_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}$$

We can solve this equation directly as,

$$\begin{aligned} v &= R + \gamma P v \\ (1 - \gamma P) v &= R \\ v &= (1 - \gamma P)^{-1} R \end{aligned}$$

Markov Decision Process

Markov Decision Process = Markov Reward Process + Action

Definition

A Markov process (or Markov chain) is defined as a tuple $\langle S, A, P, R, \gamma \rangle$, where

- S is a finite set of states $S_1, S_2, S_3, ...$
- A is a finite set of actions
- P is a *state transition probability matrix*

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

- R is Reward function, $R_s^a = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$
- γ is discount factor, $\gamma \epsilon [0, 1]$

Policies

Definition

A *policy* π is a distribution over actions given states,

$$\pi(a \mid s) = P(A_t = a \mid S_t = s)$$

- MDP policies depend on the current state (not the history)
- Given an MDP $M = \langle S, A, P, R, \gamma \rangle$ and a policy π
- Can be written as,

$$M = \langle S, A, P^{\pi}, R^{\pi}, \gamma \rangle$$

where

$$P^{\pi}_{ss'} = \sum_{a \epsilon A} \pi(a \mid s) P^{a}_{ss'}$$

$$R^{\pi}_{s} = \sum_{a \epsilon A} \pi(a \mid s) R^{a}_{s}$$

Value Function

State-Value Function

A *state-value* $V_\pi(s)$ of an MDP is the expected return starting from state s, and then following policy π thereafter,

$$V_\pi = \mathbb{E}_\pi[G_t | S_t = s]$$

Action-Value Function

A *state-value* $q_\pi(s, a)$ of an MDP is the expected return starting from state s, taking action a and then following policy π thereafter,

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

Bellman Expectation Equation for V_π

$$V_\pi(s) = \sum_{a \epsilon A} \pi(a \mid s) q_\pi(s, a)$$

We have Bellman Equation for value function as,

$$q_\pi(s, a) = R_s^a + \gamma \sum_{s' \epsilon S} P_{ss'}^a v_\pi(s')$$

So,

$$V_\pi(s, a) = \sum_{a \epsilon A} \pi(a \mid s) \left(R_s^a + \gamma \sum_{s' \epsilon S} P_{ss'}^a v_\pi(s') \right)$$

And,

$$q_\pi(s, a) = R_s^a + \gamma \sum_{s' \epsilon S} P_{ss'}^a \sum_{a' \epsilon A} \pi(a' \mid s') q_\pi(s', a')$$

Optimal Value Function

Definition

The *optimal state-value* function $v_*(s)$ is the highest expected return starting from state s, assuming the agent follows optimal policy π from that point onward.

$$v_*(s) = \max_{\pi} v_\pi(s)$$

The *optimal action-value* function $q_*(s)$ is the highest expected return starting from state s, taking action a and assuming the agent follows optimal policy π from that point onward.

$$q_*(s, a) = \max_{\pi} q_\pi(s, a)$$

- An MDP is "solved" when we know the optimal value function

Bellman Optimality Equation

The value of a state $v_*(s)$ under the optimal policy is simply the value of taking the best action in that state

$$v_*(s) = \max_a q_*(s, a)$$

And, our optimal policy,

$$q_*(s, a) = R_s^a + \gamma \sum_{s' \epsilon S} P_{ss'}^a v_*(s')$$

Finally,

$$v_*(s) = \max_a \left(R_s^a + \gamma \sum_{s' \epsilon S} P_{ss'}^a v_*(s') \right)$$

And,

$$q_*(s, a) = R_s^a + \gamma \sum_{s' \epsilon S} P_{ss'}^a \max_a q_*(s', a')$$

Optimal Policy

Definition

The policy that maximizes the expected cumulative reward for an agent starting from any state. Or, the best strategy for the agent to follow in order to achieve the highest possible long-term award.

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'(s)}$$

An optimal policy can be found by maximising over $q_*(s, a)$,

$$\pi_*(a \mid s) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in \mathcal{A}} q_*(s, a) \\ 0 & \text{otherwise} \end{cases}$$

Partially Observable MDP

MDP with hidden states

Definition

A POMDP is defined as a tuple $\langle S, A, O, P, R, Z, \gamma \rangle$, where

- S is a finite set of states $S_1, S_2, S_3, \ldots$
- A is a fine set of actions
- O is a fine set of observations
- P is a *state transition probability matrix*, $P^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$
- R is Reward function, $R^a_s = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$
- Z is an observation function

$$Z^a_{s'o} = \mathbb{P}[O_{t+1} = o \mid S_{t+1} = s', A_t = a]$$

- γ is discount factor, $\gamma \epsilon [0, 1]$

DP for solving Bellman Optimality Equation

We can use dp to solve the MDP,

- Policy Iteration
- Value Iteration

Policy Iteration

- Policy Evaluation
  - Compute value function $v_\pi(s)$ for a given policy π

$$V_\pi(s,a) = \sum_{a \epsilon A} \pi(a \mid s) \left(R_s^a + \gamma \sum_{s' \epsilon S} P_{ss'}^a v_\pi(s') \right)$$

  - Repeat until value converges, difference between value function at iteration (k+1) and (k) is less than threshold

$$\left| V_\pi^{(k+1)}(s) - V_\pi^{(k)}(s) \right| < \epsilon$$

- Policy Improvement
  - *Act greedily*, for each state s, update the policy by choosing the action that maximizes the expected return

$$\pi' = greedy(v_\pi)$$

- Repeat, until no improvement in the policy

Value Iteration

- Value Function Update
  - The value of each state $v(s)$ is updated by taking the maximum expected return over all possible actions

$$v_{k+1}(s) = \max_a \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + \gamma v_k(s') \right]$$

  - Repeat until value converges, difference between value function at iteration (k+1) and (k) is less than threshold

$$\left| V_\pi^{(k+1)}(s) - V_\pi^{(k)}(s) \right| < \epsilon$$

- Policy Extraction
  - *Act greedily*, for each state s, update the policy by choosing the action that maximizes the expected return

$$\pi' = greedy(v_\pi)$$

- Repeat, until no improvement in the policy

Motivation

- Unknown and complex environment, eg. Self driving car, stock trading
- Avoid the cost of Building a Model, eg. Alpha Go
- Learning from experience, eg. Personal Assistant

Monte-Carlo Reinforcement Learning

- Learn directly from episodes of experience so needs clear start and termination state
- Learns from complete episodes: no bootstrapping
- Uses simplest possible idea: value = mean return
- Caveat: can only apply MC to episodic MDPs
  - All episodes must terminate

Monte-Carlo Policy Evaluation

- Goal: learn v_π from episodes of experience under policy π

$$S_1, A_1, R_2, ..., S_k \sim \pi$$

- Return: the total discounted reward:

$$G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T$$

- Uses *epirical mean* return instead of *expected* return as policy evaluation

Monte-Carlo Policy Evaluation

First Visit

- To evaluate state s
- The first time-step t that s is visited in an episode,
  - Increment counter $N(s) \leftarrow N(s) + 1$
  - Increment total return $S(s) \rightarrow S(s) + G(t)$

Every-Visit

- To evaluate state s
- Every time-step t that s is visited in an episode,

Now, once enough episodes have been observed,

- Value is estimated by mean return $V(s) = S(s)/N(s)$
- By law of large numbers, $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$ (More sample)

Waiting to compute average over many episodes

Incremental Monte-Carlo Updates

- Update $V(s)$ incrementally after episode $S_1, A_1, R_2, ..., S_T$
- For each state S_t with return G_t

$$N(S_t) \leftarrow N(S_t) + 1$$
$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$$

- In non-stationary problems, it can be useful to track a running mean, i.e. forget old episodes

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

Note: $(G_t - V(S_t))$ represents the difference between the **actual return** *and the* **current estimated value** *of a state, called Prediction Error/ Update Signal.*

Temporal Difference Learning

- TD learns from *incomplete episodes*, by *bootstraping*

Simplest temporal-difference learning: TD(0)

- Update value $V(S_t)$ toward *estimated* return $R_{t+1} + \gamma V(S_{t+1})$

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

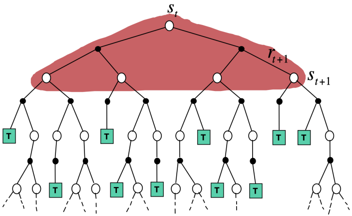
- $R_{t+1} + \gamma V(S_{t+1})$ is called the *TD target*
- $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the *TD error*
- In $TD(\lambda)$, λ controls how far into the future the updates look, $TD(0)$ (one-step look-ahead) and Monte Carlo (complete episode look-ahead)

Learning Methods in RL

- Monte Carlo methods
- Temporal Difference (TD) Learning

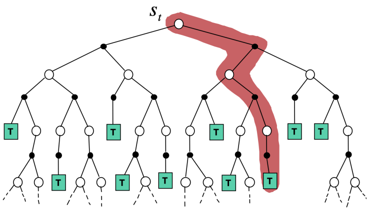
Dynamic Programming

$V(S_t) \leftarrow \mathbb{E}_\pi \left[R_{t+1} + \gamma V(S_{t+1})\right]$



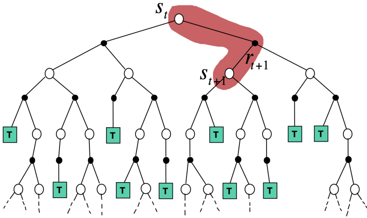
Monte Carlo Learning

$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t - V(S_t)\right)$



TD Learning

$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$



Exploration v/s Exploitation

Exploration

- Try out different actions to gather information about environment
- The agent may potentially find better actions which lead to higher reward in the future
- Trying different restaurants sometime

Exploitation

- Choose best-known action based on its current knowledge
- The agent try to maximize the immediate rewards using learned policy
- Going to the favorite restaurants
- You might miss out on a restaurant you've never tried before

ϵ-greedy

- Simple and effective idea to balance exploration v/s exploitation trade-off

For m possible actions,

- With probability $1 - \epsilon$, choose greedy action
- With probability ϵ, choose random action

$$\pi(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{m}, & \text{if } a = \arg\max_a Q(s, a) \\ \frac{\epsilon}{m}, & \text{otherwise} \end{cases}$$

Example, $\epsilon = 0.1$,

- 90% of the time, agent will take best-known action
- 10% of the time, agent will try random action

On-policy and off-policy learning

On-policy learning

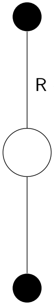
- Policy used for learning $=$ Policy used for acting
- The agent **evaluates and improves** the policy that is currently followed
- Eg. SARSA

Off-policy learning

- Policy used for learning $\neq$ Policy used for acting
- The agent can **evaluate and improve** the target policy while following a different policy to gather data
- Eg. Q-learning

SARSA (State-Action-Reward-State-Action)

- On-policy TD learning



$$Q(s, a) \leftarrow Q(s, a) + \alpha\,[R_{t+1} + \gamma Q(s', a') - Q(s, a)]$$

$$Q(s, a) \leftarrow Q(s, a) + \alpha \times \text{TD Error}$$

Q Learning

- Off-policy TD learning

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R_{t+1} + \gamma \max_{a'} Q(s\prime, a\prime) - Q(s,a) \right]$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha \times \text{TD Error}$$

Large-Scale Reinforcement Learning

- We have represented value function by a *lookup table*
  - Every state s has an entry $V(s)$
  - Or every state-action par s, a has an entry $Q(s, a)$
- Problem with large MDPs:
  - There are too many states and/or actions to store in memory
  - It is too slow to learn the value of each state individually
- Solution for large MDPs would use Function approximators
  - Linear combinations of features
  - Neural network
  - Decision tree
  - Nearest neighbour
  - ...

Value Function Approximation

- Solution for large MDPs:
  - Estimate value function with *function approximation*

    $$\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$$

    $$\hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$$

    where **w** represents the parameters of the approximation (such as the weights of a neural network or linear model)
  - *Generalize* unseen states from seen states
  - Use MC or TD learning to *update* **w**
- Incremental Methods
- Batch Methods

Gradient Descent

- Let $J(\mathbf{w})$ be differential function of the parameter vector $\mathbf{w}$
- Define the gradient of $J(w)$,

$$\nabla_w J(w) = \begin{pmatrix} \frac{\partial J(w)}{\partial w_1} \\ \frac{\partial J(w)}{\partial w_2} \\ \vdots \\ \frac{\partial J(w)}{\partial w_n} \end{pmatrix}$$

To find a local minimum of $J(w)$

- Adjust w in the direction of -ve gradient

$$\Delta w = -\frac{1}{2}\alpha \nabla_w J(w)$$

Where, α is a step-size parameter

Value Function Approx. using Stochastic Gradient Descent

- Goal: to find the parameter vector w that minimizes the mean-squared error between the true value function $v_\pi(s)$ and the approximate value function $\hat{v}(s, w)$

$$J(w) = \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, w))^2 \right]$$

- Gradient descent finds a local minimum

$$\begin{aligned} \Delta w &= -\frac{1}{2} \alpha \nabla_w J(w) \\ &= \alpha \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, w)) \nabla_w \hat{v}(S, w) \right] \end{aligned}$$

- Stochastic gradient descent samples randomly over dataset (or states)

$$\Delta w = \alpha (v_\pi(S) - \hat{v}(S, w)) \nabla_w \hat{v}(S, w)$$

Incremental Prediction Algorithms

- In supervised learning we will have target or actual o/p to compare prediction and find error
- But in RL there is no supervisor, only rewards
- So, we substitute a *target* for $v_\pi(s)$
  - For MC, the target is the return G_t

$$\Delta w = \alpha(G_t - \hat{v}(S, w))\nabla_w \hat{v}(S, w)$$

  - For TD(0)), the target is the TD target

$$\Delta w = \alpha(R_{t+1} + \gamma \hat{v}(S_{t+1}, w) - \hat{v}(S, w))\nabla_w \hat{v}(S, w)$$

Batch Reinforcement Learning

- Gradient descent didn't make the maximum use of the experiences
- Batch methods seek to find the best fitting value function

Stochastic Gradient Descent with Experience Replay

Given experience consisting of $\langle state, value \rangle$ pairs

$$\mathcal{D} = \langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, \ldots \langle s_T, v_T^\pi \rangle$$

Repeat:

1. Sample state, value from experience

$$\langle s_1, v_1^\pi \rangle \sim \mathcal{D}$$

2. Apply SGD update

$$\Delta w = \alpha(v_\pi(S) - \hat{v}(S, w))\nabla_w \hat{v}(S, w)$$

Deep Q-Networks (DQN)

DQN uses experience replay and fixed Q-targets

- Select an action using the epsilon-greedy policy.
- Store the experience $(s_t, a_t, r_{t+1}, s_{t+1})$ in the replay memory $\mathcal{D}$.
- Sample a mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
- Compute Q-learning targets w.r.t. old, fixed parameters w^-
- Optimize MSE between Q-network (prediction) and Q-learning targets

$$L_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}i}\left[\left(r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i)\right)^2\right]$$

Where, Q-learning Target $= r + \gamma \max_{a\prime} Q(s\prime, a\prime; w_i^-)$
Q-value from Q-network $= Q(s, a; w_i)$

- Use different variant of Gradient Descent

Policy Gradient

- Goal: given policy $\pi_\theta(s,a)$ with parameters θ, find best θ
- How do we know the quality of a policy π_θ?
- Reward function/ objective function,

$$J(\theta) = \sum_s d^{\pi_\theta}(s) V^{\theta_\pi}(s) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s,a) R_s^a$$

where $d^{\pi_\theta}(s)$ is stationary distribution of Markov chain for π_θ

Policy Gradient

- Policy gradient algorithms search for a local maximum in $J(\theta)$ by ascending the gradient of the policy, w.r.t. parameters θ

$$\Delta\theta = \alpha \nabla_\theta J(\theta)$$

- Where $\nabla_\theta J(\theta)$ is the policy gradient

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \frac{\partial J(\theta)}{\partial \theta_2} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$$

From Policy Gradient Theorem,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta log \theta_\pi(s, a) Q^{\pi_\theta}(s, a)]$$

Where α is a step-size parameter

Actor-Critic Reinforcement Learning

- Using linear value function approximation: $Q_w(s,a) = \phi(s,a)^\top w$
  - Critic: Updates w by linear TD(0)
  - Actor: Updates θ by policy gradient

Function QAC

Initialise s, θ

Sample $a \sim \pi_\theta$

for each step of episode **do**

Sample reward $r = \mathcal{R}^a$; **Sample transition** $s' \sim P_s^a$

Sample action $a' \sim \pi_\theta(s', a')$

$\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$

$\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$

$w \leftarrow w + \beta \delta \phi(s, a)$

$a \leftarrow a'$, $s \leftarrow s'$

end for

end for

References

- https://www.davidsilver.uk/teaching/
- https://www.samyzaf.com/ML/rl/qmaze.html

Thank You