Course 2 - Sample-based Learning Methods

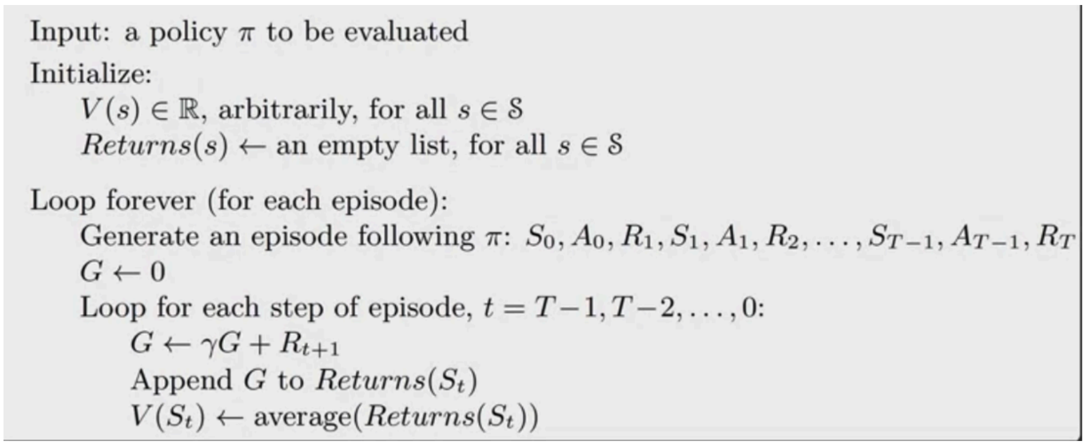
Module 2

Monte Carlo Methods

Monte Carlo methods estimate values by **averaging** over a **large number** of **random samples**. Monte Carlo method for learning a value function would first observe multiple returns from the same state.

It average those observed returns to estimate the expected return from that state

Algorithm for estimating the state value function of a policy



```
Input: a policy $\pi$ to be evaluated
Initialize:
    $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
    $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):
    Generate an episode following $\pi$: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Append $G$ to $Returns(S_t)$
        $V(S_t) \leftarrow$ average$(Returns(S_t))$
```

How Monte Carlo Estimates Value Functions

- The agent **observes multiple returns** from the same state and averages them.
- As the number of samples **increases**, the estimated value becomes more accurate.
- **Only works with episodic tasks**, since returns are calculated at the end of episodes.

Differences Between Monte Carlo and Dynamic Programming

- We do not need to keep a **large model** of the environment, but rather learn from experience
- We are estimating the value of an individual state **independently** of the values of other states

- The **computation** needed to update the value of each **state** does **not depend** on the **size** of the **MDP**

Monte-Carlo for Control

Monte-Carlo for Action Value

The method is similar to learning state values: averaging sample returns from a state-action pair.

Learning action values accurately requires trying all possible actions in a given state.

Exploration Strategies

- **Exploring Starts:** Ensures episodes **begin** in **every** possible **state-action pair**, allowing full policy evaluation.
- **Alternative Strategies (e.g., Epsilon-Greedy):** Used when setting starting states manually isn't feasible, especially for stochastic policies.

Exploring starts

- This is a way to maintain exploration.
- In exploring starts, we must guarantee that episodes start in every state-action pair.
- Afterwards, the agent simply follows its policy.

$$s_0, a_0, s_1, a_1, s_2, a_2, \cdots$$

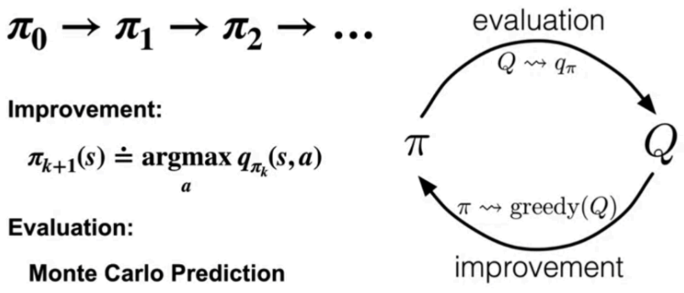
Random From π and p

Monte-Carlo for Generalized Policy Iteration

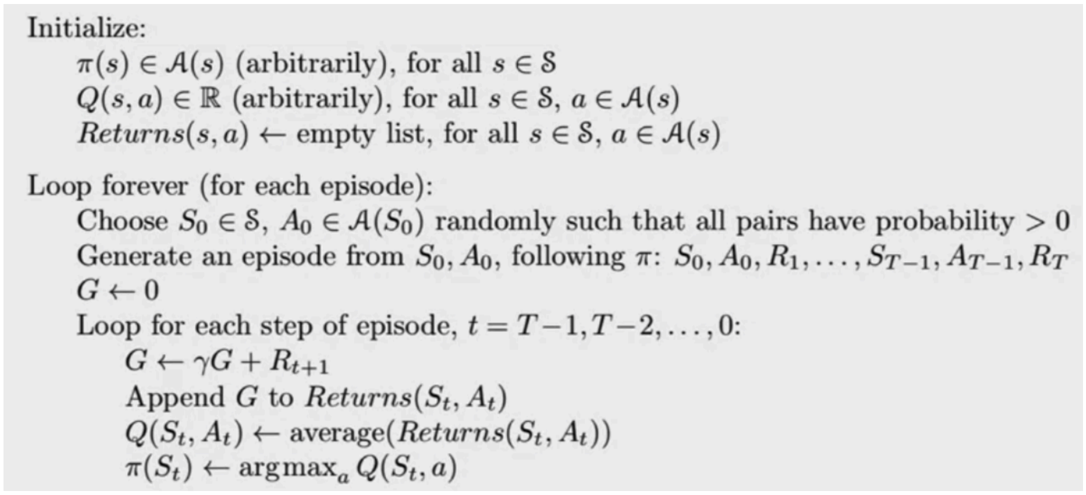
GPI includes a policy evaluation and a policy improvement step. GPI algorithms produce sequences of policies that are at least as good as the policies before them

For the policy improvement step, we can make the policy greedy with respect to the agent's current action value estimates.

For the policy evaluation step, we will use a Monte Carlo method to estimate the action values.


$\pi_0 \rightarrow \pi_1 \rightarrow \pi_2 \rightarrow \dots$
Improvement:
$\pi_{k+1}(s) \doteq \underset{a}{\operatorname{argmax}}\, q_{\pi_k}(s,a)$
Evaluation:
Monte Carlo Prediction
evaluation
$Q \rightsquigarrow q_\pi$
π
Q
$\pi \rightsquigarrow \text{greedy}(Q)$
improvement

Monte Carlo method for learning action values



```
Initialize:
    $\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$
    $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
    $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):
    Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability $> 0$
    Generate an episode from $S_0, A_0$, following $\pi$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Append $G$ to $Returns(S_t, A_t)$
        $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
        $\pi(S_t) \leftarrow \text{argmax}_a Q(S_t, a)$
```

Steps of the Algorithm

1. **Exploring Starts:** Each episode begins with a **randomly selected state-action pair**, ensuring all possibilities are explored.

2. **Episode Generation:** The agent follows its policy, recording the sequence of **states, actions, and rewards**.

3. **Computing Returns:** At the end of the episode, **returns are calculated** for each state-action pair by summing future rewards with discounting rate.

4. **Updating Action Values:** The **average return** for each state-action pair is computed and used to refine **action value estimates**.

5. **Policy Improvement:** The policy is updated to take the **greedy action**—choosing the best action based on **updated action values**.

6. **Iteration & Convergence:** This cycle continues until the policy **optimally selects actions** across all states.

Exploration Methods for Monte Carlo

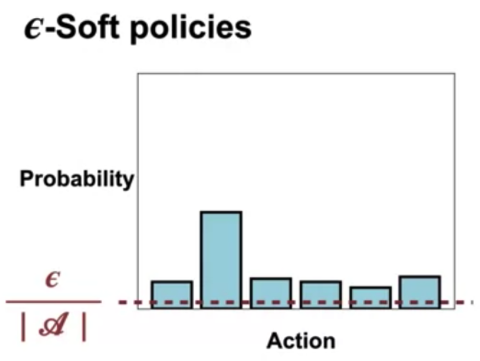
Epsilon-soft policies

Limitations of Exploring Starts

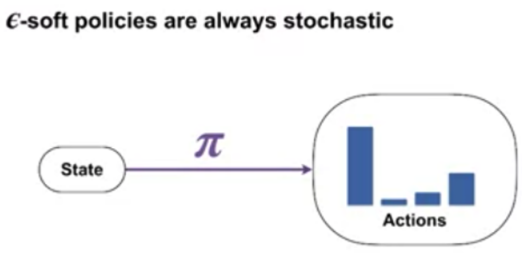
- **Exploring Starts** require starting from every possible **state-action pair**, which is often impractical.
- **Example**: Setting up a **self-driving car** in random configurations on a freeway would be unrealistic.

Epsilon greedy policies are a subset of a larger class of policies called Epsilon soft policies.

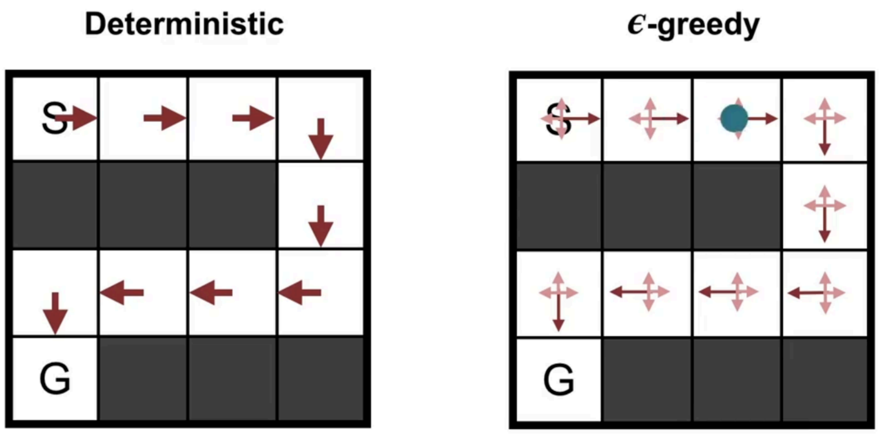
Epsilon soft policies take each action with probability at least ϵ over the number of actions.



ϵ-soft policies are always stochastic (luôn ngẫu nhiên)

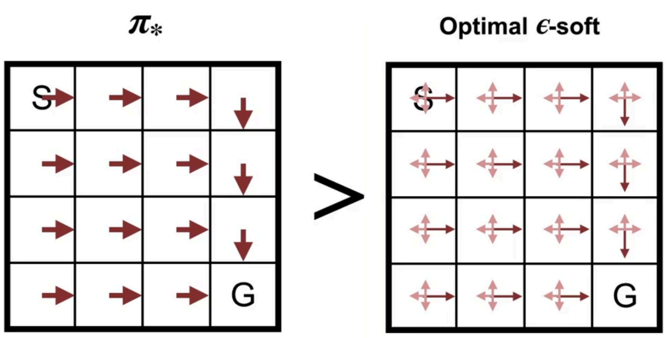


ϵ-greedy policies and deterministic policies

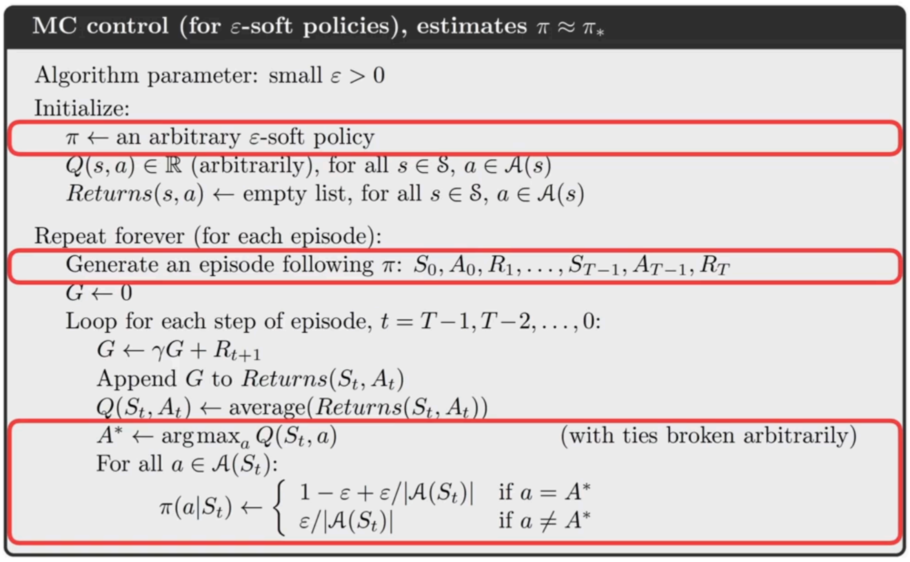


The Epsilon greedy policy has more arrows because every action has some small probability of being selected accordingly. The agent will probably follow a slightly different trajectory every episode.

ϵ-soft policies may not be optimal



Monte Carlo Control with Epsilon-Soft Policies



```
MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$
Initialize:
    $\pi \leftarrow$ an arbitrary $\varepsilon$-soft policy
    $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
    $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):
    Generate an episode following $\pi$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Append $G$ to $Returns(S_t, A_t)$
        $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
        $A^* \leftarrow \arg\max_a Q(S_t, a)$                    (with ties broken arbitrarily)
        For all $a \in \mathcal{A}(S_t)$:
            $\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$
```

Epsilon-soft policies eliminate the need for **Exploring Starts** in Monte Carlo learning.

The algorithm follows these modified steps:

1. **Start with an Epsilon-soft policy** (e.g., uniform random policy).
2. **Generate an episode** using the Epsilon-soft policy instead of Exploring Starts.
3. **Update action values based on returns** from past experiences.
4. **Improve the policy** by making it **Epsilon-greedy** with respect to learned action values.

Differences Between Exploring Starts & Epsilon-Soft Policies

- **Exploring Starts** find the **optimal deterministic policy** by directly visiting all state-action pairs.

- **Epsilon-soft policies** instead converge to the **optimal Epsilon-soft policy**, which may perform slightly worse but is easier to implement.

Practical Implications

- Although Epsilon-soft policies **may not find the best possible policy**, they allow for safe and flexible exploration.
- Future algorithms like **Q-learning** will build upon these ideas to find **optimal policies**.

Off-policy Learning for Prediction

Off-policy learning is a reinforcement learning method where the agent learns about a **target policy** different from the **behavior policy** used to generate data. This allows learning optimal behaviors without directly following the policy being optimized.

On-Policy vs Off-Policy Learning

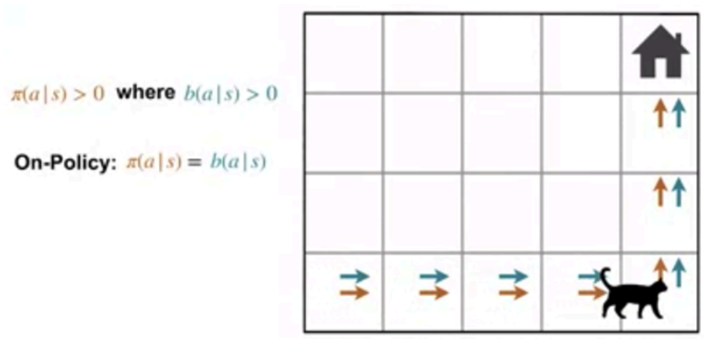
- **On-policy:** Agent learns from data generated by the same policy it is improving.
- **Off-policy:** Agent learns from data generated by a different policy (behavior policy), enabling more flexible learning.
- **Target policy (π):** The policy the agent aims to learn or improve.
- **Behavior policy (b):** The policy used to collect experience (may be exploratory or random), must sufficiently explore all actions the target policy might take.

Advantages

- Enables **learning from past experiences or demonstrations** (historical data) collected under different policies.
- Supports **continual exploration** and better coverage of state-action space.
- Allows **sample reuse** and improves **data efficiency**.
- Facilitates **parallel learning** of multiple policies.
- Useful in real-world domains where exploration is costly or risky (e.g., healthcare, robotics, finance).

Key Requirements

- The behavior policy must **cover the target policy's action space** to ensure effective learning.
    - If the target policy assigns a non-zero probability to an action in a given state, the behavior policy must also allow that action. (Note: off policy learning is a strict generalization of on policy learning on policies the specific case where the target policy is equal to the behavior policy)


$\pi(a|s) > 0$ where $b(a|s) > 0$
On-Policy: $\pi(a|s) = b(a|s)$

- Importance sampling or other correction methods are often used to handle the mismatch between behavior and target policies.

Importance Sampling

Derivation of Importance Sampling

Sample: $x \sim b$

Estimate: $\mathrm{E}_\pi[X]$

Cause x is sampled from b, cannot use the sample average to compute the expectation under Pi

$$\begin{aligned}\mathrm{E}_\pi[X] &\doteq \sum_{x \in \mathrm{X}} x\pi(x) \\ &= \sum_{x \in \mathrm{X}} x\pi(x)\frac{b(x)}{b(x)} \\ &= \sum_{x \in \mathrm{X}} x\frac{\pi(x)}{b(x)}b(x) \\ &= \sum_{x \in \mathrm{X}} x\rho(x)b(x)\end{aligned}$$

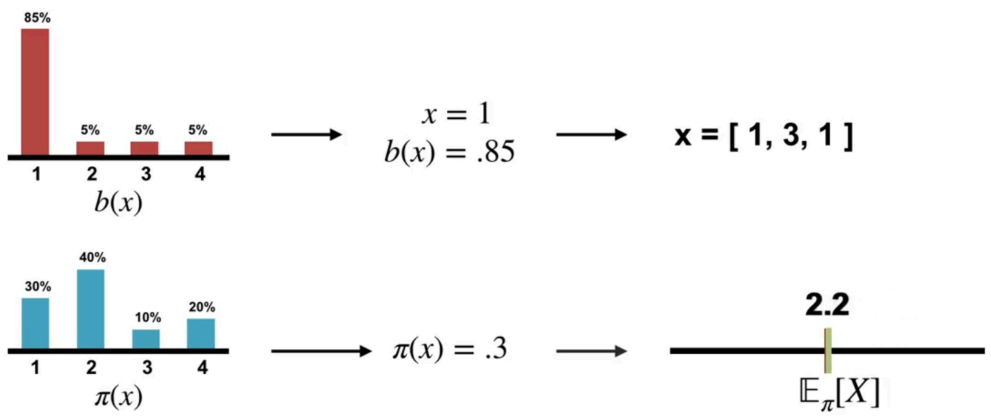
$\rho(x)$ is importance sampling ratio

$$\mathrm{E}_\pi[X] = \sum_{x \in X} x\rho(x)b(x) \qquad\qquad \rho(x) = \frac{\pi(x)}{b(x)}$$

$$= \mathrm{E}_b[X_\rho(X)] \approx \frac{1}{n}\sum_{i=1}^{n} x_i\,\rho(x_i)$$

$$x_i \sim b$$

Example:



$$\frac{1}{n}\sum_{1}^{n} x\rho(x) \rightarrow \frac{(1 \times \frac{.3}{.85}) + (3 \times \frac{.1}{.05}) + (1 \times \frac{.3}{.85})}{3} \approx 2.24$$

Off-Policy Monte Carlo Prediction

$$\begin{aligned} v_\pi(s) &\doteq \mathrm{E}_\pi\left[G_t \mid S_t = s\right] \\ &\approx \text{average}(\\ &\qquad \rho_0 \text{Returns}[0], \\ &\qquad \rho_1 \text{Returns}[1], \\ &\qquad \rho_2 \text{Returns}[2] \\ &) \end{aligned}$$

$$\rho = \frac{\text{P(trajectory under } \pi)}{\text{P(trajectory under } b)}$$

$$v_\pi(s) = \mathrm{E}_b\left[\rho G_t \mid S_t = s\right]$$

Probability distribution over episodic trajectories

Probability of starting in state S_t, taking action A_t, transitioned into state S_{t+1}, and so on ... until reaching terminal state S_T (T is terminal timestep at the end of each tasks episode) under behavior policy b

$$P(A_t, S_{t+1}, A_{t+1}, \ldots, S_T \mid S_t, A_{t:T}) =$$

$$b(A_t \mid S_t)\, p(S_{t+1} \mid S_t, A_t)\, b(A_{t+1} \mid S_{t+1})\, p(S_{t+2} \mid S_{t+1}, A_{t+1}) \;\ldots\; p(S_T \mid S_{T-1}, A_{T-1})$$

$$\mathrm{P}(\textbf{trajectory under } b) = \prod_{k=t}^{T-1} b(A_k \mid S_k)\, p(S_{k+1} \mid S_k, A_k)$$

$$\begin{aligned}\rho_{t:T-1} &\doteq \frac{\mathrm{P}(\text{trajectory under } \pi)}{\mathrm{P}(\text{trajectory under } b)} \\ &\doteq \prod_{k=t}^{T-1} \frac{\pi(A_k \mid S_k)\, p(S_{k+1} \mid S_k, A_k)}{b(A_k \mid S_k)\, p(S_{k+1} \mid S_k, A_k)} \\ &\doteq \prod_{k=t}^{T-1} \frac{\pi(A_k \mid S_k)}{b(A_k \mid S_k)}\end{aligned}$$

$$\mathrm{E}_b\left[\rho_{t:T-1} G_t \mid S_t = s\right] = v_\pi(s)$$

Off-policy every-visit MC prediction, for estimating $v \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

- $V(s) \in \mathrm{R}$, **arbitrarily, for all** $s \in \mathrm{S}$
- $\mathrm{Returns}(s) \leftarrow$ **an empty list**, for all $s \in \mathrm{S}$

Loop forever (for each episode):

- **Generate an episode following** b: $S_0, A_0, R_1, S_1, \ldots, S_{T-1}, A_{T-1}, R_T$
- $G \leftarrow 0 \quad W \leftarrow 1$
- **Loop for each step of episode,** $t = T-1, T-2, \ldots, 0$
    - $G \leftarrow \gamma W G + R_{t+1}$
    - **Append** G **to** $\mathrm{Returns}(S_t)$
    - $V(S_t) \leftarrow \textbf{average}(\mathrm{Returns}(S_t))$
    - $W \leftarrow W \dfrac{\pi(A_t \mid S_t)}{b(A_t \mid S_t)}$

Video: Emma Brunskill: Batch Reinforcement Learning

Domain that require real human-interacting data (clinic, social); need carefully collection

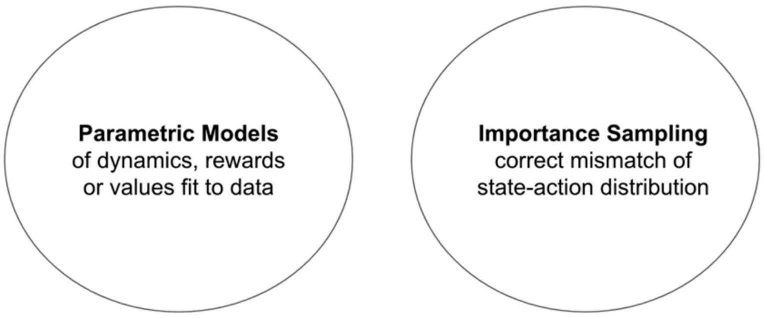
How to minimize data the agent need to learn

Issues:

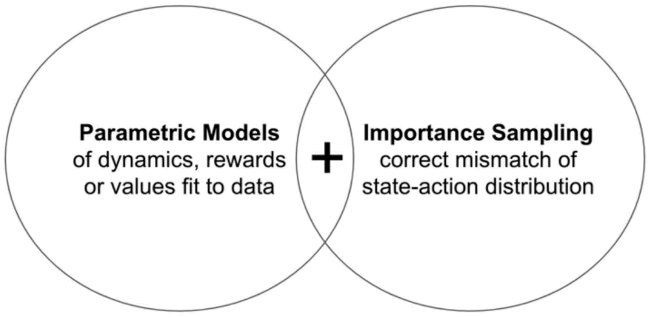
- Collect enough possible scenarios (state-action pair) for agent to learn
- Generalization

Policy Evaluation techniques issues:

- Importance sampling (for distribution mismatching (the existed behavior policy state-action distribution is different from the evaluating/learning target policy's):
    - High variance (although produces an unbiased estimator) causes by too small amount of data
- Parametric model:
    - Lower variance
    - Bias (unless realizable)



- **Doubly Robust (best of both worlds):**



    - Smaller variance than importance sampling
    - Unbiased if either model realizable or behavior policy known

Module 3

Temporal Difference (TD) learning

Temporal Difference (TD) learning is a reinforcement learning technique that combines elements of Monte Carlo methods and dynamic programming. It is a model-free approach used to estimate value functions or policies directly from experience, without requiring a model of the environment's dynamics.

Review: Estimating Values from Returns

$$v_\pi(s) \doteq \mathrm{E}_\pi\left[G_t \mid S_t = s\right]$$

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

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

TD-error (Temporal Difference Error)

The TD-error is the difference between the expected return and the current estimate. The agent updates the value estimate towards a better estimate of the true value, based on the difference between observed rewards and the predictions made by the current estimate.

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

Where:

- δ_t is the TD error at time step t.
- r_{t+1} is the immediate reward received after taking an action from state s_t and transitioning to state s_{t+1}.
- γ is the discount factor, which represents the importance of future rewards relative to immediate rewards.
- $V(s_t)$ is the estimated value of state s_t at time step t.
- $V(s_{t+1})$ is the estimated value of state s_{t+1} at time step $t + 1$.

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated

Algorithm parameter: step size $\alpha \in (0, 1]$

Initialize $V(s)$, for all $s \in \mathrm{S}^+$, arbitrarily except that $V(\text{terminal}) = 0$

Loop for each episode:

- **Initialize** S
- **Loop for each step of episode:**
    - $A \leftarrow$ action given by π for S
    - Take action A, observe R, S'
    - $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
    - $S \leftarrow S'$
- until S is terminal

Video: Rich Sutton: The Importance of TD Learning

1. What is TD Learning?

- TD Learning is a **prediction-based learning method**.
- It is **highly scalable** with computation, making it crucial for AI.
- Unlike supervised learning, TD Learning does not require labeled training sets or predefined objectives.

2. Prediction Learning vs. Supervised Learning

- **Prediction learning** involves making a prediction, waiting for the outcome, and learning from the result.
- **Supervised learning** requires a teacher or dataset providing correct answers.
- Sutton describes prediction learning as **"unsupervised supervised learning"** because it learns from real-world outcomes.

3. Biological Relevance

- TD Learning is fundamental in **neuroscience**.
- It plays a major role in **reward systems in the brain**.
- The **dopamine neurotransmitter** carries the TD error signal, driving learning.
- Other species, like bees, use **octopamine** for similar learning processes.

4. AI vs. Neuroscience Perspectives

- In AI, TD Learning is valued for its **powerful and widely applicable learning capabilities**.
- In neuroscience, TD Learning is celebrated for its **accurate modeling of animal learning behavior**.

5. Conclusion

- **Prediction learning is the key to AI**.
- **TD Learning is a specialized method** that leverages the temporal structure of learning.
- It is **important in both AI and neuroscience**, making it a crucial concept in understanding intelligence.

Advantages of TD

1. Introduction to TD Learning

- **Temporal Difference (TD) Learning** combines ideas from **dynamic programming** and **Monte Carlo methods**.
- Like **dynamic programming**, TD uses **bootstrapping** (updating estimates based on other estimates).
- Like **Monte Carlo**, TD learns **directly from experience**.

2. Real-Life Example: Predicting Travel Time

- Example: Predicting how long it takes to drive home.
- Predictions adjust dynamically based on **new observations** (e.g., rain, traffic).
- Learning happens **incrementally**, rather than waiting for the final outcome.

3. Comparison: Monte Carlo vs. TD Learning

- **Monte Carlo methods** update estimates **only at the end of an episode**.
- **TD Learning updates estimates at every step**, making it more efficient.
- TD allows **online learning**, meaning updates happen **without waiting for the final result**.

4. Advantages of TD Learning

- **Does not require a model** of the environment (unlike dynamic programming).
- **Learns directly from experience**.
- **Updates values at every step** (unlike Monte Carlo).
- **Bootstrapping improves efficiency**.

- **Converges faster** than Monte Carlo methods.

Video: Comparing TD and Monte Carlo

1. Introduction

- Temporal Difference (TD) Learning and Monte Carlo (MC) methods are compared in a scientific experiment.
- The goal is to evaluate their empirical benefits in estimating value functions.

2. Experiment Setup

- A **five-state Markov Decision Process (MDP)** is used.
- The agent follows a **uniform random policy**, starting in state C.
- Episodes terminate on the left or right, with a reward of **+1 for right termination**.
- The **discount factor is set to 1**, meaning values represent the probability of terminating on the right.

3. TD vs. Monte Carlo Updates

- **TD updates values at each step**, using the next state's value.
- **Monte Carlo updates values only at the end of an episode**, averaging returns.
- TD initially updates fewer states but eventually adjusts values at every step.

4. Learning Speed and Accuracy

- TD estimates **converge faster** toward true values.
- Monte Carlo waits until the episode ends, making it **slower to adjust**.
- TD performs **consistently better**, reducing error more efficiently.

5. Impact of Learning Rate

- **Higher learning rates** speed up convergence but result in **higher final error**.
- **Lower learning rates** slow learning but achieve **better final accuracy**.

6. Conclusion

- TD Learning **converges faster** and achieves **lower final error** than Monte Carlo in this experiment.

Video: Andy Barto and Rich Sutton: More on the History of RL

1. Early Days of Reinforcement Learning

- Rich Sutton was Andy Barto's first graduate student.
- The field of **reinforcement learning (RL)** was initially unpopular.
- Early research focused on **supervised learning**, overshadowing RL.

2. Distinction Between RL and Supervised Learning

- Early AI research in the **1950s and 1960s** explored reward-based learning but shifted toward **supervised learning**.
- Many researchers misunderstood the difference between **trial-and-error learning** and **error correction**.

3. Struggles and Persistence in RL Research

- RL was **not a recognized field** in the 1980s.
- Sutton and Barto pursued RL despite its **lack of mainstream acceptance**.
- They believed **common-sense RL principles** were being ignored.

4. Influence of Behaviorism

- RL shares elements with **behaviorism**, but pure behaviorists rejected models like **value functions**.
- Sutton learned valuable insights from behaviorist principles, which influenced **Temporal Difference (TD) Learning**.

5. Intrinsic Motivation in Research

- Sutton and Barto were **intrinsically motivated** to study RL, even when it lacked external recognition.
- They avoided **trendy research topics**, focusing on **long-term foundational ideas**.
- Intrinsic motivation plays a role in **skill-building**, similar to how animals develop competencies.

6. Conclusion

- RL has now achieved **great success**, validating their early persistence.
- Sutton and Barto celebrate the progress of RL as a **recognized and impactful field**.