@SSRoy,26th OCT,2025

The **multi-armed bandit** is a simple version of reinforcement learning where you choose actions to balance trying new things (exploration) and picking the best-known option (exploitation).Algorithms like UCB and Thompson Sampling help decide which action to take by either being optimistic or using probability.These ideas are used in **bigger RL problems** to choose actions and improve learning over time.

1) Multi-Armed Bandits (Exploration–Exploitation)

Core concept

Choose among k actions (arms) with unknown reward distributions to maximize expected cumulative reward. No state transitions, only action selection and immediate rewards.

Variables

- k: number of arms
- $a \in \{1, \ldots, k\}$: action/arm
- R_t: reward at time t after choosing A_t
- $Q_t(a)$: estimate of action value (expected reward) for arm a at time t
- $N_t(a)$: number of times arm a has been selected up to time t

Equations (action value & selection)

- **Sample-average estimate**:

$$Q_{t+1}(a) \leftarrow Q_t(a) + \frac{1}{N_t(a)}\big(R_t - Q_t(a)\big)$$

- **ε-greedy**: with prob. $1 - \varepsilon$ choose $\arg\max_a Q_t(a)$, else random explore.
- **UCB1** (optimism via confidence term) (choose at time t):

$$A_t = \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right].$$

(UCB family presented in the text as a principled exploration strategy.)

Conceptual example

Headline testing: show one of k titles; click = reward 1, no click = 0. The learner explores new titles (ε) and exploits the best so far.

Numerical example (2-armed)

- True (unknown) means: Arm 1: $\mu_1 = 0.4$, Arm 2: $\mu_2 = 0.6$.
- Initialize $Q_1(1) = Q_1(2) = 0,\ N_1(1) = N_1(2) = 0,\ \varepsilon = 0.1.$
- **t=1**: both equal → tie-break: pick arm 1. Suppose $R_1 = 1.$

$$N_2(1) = 1,\ Q_2(1) = 0 + \tfrac{1}{1}(1-0) = 1,\ Q_2(2) = 0.$$

- **t=2** (exploit): pick arm 1 (since 1>0). Suppose $R_2 = 0$:

$$N_3(1) = 2,\ Q_3(1) = 1 + \tfrac{1}{2}(0-1) = 0.5.$$

- **t=3** (maybe explore with prob 0.1; if exploit): pick arm 1 (0.5>0). Suppose $R_3 = 0$: $Q_4(1) = 0.5 + \frac{1}{3}(0-0.5) = 0.333\ldots$
  Over time, ε-greedy/UCB will shift more pulls to arm 2 as evidence grows. (UCB would also try the under-sampled arm 2 early due to the confidence bonus.)

---

2) Finite Markov Decision Processes (MDPs)

Core concept

Sequential decision problems where the **next state and reward depend only on current state and action** (Markov property). Defines the RL interface.

Variables

- $\mathcal{S}$: set of states, $s \in \mathcal{S}$
- $\mathcal{A}(s)$: actions available in state s
- $p(s', r \mid s, a)$: dynamics (transition/reward model)
- $\pi(a \mid s)$: policy (probability of action a in state s)
- $\gamma \in [0, 1)$: discount factor
- $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$: return
- $v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$, $q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$

Equations (Bellman)

- **Bellman expectation (state values)**:

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma v_\pi(s')\big].$$

- **Bellman optimality**:

$$v_*(s) = \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma v_*(s')\big],$$

$$q_*(s,a) = \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma \max_{a'} q_*(s', a')\big].$$

(Uniqueness of solution for finite MDPs.)

Conceptual example

2-room robot: choose to "work" (reward now, risk battery drain) or "recharge" (no immediate reward, but better future).

Numerical example (2-state tiny MDP)

States $\{A, B\}$, single action per state (for simplicity).
Transitions/Rewards:

- From **A** → to **B** with reward $+2$ (prob 1).
- From **B** → to **A** with reward $+1$ (prob 1).
  Let $\gamma = 0.5$. Find $v_*(A) = v(A), v_*(B) = v(B)$ (here optimal=only policy).
  Bellman equations:

$$v(A) = 2 + \gamma v(B) = 2 + 0.5\, v(B), \quad v(B) = 1 + \gamma v(A) = 1 + 0.5\, v(A).$$

Solve simultaneously: Substitute $v(B) = 1 + 0.5v(A)$ into first:
$v(A) = 2 + 0.5(1 + 0.5v(A)) = 2 + 0.5 + 0.25v(A) \Rightarrow 0.75v(A) = 2.5 \Rightarrow$
$v(A) = \frac{2.5}{0.75} = 3.\overline{3}$.
Then $v(B) = 1 + 0.5(3.\overline{3}) = 1 + 1.\overline{6} = 2.\overline{6}$.
This directly matches Bellman optimality for deterministic dynamics.

---

3) Dynamic Programming (DP)

Core concept

Model-based solution when $p(s', r \mid s, a)$ known: iteratively evaluate a policy and improve it until optimal.

Variables

- $v_k(s)$: k-th iterate of value for policy π

- $\pi'(s)$: improved greedy policy wrt q_π

Equations

- **Policy evaluation**:

$$v_{k+1}(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma v_k(s')\big].$$

- **Policy improvement**:

$$\pi'(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma v_\pi(s')\big].$$

- **Policy iteration**: alternate evaluation & improvement.
- **Value iteration**:

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma v_k(s')\big].$$

(These are the canonical DP updates in the text.)

Conceptual example

Gridworld with known transitions: compute values by sweeping the grid until convergence, then act greedily.

Numerical example (policy evaluation, 2-state)

Use the MDP from #2 and a **stochastic policy**: in **A**, do action that still leads to **B** with prob 1; in **B**, same back to **A**.
Initialize $v_0(A) = v_0(B) = 0$. One sweep of evaluation:

$$v_1(A) = \sum_{s',r} p(s', r \mid A)[r + \gamma v_0(s')] = 2 + 0.5\, v_0(B) = 2.$$

$$v_1(B) = 1 + 0.5\, v_0(A) = 1.$$

Second sweep:
$v_2(A) = 2 + 0.5\, v_1(B) = 2 + 0.5 = 2.5$,
$v_2(B) = 1 + 0.5\, v_1(A) = 1 + 1 = 2$, ...
This sequence converges to the exact solution computed earlier.

---

4) Monte Carlo (MC) Methods

Core concept

Learn value functions **from complete episodes** without a model; average **returns** observed after visiting a state (first-visit or every-visit).

Variables

- G_t: return from time t to end of episode
- $V(s)$: estimate of state value as average of returns observed after s

Equation (first-visit MC prediction)

$$V(s) \leftarrow \text{average of } G_t \text{ over first visits to } s.$$

Conceptual example

Blackjack: play many full hands; for each state (sum, dealer card, usable ace), average win/lose returns.

Numerical example (toy episodic chain)

Episodic chain: $S_0 \to S_1 \to \text{Terminal}$. Rewards: $+1$ on final transition only. $\gamma = 1$.
Episode 1 states: S_0, S_1 with terminal reward +1.

- First-visit returns: $G(S_1) = +1$ (from its time), $G(S_0) = +1$.
- After 1 episode: $V(S_0) = 1,\ V(S_1) = 1$.
  Episode 2: suppose same trajectory but terminal reward 0 (stochastic). Now returns: $G(S_1) = 0,\ G(S_0) = 0$.
- Averages (every-visit): $V(S_0) = (1+0)/2 = 0.5,\ V(S_1) = 0.5$.
  This illustrates the MC averaging principle.

---

5) Temporal-Difference (TD) Learning

Core concept

Update values **after each step** using bootstrapping from the **next state's current estimate**. Works online without a model.

Variables

- α: step size
- **TD error**: $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$

Equation (TD(0) prediction)

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

(Sarsa and Q-learning below are TD control variants.)

Conceptual example

Random walk: value of each nonterminal state is updated toward the neighbor's estimate plus reward.

Numerical example (single TD update)

From state s to s' with $R = 2,\ \gamma = 0.5,\ \alpha = 0.1,\ V(s) = 3.0,\ V(s') = 4.0$:

$$\delta = 2 + 0.5 \cdot 4 - 3.0 = 2 + 2 - 3 = 1.$$

$$V(s) \leftarrow 3.0 + 0.1 \cdot 1 = 3.1.$$

One step closer to bootstrapped target.

6) TD Control: SARSA (on-policy) & Q-Learning (off-policy)

Core concept

Learn **action-values** $Q(s, a)$ and improve a policy while acting.

- **SARSA** learns the value of the **behavior policy** (on-policy).
- **Q-Learning** learns the **greedy target** regardless of behavior (off-policy).

Variables

- $Q(s, a)$: action-value estimate
- A': next action actually taken (SARSA)
- $\max_{a'} Q(S', a')$: greedy next value (Q-learning)

Equations

- **SARSA**:

$$Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big].$$

- **Q-Learning**:

$$Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_{a'} Q(S', a') - Q(S, A)\big].$$

(These are the canonical tabular one-step TD control rules.)

Conceptual example

Windy Gridworld: SARSA learns a robust path under its exploratory behavior; Q-Learning aims for the greedy optimal path.

Numerical example

Suppose $Q(S, A) = 5.0,\ R = 1,\ \gamma = 0.9,\ \alpha = 0.2$.

- **SARSA**: given $Q(S', A') = 4.0$:
  Target $= 1 + 0.9 \cdot 4 = 4.6$.
  Update $= 5.0 + 0.2(4.6 - 5.0) = 5.0 - 0.08 = 4.92$.
- **Q-Learning**: given $\max_{a'} Q(S', a') = 6.0$:
  Target $= 1 + 0.9 \cdot 6 = 6.4$.
  Update $= 5.0 + 0.2(6.4 - 5.0) = 5.0 + 0.28 = 5.28$.
  Different targets reflect on- vs off-policy nature.

---

7) n-Step Returns & TD(λ) with Eligibility Traces

Core concept

Blend multi-step returns to trade bias/variance and speed credit assignment. **Eligibility traces** keep a decaying memory of visited states, updating many at once. (Chapter 7)

Variables

- $\lambda \in [0, 1]$: trace-decay parameter
- $e_t(s)$: eligibility trace for state s at time t

Equations (forward view sketch)

- **λ-return** (weighted mix of n-step returns):

$$G_t^{(\lambda)} = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}.$$

- **Backward view (tabular TD(λ))**:

$$e_t(s) = \gamma\lambda\, e_{t-1}(s) + \mathbf{1}\{S_t = s\}, \quad V(s) \leftarrow V(s) + \alpha\, \delta_t\, e_t(s),$$

  with $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$.
  (These are the standard forms covered in Chapter 7's forward/backward equivalence discussion.)

Conceptual example

Maze learning: TD(λ) quickly propagates credit back along the whole path after a good outcome.

Numerical example (short trajectory)

Trajectory: $S_0 \to S_1 \to S_2\ (terminal)$.
Let $\gamma = 0.9,\ \lambda = 0.8,\ \alpha = 0.1$.

Initial $V(S_0) = 2.0,\ V(S_1) = 1.0,\ V(S_2) = 0$. Rewards: $R_1 = 0$ (to S_1), $R_2 = 5$ (terminal).

Step t=0:

- $\delta_0 = R_1 + \gamma V(S_1) - V(S_0) = 0 + 0.9 \cdot 1 - 2 = -1.1$
- $e_0(S_0) = 1$, others 0.

  Updates: $V(S_0) \leftarrow 2 + 0.1(-1.1)(1) = 1.89$.

  Step t=1:
- $\delta_1 = R_2 + \gamma V(S_2) - V(S_1) = 5 + 0 - 1.0 = 4.0$
- Traces:

  $e_1(S_0) = \gamma\lambda e_0(S_0) = 0.9 \cdot 0.8 \cdot 1 = 0.72$,

  $e_1(S_1) = 1$.

  Updates:
- $V(S_1) \leftarrow 1.0 + 0.1(4.0)(1) = 1.4$
- $V(S_0) \leftarrow 1.89 + 0.1(4.0)(0.72) = 1.89 + 0.288 = 2.178$.

  Credit spreads to both states immediately (backward view).

---

8) Planning & Learning Integration (Dyna)

Core concept

Unify model-free learning with **model-based planning**: learn a model from experience, then do **simulated updates** in between real steps (e.g., Dyna-Q, prioritized sweeping). Planning, acting, and model learning form a loop.

Variables

- Learned model $\hat{p}(s', r \mid s, a)$
- Planning updates: additional TD backups using simulated transitions

Algorithm sketch (Dyna-Q cycle)

1. Real step: observe (S, A, R, S'), do Q-learning/SARSA update.
2. Update model entry $\hat{p}(S', R \mid S, A)$.
3. **k** planning sweeps: sample past (s, a), query model for (s', r), do extra TD updates.

Conceptual example

Robot vacuum: after a bump, it updates the map (model) and runs quick internal rollouts to refine Q-values before moving again.

Numerical example (single planning step)

Assume tabular Q-learning, $\alpha = 0.5,\ \gamma = 0.9$.
Real experience: from S via A, got $R = 2$ to S'; current $Q(S, A) = 1,\ Q(S', \cdot)$ has max 3.
Real update: $Q(S, A) \leftarrow 1 + 0.5\big(2 + 0.9 \cdot 3 - 1\big) = 1 + 0.5(3.7 - 1) = 1 + 1.35 = 2.35$.
Model stored: $\hat{p}(S', 2 \mid S, A) = 1$.
Planning: sample the same (S, A) from the model; redo the same Q-update (like an extra replay), pushing Q further toward its target. Prioritized sweeping would focus such replays where changes are largest.

---

9) Function Approximation (Large State Spaces)

Core concept

Replace tables by **parameterized approximators** (linear features, tile coding, neural nets). Learn parameters by **stochastic gradient** on TD errors.

Variables

- Feature map $x(s) \in \mathbb{R}^d$
- Weights $w \in \mathbb{R}^d$
- Approximate value $\hat{v}(s, w) = w^\top x(s)$

Equations (semi-gradient TD(0))

$$w \leftarrow w + \alpha\big[R + \gamma \hat{v}(S', w) - \hat{v}(S, w)\big]\nabla_w \hat{v}(S, w) = w + \alpha\,\delta\,x(S).$$

(Linear case gradient is $x(S)$.)

Conceptual example

Mountain-Car / Tile coding: many positions/speeds; tile features let local generalization work with linear learners.

Numerical example (linear features)

Let $x(S) = [1,\ s]^\top$ (bias + scalar feature), $w = [0.5,\ 1.0]^\top$.
Current state $s = 2 \Rightarrow \hat{v}(S, w) = 0.5 + 1.0 \cdot 2 = 2.5$.
Next state $s' = 1 \Rightarrow \hat{v}(S', w) = 0.5 + 1.0 \cdot 1 = 1.5$.
Reward $R = 1,\ \gamma = 0.9,\ \alpha = 0.1$.

$$\delta = R + \gamma \hat{v}(S', w) - \hat{v}(S, w) = 1 + 0.9 \cdot 1.5 - 2.5 = 1 + 1.35 - 2.5 = -0.15.$$

Gradient $= x(S) = [1, 2]^\top$.
Update: $w \leftarrow [0.5, 1.0] + 0.1(-0.15)[1, 2] = [0.5 - 0.015,\ 1.0 - 0.03] = [0.485,\ 0.97]$.
New $\hat{v}(S, w) = 0.485 + 0.97 \cdot 2 = 2.425$ (moved toward target).

---

10) Policy Gradient & Actor–Critic

Core concept

Directly optimize the policy $\pi_\theta(a \mid s)$ using gradient ascent of performance $J(\theta)$.
Actor–Critic uses a value function (critic) to reduce variance while the actor updates policy parameters. (Chapter 11 introduces actor–critic.)

Variables

- θ: policy parameters
- $\pi_\theta(a \mid s)$: differentiable policy (e.g., softmax or Gaussian)
- $b(s)$: baseline (often $V(s)$) to reduce variance
- $\hat{A}(s, a)$: advantage estimate $\approx Q(s, a) - V(s)$

Canonical gradient (REINFORCE idea with baseline)

$$\nabla_\theta J(\theta) = \mathbb{E}\big[\,\nabla_\theta \log \pi_\theta(A \mid S)\ \hat{A}(S, A)\,\big].$$

Actor–Critic online update (one step):

- Critic TD error $\delta = R + \gamma V(S'; w) - V(S; w)$
- Value update: $w \leftarrow w + \alpha_v\, \delta\, \nabla_w V(S; w)$
- Policy update: $\theta \leftarrow \theta + \alpha_\pi\, \delta\, \nabla_\theta \log \pi_\theta(A \mid S)$

Conceptual example

Two-action softmax bandit: learn probability of choosing Left vs Right directly via gradient feedback.

Numerical example (softmax bandit, single update)

Two actions $a \in \{1, 2\}$. Preferences $h_\theta(a)$ define softmax policy:

$$\pi_\theta(a) = \frac{e^{h(a)}}{e^{h(1)} + e^{h(2)}}.$$

Let $h(1) = 0.0,\ h(2) = 0.0 \Rightarrow \pi(1) = \pi(2) = 0.5$. Choose $a = 1$, observe $R = 1$ (baseline 0 for simplicity).
Gradients for softmax (two-action case):

$$\nabla_{h(1)} \log \pi(1) = 1 - \pi(1) = 0.5, \quad \nabla_{h(2)} \log \pi(1) = -\pi(2) = -0.5.$$

With $\alpha_\pi = 0.1$, update preferences:

$$h(1) \leftarrow 0 + 0.1 \cdot 1 \cdot 0.5 = 0.05, \quad h(2) \leftarrow 0 + 0.1 \cdot 1 \cdot (-0.5) = -0.05.$$

New policy:

$$\pi(1) = \frac{e^{0.05}}{e^{0.05} + e^{-0.05}} \approx \frac{1.0513}{1.0513 + 0.9512} \approx 0.525, \quad \pi(2) \approx 0.475.$$

The policy shifts toward the rewarding action. (Actor–critic would use δ as the signal.)

(Helpful) Exploration Strategies Summary (from Bandits & Control)

- **ε-greedy**: simple and effective; persistent exploration.
- **Optimistic initialization**: initial high Q to drive early exploration.
- **UCB**: balances exploration via confidence bonus; efficient in stationary bandits.

(Helpful) How Value Functions Connect to Optimal Actions

- If you know $v_*(s)$, **one-step greedy** w.r.t. v_* is optimal.
- If you know $q_*(s, a)$, choose $\arg\max_a q_*(s, a)$ directly (no one-step lookahead needed).

Final Notes

- **Bellman equations** are the backbone (definitions & optimality).
- **TD learning** is the practical engine for online learning.
- **Dyna** shows planning ≈ learning with simulated experience.
- **Function approximation** is essential in large problems.