Lecture 7

03.11.2025

Today's plan and announcements

- Hour 1: stochastic optimal control and reinforcement learning introduction
- Hour 2: policy gradient approach to reinforcement learning

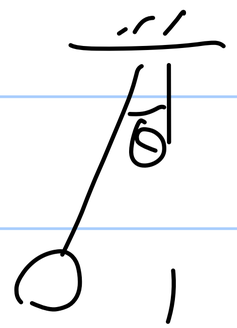
- Exercise hour this week
  - Problem set 4 & Problem Set 3
  - You can also ask questions on Problem set 3 and the python homework

Where we are

- Machine learning topics we look at
- Supervised learning $\{x_i, y_i\}$
- Reinforcement learning
- Unsupervised learning

Dynamical systems review problem

Consider the inverted pendulum model in state-space form $\ddot{\theta}(t) = mg\sin(\theta(t))$



- Discretize the dynamics.

- Derive the equilibria of the system.

Recall: a state $s \in \mathbb{R}^n$ is an equilibrium of the discrete-time dynamical system $s_{t+1} = f(s_t)$ if $s_e = f(s_e)$ (implies $s_t = s_e \ \forall t$). Contrast to continuous time: $\dot{s}(t) = f(s_e) = 0_{n\times 1}$.

- Assume there is additive uncertainty affecting the position and velocity of the pendulum, $w_1 \in \mathcal{N}(0,\sigma_1), w_2 \in \mathcal{N}(0,\sigma_2)$, respectively. Derive the stochastic dynamical system model of the system: $s_{t+1} \sim P(\cdot \mid s_t)$

Dynamical systems review problem solution

Discretization

$s_1 = \theta$, $s_2 = \dot{\theta}$

$$\left.\begin{aligned} \dot{s}_1 &= s_2 \\ \dot{s}_2 &= mg \sin(s_1) \end{aligned}\right\}$$ state-space dynamics

$$\left.\begin{aligned} s_{1,k+1} &= s_{1,k} + \delta s_{2,k} \\ s_{2,k+1} &= s_{2,k} + \delta mg \sin(s_{1,k}) \end{aligned}\right\}$$ Euler discretization with discretization step δ

$s_{k+1} = f_{\text{discrete}}(s_k)$. (there's no control input here)

Dynamical systems review problem solution

Equilibria

Let $s_e = (s_{1,e}, s_{2,e})$ be the equilibrium.

$$s_{1,k+1} = s_{1,k} + \delta s_{2,k}$$

$$s_{2,k+1} = s_{2,k} + \delta m \sin(s_{1,k})$$

$$s_{1,e} = s_{1,e} + \delta s_{2,e} \Rightarrow s_{2,e} = 0 \Rightarrow \text{angular velocity } 0$$

$$s_{2,e} = s_{2,e} + \delta mg \sin(s_{1,e}) \Rightarrow$$

$$\sin(s_{1,e}) = 0 \Rightarrow s_{1,e} = 0, \pi, 2\pi, \ldots$$

Dynamical systems review problem solution

Discrete-time stochastic dynamics

uncertainty / noise

$$s_{1,k+1} = s_{1,k} + \delta s_{2,k} + w_1$$

$$s_{2,k+1} = s_{2,k} + \delta\, mg \sin(s_{1,k}) + w_2$$

variance of the noise

$$w_i \sim N(0, \sigma_i^2)$$

$$s_{k+1} = \begin{bmatrix} s_{1,k+1} \\ s_{2,k+1} \end{bmatrix} \sim N\left(\begin{bmatrix} s_{1,k} + \delta s_{2,k} \\ s_{2,k} + \delta mg \sin s_{1,k} \end{bmatrix}, \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix} \right)$$

mean of distribution : deterministic

Stochastic optimal control: dynamic programming

expectation (average)

$$J(s_0) = \min \mathbb{E}[\sum_{t=0}^{K-1} c(s_t, a_t) + c_K(s_K)], \quad s_{t+1} \sim P(\cdot \mid s_t, a_t)$$, $\pi_k : S \longrightarrow A$

- Principle of optimality: tail of an optimal policy is optimal for the tail subproblem
- Define a cost-to-go function $V_t(s) = \mathbb{E}[\sum_{k=t}^{K-1} c(s_k, a_k)] + c_K(s_K)$
- Then, the optimal cost-to-go and optimal control policy is computed by a backward recursion starting from the final time $V_K(s) = c_K(s)$.
- Theory extends to infinite horizon, objectives : $E[\sum_{t=0}^{\infty} c(s_t, a_t)]$

Solving the stochastic optimal control problem

- Dynamic programming
  - Can be used to tractably solve the problem for few special cases (small dimensions, linear dynamics and quadratic costs)
  - For more general cases, need for approximation approaches

background theory for

- Reinforcement learning: an approximation approach for dynamic programming
  - It can be model-free, useful if model is complex or high dimensional
  - Provable theory for few special cases
  - Requires a very large number of samples

Reinforcement Learning

Recent advances of reinforcement learning

2013

Atari

Deep Q-learning for Atari games [1].

2016

Energy saving

DeepMind AI reduces Google data centre cooling bill by 40% [3].

2017

AlphaGo/ AlphaZero

AI achieving grand master level in **chess, go,** and **shogi** [4,5].

2018

OpenAI Five

Training five artificial intelligence agents to play the **Dota 2** [6].

2019

Alpha Star

AI achieving grand master level in **StarCraft II** game [7].

Rubik's Cube

Solving **Rubik's Cube** with a human-like robot hand [8].

2022

AlphaTensor

Discovering faster **matrix multiplication** algorithms [9].

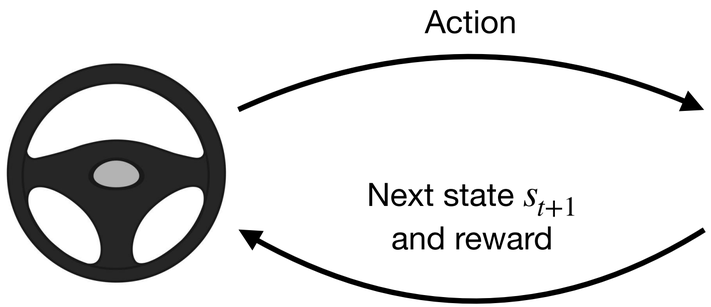
ChatGPT

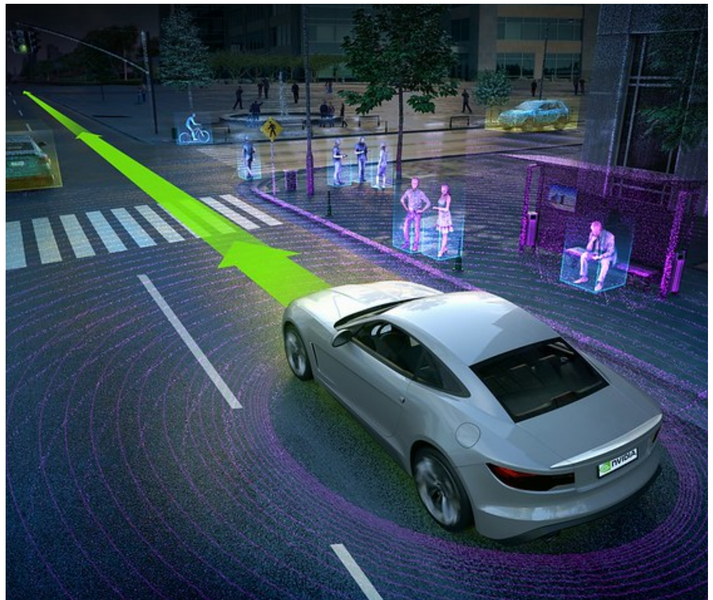
A **language model** trained to generate human-like responses to text input [10].

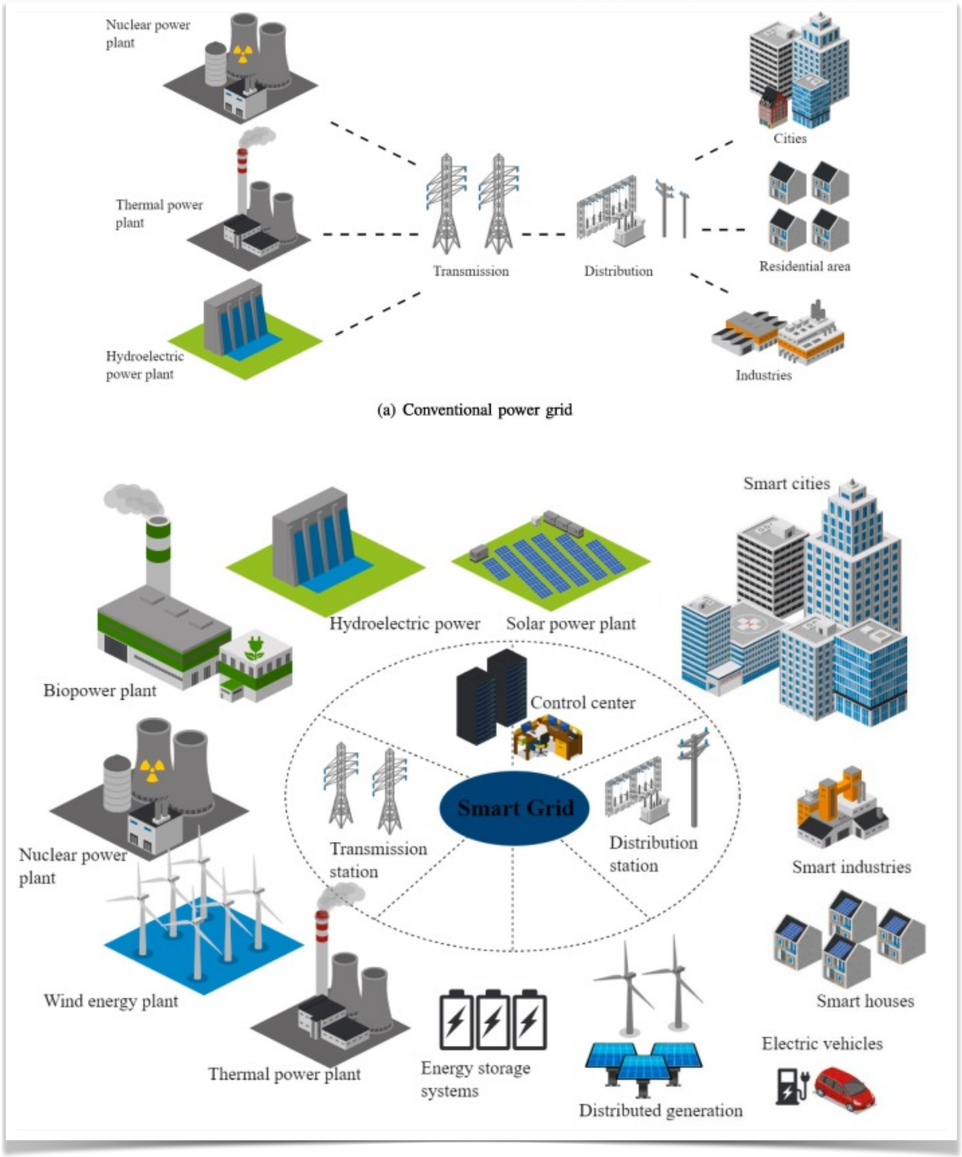
to date,
impressive success
but mainly in
non-safety-critical systems

Potential Applications

Teaching robot to walk [11]







Control of power grids [12].

Markov decision process

A **Markov decision process (MDP)** is specified by a tuple (S, A, P, ρ) where…

- S: set of possible states (state space)
- A: set of possible actions (action space)
- $P(\cdot \mid s, a)$: probabilistic transition law (transition kernel), a probability measure over the next state

  Finite state space, probability mass: $P(s' \mid s, a) \geq 0, \sum_{s' \in S} P(s' \mid s, a) = 1, \forall a \in A, s \in S$

  Continuous state space, probability density: $P(s' \mid s, a) \geq 0, \int_S P(s' \mid s, a) ds = 1, \forall a \in A, s \in S$

- ρ: initial state distribution, a probability measure over the state space (similarly, a probability mass or density function depending on the state space)

Markov property

- Future depends on the present, not how we got to the present

$$P(s_{t+1} \mid s_t, s_{t-1}, \ldots, s_0, a_t, a_{t-1}, \ldots, a_0) = P(s_{t+1} \mid s_t, a_t)$$

- State and action at each time are sufficient to determine probabilistic evolution of the system (similar to dynamical systems in state-space form)

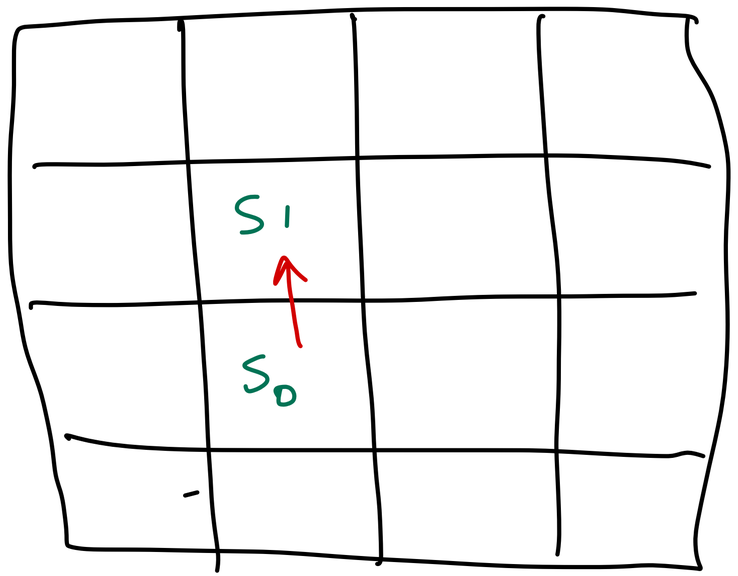
- Implies a Markov decision process is equivalent to a discrete-time stochastic dynamical system (under mild conditions on the noise affecting the discrete-time system

Example: Stochastic grid world

Time evolution

- Start in $s_0 \sim \rho$
- At each time t
  - take action $a_t \in A = \{\uparrow, \downarrow, \rightarrow, \leftarrow\}$
  - state transitions probabilistically:
    $s_{t+1} \sim P(\cdot \mid s_t, a_t)$

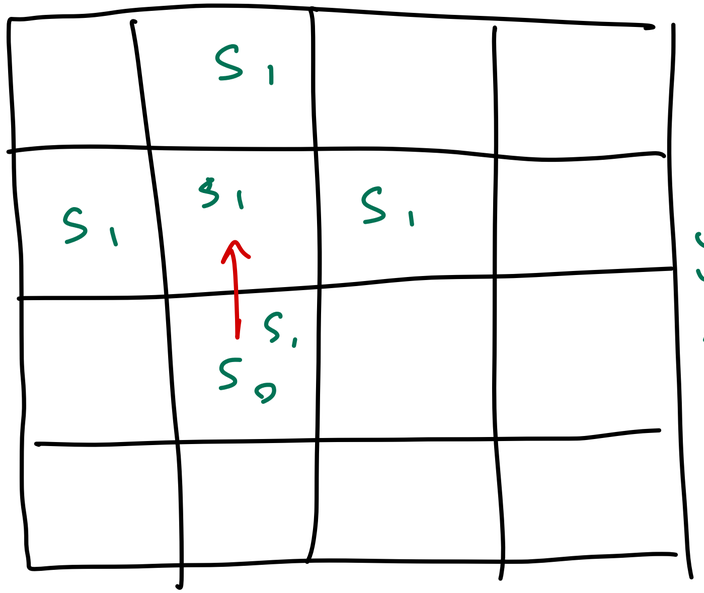
16 states, actions: $\{\uparrow, \downarrow, \rightarrow, \leftarrow\}$



Deterministic

Motivation: perhaps in the robot dynamics, such as the unicycle model, there is some noise (imperfect modeling, disturbances affecting the system)

with some probability $1-p$, you (and in the intended grid point, and with $\frac{p}{N}$ you end up in a neighboring grid, where N is # of neighboring grids.



stochastic

Control problem

Same as a stochastic optimal control, but in reinforcement learning, we use “Reward” instead of cost, namely, rather than minimizing a cost, we maximize a reward

Reward and discount factor

- Reward function $r : S \times A \to \mathbb{R}$
- Discount rate $\gamma \in (0,1)$, to ensure cost over infinite horizon is summable.

Objective function (we can also formulate finite horizon problems similar to stochastic control)

The goal is to find a policy $\pi : S \to A$ maximizing

$$J(\pi) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,|\, s_0 \sim \rho, a_t = \pi(s_t), s_{t+1} \sim P(\,\cdot\,|\, s_t, a_t)\right]$$

Reinforcement learning vs. stochastic control

- Same dynamics and objective (maximizing a function $f(x)$ is equivalent to minimizing $-f(x)$)

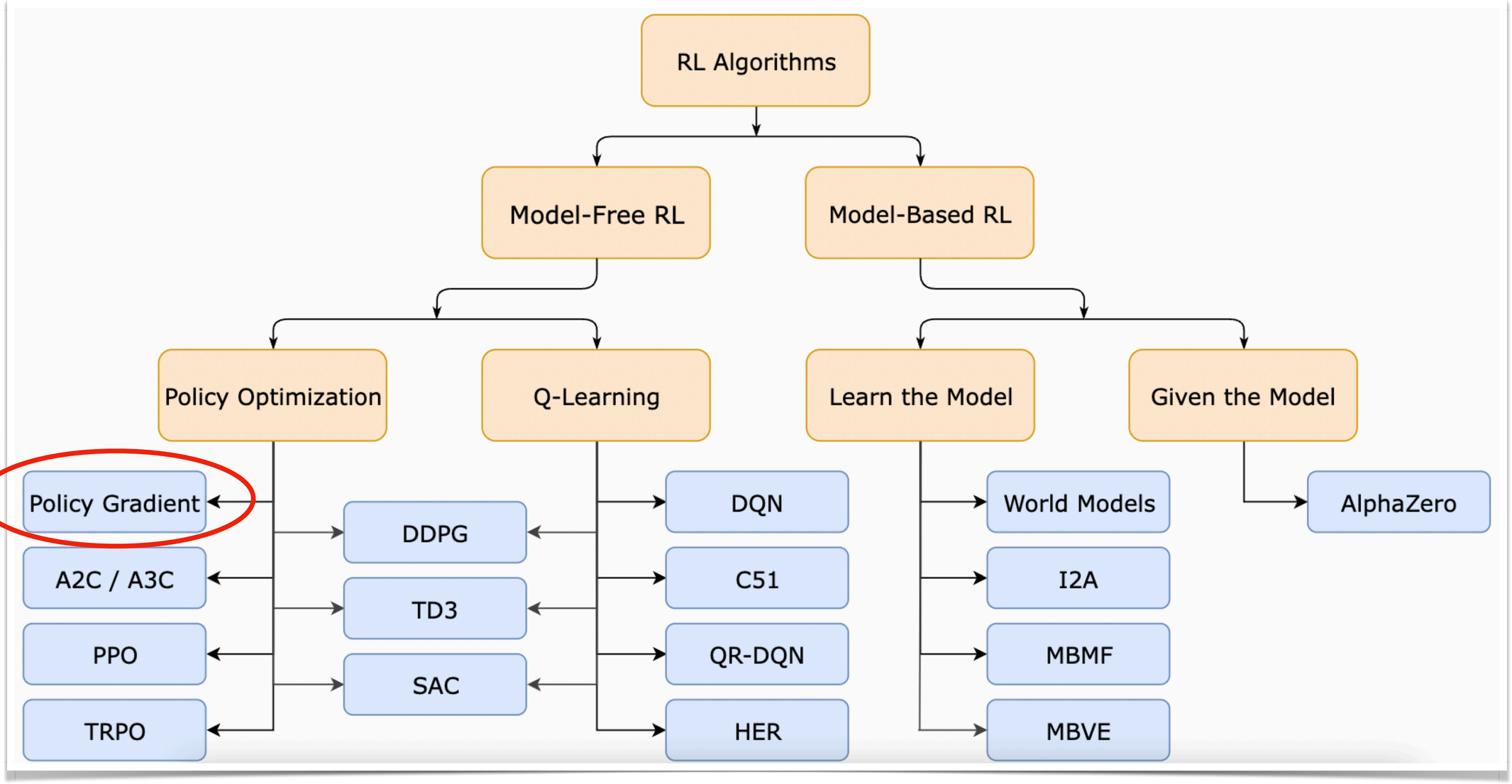
$$J(\pi) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,|\, s_0 \sim \rho, a_t = \pi(s_t), s_{t+1} \sim P(\,\cdot\,|\, s_t, a_t)\right]$$

control policy : map from state to action

- Stochastic control: assumes known model. Focuses on proving existence of optimal policy, characterizing the optimal policy, and computation of the optimal policy

- Reinforcement learning: assumes unknown model. Uses simulator or real-system data to compute the ophmal policy.

Reinforcement learning approaches



Taxonomy of reinforcement learning approaches [14].

Policy gradient approach to reinforcement learning

Class of policies to optimize over

- Search over all functions $\pi : S \to A$ is hopeless (too many functions)
  - except for specific problem classes such as linear dynamics and quadratic costs

- Policy Optimization: parameterize the policy as $\pi_\theta(s)$, where $\theta \in \mathbf{R}^d$

$$\max_\theta J(\pi_\theta) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,|\, s_0 \sim \rho, a_t = \pi_\theta(s_t)\right]$$

- Above is a finite dimensional optimization

Stochastic policy parametrization

- In reinforcement learning, we often search for stochastic policies
  - ensures gradients are well-defined (we will see this later)
  - ensures sufficient exploration (since we assume we don't have a model)
  - Gridworld example: at each grid point, choose a direction with some probability

- The policy outputs a distribution over actions
  - Finite action space: probability mass function over actions

$$\pi(a \mid s) \geq 0, \forall a \in A, \sum_{a \in A} \pi(a \mid s) = 1, \forall s \in S$$

  - Continuous action space: probability density function over actions

$$\pi(a \mid s) \geq 0, \forall a \in A, \int_{A} \pi(a \mid s) da = 1, \forall s \in S$$

Examples of policy parameterization (continued)

- Direct policy parametrization, needs finite state and action spaces , example : grid world

$\pi_\theta(a \mid s) = \theta_{s,a}$, $\theta \in \mathbb{R}^{|S| \times |A|}$, $\theta_{s,a} \geq 0$ and $\sum_{a \in A} \theta_{s,a} = 1$, $\forall s \in S$.

- Softmax parameterization: needs finite state and action

$$\pi_\theta(a \mid s) = \frac{\exp\left(\theta_{s,a}\right)}{\sum_{a' \in A} \exp\left(\theta_{s,a'}\right)}, \theta \in \mathbb{R}^{|S| \times |A|}$$, observe $\pi_\theta(a|s) \geq 0$, $\sum_{a \in A} \pi_\theta(a|s) = 1$

- Neural softmax parameterization, $\theta \in \mathbb{R}^d$, finite action set A

$$\pi_\theta(a \mid s) = \frac{\exp\left(f_\theta(s, a)\right)}{\sum_{a' \in A} \exp\left(f_\theta(s, a')\right)}$$, where $f_\theta(s, a)$ represents a neural network

- Gaussian policy parameterization, $\theta \in \mathbb{R}^d$, general state and action set

$\pi_\theta(a \mid s) \sim \mathcal{N}(\mu_\theta(s), \Sigma_\theta(s))$

Policy gradient approach

A finite dimensional optimization problem

$$\max_{\theta} J(\pi_\theta) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,|\, s_0 \sim \rho, a_t \sim \pi_\theta(\cdot \,|\, s_t)\right]$$

- Parameterize the policy as $\pi_\theta(a \,|\, s)$
- Using gradient ascent method to find best policy π^*

Pseudo-code for policy gradient method

$K \leftarrow$ number of training iterations, θ initialized randomly, $\alpha \leftarrow$ step size
for $i = 1, 2, \ldots, K$ **do**
    Calculate the gradient $\nabla_\theta J(\pi_\theta)$
    $\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta)$
end for

Policy gradient approach

1. How do we compute the gradients of the objective function? $\nabla_\theta J(\theta)$
2. Does it converge to the optimal policy? function is non. convex.

$$\max_\theta J(\pi_\theta) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,|\, s_0 \sim \rho, a_t \sim \pi_\theta(\,\cdot\,|\,s_t)\right]$$

$K \leftarrow$ number of training iterations, θ initialized randomly, $\alpha \leftarrow$ step size
for $i = 1, 2, \ldots, K$ **do**
    Calculate the gradient $\nabla_\theta J(\pi_\theta)$
    $\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta)$
end for

Deriving the gradient

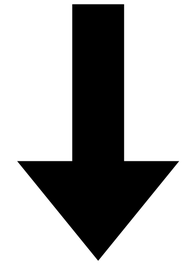
- Truncate the trajectory to finite horizon

- Compute the gradient with respect to the parameters using the policy gradient theorem -> we will go through this

- Make a limiting argument for the horizon approaching infinite (we don't go through this step)

$$J(\pi_\theta) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,|\, s_0 \sim \rho, a_t \sim \pi_\theta(\,\cdot\,|\, s_t)\right]$$

$$J(\pi_\theta) \approx J_H(\pi_\theta)$$



$$J_H(\pi_\theta) := \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t r(s_t, a_t) \,|\, s_0 \sim \rho, a_t \sim \pi_\theta(\,\cdot\,|\, s_t)\right]$$

Policy gradient theorem

$$\nabla_\theta J_H(\pi_\theta) = \mathbb{E}_{\tau_H \sim p_\theta}\left[\left(\sum_{t=0}^{H} \gamma^t r(s_t, a_t)\right) \times \left(\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right)\right]$$

discount factor $\gamma \in (0,1) \subset \mathbb{R}$

it is a constant.

$\tau_H \sim p_\theta$ → trajectory of the system τ_H, with probability p_θ.

- To prove the above, we will
  - Use Markovian property to derive probability of a trajectory
  - Use an identity for gradient of expectation
  - Do some algebra

Probability of a trajectory

- Trajectory over a finite horizon: $\tau_H = \left(s_0, a_0, s_1, a_1, \ldots, s_H, a_H, s_{H+1}\right)$

Fact 1: probability of the trajectory: $p_\theta\left(\tau_H\right) := \rho\left(s_0\right) \prod_{t=0}^{H} \pi_\theta\left(a_t \mid s_t\right) P\left(s_{t+1} \mid s_t, a_t\right).$

- Proof: by induction, $K = 0$

Observe: $P(s_1, a_0, s_0) = P(s_1 \mid a_0, s_0) P(a_0, s_0) = P(s_1 \mid a_0, s_0) \pi_\theta(a_0 \mid s_0) \rho(s_0)$

Bayes rule

Bayes rule again

Induction step

Suppose $P_\theta(\tau_{K-1}) = \rho(s_0) \prod_{t=0}^{K-1} \pi_\theta(a_t \mid s_t) P(s_{t+1} \mid s_t, a_t)$. We want to show

$$P_\theta(\tau_K) = \rho(s_0) \prod_{t=0}^{K} \pi_\theta(a_t \mid s_t) P(s_{t+1} \mid s_t, a_t)$$

Probability of a trajectory

- Trajectory over a finite horizon: $\tau_H = \left(s_0, a_0, s_1, a_1, \ldots, s_H, a_H, s_{H+1}\right)$

Fact 1: probability of the trajectory: $p_\theta\left(\tau_H\right) := \rho\left(s_0\right) \prod_{t=0}^{H} \pi_\theta\left(a_t \mid s_t\right) P\left(s_{t+1} \mid s_t, a_t\right).$

- Proof: for induction step :

$$p_\theta(\tau_{k+1}) = p(s_0, a_0, s_1, a_1, \cdots, s_k, a_k, s_{k+1})$$

$$= p(s_{k+1} \mid a_k, s_k, a_{k-1}, s_{k-1}, \cdots, s_0, a_0)\, p(a_k, s_k, a_{k-1}, \cdots, s_{k-1}, \cdots, s_0, a_0)$$

↓ by Markov property

$$= p(s_{k+1} \mid a_k, s_k)\, p(a_k, s_k, a_{k-1}, \cdots, s_0, a_0)$$

$$= p(s_{k+1} \mid a_k, s_k)\, p(a_k \mid s_k, a_{k-1}, \cdots, s_0, a_0)\, p(s_k, a_{k-1}, \cdots, s_0, a_0)$$

Probability of a trajectory

- Trajectory over a finite horizon: $\tau_H = \left(s_0, a_0, s_1, a_1, \ldots, s_H, a_H, s_{H+1}\right)$

Fact 1: probability of the trajectory: $p_\theta\left(\tau_H\right) := \rho\left(s_0\right) \prod_{t=0}^{H} \pi_\theta\left(a_t \mid s_t\right) P\left(s_{t+1} \mid s_t, a_t\right).$

- Proof: induction step, continued

$= P(s_{k+1} \mid a_k, s_k)\, \pi_\theta(a_k \mid s_k)\, P(s_k, a_{k-1}, \ldots, s_0, a_0)$

↑ since for $k-1$ the statement holds.

$= P(s_{k+1} \mid a_k, s_k)\, \pi_\theta(a_k \mid s_k) \prod_{t=0}^{k} P(s_t \mid s_{t-1}, a_{t-1})\, \pi_\theta(a_t \mid s_t)\, \rho(s_0)$

$= \rho(s_0) \prod_{t=0}^{k} \pi_\theta(a_t \mid s_t)\, P(s_{t+1} \mid s_t, a_t)$, as desired.

Expected reward of a trajectory

- Reward from trajectory $\tau_H = \left(s_0, a_0, s_1, a_1, \ldots, s_H, a_H, s_{H+1}\right)$

$$R\left(\tau_H\right) := \sum_{t=0}^{H} \gamma^t r(s_t, a_t)$$

- Expected reward:

$$J_H(\pi_\theta) := \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t r(s_t, a_t) \,|\, s_0 = s, \pi\right] = \mathbb{E}_{\tau_H \sim p_\theta}\left[R(\tau_H)\right]$$

- We need to compute $\nabla_\theta J_H(\pi_\theta) = \nabla_\theta \mathbb{E}_{\tau_H \sim p_\theta}[R(\tau_H)]$

Deriving the policy gradient

Fact 2: $\nabla_\theta p_\theta(\tau) = \nabla_\theta p_\theta(\tau) \frac{p_\theta(\tau)}{p_\theta(\tau)} = \nabla_\theta \log p_\theta(\tau) p_\theta(\tau)$

for any function
$\nabla_x \log f(x) = \frac{\nabla_x f(x)}{f(x)}$

- ~~Example~~: Implication

$$\nabla_\theta \mathop{E}_{\tau_H \sim p_\theta} [R(\tau_H)] = \nabla_\theta \left[\sum_{\tau_H} R(\tau_H) P_\theta(\tau_H) \right]$$

$$= \sum_{\tau_H} R(\tau_H) \nabla_\theta P_\theta(\tau_H)$$

$$= \sum_{\tau_H} \left(R(\tau_H) \nabla_\theta \log P_\theta(\tau_H) \right) P_\theta(\tau_H)$$

$$= \mathop{E}_{P_\theta} \left(R(\tau_H) \nabla_\theta \log P_\theta(\tau_H) \right)$$

Deriving the policy gradient

Fact 2: $\nabla_\theta p_\theta(\tau) = \nabla_\theta p_\theta(\tau)\frac{p_\theta}{p_\theta} = \nabla_\theta \log p_\theta(\tau) p_\theta(\tau)$

- Example: Suppose we have two actions a_1, a_2,

$$p(a_1) = \theta \implies p(a_2) = 1-\theta$$

$$J(\theta) = E[R(a)] = \theta R(a_1) + (1-\theta)R(a_2)$$

$\nabla_\theta J(\theta) = R(a_1) - R(a_2)$. Furthermore,

$$\nabla_\theta \log \pi_\theta(a_1) = \nabla_\theta \log \theta = \frac{1}{\theta}$$

$$\nabla_\theta \log \pi_\theta(a_2) = \nabla_\theta \log (1-\theta) = \frac{-1}{1-\theta}$$

. We can verify, $\nabla_\theta J(\theta) =$

$$E_{a\sim\pi_\theta}[R(a)\nabla_\theta \log \pi_\theta(a)] = \theta R(a_1)\frac{1}{\theta} + (1-\theta)R(a_2)\frac{-1}{1-\theta} = R(a_1) - R(a_2)$$

Deriving the policy gradient

- Need to compute $\nabla_\theta J_H(\pi_\theta) = \nabla_\theta \mathbb{E}_{\tau_H \sim p_\theta}[R(\tau_H)]$
- Using Fact 1 and Fact 2

Estimating gradient from samples

- Sample N trajectories by interacting with the system or simulator

$$\left(s_0^i, a_0^i, s_1^i, a_1^i, \ldots, s_H^i, a_H^i\right), \quad i = \{1, \ldots, N\}$$

- Empirical estimate of $\nabla_\theta J(\pi_\theta)$

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &\approx \nabla_\theta J_H(\pi_\theta) \\
&= \mathbb{E}_{\tau_H \sim p_\theta}\left[\left(\sum_{t=0}^{H} \gamma^t r(s_t, a_t)\right) \times \left(\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right)\right] \\
&\approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=0}^{H} \gamma^t r(s_t^i, a_t^i)\right)\left(\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i)\right)
\end{aligned}$$

Policy gradient with gradient estimator

$K \leftarrow$ number of training iterations, θ initialized randomly, $\alpha \leftarrow$ step size
for $i = 1, 2, \ldots, K$ **do**
    Calculate the gradient $\nabla_\theta J(\pi_\theta)$
    $\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta)$
end for

- Convergence under certain conditions on the Markov decision process and/or policy parameterization
- In practice, requires very large number of samples for good gradient estimation and convergence

- If you have a model of the system, model-based are better -> optimization, (approximate) dynamic programming, model predictive control

Summary

- Reinforcement learning
  - A model-free approach to stochastic optimal control
  - Policy gradient: an approach to do reinforcement learning

- Your tasks this week
  - Go through Problem set 3
  - Start working on python homework
  -