A brief introduction to

Reinforcement Learning

Aadarsh Ram
Ibot, IITM

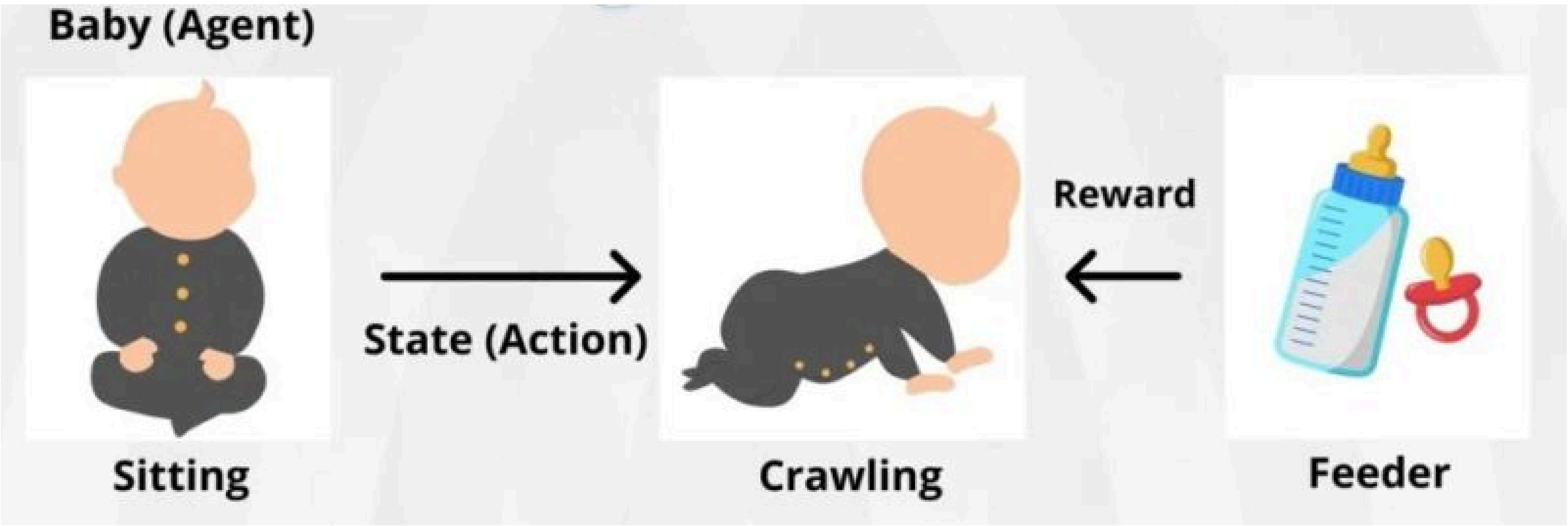
Before we start...

Useful resources to follow along:

- https://huggingface.co/learn/deep-rl-course/en/unit0/introduction/
- https://github.com/aadarshram/RL_basics/blob/main/NOTES.md (My concise notes)

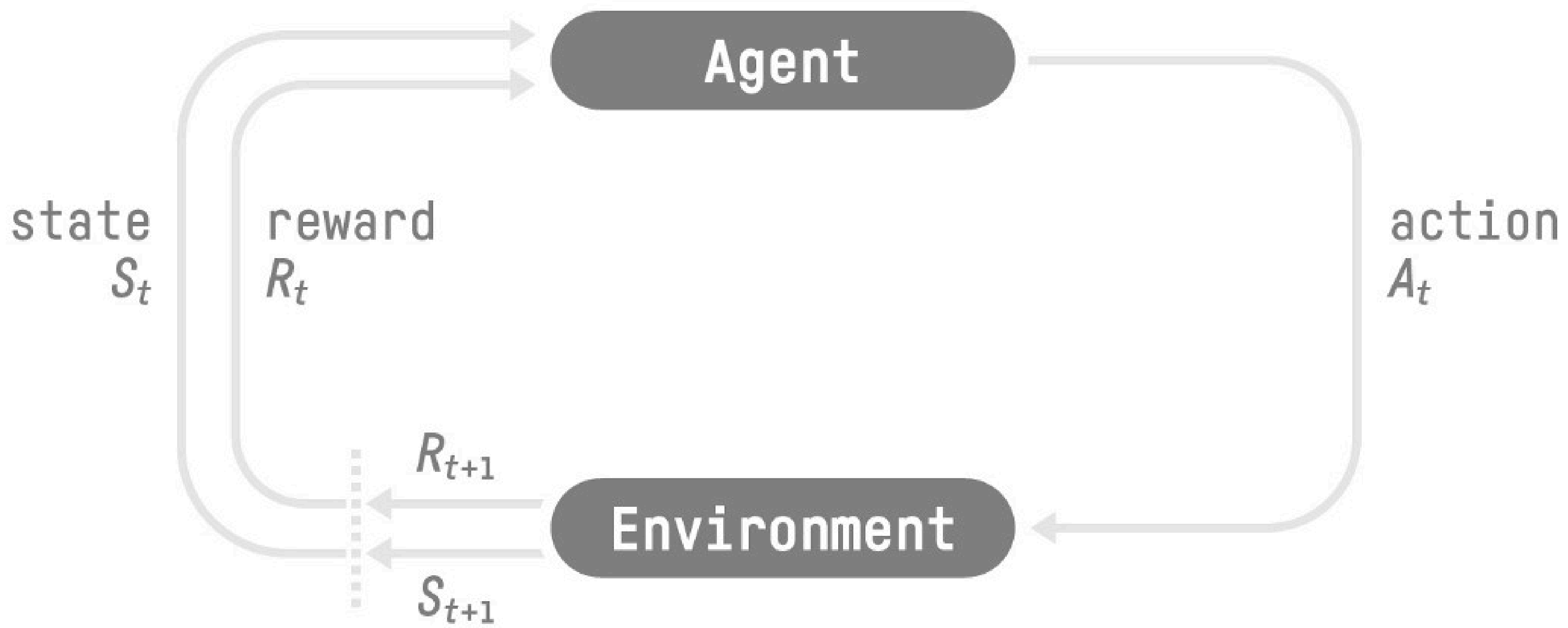
Motivation: What is Reinforcement Learning?

- Learning by trial and error
- Receive feedback in form of rewards/punishments
- Goal: learn behaviour that maximizes expected cumulative reward
- Inspired by how animals learn.



The RL Interaction Loop

- RL setup has:
  - Agent: learner/decision-maker
  - Environment: world it interacts with
  - Policy: “The brain” (S-> A)
- At each step t:
  - Agent observes state St
  - Chooses action At
  - Environment gives reward Rt+1
  - Moves to next state St+1 with some transition probability



The Reward Hypothesis

- “All goals can be framed as maximizing expected cumulative reward.”
- RL converts any task → reward accumulation
- Discounted return:

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

SOME NOTES

- **Observation vs. State:**A policy maps observations → actions, not states → actions.
- **Discounting Future Rewards:** In stochastic environments, future rewards are uncertain → discounting reduces risk. Even in deterministic, infinite-horizon settings, discounting ensures convergence of cumulative returns.
- **Episodic vs. Continuing Tasks:**
  - Episodic tasks: interactions end at a terminal state → episode resets.
  - Continuing tasks: no terminal state; agent interacts indefinitely.

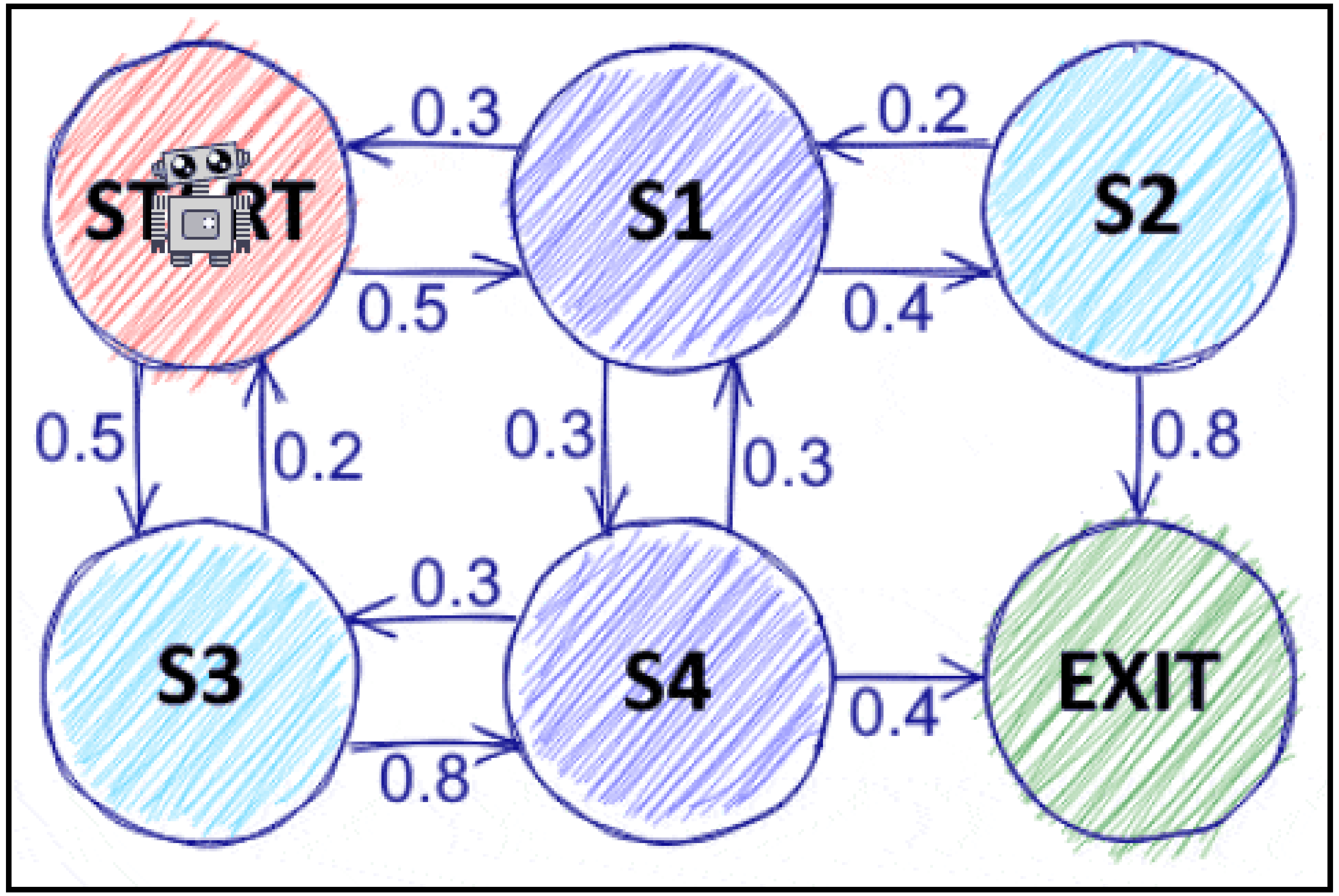
SOME NOTES

- **Exploration vs. Exploitation:** At each step an agent must choose between:
  - Exploit: choose the best-known action (max reward now).
  - Explore: try new actions (discover better long-term strategies).
- Trade-off is essential; too much exploitation → local optima, too much exploration → slow learning.
- **Policy-based Vs Value-based methods:** Learn mapping from state to actions directly or learn state/state-action value functions.
- **Deterministic vs. Stochastic Policies:** Unique action for each state or a distribution of possibilities.
- **Markov Property:** State must contain all information necessary to predict the next step. Can be the history of the past k observations if a single observation is insufficient.

Markov Decision Processes (MDPs)

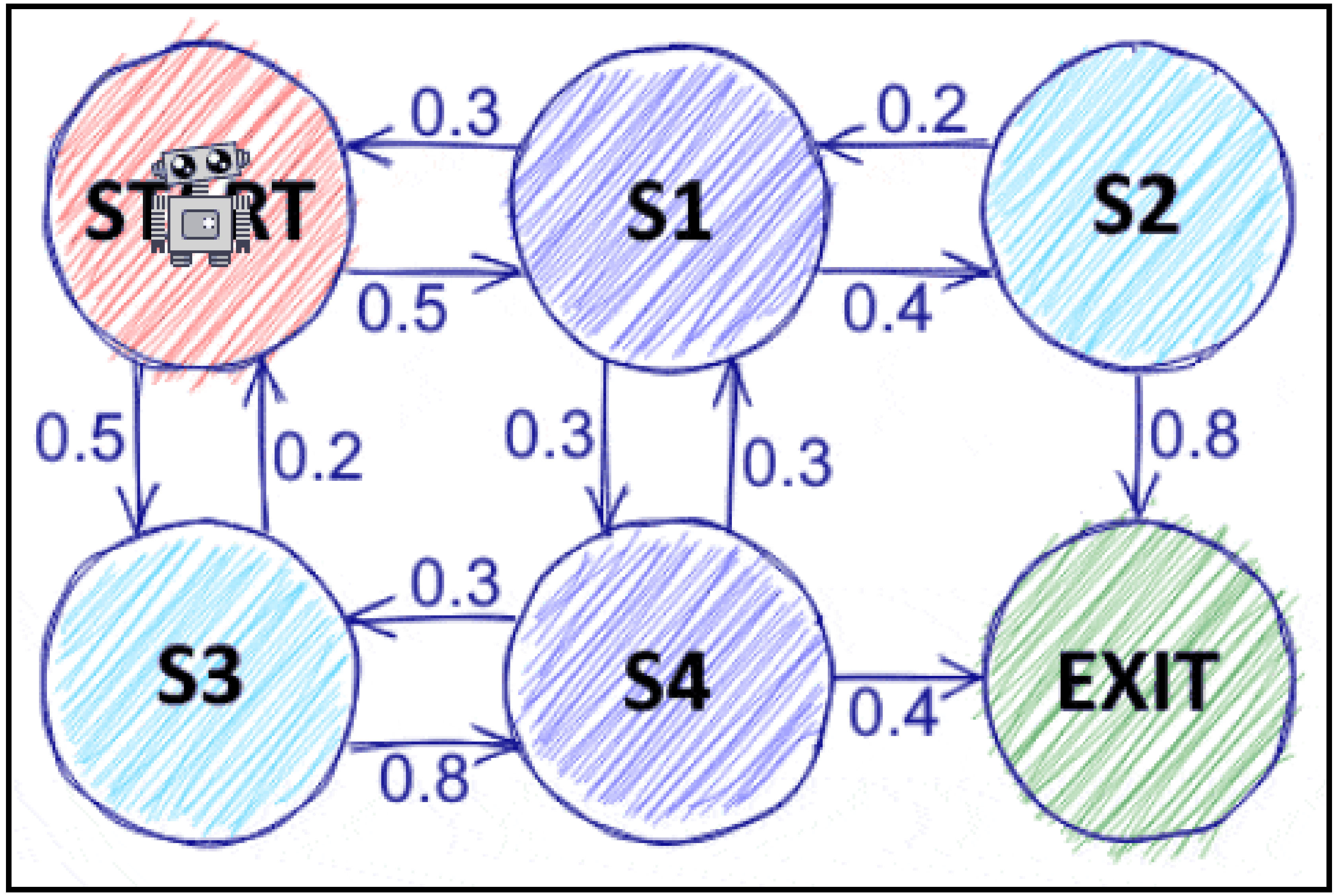
- RL is formally built on **MDPs**
- MDP assumptions:
  - **Markov property**: next state depends only on current state & action
- Defined by tuple

$$(S, A, P, R, \gamma)$$



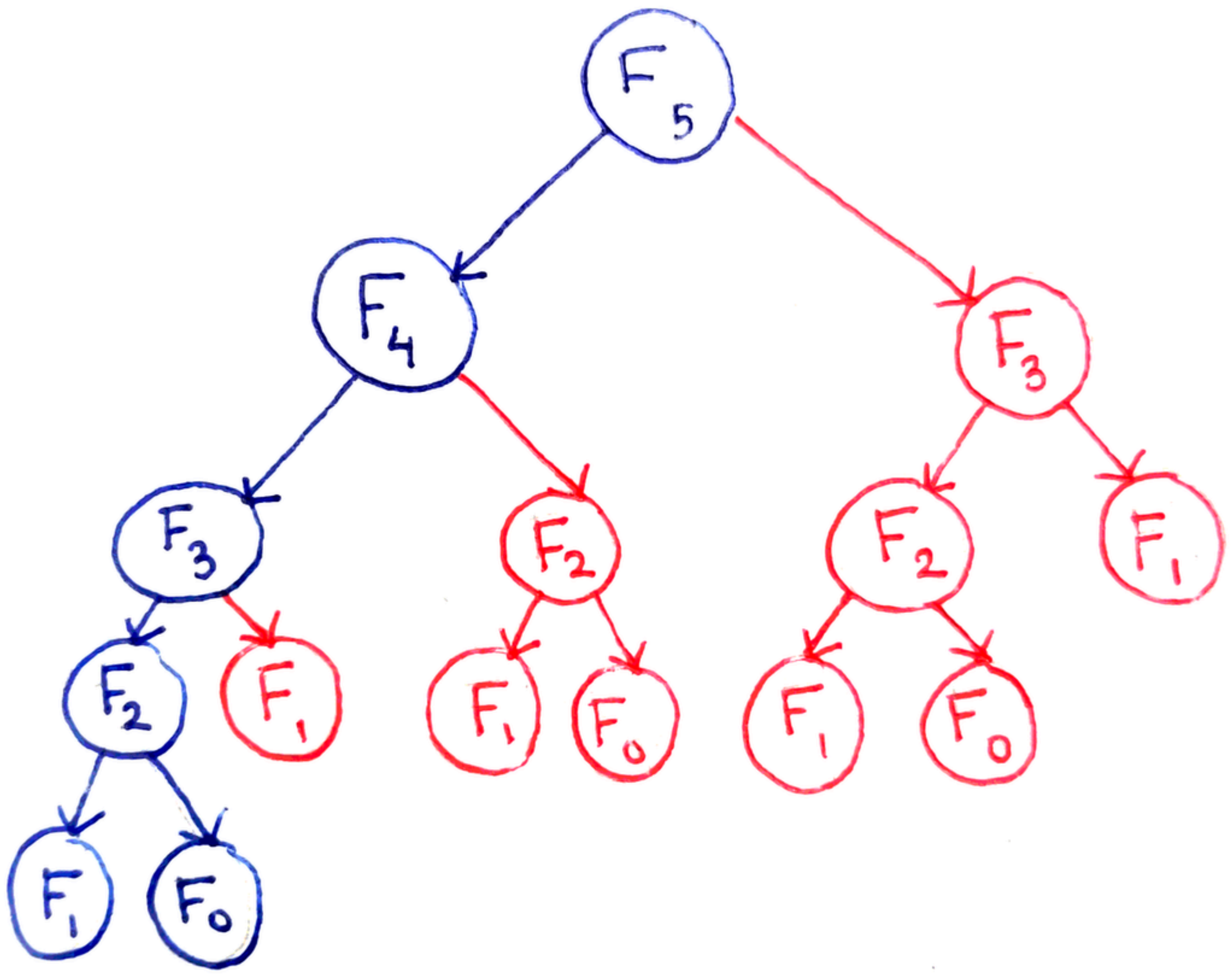
MDP Components

- States S
- Actions A
- Transitions $P(s'|s,a)$
- Rewards $R(s,a)$
- Discount γ



Finite Horizon MDPs

- Episode ends after N steps
- Use **Dynamic Programming (DP)** backwards in time
- **Optimality principle:**
  - "Every tail of an optimal trajectory is optimal."



Dynamic Programming Algorithm

$$V_k(s) = \max_a \mathbb{E}[r(s,a) + V_{k+1}(s')]$$

Steps:

1. Initialize **terminal reward**
2. For k = N–1 to 0:
   - Compute V_k(s) via Bellman backup
3. Recover optimal policy from argmax

$$\text{Initialize:} \qquad V_N(x_N) = r(x_N).$$

$$\text{For } k = N-1,\, N-2,\, \ldots,\, 0: \qquad V_k(x_k) \;=\; \max_{a \in \mathcal{A}} \; \mathbb{E}\big[\, r(x_k, a) + V_{k+1}(x_{k+1}) \,\big|\, x_k, a \big] \,.$$

Infinite Horizon MDPs

- Episodes don't end
- Use **Bellman Optimality Equation:**

$$V^*(s) = \max_a \mathbb{E}[r + \gamma V^*(s')]$$

- Solve using **fixed-point iteration**

Value Iteration

- Iteratively apply Bellman operator:

$$V_{k+1}(s) = \max_a \mathbb{E}[r + \gamma V_k(s')]$$

- Converges to **optimal value function**

Policy Iteration

Two steps repeatedly:

1. **Policy Evaluation:** compute V^π
2. **Policy Improvement:**

$$\pi' = \arg\max_a \mathbb{E}[r + \gamma V^\pi]$$

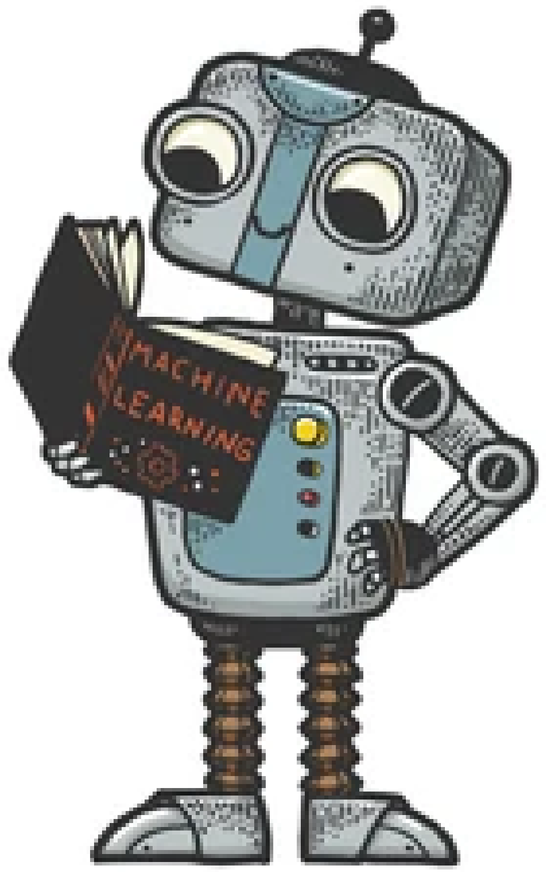
SOME NOTES

- **Variants of Value and Policy iteration:** Synchronous, Asynchronous
- RL algorithms are of 2 types- 1. **Evaluation** 2. **Control.** Evaluation algorithms find value function for given policy. Control algorithms solves for a better policy.

LEARNING FROM EXPERIENCE

Why Learn Instead of Compute?

- In real world:
  - Transitions are unknown
  - Reward is unknown before acting
  - Instead of solving MDP, agent samples trajectories and learns



Stochastic Fixed Point Iterations

- Many RL algorithms rely on solving expected Bellman equations.
- Exact expectations are intractable in most environments.
- Solution: Replace expectations with single samples obtained through interaction.
- Under standard assumptions (step-size conditions, contraction, bounded noise), these stochastic iterative algorithms converge to the optimal value or policy.

Fixed-Point Iteration Template

To solve a fixed-point equation:

$$Hr^* = r^*$$

we iterate:

$$r_{k+1} = r_k + \alpha_k \big(h(r_k) - r_k\big)$$

where:

$$h(r) = Hr.$$

- The fixed point satisfies:

$$h(r^*) = r^*, \quad \text{or equivalently } h(r) - r = 0.$$

Stochastic Version Template

- In RL, the operator H (expected Bellman backup) is **unknown**.
- Instead, we use **sample-based estimates**:

$$\hat{H}, \quad \hat{h}(r) = \hat{H}r.$$

- The update becomes:

$$r_{k+1} = r_k + \alpha_k \big(\hat{h}(r_k) - r_k\big)$$

Temporal Difference Learning (TD)

- Bootstraps using current estimate
- Update:

$$V(s) \leftarrow V(s) + \alpha(r + \gamma V(s') - V(s))$$

TD Error:

$$\delta = r + \gamma V(s') - V(s)$$

Apply template to Policy Evaluation (Value iteration)

Monte Carlo Policy Evaluation

- Uses full return until episode ends:

$$V(s) \leftarrow V(s) + \alpha(G - V(s))$$

- **Unbiased**, high variance
- TD = biased/low variance
- MC = unbiased/high variance

TD(λ)

- Weighted mixture of all n-step returns
- (bootstrapping ⇔ full MC)
- Controls bias-variance trade-off

$$V_\pi^{(k+1)}(s) = V_\pi^{(k)}(s) + \alpha\,(1-\lambda)\,\sum_{n=0}^{\infty} \lambda^n\, R_k^{(n)},$$

where each n-step return is:

$$R_k^{(n)} \;=\; \sum_{i=0}^{n-1} \gamma^i\, r(s_i, a_i) \;+\; \gamma^n V_\pi^{(k)}(s_n).$$

SOME NOTES

- Alternative variants of TD and MCPE include updating on **every visit** of a state or only on **first visit** for a given episode. While the former may yield lower variance unlike latter it can be biased due to the dependent returns from the same episode.
- The TD algorithm is based on the Bellman equation and hence heavily relies on the markov property to work. Thus, for non-Markovian or partially markov environments (which is more often the case), Monte Carlo methods or TD(λ) might prove better.

THE CONTROL PROBLEM

Why Q-Values?

- Direct Bellman optimality cannot be used with unknown model
- Introduce Q-values:

$$V(s) = \max_a Q(s,a)$$

Q-Bellman Equation

$$Q^*(s,a) = \mathbb{E}\left[r(s,a) \ + \ \gamma \max_b Q^*(s',b) \,\middle|\, s,a \right],$$

Q-Learning

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_b Q(s',b) - Q(s,a) \right)$$

- Off-policy
- Model-free
- Converges under certain assumptions

Q-learning is simply, TD method involving Q-values.

Algorithm

```
Initialize Q(s,a) arbitrarily
for episode = 1 … M:
    set ε = ε_episode
    observe initial state S
    while S is not terminal:
        choose A with ε-greedy over Q
        take action A and observe R, S'
        Q(S,A) += α * (R + γ * max_a Q(S', a) - Q(S,A))
        S = S'
return Q
```

Limitation

- Not scalable: huge tables for large/continuous spaces.

That’s all for today!

What’s next?

- Deep Q Learning
- Policy-gradient methods (REINFORCE)
- Actor-Critic methods
- Proximal Policy Optimization (brief)
- Code implementation: Random policy on Frozen Lake using stable baselines

Deep Q Learning (DQN)

Why Deep Q-Learning?

- Tabular Q-learning does not scale to large or continuous state spaces.
- Need a function approximator to generalize across states.
- Deep neural networks provide a flexible mapping

$$(s) \mapsto Q(s, a; \theta) \ \forall a$$

Deep Q Learning (DQN)

- Replace Q-table with a neural network.
- Input: state/observation
- Output: Q-value for each admissible action
- Goal: minimize TD error via gradient descent:

$$L = \big(y - Q(s, a; \theta)\big)^2$$

$$y = r + \gamma \max_{a'} Q_{\hat{\theta}}(s', a')$$

Challenges

Catastrophic Forgetting

- Online updates cause the network to overwrite older knowledge.
- Solution: Experience Replay Buffer
- Store past transitions
- Train from diverse older experiences
- Repeat learning on useful transitions

Challenges

Experience Correlation

- Sequential transitions are highly correlated → biased gradient updates.
- SGD assumes i.i.d. samples.
- Solution: Replay buffer with random minibatch sampling
    - → decorrelates data
    - → stabilizes learning

Challenges

The Moving Target Problem

- TD target uses the same network being updated → unstable fixed point.
- Every weight update shifts the target.
- Solution: Target Network
  - Maintain copy $Q\theta$_hat
  - Update it only every C steps
  - Provides stable bootstrap target

Challenges

Overestimation Bias

- Max operator over noisy Q-values leads to optimistic value estimates.
- Amplified by neural networks.
- Solution:
  - Use Target Network for stable target
  - Or use **Double DQN** to reduce bias:
  -

$$y = r + \gamma Q_{\hat{\theta}}(s', \arg\max_{a'} Q_{\theta}(s', a'))$$

DQN Algorithm

1. Initialize replay buffer D, Q-network Q_θ, and target network $Q_{\hat{\theta}} \leftarrow Q_\theta$.
2. For each episode:
   - Select action using ε-greedy
   - Observe transition (s, a, r, s')
   - Store in replay buffer
   - Sample minibatch
   - Compute TD target and loss
   - Gradient descent update on θ
   - Every C steps: update target network

Policy-Based Methods

- Directly output action distribution
- Works for continuous actions
- Generally smoother learning
- Objective:

$$J(\theta) = \mathbb{E}[G]$$

$$\nabla_\theta J = \mathbb{E}\Big[\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t)\ G\Big]$$

Policy Gradient Theorem

REINFORCE Algorithm

- Sample trajectory
- Compute returns
- Update:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a|s)\, G$$

Actor-Critic Methods

- REINFORCE (pure policy gradient) suffers from:
- High variance → sample inefficient
- Slow learning due to Monte Carlo returns
- Actor–Critic solution:
- Use a Critic to estimate value function
- Use this estimate to bootstrap instead of relying purely on full-return Monte Carlo

Actor:

- Input: state S_t
- Output: action A_t

Critic:

- Input: (S_t, A_t)
- Output: $Q(S_t, A_t)$

Actor Update:

$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(a_t|s_t)\, Q(s_t, a_t)$$

Critic update:

Transition from environment:

$$(S_t, A_t, R_{t+1}, S_{t+1})$$

Target:

$$R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$$

Critic update:

$$\Delta w = \beta \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right] \nabla_w Q(S_t, A_t)$$

Advantage Actor-Critic

Motivation:

- Using **absolute** $Q(s, a)$ is noisy
- Better signal: **relative improvement** → Advantage function

Advantage:

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$$

Using

$$Q(s_t, a_t) = R(s_t, a_t) + \gamma V(s_{t+1})$$

Simplifies to:

$$A(s_t, a_t) = \delta_t = \underbrace{R_{t+1} + \gamma V(s_{t+1}) - V(s_t)}_{\text{TD Error}}$$

Thus: **Advantage = TD Error**

Advantage Actor-Critic (A2C)

Actor Update:

$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t)$$

Critic:

Predicts $V(s)$ using TD error as target.

Outcome:

- Lower variance
- More stable learning
- Works with synchronous/asynchronous variants (A3C, etc.)

Proximal Policy Optimization (PPO)

Problem with vanilla PG / Actor–Critic:

- A single sample may **misrepresent** policy improvement
- Risk of **large destructive policy updates**

PPO Goal:

- Keep updates **conservative**
- Encourage **stable monotonic improvement**
- Without the complexity of TRPO

Define probability ratio:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}$$

REINFORCE update:

$$\nabla_\theta J = \mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_t|s_t) A_t\right]$$

Replace log-prob with ratio:

$$r_t(\theta) A_t$$

Proximal Policy Optimization (PPO)

$$J(\theta) = \mathbb{E}\left[\min(r_t(\theta)A_t,\ \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)\right]$$

- Prevents ratio from deviating too far
- Ensures stable updates
- Encourages improvement without collapse

Typically: $\epsilon = 0.2$

Summary

- Any Reinforcement Learning (RL) problem is fundamentally a sequential decision-making task where the goal is to maximize the expected cumulative reward.
- Formally, RL is modeled using a Markov Decision Process (MDP), for which exact mathematical solutions exist when the environment’s dynamics are known.
- When the model is unknown, we rely on learning-based methods that estimate these quantities from sampled trajectories.
- Popular approaches include Q-Learning, DQN, Actor−Critic methods, REINFORCE, and PPO.
- In essence: RL is about writing down the expected cumulative reward — and then figuring out clever (and often mathematically messy) ways to maximize it, which is why all the “fancy algorithms” exist in the first place.