EC332 Machine Learning

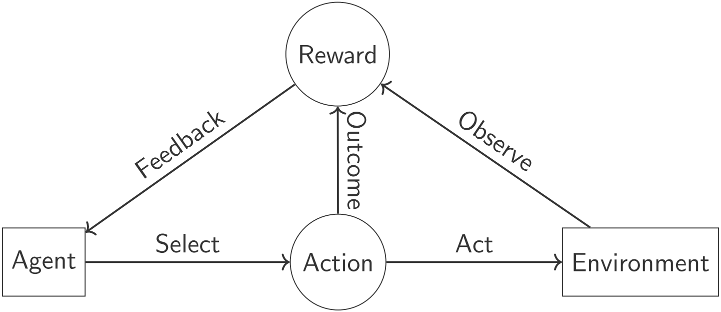
Reinforcement Learning: Derivation of Update Rules

Nazar Khan
Department of Computer Science
University of the Punjab

Bandit Problems

What is a Bandit Problem?

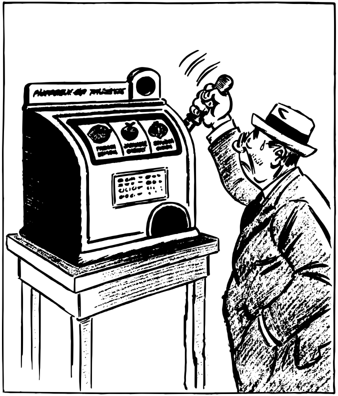
- A bandit problem is a simplified reinforcement learning problem.
- At each time step:
  1. You choose one action from a set of actions $\{a_1, a_2, \ldots, a_n\}$.
  2. You receive a reward r_t based on the chosen action.
- Goal: Maximize the total reward over time by learning the best action.



Introduction to Bandit Problems

- The term *bandit problem* originates from the analogy to a *one-armed bandit*.
- A one-armed bandit is a colloquial term for a slot machine used for gambling.
- Slot machines are designed to “steal” money while offering a chance of reward.

What Is a One-Armed Bandit?



- A slot machine with a single lever (or "arm") that a player can pull.
- Known as a "bandit" because it often takes more money than it gives.
- Offers the player an uncertain reward based on fixed probabilities.

Multi-Armed Bandit Problem

- Extends the analogy to multiple slot machines ("arms").
- Each arm has an unknown probability of giving a reward.
- The challenge: Choose which arm to pull to maximize overall reward.

Core Challenge: Exploration vs. Exploitation

- **Exploration:** Try different machines to gather information about their reward probabilities.
- **Exploitation:** Stick with the machine that seems to give the best rewards based on current knowledge.

Key Quantities in a Bandit Problem

- **Action-value function** $Q(a)$:
  - $Q(a)$ is the expected reward when choosing action a.
  - $Q(a) = \mathbb{E}[r_t | a_t = a]$, where r_t is the reward at time t.
- **Objective:** Learn $Q(a)$ for all actions a to identify the optimal action a^*:

$$a^* = \arg\max_a Q(a)$$

How to Estimate $Q(a)$?

- We do not know the true $Q(a)$ values. Instead, we estimate them iteratively.
- Let $\hat{Q}_t(a)$ be the estimate of $Q(a)$ at time t.
- After choosing action a at time t, we observe reward r_t.

Simple Update Rule

Update the estimate $\hat{Q}_t(a)$ as:

$$\hat{Q}_{t+1}(a) = \hat{Q}_t(a) + \alpha(r_t - \hat{Q}_t(a))$$

Derivation of the Update Rule

- Let $N_t(a)$ be the number of times action a has been selected up to time t.
- The empirical estimate of $Q(a)$ is the *average reward so far*:

$$\hat{Q}_t(a) = \frac{1}{N_t(a)} \sum_{i=1}^{N_t(a)} r_{t_i}$$

- If action a was selected 4 times until time t then $N_t(a) = 4$.
- If action a was selected at time steps $3, 6, 17$, and 24, then $t_1 = 3, t_2 = 6, t_3 = 17$, and $t_4 = 24$.
- Adding the latest reward r_t:

$$\hat{Q}_{t+1}(a) = \frac{1}{N_t(a) + 1} \left(N_t(a) \hat{Q}_t(a) + r_t \right)$$

Rewriting the Update Rule

- Simplify the expression:

$$\hat{Q}_{t+1}(a) = \frac{N_t(a)}{N_t(a)+1}\hat{Q}_t(a) + \frac{1}{N_t(a)+1}r_t$$
$$= \left(1 - \frac{1}{N_t(a)+1}\right)\hat{Q}_t(a) + \frac{1}{N_t(a)+1}r_t$$
$$= \hat{Q}_t(a) + \frac{1}{N_t(a)+1}(r_t - \hat{Q}_t(a))$$

- Generalize by replacing $\frac{1}{N_t(a)+1}$ by a step size α:

$$\hat{Q}_{t+1}(a) = \hat{Q}_t(a) + \alpha(r_t - \hat{Q}_t(a))$$

- Step size α controls how much the new reward influences the estimate.

Understanding the Update Rule

- $\hat{Q}_t(a)$: Current estimate of the action-value function for action a.
- r_t: Reward observed after taking action a at time t.
- α: Step size (learning rate), typically $\alpha = \frac{1}{N_t(a)}$.
- $(r_t - \hat{Q}_t(a))$: Difference between observed reward and current estimate (the error).

Intuition

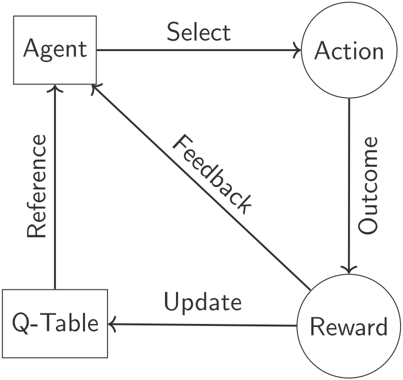
- If the reward r_t is higher than $\hat{Q}_t(a)$, increase $\hat{Q}_t(a)$.
- If the reward r_t is lower than $\hat{Q}_t(a)$, decrease $\hat{Q}_t(a)$.
- Observed reward r_t serves as a "target" for current esitmate $\hat{Q}_t(a)$.

Summary of Update Rule for Bandit Problems

- We derived the Q-update rule iteratively using rewards and counts.
- The rule:

$$\hat{Q}_{t+1}(a) = \hat{Q}_t(a) + \alpha(r_t - \hat{Q}_t(a))$$

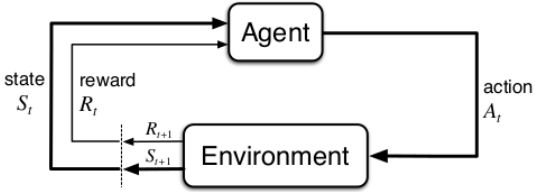
- This rule helps estimate the true action-value function $Q(a)$ over time.



Markov Decision Processes

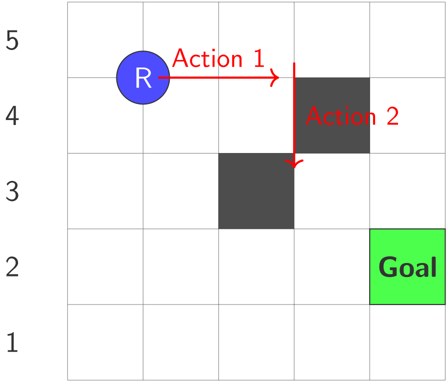
What is a Markov Decision Process?

- A framework for modeling decision-making in environments with:
  - Actions a
  - Rewards r
  - *States* s
  - *State transitions* $P(s'|s, a)$
- Objective: Maximize cumulative reward over time.



Example: Robot Navigation

- States (S): Positions on a grid.
- Actions (A): Move up, down, left, right.
- Transition probabilities (P): Probability of moving to the intended position vs slipping.
- Rewards (R): +10 for reaching the goal, -1 for each step.



Comparison: Bandit Problems vs MDPs

Aspect	Bandit Problem	Markov Decision Process (MDP)
State Dependence	No states; static actions	State transitions influence outcomes
Temporal Dependency	Independent decisions	Sequential decisions with future impact
Objective	Maximize immediate reward	Maximize long-term cumulative reward
Decision Horizon	Static, no future consideration	Dynamic, future actions considered
Policy	Strategy for choosing arms	Mapping from states to actions
Transition Dynamics	Not applicable	Defined by $P(s'\|s,a)$
Example Problem	Slot machines	Grid navigation or robot control

Table: Key Differences Between Bandit Problems and MDPs

The Update Rule for MDPs

Q-Learning Update Rule

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

- $Q(s,a)$: Current estimate of the action-value function.
- r: Immediate reward for taking action a in state s.
- γ: Discount factor (importance of future rewards).
- $\max_{a'} Q(s',a')$: Maximum future reward for next state s'.
- α: Learning rate (controls update magnitude).

Breaking Down the Rule

- Current Estimate: $Q(s, a)$
  - Represents the expected cumulative reward for state s, action a.
- Target Value: $r + \gamma \max_{a'} Q(s', a')$
  - Combines immediate reward r and discounted future rewards.
- Update Step:
  - Adjust $Q(s, a)$ towards the target value with learning rate α:

$$\Delta Q(s, a) = \alpha \left[\text{Target} - Q(s, a) \right]$$

Example of the Update Rule

- Current state: $s = s_1$, action: $a = a_1$
- Reward received: $r = 10$
- Next state: $s' = s_2$
- $Q(s_2, a')$: $\{Q(s_2, a_1) = 20,\ Q(s_2, a_2) = 15\}$
- Parameters: $\alpha = 0.1, \gamma = 0.9$

Update Calculation

$$\begin{aligned}
\text{Target} &= r + \gamma \max_{a'} Q(s', a') \\
&= 10 + 0.9 \times 20 \\
&= 28 \\
\Delta Q(s, a) &= \alpha \big[\text{Target} - Q(s, a) \big] \\
&= 0.1 \big[28 - Q(s_1, a_1) \big]
\end{aligned}$$

Instability in Q-Learning

- **Bootstrap Updating:**

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

  - Relies on current (and often noisy) Q-values.
  - Errors propagate during learning, causing instability.
- **Feedback Loops:**
  - Errors in Q-values are fed back into future updates.
  - Leads to oscillations or divergence in updates.
- **Overestimation Bias:**
  - $\max_{a'} Q(s',a')$ can include noisy overestimates.

Stabilization in Deep Q-Learning

- **Target Network:**
  - Maintains a separate network for target Q-values:

$$y = r + \gamma \max_{a'} Q_{\text{target}}(s', a'; \theta^-)$$

  - Target network updates less frequently.
  - Reduces feedback loop instability.
- **Experience Replay:**
  - Stores experiences in a replay buffer.
  - Samples random batches for training, breaking temporal correlation.
  - Provides stable gradient updates.

Comparison: Q-Learning vs Deep Q-Learning (DQN)

Aspect	Q-Learning	Deep Q-Learning (DQN)
Target Formation	Relies on noisy Q-values from the same table	Uses a separate, stable target network
Error Propagation	Errors propagate quickly, leading to instability	Errors are controlled with target network
Overestimation Bias	High (noisy max Q-values)	Reduced (stable reference for max Q-values)
Stabilizing Techniques	None	Target network, experience replay

Conclusion

- **Q-Learning:**
  - Prone to instability due to bootstrapping directly from its own estimates.
  - Faster error propagation, oscillations, or divergence.
- **Deep Q-Learning:**
  - Uses a target network and experience replay for stabilization.
  - Significantly more stable, especially for complex or high-dimensional environments.