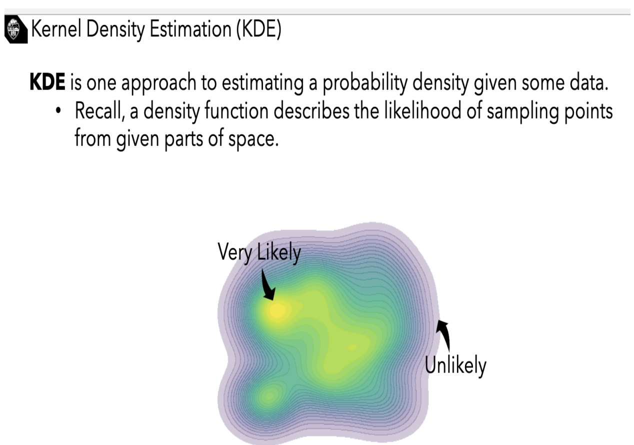
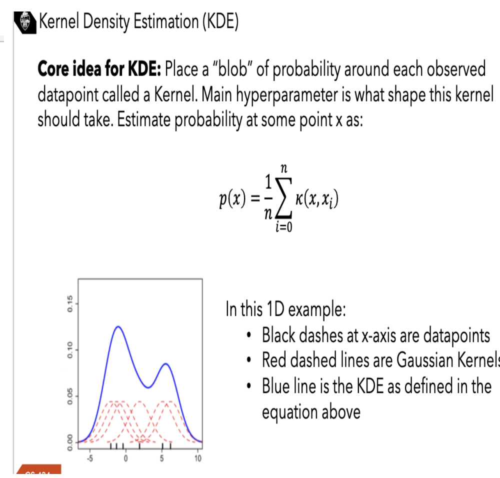
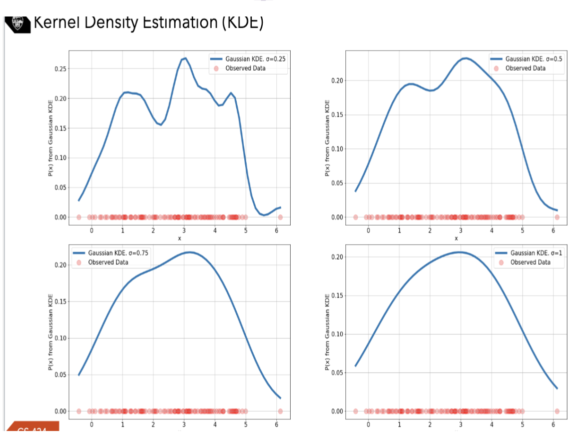
Lecture 10.1: Kernel Density Estimation and A Tiny Sliver of Reinforcement Learning

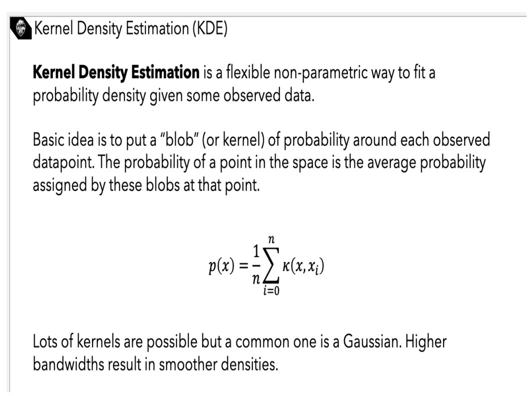
Be able to answer:

- What is Kernel Density Estimation?
- What is a sequential decision problem?
- What is a Markov Decision Process?
- What is a state?
- What is a reward?
- What is an action?
- What is a transition function?
- What reinforcement learning generally?
- What are policy gradient methods?
- What is imitation learning?
- What is behavior cloning?









Sequential Decision Making

So far, we've mostly considered ML tasks where our algorithms are trying to reduce error when

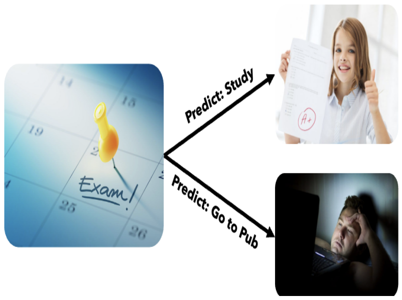
making predictions independently for each datapoint.

For example, predicting the first image below as a cat doesn't change the next image or how our algorithms would classify it Prediction à Cat Prediction à Dog

Sequential Decision Making

Contrast this with a **sequential decision making** task where an algorithm's output changes the next input that will be received.

For example, deciding what to do the night before the exam:



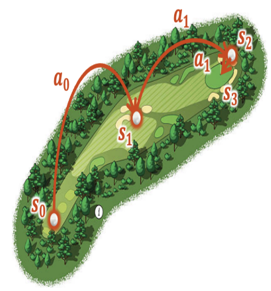
Sequential Decision Making

In **sequential decision making tasks**, our algorithms will need to make a **sequence** of correct predictions in order to be successful. Further – inputs our algorithms receive will change based (in part) on the sequence of actions that have been taken thus far!

Imagine we have a ML model that predicts how to hit a golf ball (angle/force) given its current position / conditions on a course.

Call the current position the **state**, call the hits **actions**, and call our model a **policy** that maps from states to actions

To be "good" at this, our policy really needs to make a sequence of good predictions. In this case, bad predictions led to worse states!



Markov Decision Processes

We will very often consider these sorts of problems within the framework of Markov Decision Process (MDP)s or Partially Observable Markov Decision Processes (POMDPs)

Decision process defined by

- $\mathcal{S}$ Set of possible **states**
- $\mathcal{A}$ Set of possible **actions**
- $\mathrm{R}: \mathcal{S} \rightarrow \mathbb{R}$ **Reward function** – mapping between states and rewards
- $\mathrm{T}: \mathcal{S} \times \mathcal{A} \rightarrow \Omega_{\mathcal{S}}$ **Transition** function – mapping between state-action pairs and a distribution over next states

"Markov" assumption says transitions depend only on current state – mathematically that $P(s_{t+1} | a, s_0, \ldots, s_t) = P(s_{t+1} | a, s_t)$

Markov Decision Processes

We will often be interested in finding behavior policies that perform well in these environments. To operationalize this idea, we'll need to define some things:

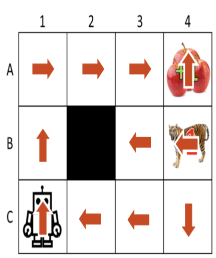
1. **What is a behavior policy?**

2. **What do we mean by "performs well"?**

3. **What algorithms can we use to find a behavior policy that performs well?**

Behavior Policies

1. **What is a behavior policy?**



A behavior policy is a mapping from states to actions (or distributions over actions):

Usually use the symbol π to denote policy:

Deterministic policy: $a = \pi(s)$

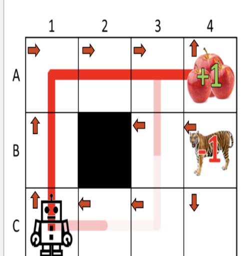
or

Stochastic policy: $P(A \mid s) = \pi(s)$

Denote the "best' behavior policy as π^*

Behavior Policies

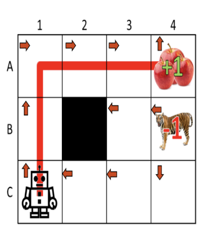
1. **What is a behavior policy?**



Recall that our environment has some randomness in the action space. Running the same policy from the same start, may result in different trajectories.

Intensity of red line show fraction of 1000 simulations that took that path.

Reward of a Trajectory



Given a trajectory s_0, s_1, s_2, s_3, s_4 ... can compute the **additive reward** as:

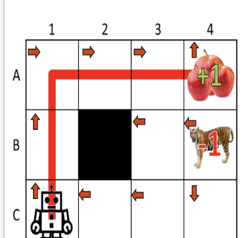
$$\sum_{t=0}^{\infty} R(s_t) = R(s_0) + R(s_1) + R(s_2) + \cdots$$

Will usually want to use **discounted rewards**:

$$\sum_{t=0}^{\infty} \gamma^t R(s_t) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$$

where $0 \geq \gamma \geq 1$ is the discount factor.

Reward of a Trajectory



More about **discounted rewards**:

$$\sum_{t=0}^{\infty} \gamma^t R(s_t) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$$

where $0 \geq \gamma \geq 1$ is the discount factor.

- Describes preference for current reward over future reward
  - **Biology:** Compensates for uncertainty in available time (model mortality)
  - **Financial:** Money gained today can be invested and make more money

Question Break!

Consider **discounted rewards**:

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \gamma^3 R(s_3) + \cdots$$

If γ is near 0, this encodes the idea that ____________.

A Future and current rewards are equally important.

B Future rewards don't matter - only immediate payoff.

C

D

CS 434

Reward of a Trajectory

2. What do we mean by policy "performs well"?

How much reward can we *expect* a policy to collect on average if we run it many times?

- Each time we run the policy, will get a trajectory of states. However, which precise trajectory is random due to the environment (and/or the policy if it is stochastic).
  - Can write a state sequence s that comes from running π as $s \sim \pi$

Write expected discounted reward as:

$$\mathbb{E}_{s\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)\right]$$

Optimal policy π^* is the policy that maximizes expected discounted reward:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{s\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)\right]$$

Markov Decision Processes

We will often be interested in finding behavior policies that perform well in these environments. To operationalize this idea, we'll need to define some things:

1. **What is a behavior policy?**

   A policy is a mapping from states to actions (or distributions over actions).

2. **What do we mean by "performs well"?**

   Expected discounted reward of running the policy in the environment.

3. **What algorithms can we use to find a behavior policy that performs well?**

Markov Decision Processes

What algorithms can we use to find a behavior policy that performs well?

If we know the transition function and reward function (and state space isn't huge):

Value Iteration / Policy Iteration. No real learning happens here typically, just dynamic programming to iteratively refine our estimates.

If we don't know transition and reward functions:

Model-based Reinforcement Learning: Learn transition function and reward function from observation and then use the above techniques to find best policy.

$\boxed{\text{Model-free Reinforcement Learning: Directly learn a policy to maximize reward based on experience.}}$

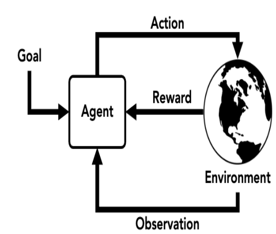
Challenges in Reinforcement Learning

- Actions have unknown and possibly non-deterministic effects which are initially unknown and must be learned
  - **Explore/Exploit Tradeoff:** How much time should we spend trying to explore the environment vs learning how to maximize reward in the area we've already explored?
- Rewards / punishments can be infrequent and are often at the end of long sequences
  - **Credit Assignment Problem:** How do we determine which actions are *really* responsible for the reward or punishment?
- The environment may be massive and exploring all possible actions and states infeasible
  - The game of Go has 2.08×10170 legal board configurations!

Model-free Reinforcement Learning:

An intuitively appealing strategy:
- If an action leads to positive reward when executed in a state, do it more often in that state.
- If an action leads to negative reward when executed in a state, do it less often in that state.

Does it make sense mathematically? Is it the "correct" thing to do?



Policy-Gradient Methods

Let's consider the "direct" approach of trying to learn some policy $\pi(s;\theta)$ that maximizes the expected discounted future rewards:

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \underbrace{\mathbb{E}_{s\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)\right]}_{J(\theta)}$$

How to optimize θ to maximize $J(\theta)$?

Gradient Ascent! $\theta' = \theta + \alpha \nabla_\theta J(\theta)$

Policy-Gradient Methods

Let's rewrite a bit:

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[r(\tau)]$$

$$= \int_\tau p(\tau;\theta) r(\tau)\, d\tau$$

Where $r(\tau)$ is the discounted reward of trajectory $\tau = (s_0, a_0, s_1, a_1, s_2, \dots)$

$$p(\tau;\theta) = \prod \underbrace{p(s_{t+1}|s_t, a_t)}_{\text{Transition}} \underbrace{\pi_\theta(a_t, s_t)}_{\text{Probability of taking action } a \text{ under the policy}}$$

Policy-Gradient Methods

Lets differentiate with respect to θ: $\nabla_\theta J(\theta) = \int_\tau \nabla_\theta p(\tau;\theta)\, r(\tau)\; d\tau$

A useful identity/trick:

$$\frac{p(\tau;\theta)}{\underbrace{p(\tau;\theta)}_{1}} \nabla_\theta p(\tau;\theta) = p(\tau;\theta) \frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta) \nabla_\theta \log p(\tau;\theta)$$

Policy-Gradient Methods

Lets differentiate with respect to θ: $\nabla_\theta J(\theta) = \int_\tau \nabla_\theta p(\tau;\theta)\, r(\tau)\; d\tau$

$$\nabla_\theta J(\theta) = \int_\tau p(\tau;\theta)\; \nabla_\theta \log p(\tau;\theta)\, r(\tau)\, d\tau$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[r(\tau)\; \nabla_\theta \log p(\tau;\theta)]$$

Policy-Gradient Methods

Expected rewards of policy $\pi(\cdot\,|\,\cdot\,;\theta)$: $J(\theta) = \int_\tau p(\tau;\theta) r(\tau) d\tau$

Lets differentiate with respect to θ: $\nabla_\theta J(\theta) = \int_\tau \underbrace{\nabla_\theta p(\tau;\theta) r(\tau) d\tau}_{\text{Intractable}}$

Policy-Gradient Methods

Lets differentiate with respect to θ: $\nabla_\theta J(\theta) = \int_\tau \nabla_\theta p(\tau;\theta)\, r(\tau)\; d\tau$

$$\nabla_\theta J(\theta) = \int_\tau p(\tau;\theta)\; \nabla_\theta \log p(\tau;\theta)\, r(\tau)\, d\tau$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[r(\tau)\; \nabla_\theta \log p(\tau;\theta)]$$

Policy-Gradient Methods

Lets differentiate with respect to θ: $\nabla_\theta J(\theta) = \int_\tau \nabla_\theta p(\tau;\theta)\, r(\tau)\; d\tau$

A useful identity/trick:

$$\frac{p(\tau;\theta)}{\underbrace{p(\tau;\theta)}_{1}} \nabla_\theta p(\tau;\theta) = p(\tau;\theta) \frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta) \nabla_\theta \log p(\tau;\theta)$$

Policy-Gradient Methods

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[r(\tau)]$$

Transformed the **gradient of an expectation** into the **expectation of a gradient**

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[r(\tau)\; \nabla_\theta \log p(\tau;\theta)]$$

Policy-Gradient Methods

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[r(\tau)\ \nabla_\theta \log p(\tau;\theta)]$$

Computing $\nabla_\theta \log p(\tau;\theta)$:

$$p(\tau;\theta) = \prod p(s_{t+1}|s_t,a_t)\,\pi_\theta(a_t,s_t)$$

$$\log p(\tau;\theta) = \sum \log p(s_{t+1}|s_t,a_t) + \log \pi_\theta(a_t,s_t)$$

$$\Rightarrow\ \nabla_\theta \log p(\tau;\theta) = \sum \nabla_\theta \log \pi_\theta(a_t,s_t)$$

No model needed! Model-free RL.

Policy-Gradient Methods

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{s_i,a_i \in \tau} \nabla_\theta \log \pi(a_i|s_i;\theta)\, r(\tau)\right]$$

Monte Carlo Approximation:

$$\nabla_\theta J(\theta) \approx \sum_n \sum_{s_i,a_i \in \tau_n} \nabla_\theta \log \pi(a_i|s_i;\theta)\, r(\tau_n)$$

Policy-Gradient Methods

Intuition:

$$\nabla_\theta J(\theta) \approx \sum_n \sum_{s_i,a_i \in \tau_n} \nabla_\theta \log \pi(a_i|s_i;\theta)\, r(\tau_n)$$

- If trajectory reward is positive, push up the probabilities of all the actions involved
- If trajectory reward is negative, push down the probabilities of all the action involve

All actions in trajectory move in same direction based on reward?!?

I know it seems too simple but it averages out.

Policy-Gradient Methods

REINFORCE Algorithm (Williams, 1992)

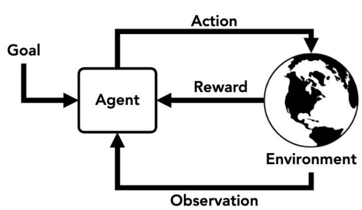
While not converged:

1. Run policy to collect trajectories $\tau_j = (s_{j0}, a_{j0}, s_{j1}, a_{j1}, s_{j2}, \dots)$ and reward $r(\tau_j)$
2. Compute gradient estimate $\nabla_\theta J(\theta) \approx \sum_j \sum_{s_{ji},a_{ji} \in \tau_j} \nabla_\theta \log \pi(a_{ji}|s_{ji};\theta)\, r(\tau_j)$
3. Update policy parameters $\theta' = \theta + \alpha \nabla_\theta J(\theta)$

Imitation Learning

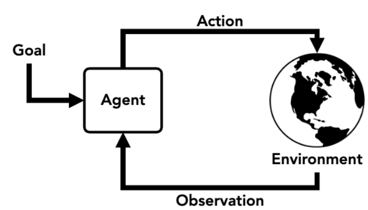
Reinforcement Learning

- Environment provides feedback
- No examples of optimal policy



Imitation Learning

- Have expert demonstrations (possibly interactive)



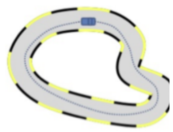
Imitation Learning

Imitation Learning

- Assume access to an expert demonstrator $\boldsymbol{\pi}^*$ at some point or another and to varying levels of interactivity. **No reward signals provided.**

Imitation Learning

Example #1: Racing Game
(Super Tux Kart)

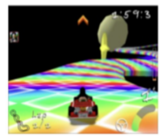


s = game screen

a = turning angle

Training set: D={**τ**:=(**s**,**a**)} from π^*

- **s** = sequence of s
- **a** = sequence of a



Goal: learn $\pi_\theta(s) \rightarrow a$

Imitation Learning

Behavior Cloning

- Given dataset of trajectories $D = \{(s_0, a_0, s_1, a_1, \dots, s_T, a_T)_i\}_{i=1}^N$ from an expert demonstration policy π^*
- Break things down to individual state-action pairs s_t, a_t and directly train a policy $\hat{a}_t = \pi(s_t)$ using supervised learning:

$$\theta^* = argmin_\theta \mathbb{E}_{s \sim \pi^*}[\ L(\pi_\theta(s), \pi^*(s))\ ]$$

$$\theta^* = argmin_\theta \sum_i L(\pi_\theta(a_t|s_t), \pi^*(a_t|s_t))$$

- Interpretations:
  - Assuming perfect imitation so far, learn to continue imitating perfectly
  - Minimize 1-step deviation for states the expert visits

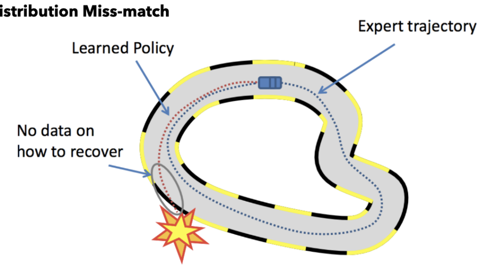
Imitation Learning - Behavior Cloning

Behavior Cloning

- **Strengths:**
  - Dead simple. Seriously. It is just supervised learning.
  - Works well when minimizing 1-step deviation is sufficient.
- **Weaknesses:**
  - Compounding errors.
  - Data distribution miss-match.

Imitation Learning - Behavior Cloning

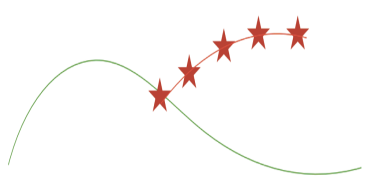
Data Distribution Miss-match



Imitation Learning - Behavior Cloning

Compounding Errors

Suppose π_θ achieves an error rate of ϵ for states induced by π^*, then over a T length trajectory the expected number of errors is $E_\pi[mistakes] = O(T^2\epsilon)$



Efficient reductions for imitation learning, 2010.

Summary of Behavior Cloning

Use set of demonstrations as if they were targets for a supervised learning task and minimizing 1-step error for your policy. Super simple but has trouble with dataset miss-match.

- **When to use this?**
  - When the state space is well-covered by the demonstrator.
  - When recovering from 1-step deviations is easy.
  - To pre-train before doing a full RL approach.