Reinforcement Learning
Learning and Planning

S. M. Ahsan Kazmi

Recap

Last lecture:

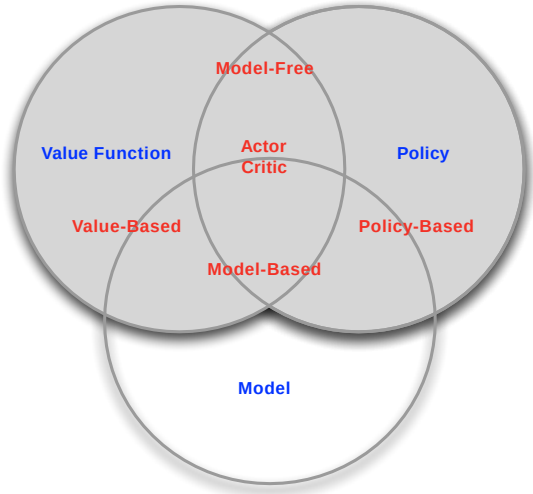
- Function Approximation
  - Motivation
  - Incremental Methods
  - Batch Methods

This lecture:

- Model-Based RL
  - Learn a model from experience
  - Plan value functions using the learned model

Value-Based and Policy-Based RL

- Value Based
  - No Policy (Implicit)
  - Value Function
- Policy Based
  - Policy
  - No Value Function
- Actor Critic
  - Policy
  - Value Function



Model-Based Reinforcement Learning

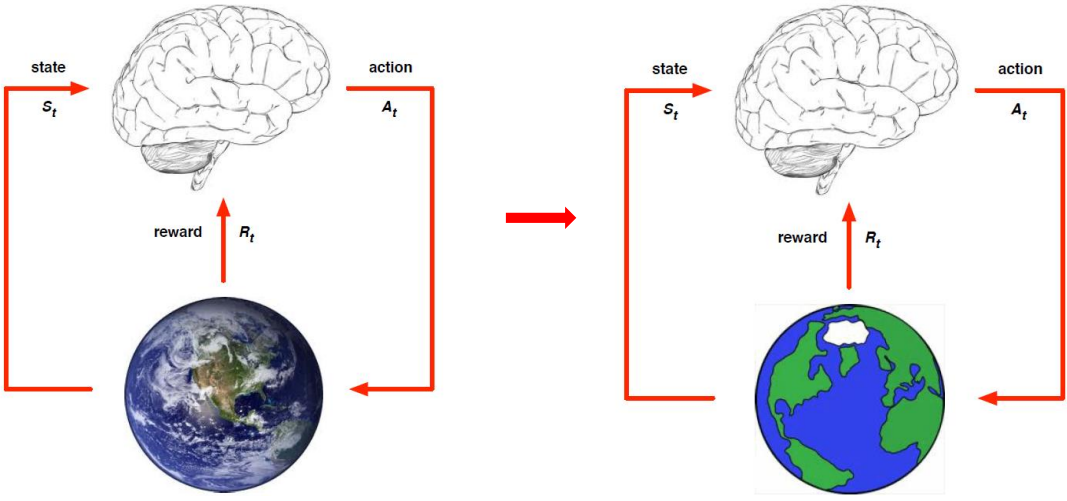
Model-Based and Model-Free RL

- Learn the *model directly from experience* and use planning to construct a value function or policy

- Model-Free RL
  - No model
  - Learn value function (and/or policy) from experience

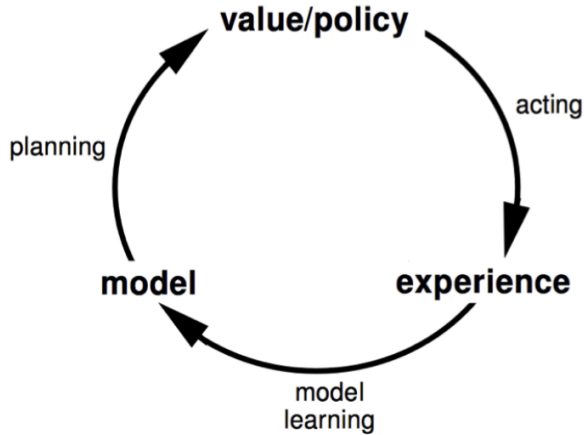
- Model-Based RL
  - Learn a model from experience
  - Plan value function (and/or policy) from model

Model-Free RL to Model-Based RL



Model-Based RL

- Advantages:
  - Can efficiently learn models by supervised learning methods
  - Can reason about model uncertainty
- Disadvantages:
  - First learn a model, then construct a value function
  - two sources of the approximation error



What is a Model?

- A *model* $\mathcal{M}$ is a representation of an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, parametrized by η
- We will assume state space $\mathcal{S}$ and action space $\mathcal{A}$ are known
- So a model $\mathcal{M} = \langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$ represents state transitions $\mathcal{P}_\eta \approx \mathcal{P}$ and rewards $\mathcal{R}_\eta \approx \mathcal{R}$

$$S_{t+1} \sim \mathcal{P}_\eta(S_{t+1} \mid S_t, A_t)$$
$$R_{t+1} = \mathcal{R}_\eta(R_{t+1} \mid S_t, A_t)$$

- Typically assume conditional independence between state transitions and rewards

$$\mathbb{P}\left[S_{t+1}, R_{t+1} \mid S_t, A_t\right] = \mathbb{P}\left[S_{t+1} \mid S_t, A_t\right] \mathbb{P}\left[R_{t+1} \mid S_t, A_t\right]$$

Model Learning

- Goal: estimate model $\mathcal{M}_\eta$ from experience $\{S_1, A_1, R_2, ..., S_T\}$
- This is a supervised learning problem

$$
\begin{aligned}
S_1, A_1 &\rightarrow R_2, S_2 \\
S_2, A_2 &\rightarrow R_3, S_3 \\
&\vdots \\
S_{T-1}, A_{T-1} &\rightarrow R_T, S_T
\end{aligned}
$$

- Learning $s, a \rightarrow r$ is a *regression* problem
- Learning $s, a \rightarrow s'$ is a *density estimation* problem
- Pick loss function, e.g. mean-squared error, KL divergence, ...
- Find parameters η that minimise empirical loss

Examples of Models

- We typically decompose the dynamics into separate parametric functions
  - transition and reward dynamics

- For each of these we can then consider different options:
  - Table Lookup Model
  - Linear Expectation Model
  - Deep Neural Network Model
  - Linear Gaussian Model
  - Gaussian Process Model
  - Deep Belief Network Model
  - …

Table Lookup Model

- Model is an explicit MDP, $\hat{\mathcal{P}}, \hat{\mathcal{R}}$
- Count visits $N(s, a)$ to each state action pair

$$\hat{\mathcal{P}}^a_{s,s'} = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbf{1}(S_t, A_t, S_{t+1} = s, a, s')$$

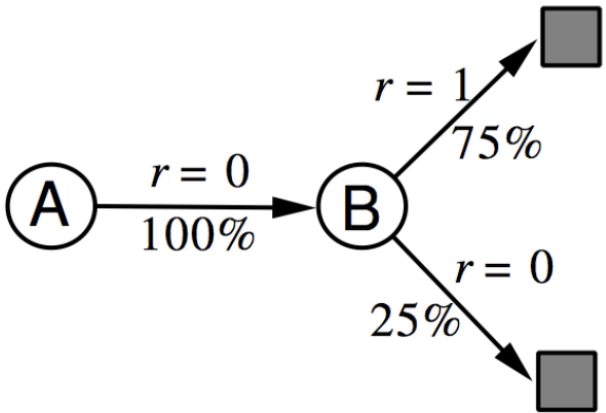
$$\hat{\mathcal{R}}^a_s = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbf{1}(S_t, A_t = s, a) R_t$$

- Alternatively
  - At each time-step t, record experience tuple $\langle S_t, A_t, R_{t+1}, S_{t+1} \rangle$
  - To sample model, randomly pick tuple matching $\langle s, a, \cdot, \cdot \rangle$

Example

Two states A, B; no discounting; 8 episodes of experience

A, 0, B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0



We have constructed a table lookup model from the experience

Planning with a Model

- Given a model $\mathcal{M}_\eta = \langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$
- Solve the MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$
- Using favourite planning algorithm
  - Value iteration
  - Policy iteration
  - Tree search
  - ...

Sample-Based Planning with a learned Model

- A simple but powerful approach to planning
- Use the model only to generate samples
- Sample experience from model

$$S_{t+1} \sim \mathcal{P}_\eta(S_{t+1} \mid S_t, A_t)$$
$$R_{t+1} = \mathcal{R}_\eta(R_{t+1} \mid S_t, A_t)$$

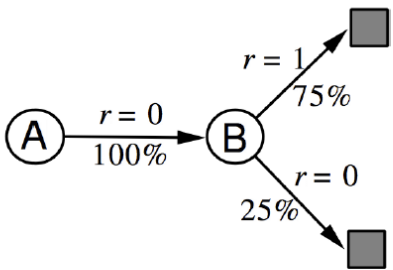
- Apply model-free RL to samples, e.g.:
  - Monte-Carlo control
  - Sarsa
  - Q-learning
- Sample-based planning methods are often more efficient

AB Example

- Construct a table-lookup model from real experience
- Apply model-free RL to sampled experience

Real experience

A, 0, B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0



Sampled experience

B, 1
B, 0
B, 1
A, 0, B, 1
B, 1
A, 0, B, 1
B, 1
B, 0

e.g. Monte-Carlo learning: $V(A) = 1, V(B) = 0.75$

Limits of Planning with an Inaccurate Model

- Given an imperfect model $\langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle \neq \langle \mathcal{P}, \mathcal{R} \rangle$
- Performance of model-based RL is limited to optimal policy for approximate MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$

  i.e. Model-based RL is only as good as the estimated model
- When the model is inaccurate, planning process will compute a suboptimal policy
- Solution 1: when model is wrong, use model-free RL
- Solution 2: reason explicitly about model uncertainty (e.g. Bayesian methods)

Integrating Learning and Planning

Real and Simulated Experience

- We consider two sources of experience
- Real experience Sampled from environment (true MDP)

$$S' \sim \mathcal{P}^a_{s,s'}$$
$$R = \mathcal{R}^a_s$$

- Simulated experience Sampled from model (approximate MDP)

$$S' \sim \mathcal{P}_\eta(S' \mid S, A)$$
$$R = \mathcal{R}_\eta(R \mid S, A)$$

Integrating Learning and Planning

Model-Free RL

- No model
- Learn value function (and/or policy) from real experience

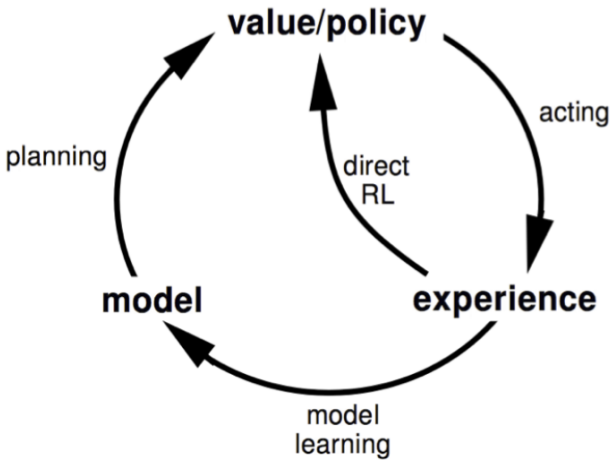
Model-Based RL (using Sample-Based Planning)

- Learn a model from real experience
- Plan value function (and/or policy) from simulated experience

Dyna

- Learn a model from real experience
- Learn and plan value function (and/or policy) from real and simulated experience

Dyna Architecture



Dyna-Q Algorithm

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Do forever:

(a) $S \leftarrow$ current (nonterminal) state

(b) $A \leftarrow \varepsilon$-greedy(S,Q)

(c) Execute action A; observe resultant reward, R, and state, S'

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

(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)

(f) Repeat n times:

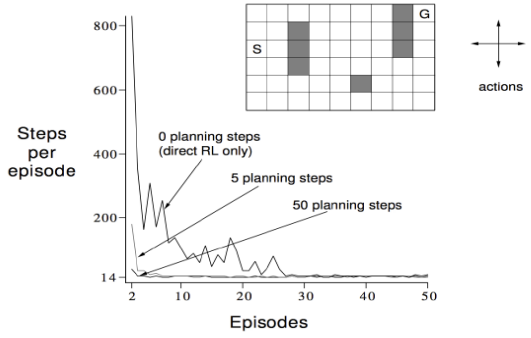
$S \leftarrow$ random previously observed state

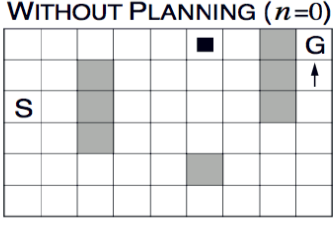
$A \leftarrow$ random action previously taken in S

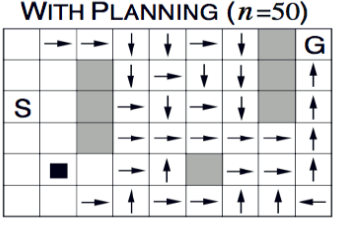
$R, S' \leftarrow Model(S,A)$

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

Dyna-Q on a Simple Maze







Dyna-2

- In Dyna-2, the agent stores two sets of feature weights
  - Long-term memory
  - Short-term (working) memory
- Long-term memory is updated from real experience using TD learning
  - General domain knowledge that applies to any episode
- Short-term memory is updated from simulated experience using TD search
  - Specific local knowledge about the current situation
- Over value function is sum of long and short-term memories

Policy-Based Reinforcement Learning

Policy Gradient Methods?

- Before: Learn the values of actions and then select actions based on their estimated action values. The policy was generated directly from the value function.
- We want to learn a parameterized policy that can select actions without consulting a value function. The parameters of the policy are called policy weights
- A value function may be used to learn the policy weights, but this is not required for action selection
- Policy gradient methods are methods for learning the policy weights using the gradient of some performance measure with respect to the policy weights
- Policy gradient methods seek to maximize performance and so the policy weights are updated using gradient ascent

Policy-Based Reinforcement Learning

- In the last lecture we approximated the value or action-value function using parameters θ,

$$V_\theta(s) \approx V^\pi(s)$$
$$Q_\theta(s, a) \approx Q^\pi(s, a)$$

- A policy was generated directly from the value function
  e.g. using ϵ-greedy
- In this lecture we will directly parametrise the policy

$$\pi_\theta(s, a) = \mathbb{P}[a \mid s, \theta]$$

- We will focus again on model-free reinforcement learning

Policy Optimisation

- Policy based reinforcement learning is an optimisation problem
- Find θ that maximises $J(\theta)$
- Some approaches do not use gradient
  - Hill climbing
  - Simplex / amoeba / Nelder Mead
  - Genetic algorithms
- Greater efficiency often possible using gradient
  - Gradient descent
  - Conjugate gradient
  - Quasi-newton

Example of Policy Gradient in Action

Suppose a game where a ball is thrown toward a moving target, and the goal is to hit the target by choosing the correct angle (π_θ).

Policy: Represent the policy as π_θ , where π_θ is the angle of the throw?
Reward: A reward of +10 for hitting the target and 0 otherwise.

Training:

- Randomly sample angles to throw the ball (exploration).
- Record outcomes and rewards for each angle.
- Adjust π_θ to increase the probability of successful angles using the policy gradient.

Advantages of Policy-Based RL

- Sometimes learning a model is easier (e.g., simple dynamics)
- Sometimes learning a policy is easier (e.g., "always move forward" is optimal)

- Advantages:
  - Better convergence properties
  - Effective in high-dimensional or continuous action spaces
  - Can learn stochastic policies

- Disadvantages:
  - Typically converge to a local rather than global optimum
  - Evaluating a policy is typically inefficient and high variance

Stochastic policies

- Why could we need stochastic policies?
  - In MDPs, there is always an optimal deterministic policy
  - But, most problems are not fully observable
  - This is the common case, especially with function approximation
  - The optimal policy may then be stochastic

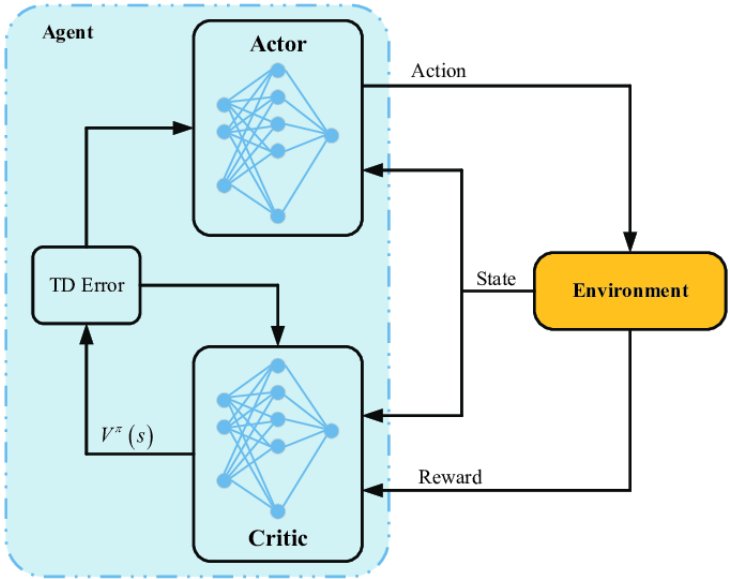
- Provides some ‘exploration’ during learning

When Policy-Based Methods Shine

Scenario	Why Policy-Based is Better
Continuous Action Spaces	Avoids discretization, enabling precise control.
Stochastic Policies	Naturally models randomness for strategies like exploration or mixed strategies.
Sparse Rewards	Focuses directly on optimizing trajectories, bypassing inefficient value estimations.
Multi-Agent Environments	Adapts dynamically to other agents' stochastic behaviors, promoting collaboration.

Actor-Critic Reinforcement Learning

Actor-Critic Methods



Actor-Critic Methods

- Methods that learn approximations to both policy and value functions are called actor-critic methods
- Actor refers to the learned policy
- Critic refers to the learned value function, which is usually a state-value function
- The critic is bootstrapped – the state values are updated using the estimated state values of subsequent states
- The number of steps in the actor-critic method controls the degree of bootstrapping

Actor-Critic Methods

- A critic is a value function, learned via policy evaluation

- This problem was explored in our previous sessions:
  - Monte-Carlo policy evaluation
  - Temporal-Difference learning
  - n-step TD

- Critic: Update parameters w of v_w by TD (e.g., one-step) or MC
- Actor: Update θ by policy gradient

One-step Actor-Critic (episodic)

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta}), \forall a \in \mathcal{A}, s \in \mathcal{S}, \boldsymbol{\theta} \in \mathbb{R}^n$
Input: a differentiable state-value parameterization $\hat{v}(s,\mathbf{w}), \forall s \in \mathcal{S}, \mathbf{w} \in \mathbb{R}^m$
Parameters: step sizes $\alpha > 0$, $\beta > 0$

Initialize policy weights $\boldsymbol{\theta}$ and state-value weights $\mathbf{w}$
Repeat forever:
  Initialize S (first state of episode)
  $I \leftarrow 1$
  While S is not terminal:
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \beta \delta \nabla_{\mathbf{w}} \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha I \delta \nabla_{\boldsymbol{\theta}} \log \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

Policy gradient variations

- Many extensions and variants exist to stabilize learning
- Take care: bad policies lead to bad data
- This is different from supervised learning (where learning and data are independent)
  - Trust Region Policy Optimization (TRPO, Schulmanetal, 2015)
  - Proximal Policy Optimization (PPO, Abbeel & Schulman, 2016)
  - Maximum Posteriori Policy Optimisation (MPO, Abdol Malekietal, 2018)

Thanks