Machine Learning and Data Mining

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

CS 434

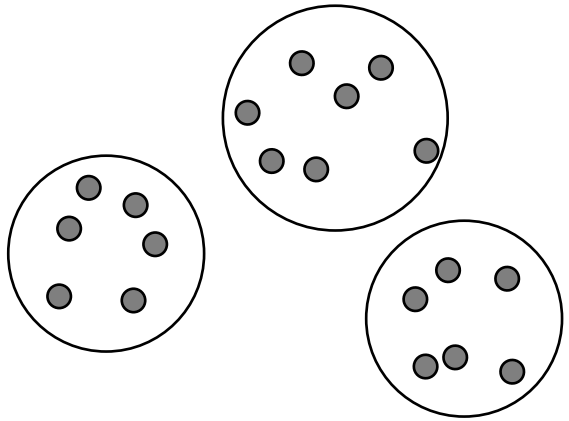
RECAP

From Last Lecture

Two Categories of Clustering Algorithms

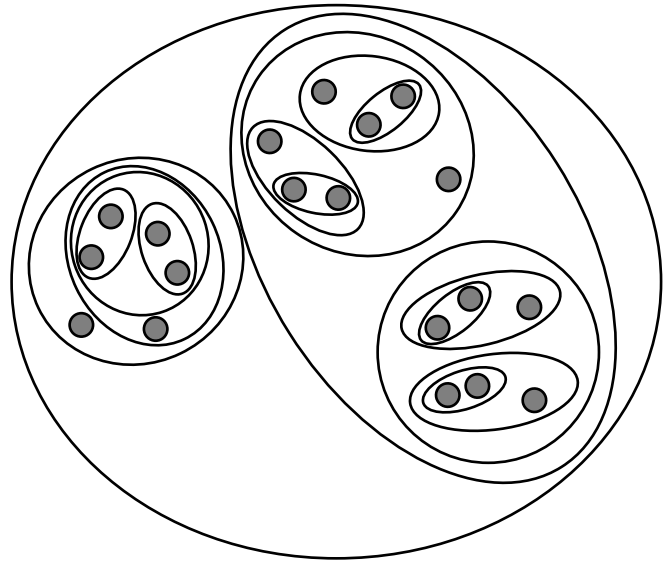
Partition Algorithms ("Flat" Clusterings)

- k-Means / k-Medians
- Gaussian Mixture Models



Hierarchical Algorithms

- Bottom-up – Agglomerative
- Top-down – Divisive

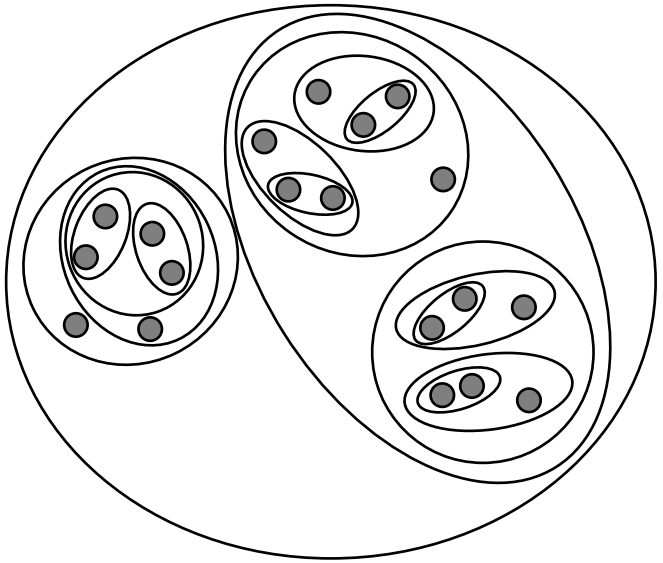


Slide adapted from Xiaoli Fern

Hierarchical Agglomerative Clustering (HAC)

Given a dataset $X = \{\boldsymbol{x}_i\}_{i=1}^{n}$ where $\boldsymbol{x}_i \in \mathbb{R}^d$ and some distance function $d(x_i, x_j)$ which measures the distance between two points:

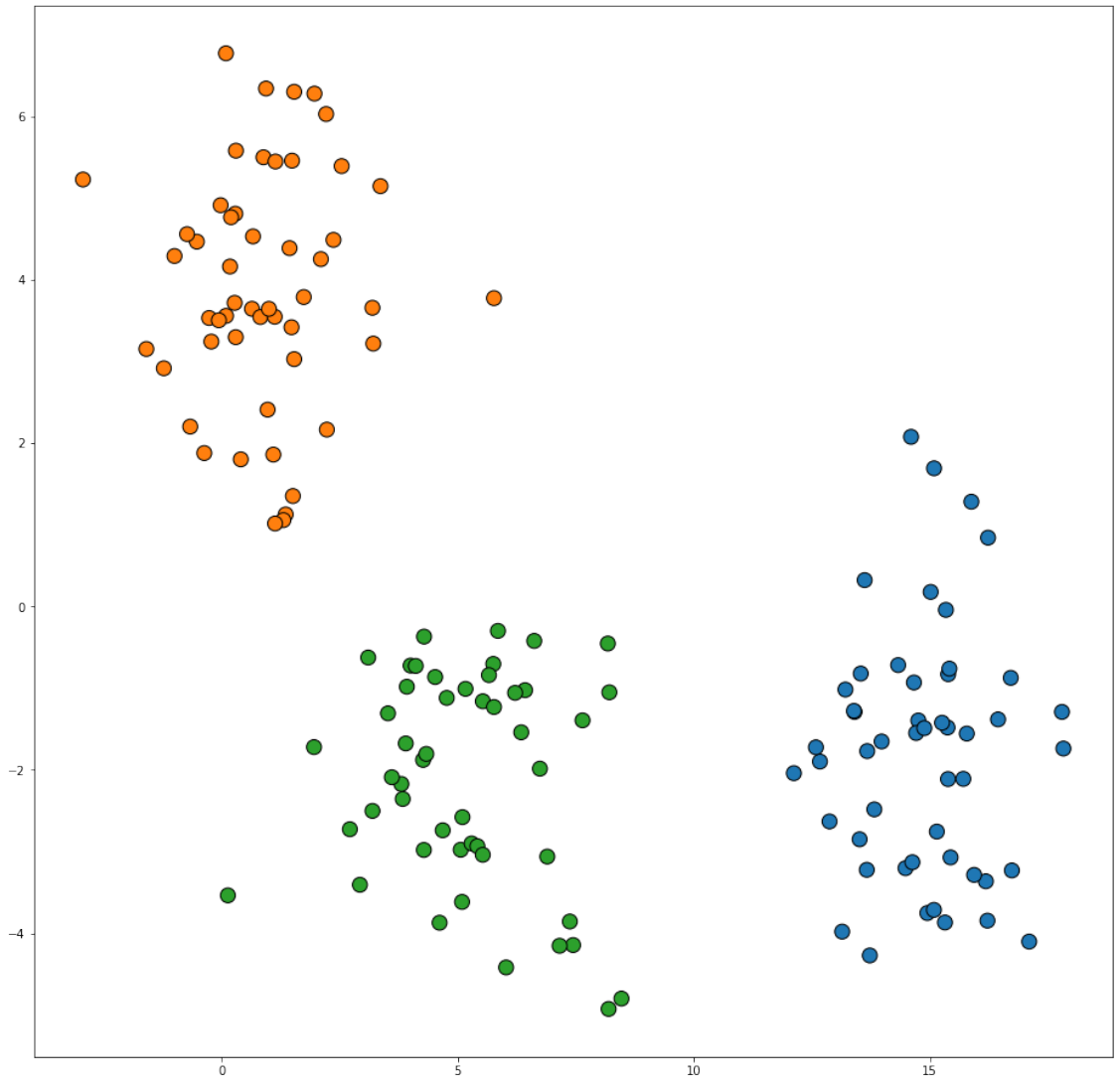
1. Initialize every datapoint as its own cluster
2. Until there is only one cluster remaining:
    a) Merge the two closest clusters

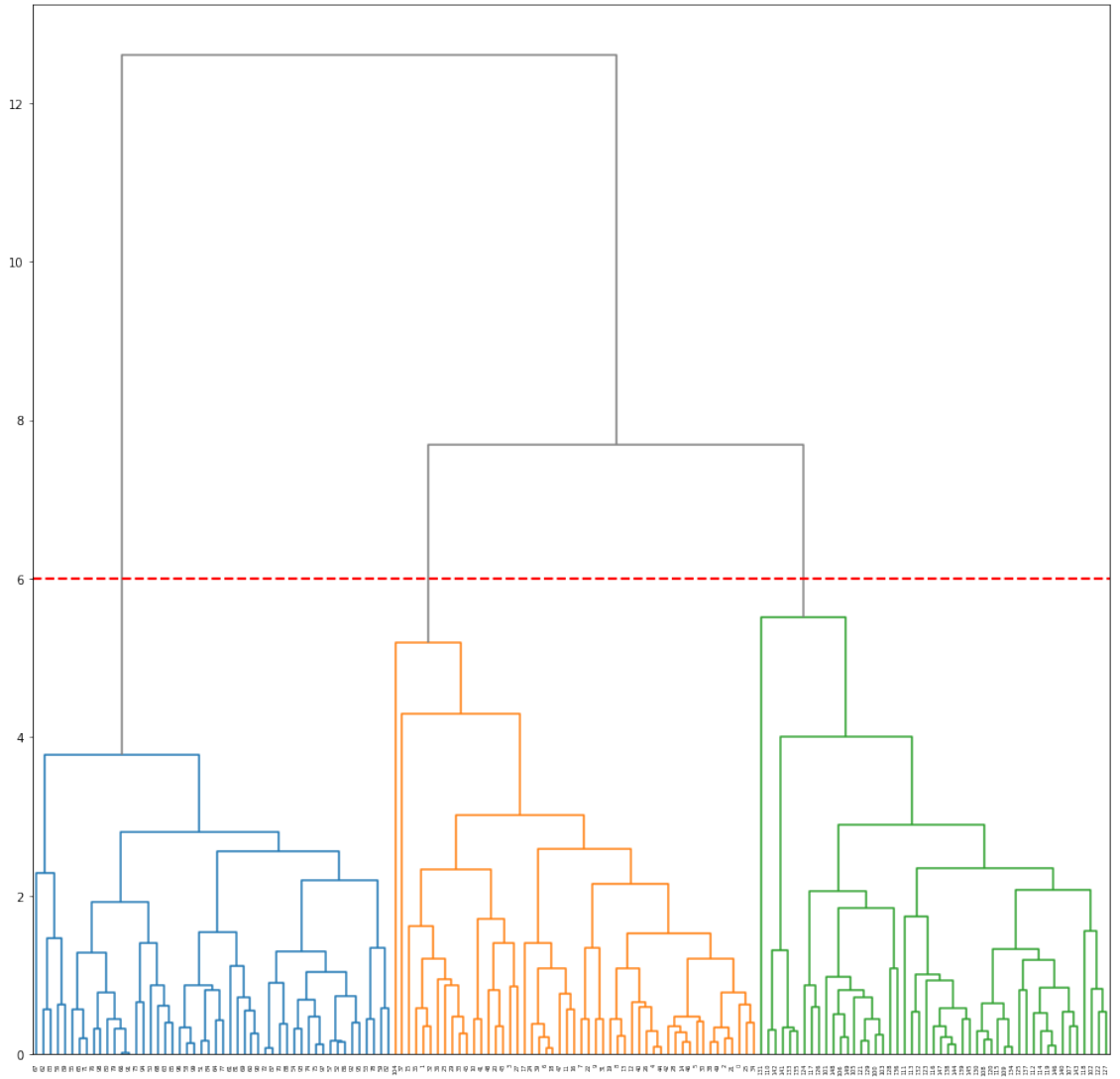


Notice it doesn't output a specific number of clusters. Can save intermediate clusterings from k=n to k=1.

Question: We have a measure of distance between points, how do we use it to measure "closeness" of clusters?

Visualizing Hierarchical Clusterings: Dendrogram





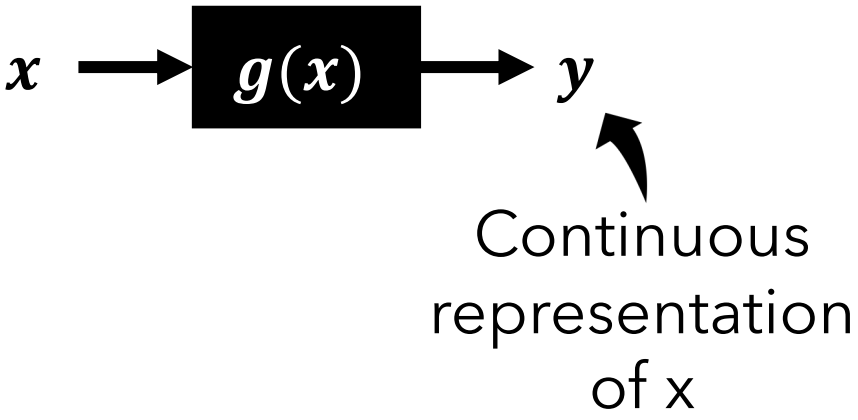
Average-link example

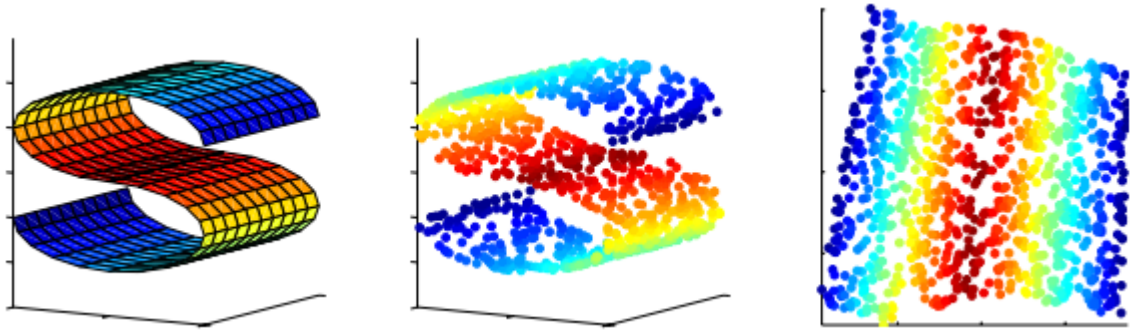
New Topic: Dimensionality Reduction

Unsupervised Learning

Only given a set of input instances (x)

Dimensionality Reduction: Represent high-dim data as low-dim data



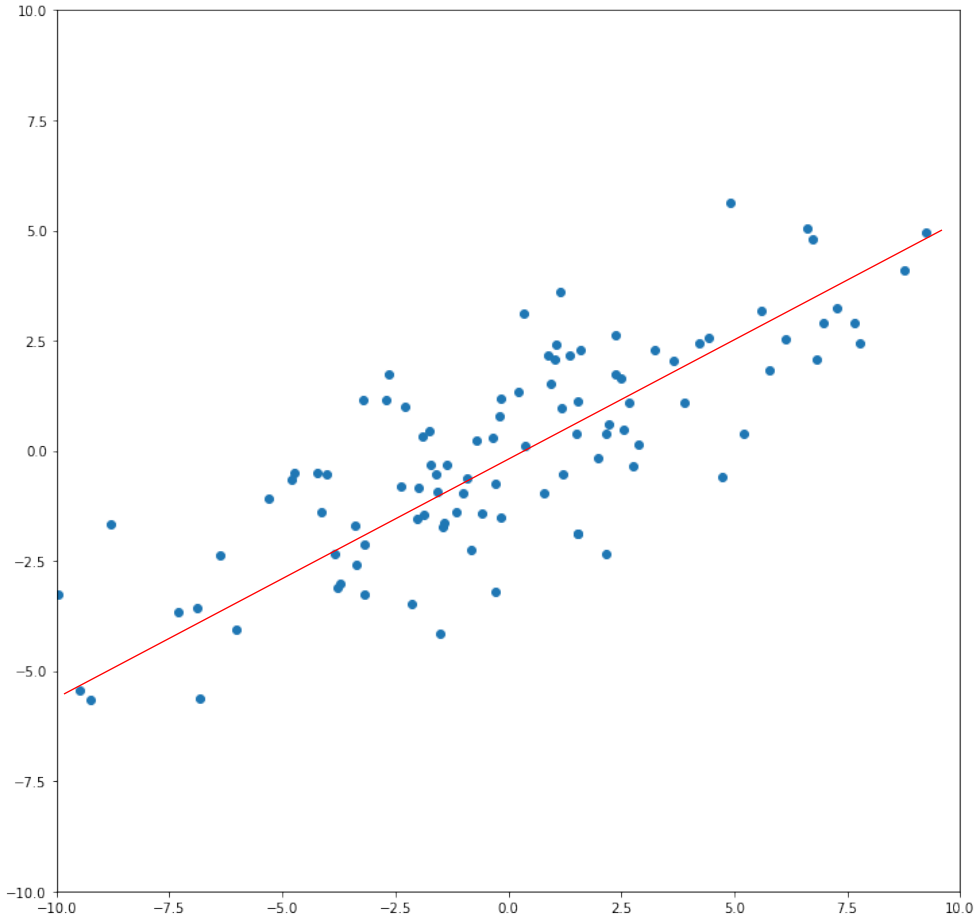


Example: Finding a 2D subspace in a 3D feature space

Principal Component Analysis (PCA) -- Conceptually

Consider doing the following:

- Find a line such that most of the variance of the data follows along that line
- Or equivalently, a line where the dataset projected onto that line maintains as much variance.
- Call this line your first principal component.



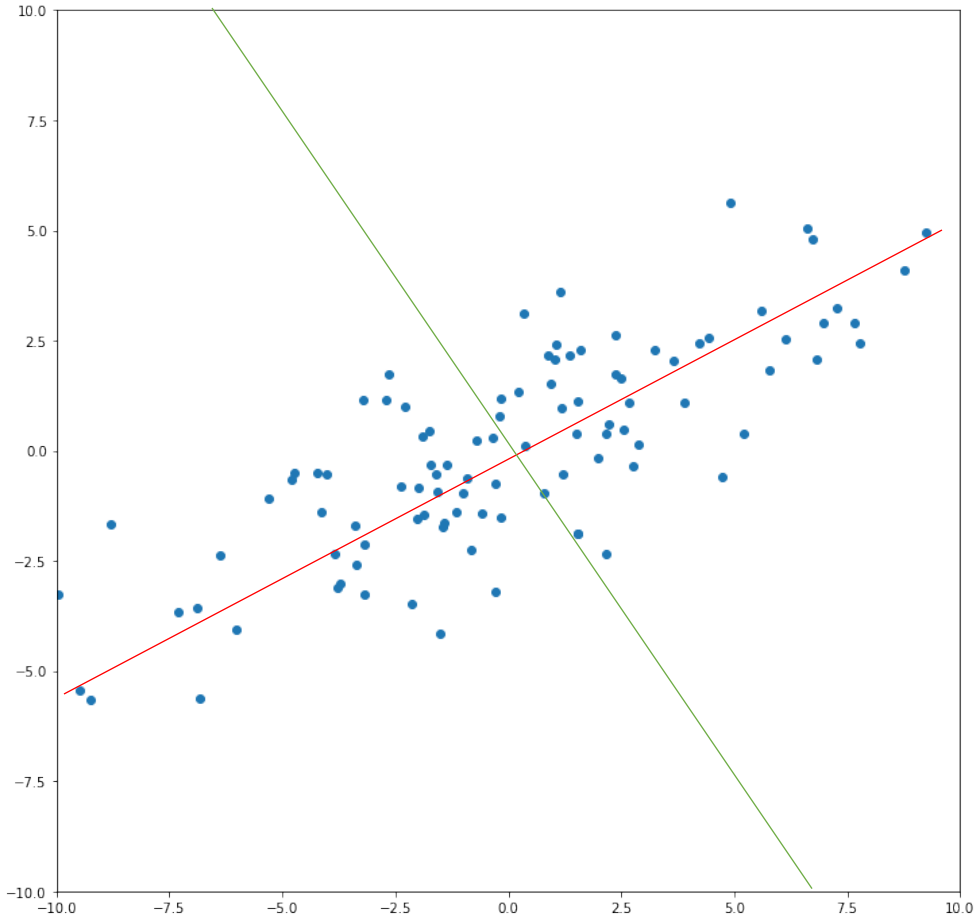
Principal Component Analysis (PCA) -- Conceptually

Consider doing the following:

- Repeat this by finding the line orthogonal from this that captures the most variance.
  - 2nd principal component
  - 3rd …
  - 4th …

- In 2D, there is only one line orthogonal to our 1st principal component so we must stop after 2.

- In higher dims, there will be many.

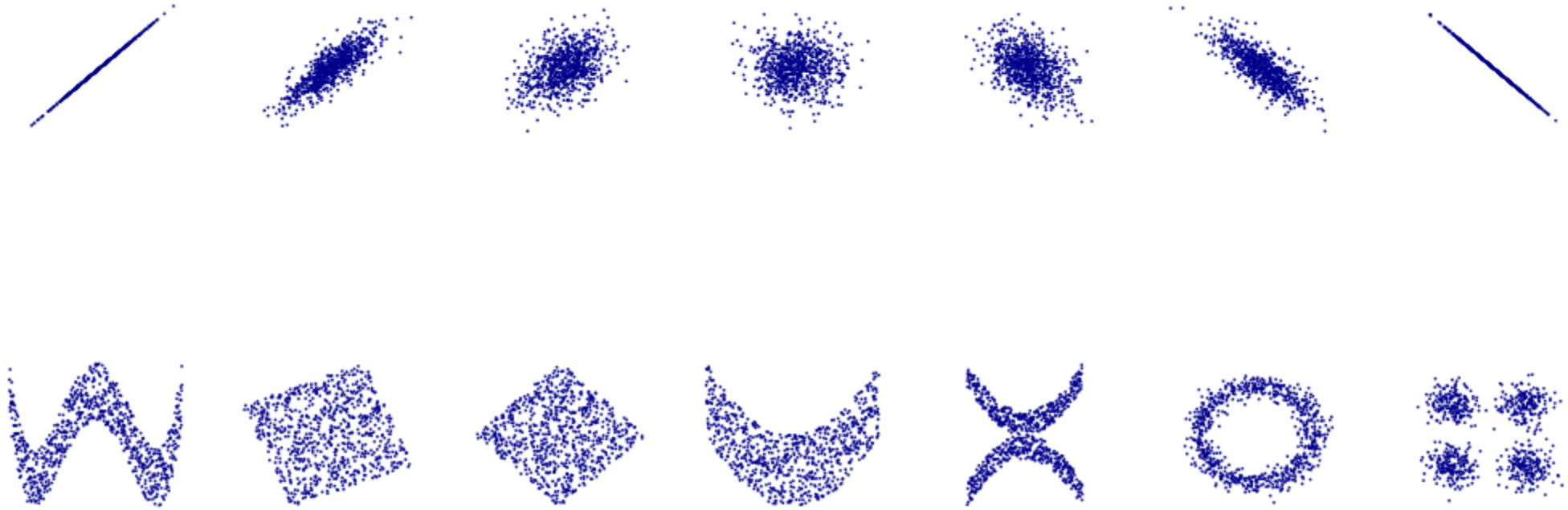


Principal Component Analysis (PCA) -- Demo

https://colab.research.google.com/drive/12RHEzkN0w1JgviAOeCBh6lx-KFDJ_OQV?usp=sharing

PCA is a Linear Dimensionality Reduction Model

Only works at finding linear subspace…



Methods for Non-linear Dimensionality Reduction

PCA only works on finding linear subspaces – may be limited for very high-dimensional data.

Non-linear Alternatives: (not an exhaustive list)

- **Kernel PCA** exists! With some clever math you can show PCA is also dependent on dot products between inputs (think about the covariance matrix) and thus replace them with a kernel to do PCA to find linear subspace in non-linear projections of the data.

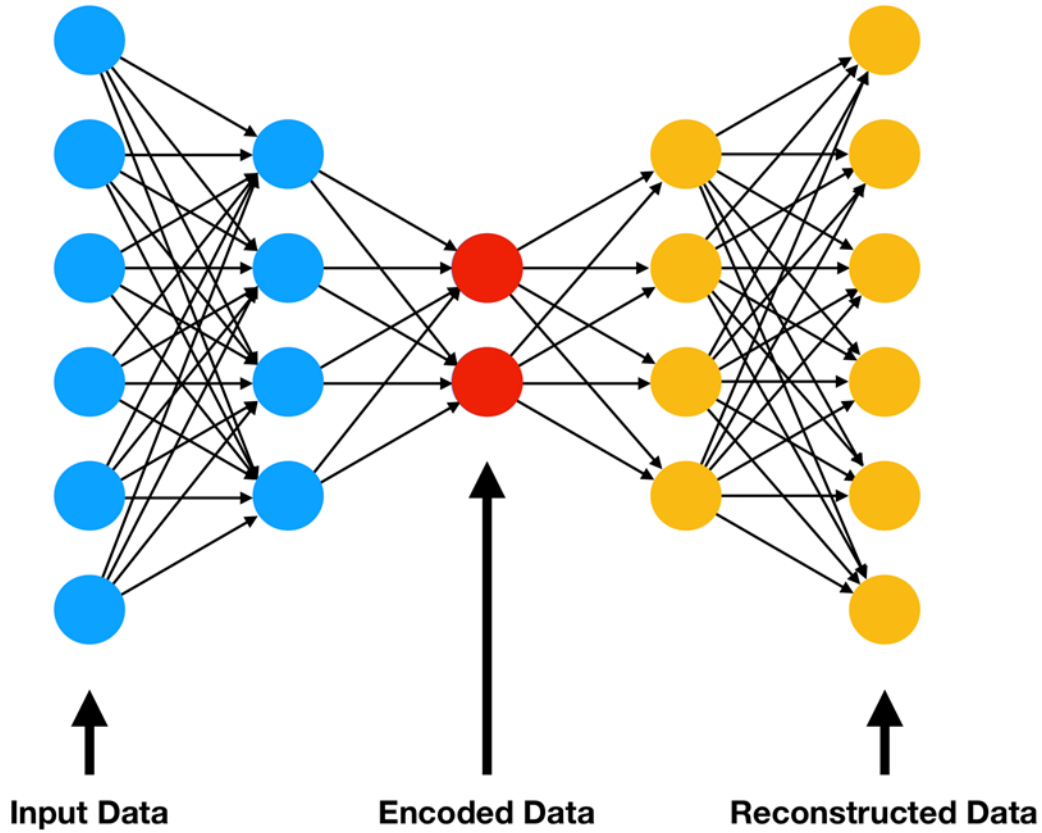
- **t-distributed Stochastic Neighbor Embedding (t-SNE)** is a non-linear method that optimizes the low-dimensional position of each point independently. The math isn't all that bad and it's solved with gradient descent, just don't have time for it.

    - **Intuition:** Find a k-dimensional representation of each datapoint such that relative distances between points in the low k-dimensional space match relative distances in the high dimensional space.

- **Autoencoders.** Next slide.

Autoencoders

A special form of neural network that takes an input and then tries to predict that same input as the output. In-between, there is a "bottleneck" layer with lower dimensionality than the input.



Today's Learning Objectives

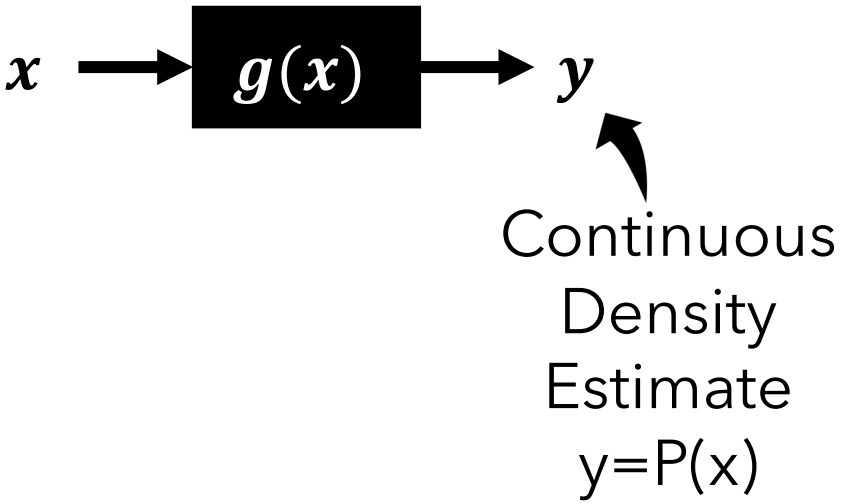
Be able to answer:

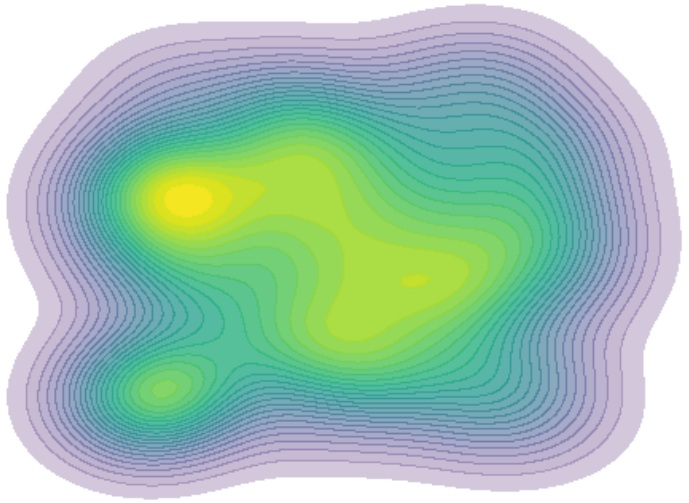
- What is Kernel Density Estimation?
- What is a sequential decision problem?
- What is a Markov Decision Process?
  - What is a state?
  - What is a reward?
  - What is an action?
  - What is a transition function?
- What reinforcement learning generally?
  - What are policy gradient methods?
- What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

Now, We Are Turning Towards Unsupervised Learning

Unsupervised Learning
Only given a set of input instances (x)

Density Estimation: Estimate the underlying distribution generating our data



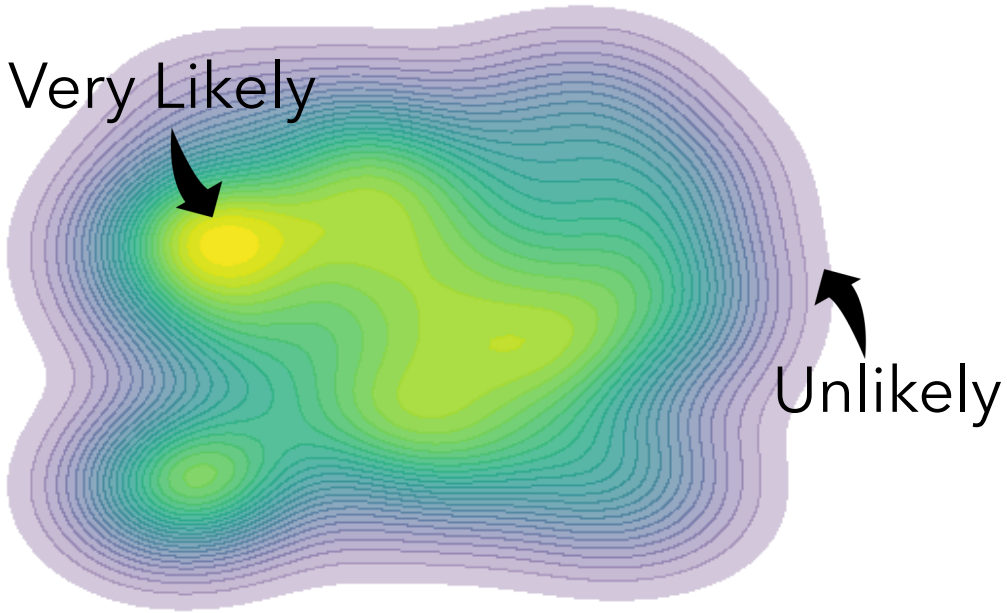


Example: Estimate how likely is a given house with these properties

Kernel Density Estimation (KDE)

KDE is one approach to estimating a probability density given some data.

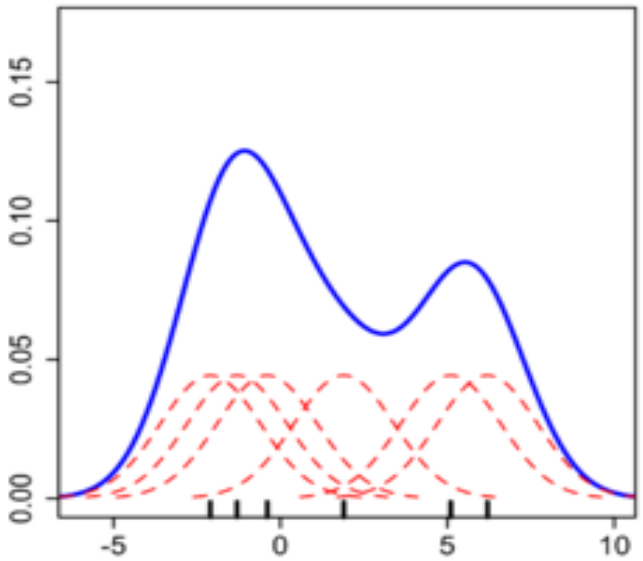
- Recall, a density function describes the likelihood of sampling points from given parts of space.



Kernel Density Estimation (KDE)

Core idea for KDE: Place a "blob" of probability around each observed datapoint called a Kernel. Main hyperparameter is what shape this kernel should take. Estimate probability at some point x as:

$$p(x) = \frac{1}{n}\sum_{i=0}^{n} \kappa(x, x_i)$$



In this 1D example:

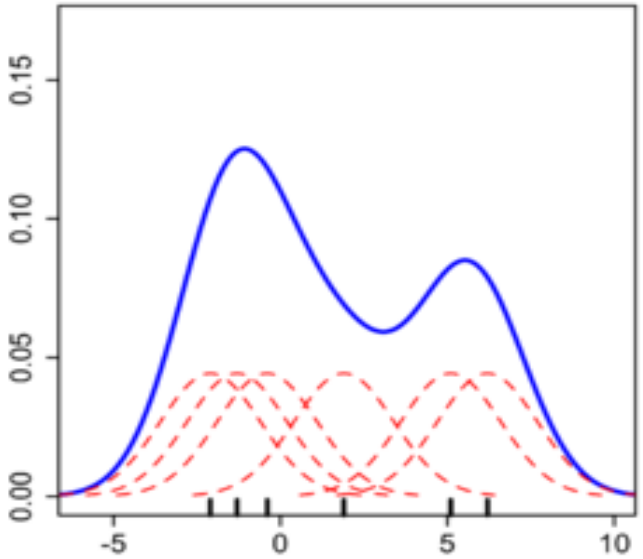
- Black dashes at x-axis are datapoints
- Red dashed lines are Gaussian Kernels
- Blue line is the KDE as defined in the equation above

Kernel Density Estimation (KDE)

A popular kernel is the Gaussian such that:

$$p(x) = \frac{1}{n}\sum_{i=0}^{n}\kappa(x, x_i) = \frac{1}{n}\sum_{i=0}^{n}\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-x_i)^2}{2\sigma}}$$

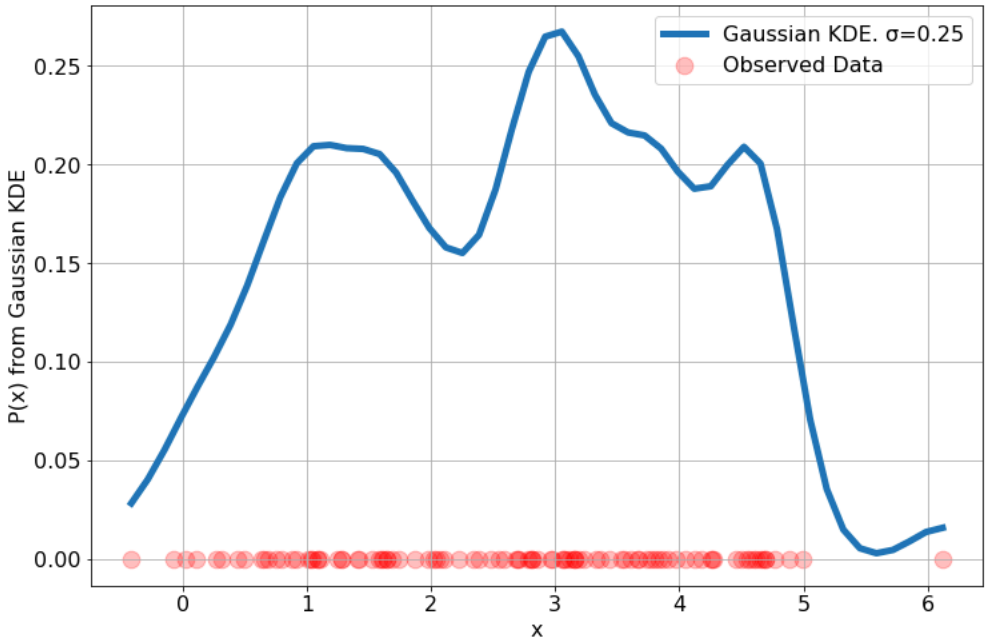
How to set the variance sigma? (Often referred to as a bandwidth)

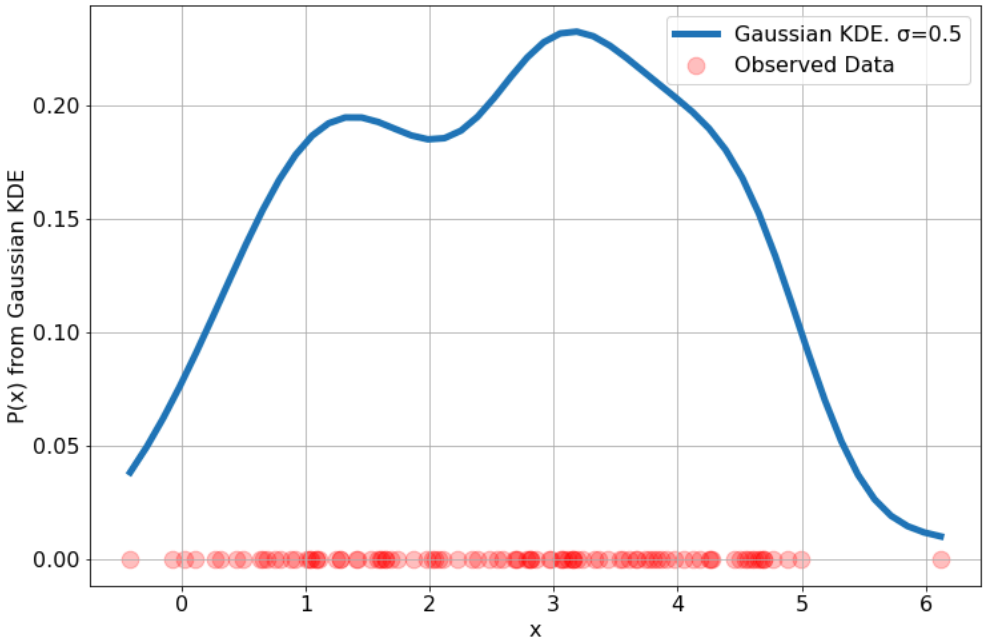


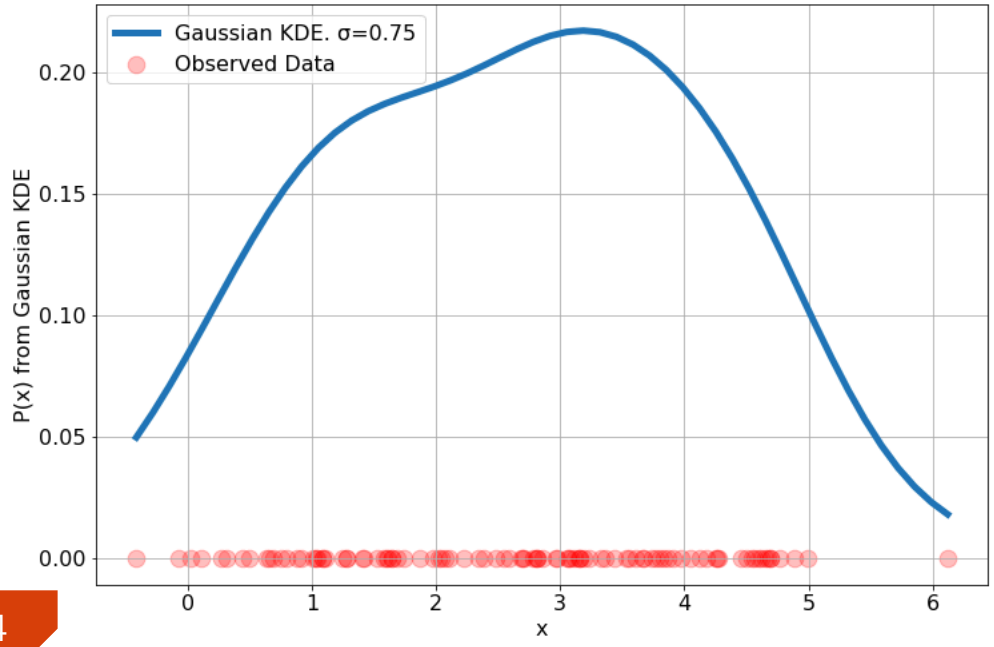
In this 1D example:

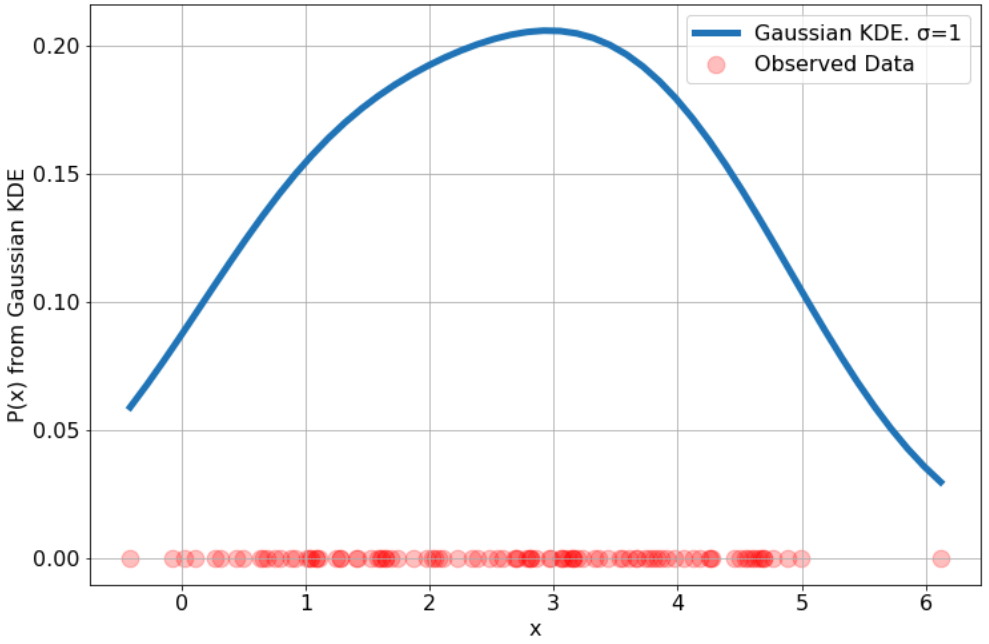
- Black dashes at x-axis are datapoints
- Red dashed lines are Gaussian Kernels
- Blue line is the KDE as defined in the equation above

Kernel Density Estimation (KDE)









Kernel Density Estimation (KDE)

Kernel Density Estimation is a flexible non-parametric way to fit a probability density given some observed data.

Basic idea is to put a "blob" (or kernel) of probability around each observed datapoint. The probability of a point in the space is the average probability assigned by these blobs at that point.

$$p(x) = \frac{1}{n}\sum_{i=0}^{n} \kappa(x, x_i)$$

Lots of kernels are possible but a common one is a Gaussian. Higher bandwidths result in smoother densities.

Today's Learning Objectives

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?

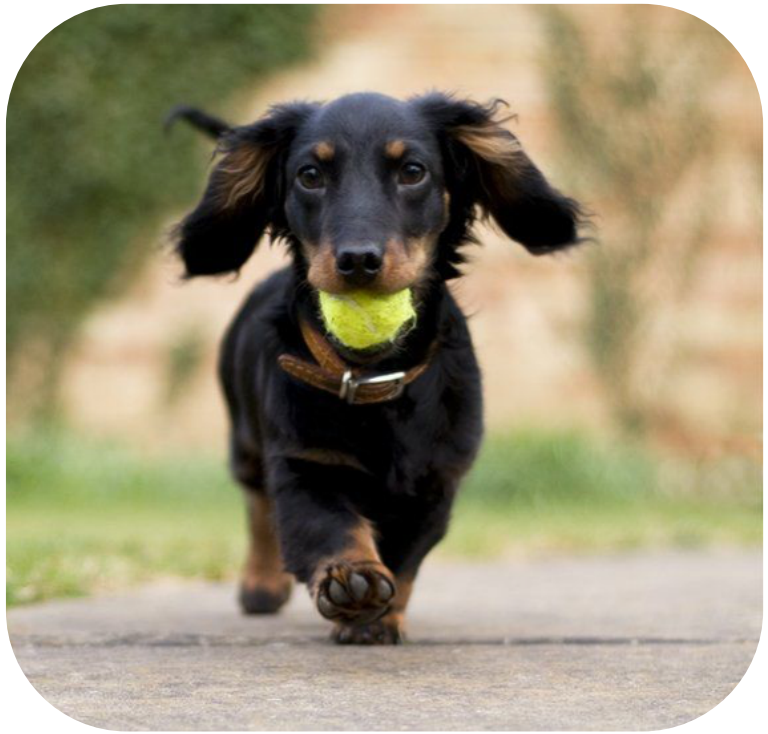
Sequential Decision Making

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 → Cat

Prediction → Dog



Sequential Decision Making

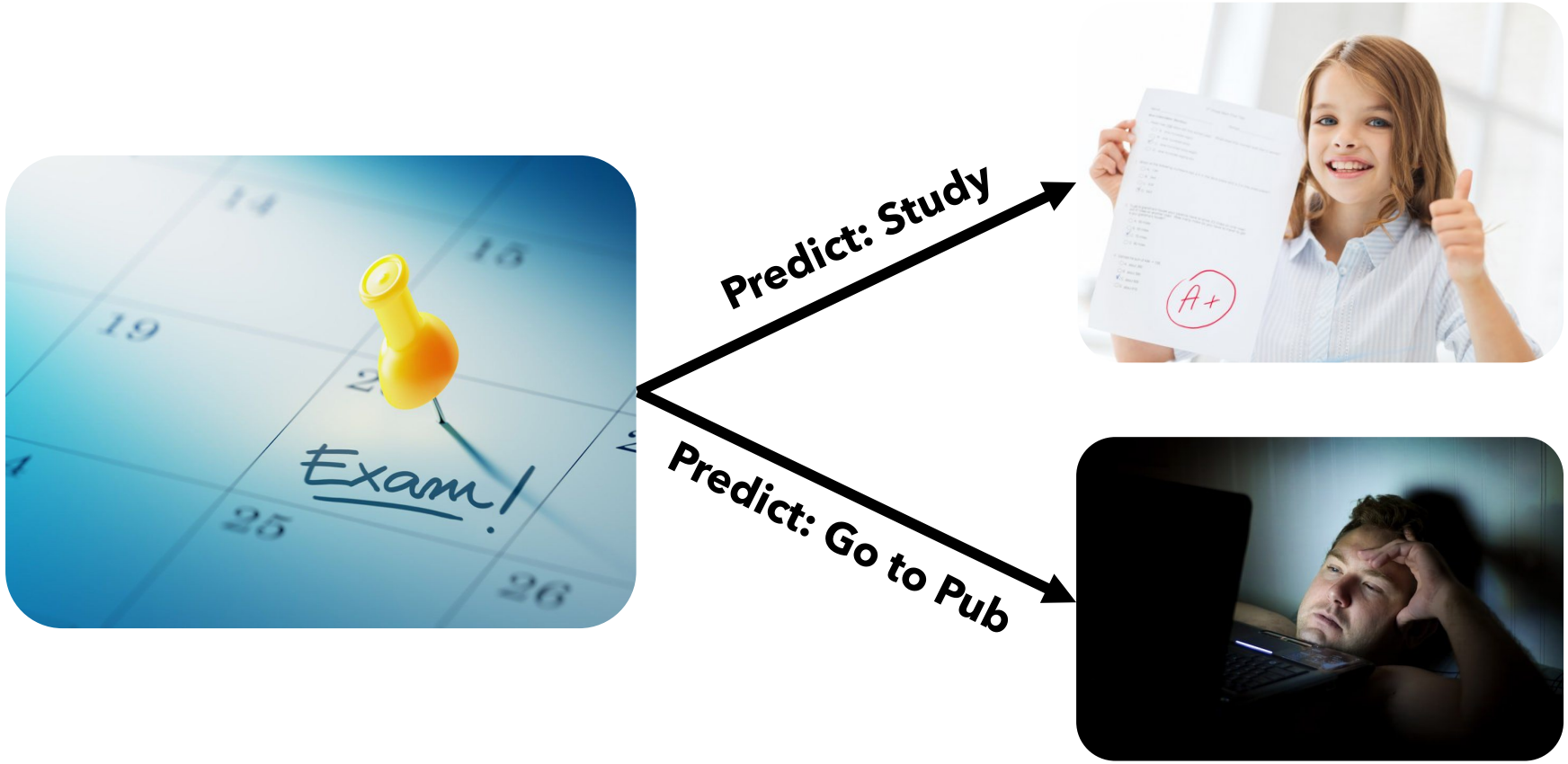
Contrast this with a **sequential decision making** task where an algorithm's output changes the next input that will be received.

For example, a robot walking along a narrow cliff path:

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:



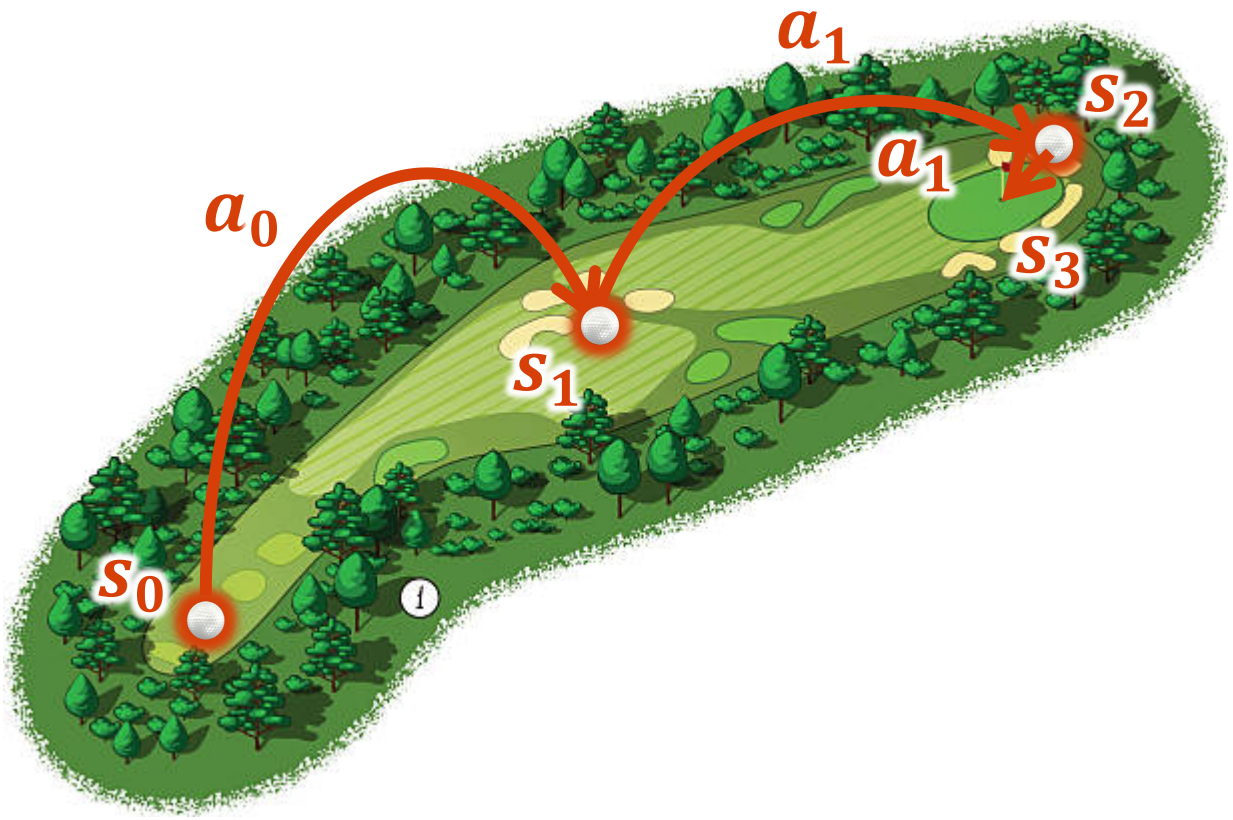
Sequential Decision Making

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!

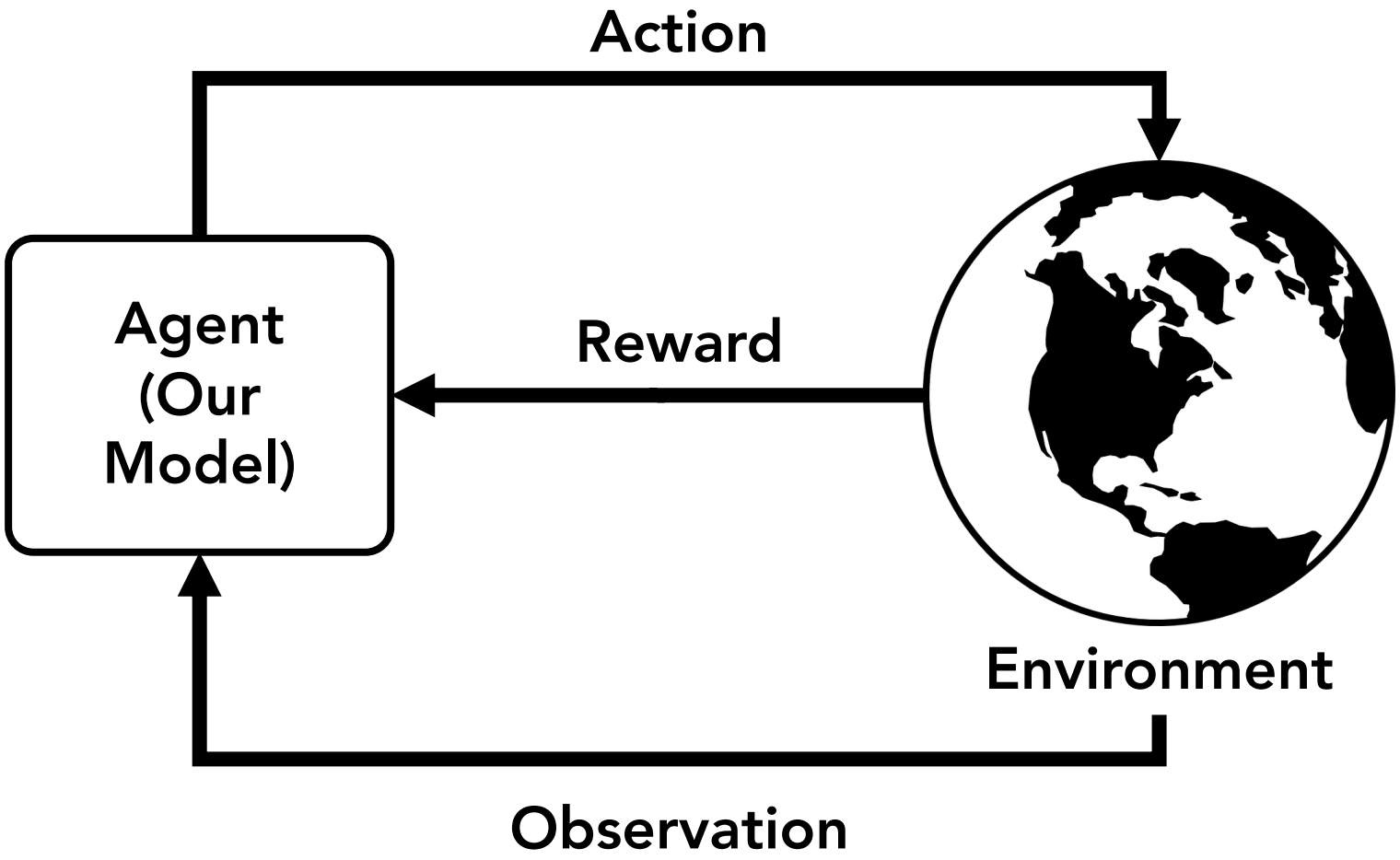
Imagine we have a ML model that predicts how to hit a golf ball (angle/force) given its current position / conditions on a course.

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

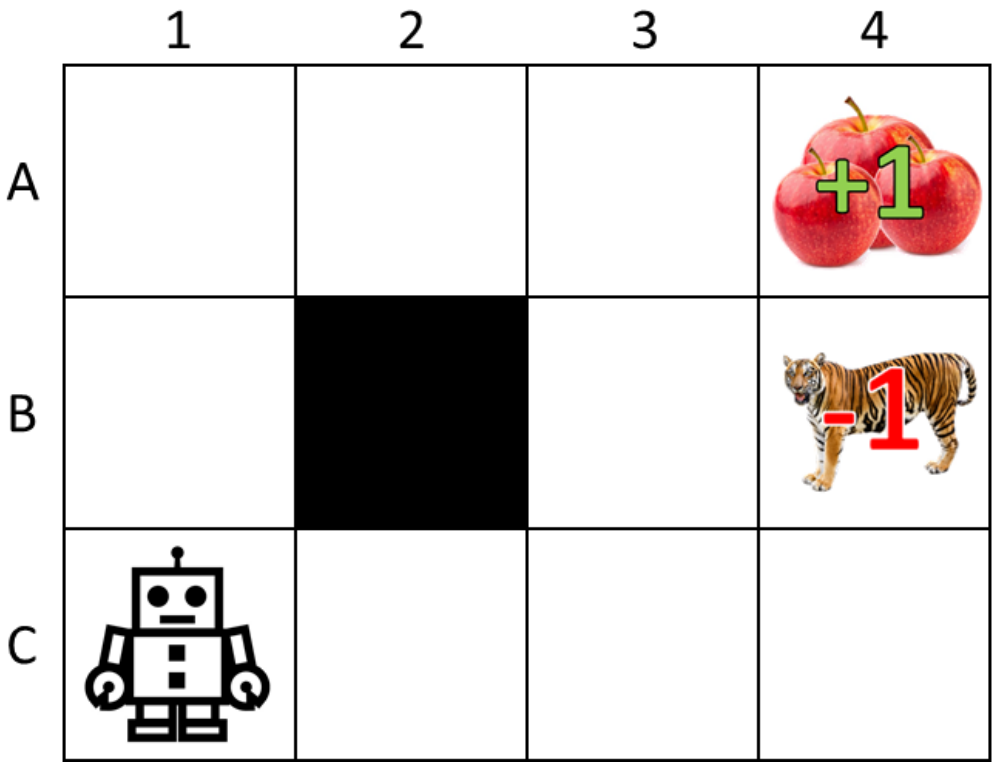
To be "good" at this, our policy really needs to make a sequence of good predictions. In this case, bad predictions led to worse states!



Sequential Decision Making



Let's start thinking about sequential decisions using this simple grid world.
(The same ideas apply to more abstract things than motion like "when to raise or lower control rods in a nuclear reactor")

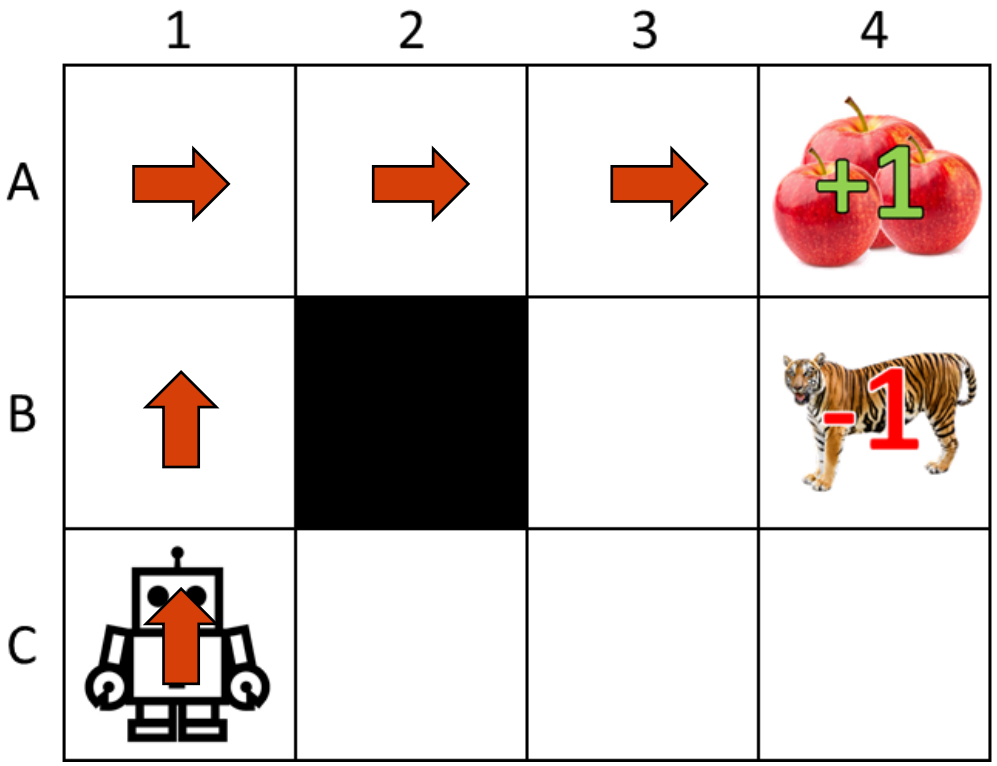


- Assume a fully-observable, deterministic environment shown to the left.
- Each grid cell is a **state**
- At each time step, agent can take **actions**: Up, Down, Left, Right
- Positive **reward** for getting apples, negative for entering the tiger cell.

How do you get to the apples?

Sequential Decision Making

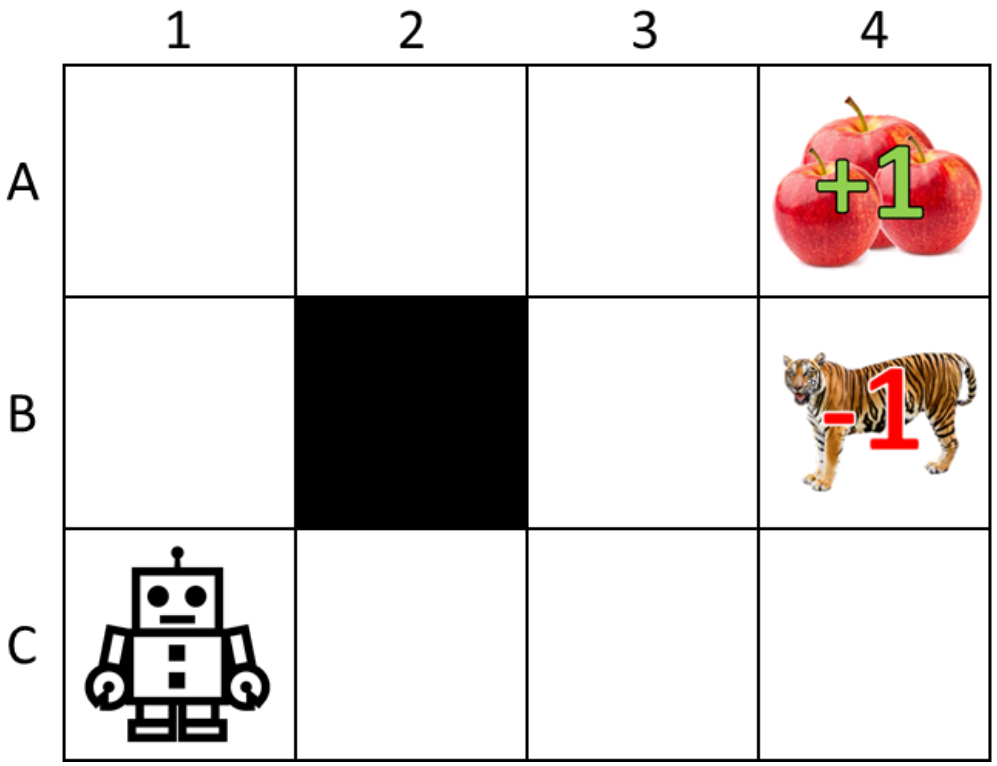
Let's start thinking about sequential decisions using this simple grid world.
(The same ideas apply to more abstract things than motion like "when to raise or lower control rods in a nuclear reactor")



- Assume a fully-observable, deterministic environment shown to the left.
- Each grid cell is a **state**
- At each time step, agent can take **actions**: Up, Down, Left, Right
- Positive **reward** for getting apples, negative for entering the tiger cell.

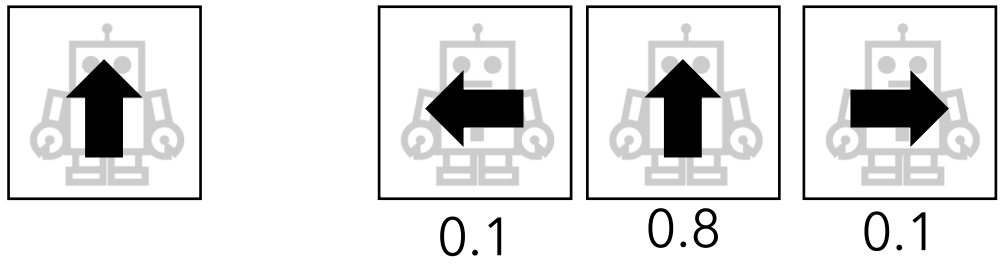
How do you get to the apples?

Let's start thinking about sequential decisions using this simple grid world.
(The same ideas apply to more abstract things than motion like "when to raise or lower control rods in a nuclear reactor")



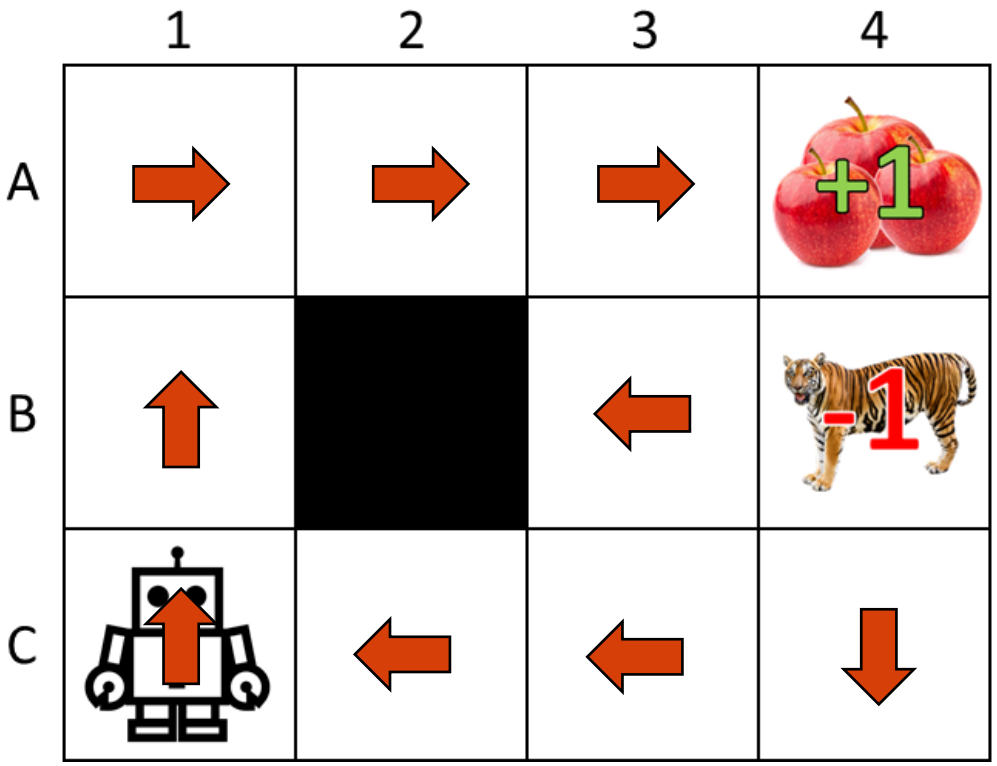
Suppose your algorithm is controlling a slightly broken robot…

- With 80% probability, will go where told.
- With 20% probability, will go in a direction 90 degrees from the instructed direction (10% each)



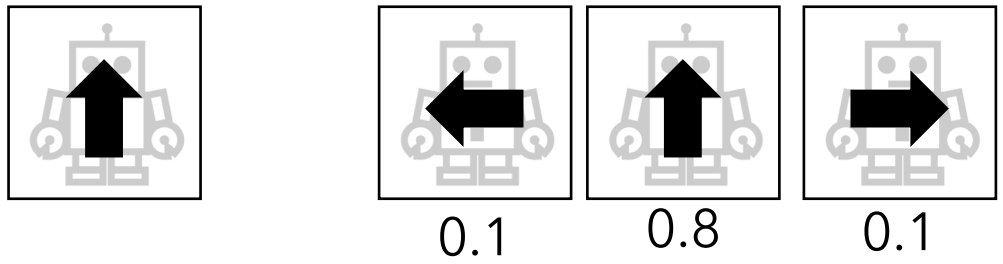
How do you get to the apples?

Let's start thinking about sequential decisions using this simple grid world.
(The same ideas apply to more abstract things than motion like "when to raise or lower control rods in a nuclear reactor")



Suppose your algorithm is controlling a slightly broken robot…

- With 80% probability, will go where told.
- With 20% probability, will go in a direction 90 degrees from the instructed direction (10% each)



How do you get to the apples?

Today's Learning Objectives

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?

Markov Decision Processes

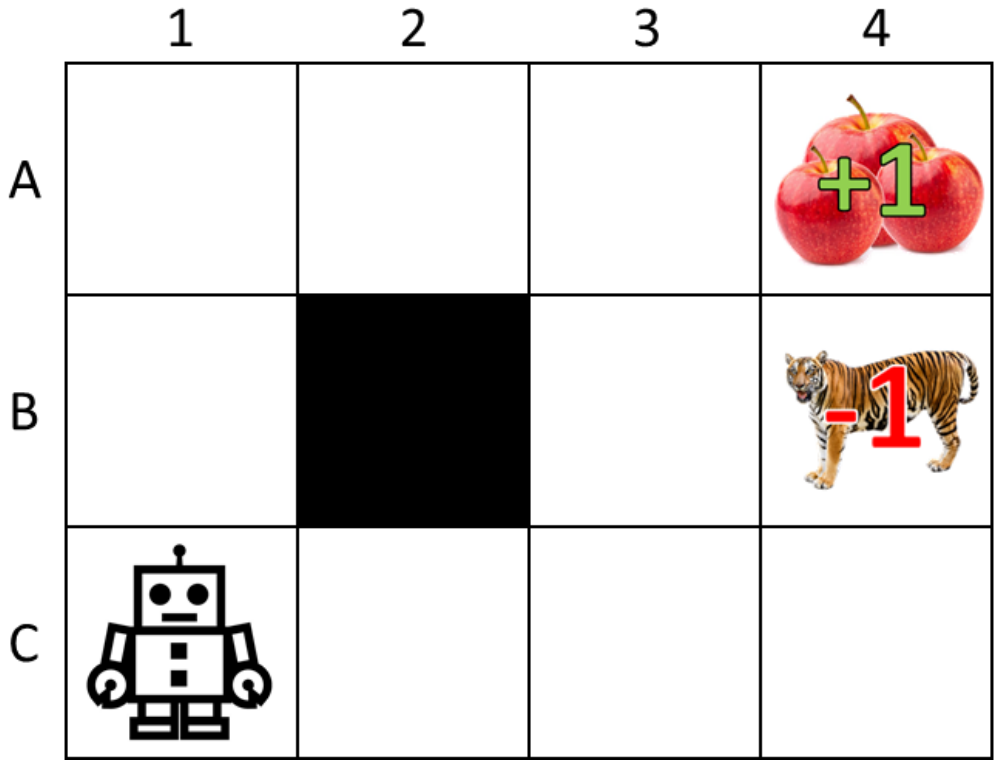
We will very often consider these sorts of problems within the framework of Markov Decision Process (MDP)s or Partially Observable Markov Decision Processes (POMDPs)

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 $P(s_{t+1}|a, s_0, \ldots, s_t) = P(s_{t+1}|a, s_t)$

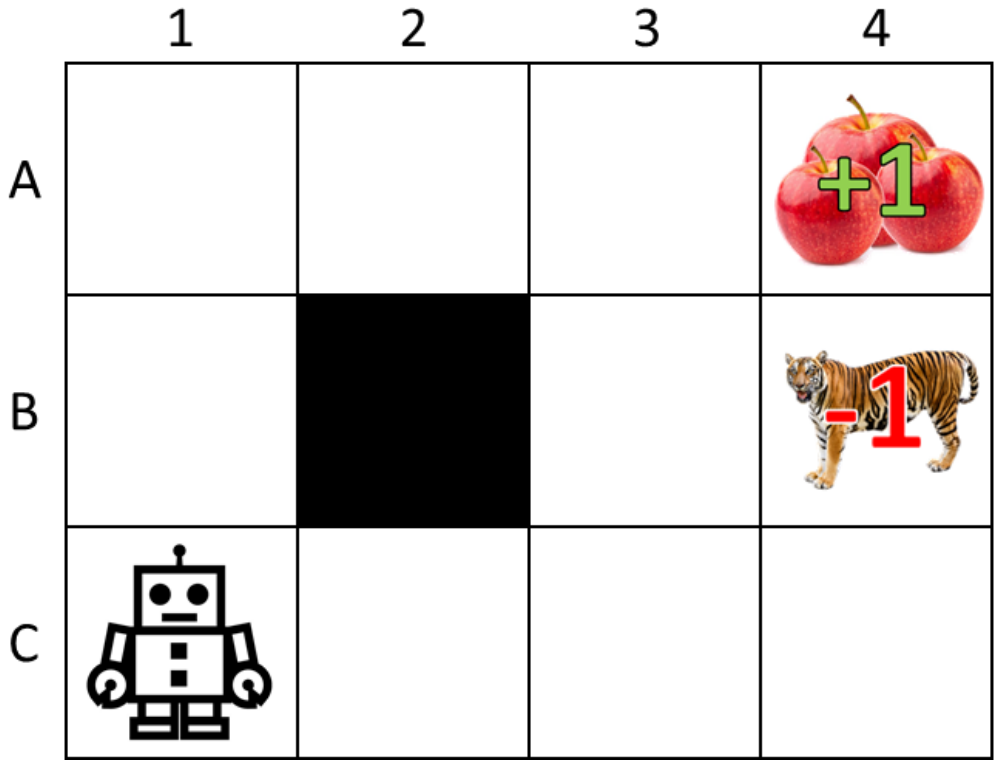
Sequential Decision Making



States $\mathcal{S}$:

$$\mathcal{S} = \{A1, A2, A3, A4, B1, B3, B4, C1, C2, C3, C4\}$$

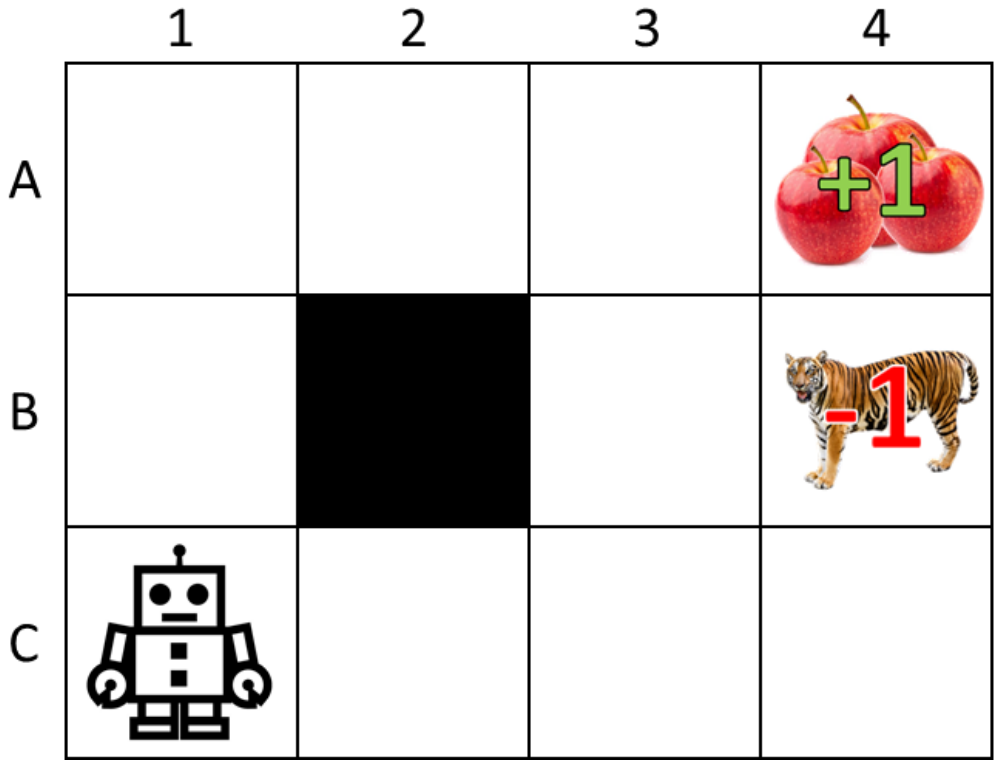
Sequential Decision Making



Actions $\mathcal{A}$:

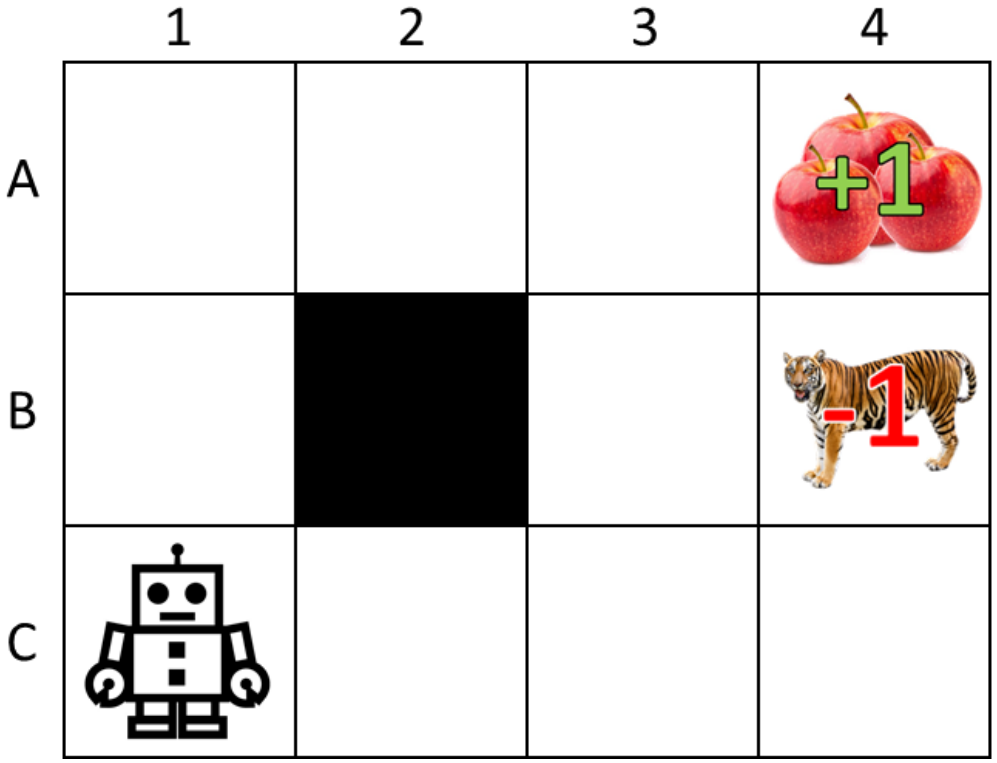
$$\mathcal{A} = \{Up, Down, Left, Right\}$$

Sequential Decision Making



Reward Function $R: \mathcal{S} \rightarrow \mathbb{R}$:

$$R = \{A4 = 1, B4 = -1, Others = -0.0001\}$$

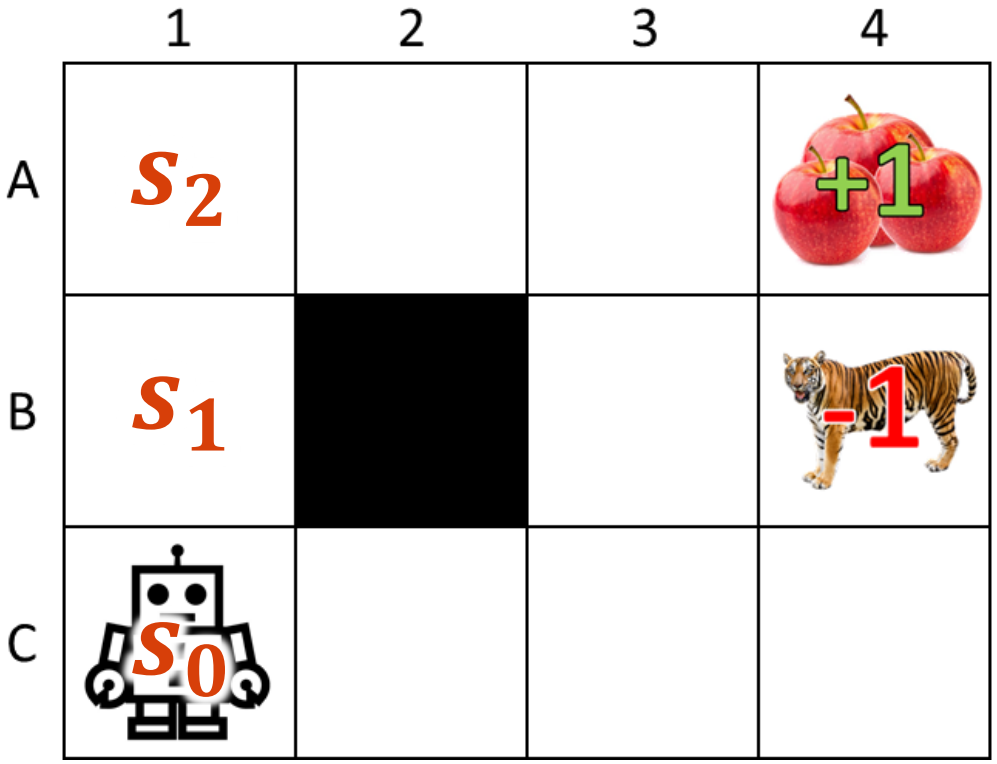


Transition Function $T: \mathcal{S} \times \mathcal{A} \rightarrow \Omega_{\mathcal{S}}$:

$T(s, a, s')$ is probability of arriving at state s' after performing action a in state s

$$T(C1, Up, C2) = 0.1$$
$$T(C1, Up, B1) = 0.8$$
$$T(C1, Up, A2) = 0$$
$$\vdots$$

Sequential Decision Making



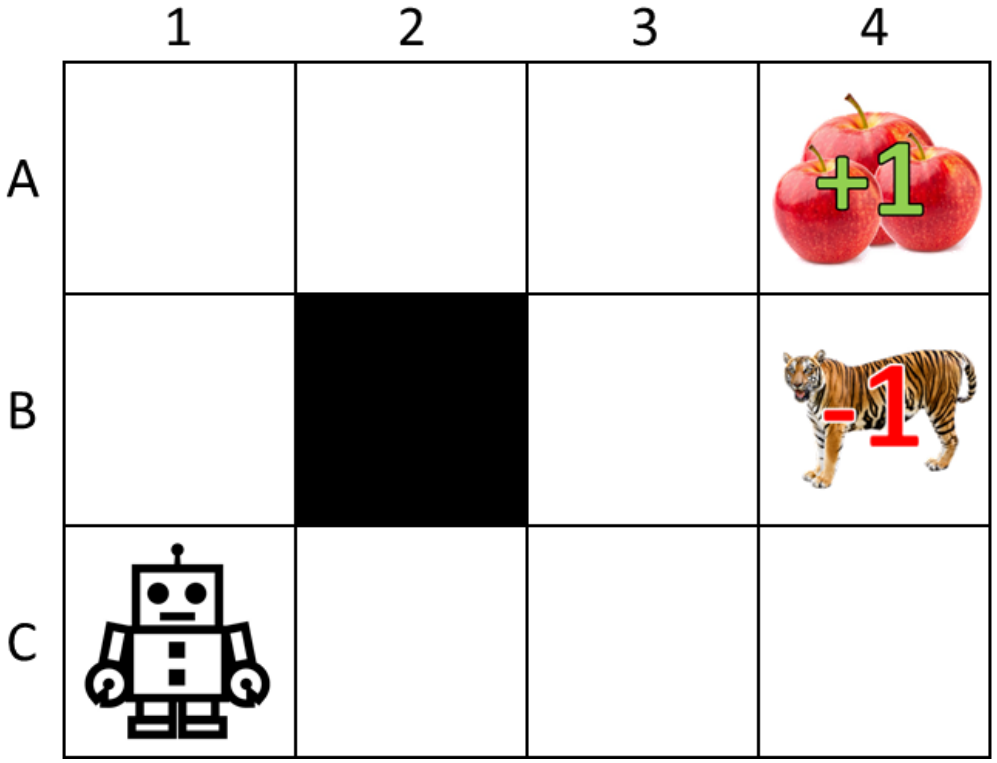
"Markov" assumption says transitions depend only on current state

$$P(s_{t+1}|a, s_0, \dots, s_t) = P(s_{t+1}|a, s_t)$$

If the agent takes the "Right" action at s_2, the probability of arriving at $s_3 = A2$ only depends on the fact that the agent is at s_2 and not the entire history $\{s_0, s_1, s_2\}$

Sequential Decision Making

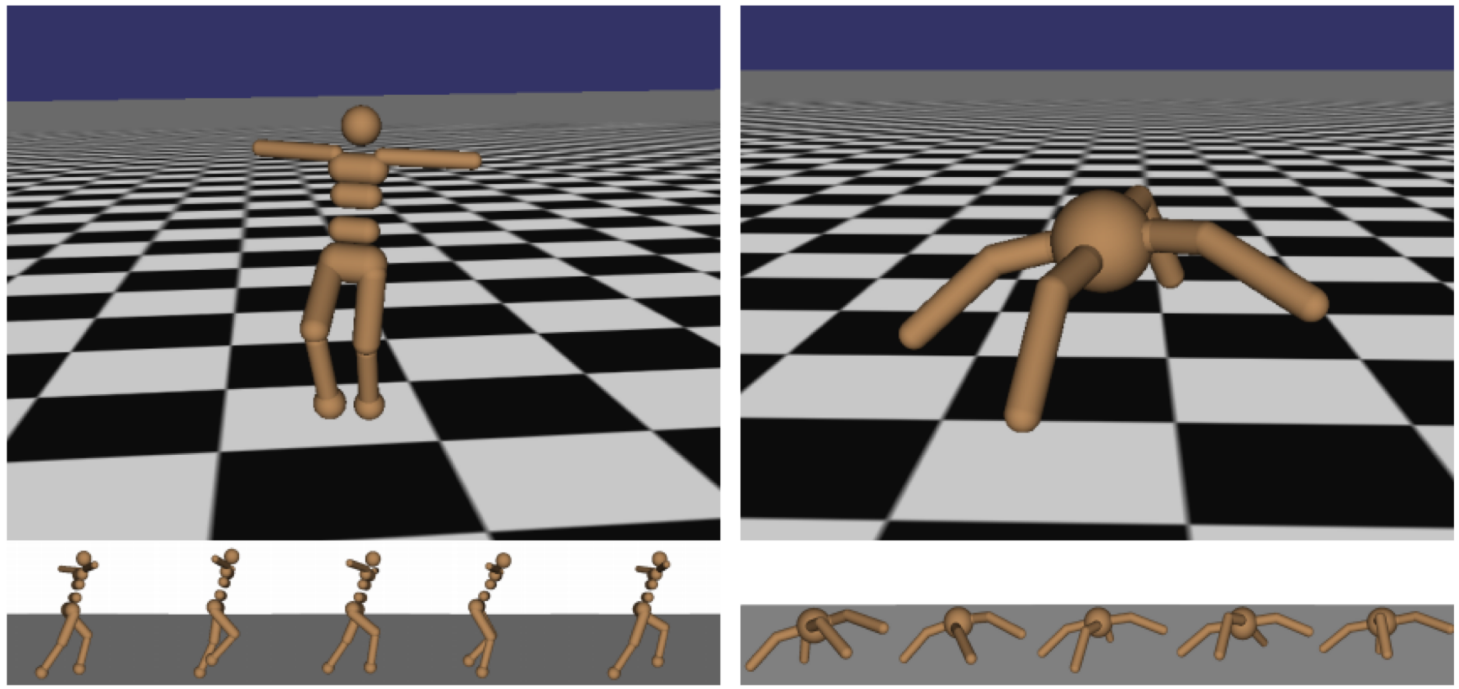
For our little broken-robot example:



Decision process defined by

- $\mathcal{S} = \{A1, A2, A3, A4, B1, B3, B4, C1, C2, C3, C4\}$
- $\mathcal{A} = \{Up, Down, Left, Right\}$
- $R = \{A4 = 1, B4 = -1, Others = -0.0001\}$
- $T = \left\{ \begin{matrix} C1, Up \rightarrow \{B1\colon 0.8, C1\colon 0.1, C2\colon 0.1\} \\ C1, Left \rightarrow \{C1\colon 0.9, B1\colon 0.1\} \\ \vdots \end{matrix} \right\}$

Robot Locomotion



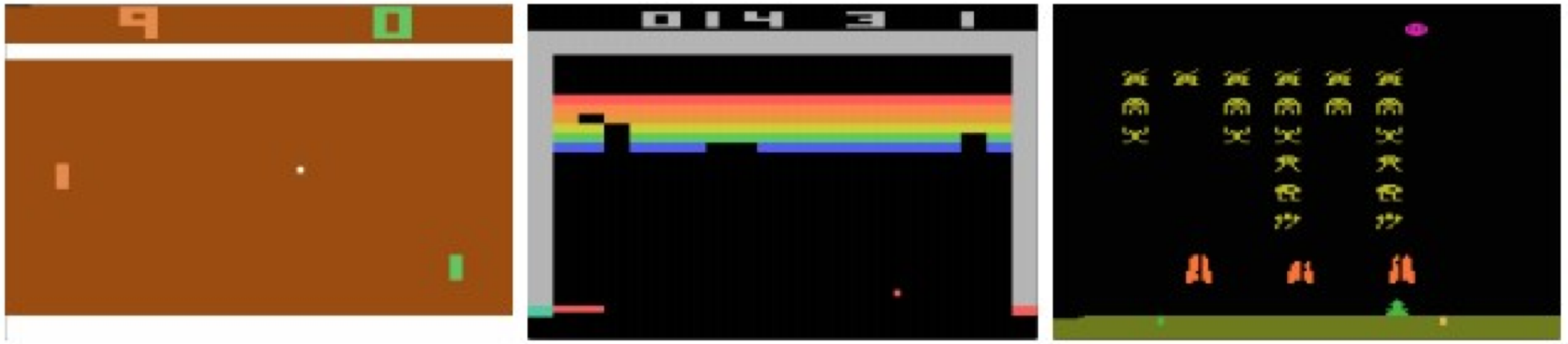
Figures copyright John Schulman et al., 2016. Reproduced with permission.

Make the robot move forward

State Space: Angle and position of the joints
Action Space: Torques applied on joints
Reward Function: 1 at each time step upright + forward movement
Transition Function: Deterministic simulator

Slide Credit: Fei-Fei Li, Justin Johnson-CS 231n

Atari Games



Figures copyright Volodymyr Mnih et al., 2013. Reproduced with permission.

Complete the game with the highest score

State Space: Raw pixel inputs of the game state
Action Space: Game controls e.g. Left, Right, Up, Down
Reward Function: Score increase/decrease at each time step
Transition Function: Game engine

Slide Credit: Fei-Fei Li, Justin Johnson-CS 231n

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:

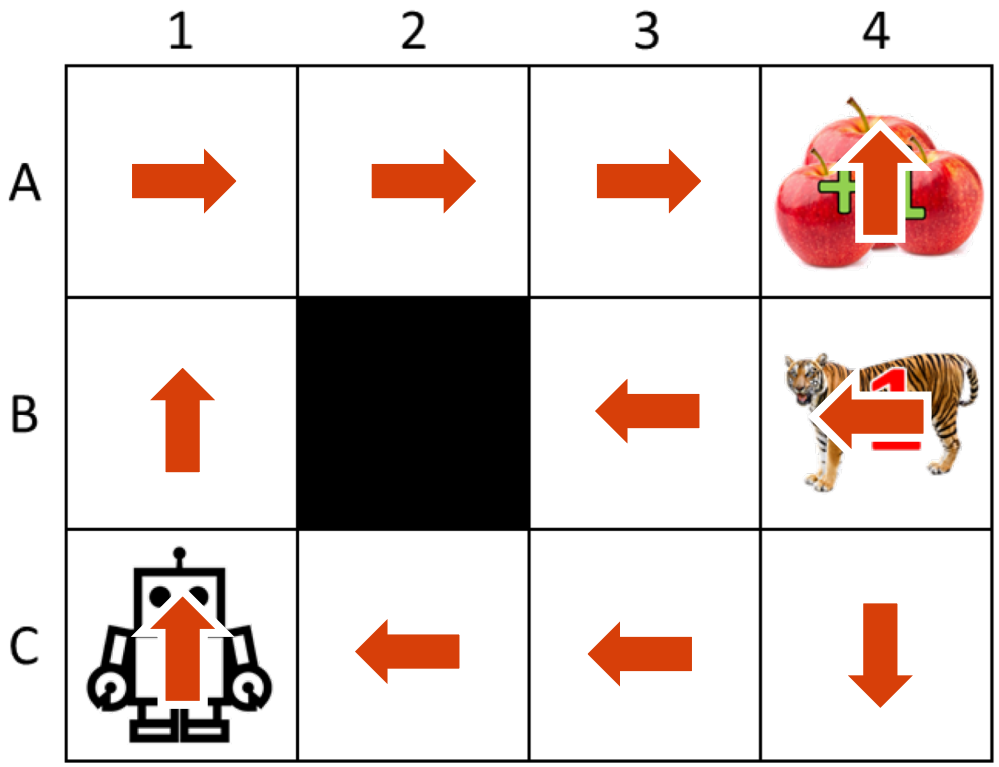
1. **What is a behavior policy?**

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: $\mathbf{a} = \boldsymbol{\pi}(\boldsymbol{s})$

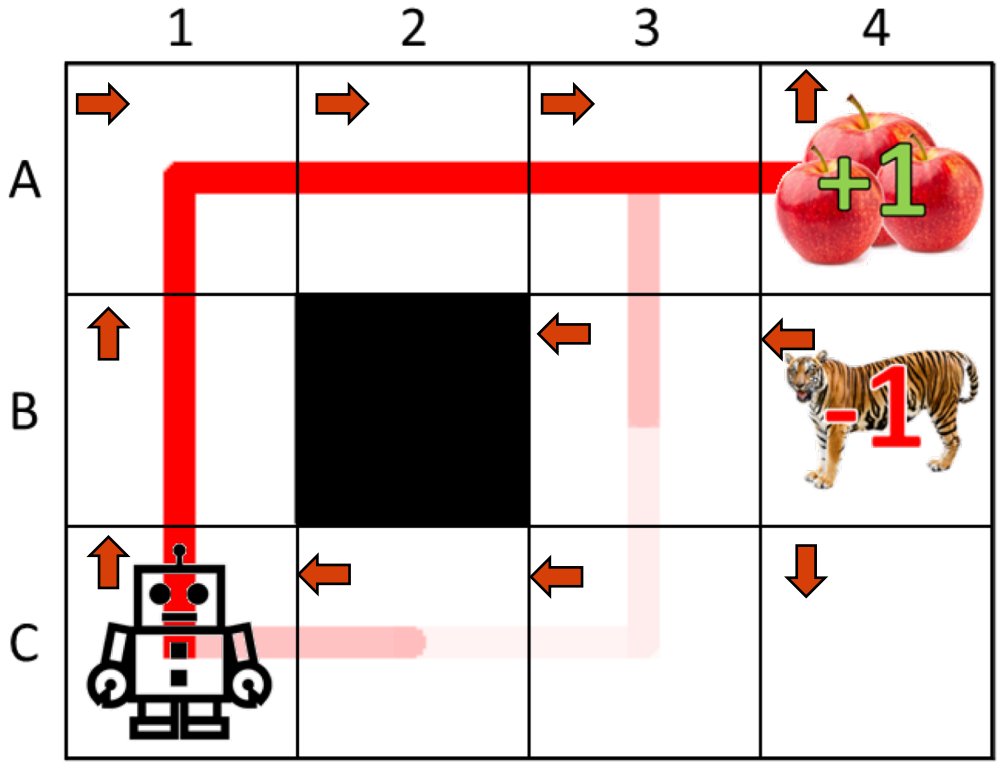
or

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

Denote the "best' behavior policy as π^*

Behavior Policies

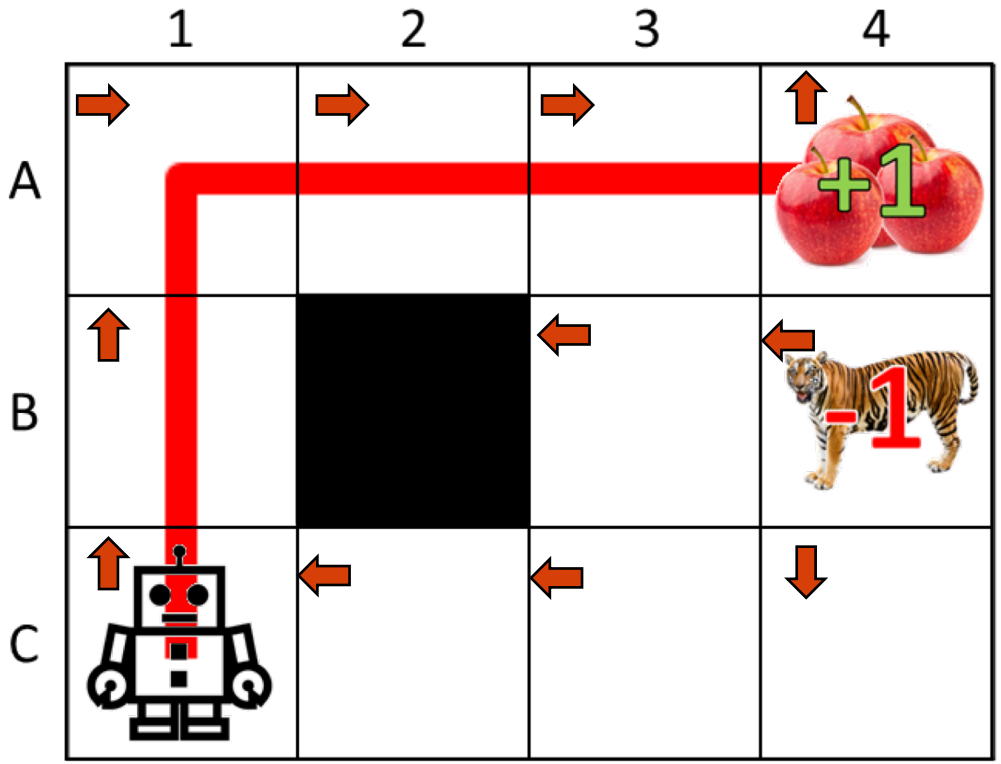
1. What is a behavior policy?



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:

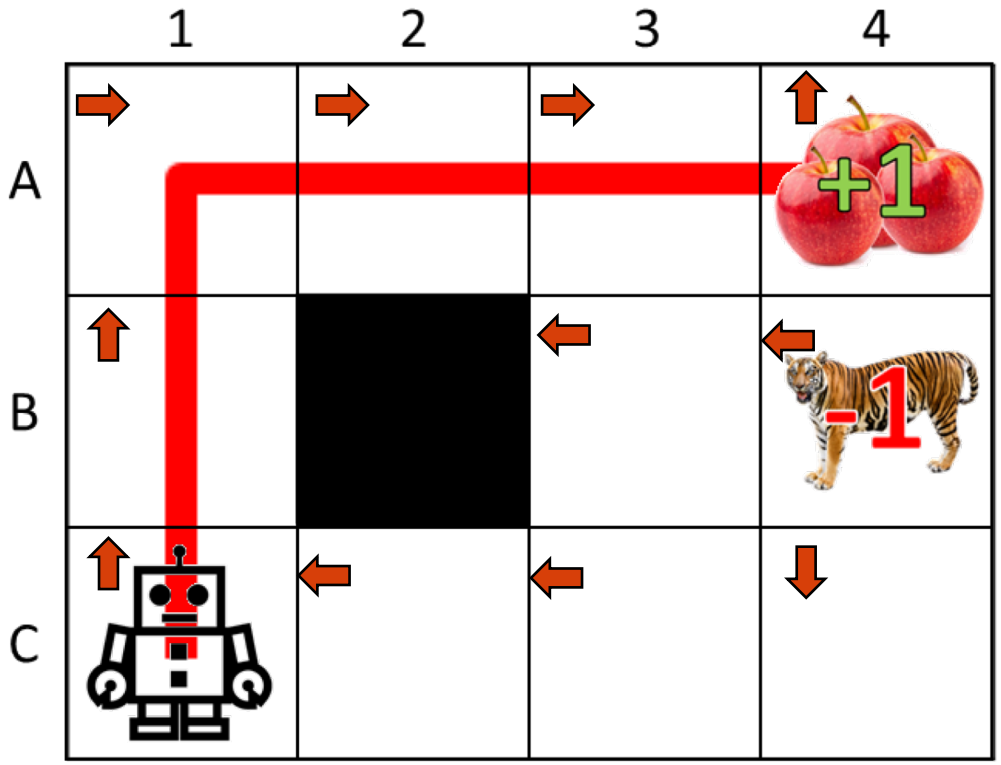
$$\sum_{t=0}^{\infty} R(s_t) = R(s_0) + R(s_1) + R(s_2) + \cdots$$

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

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:

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

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?**

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.

Model-free Reinforcement Learning: Directly learn a policy to maximize reward based on experience.

Today's Learning Objectives

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?

Imagine playing a new game whose rules you don't know;

after a hundred or so moves, your opponent announces,

"You lose".

- Russel and Norvig – Introduction to Artificial Intelligence

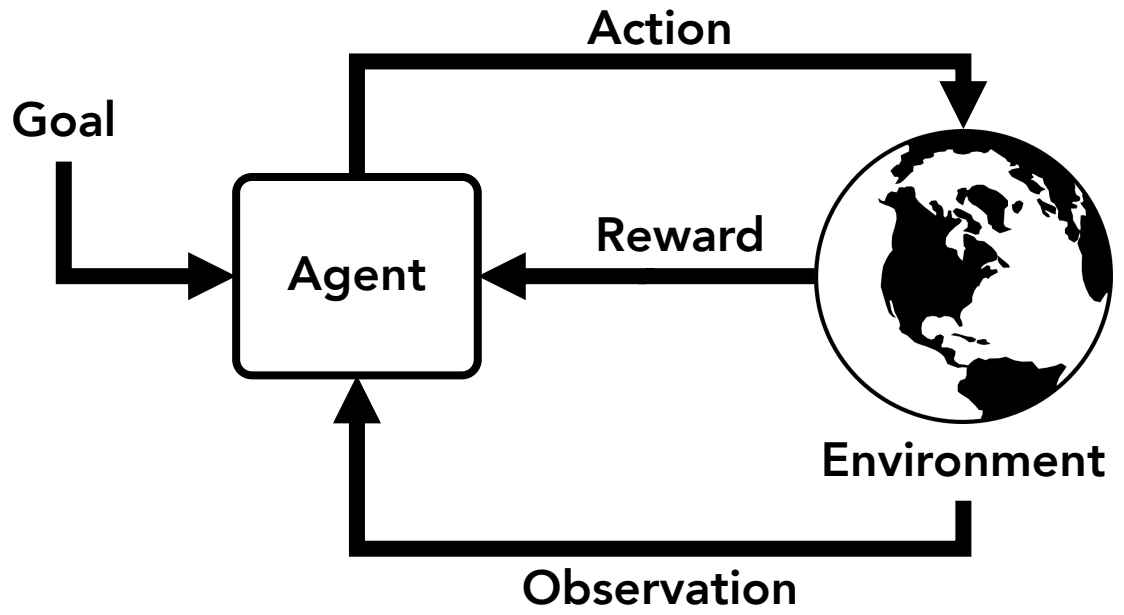
Reinforcement learning describes algorithms for producing good policies in MDPs when transition dynamics and reward functions are initially unknown.

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:

In model-free reinforcement learning, we directly try to estimate a good policy from experience interacting with the environment without trying to explicitly model the transition or reward functions.

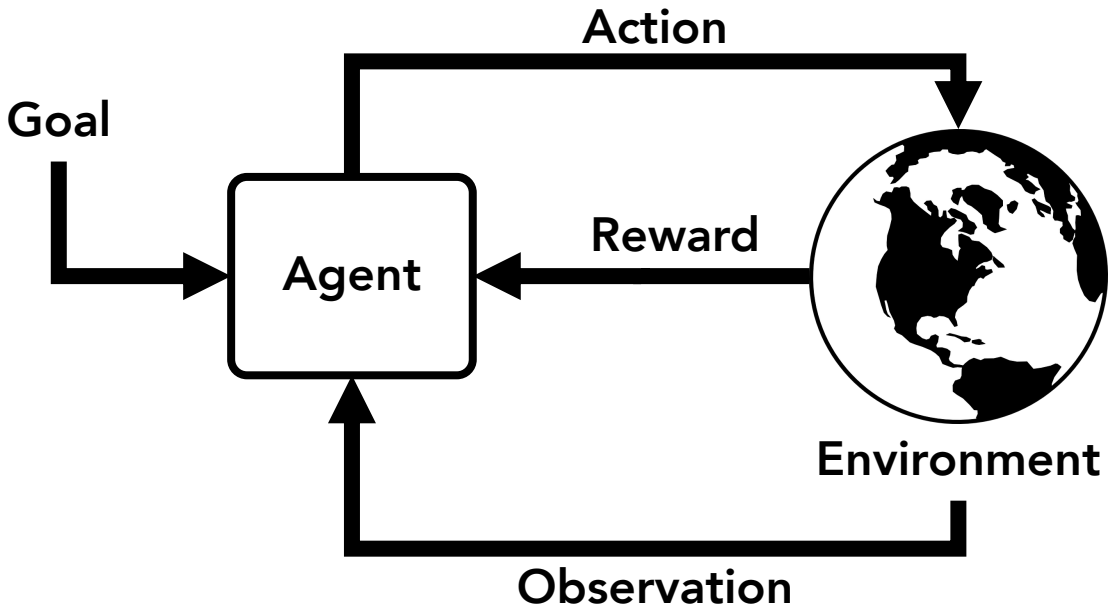


Model-free Reinforcement Learning:

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:

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

$$J(\theta)$$

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

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

Slide adapted from Dhruv Batra

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 $\tau = (s_0, a_0, s_1, a_1, s_2, \ldots)$

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

Slide adapted from Dhruv Batra

Expected rewards of policy $\pi(\cdot \mid \cdot\,; \theta)$:

$$J(\theta) = \int_{\tau} p(\tau; \theta) r(\tau) d\tau$$

Lets differentiate with respect to θ:

$$\nabla_\theta J(\theta) = \int_{\tau} \underbrace{\nabla_\theta p(\tau; \theta) r(\tau) d\tau}_{\text{Intractable}}$$

Slide adapted from Dhruv Batra

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:

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

Slide adapted from Dhruv Batra

Lets differentiate with respect to θ:

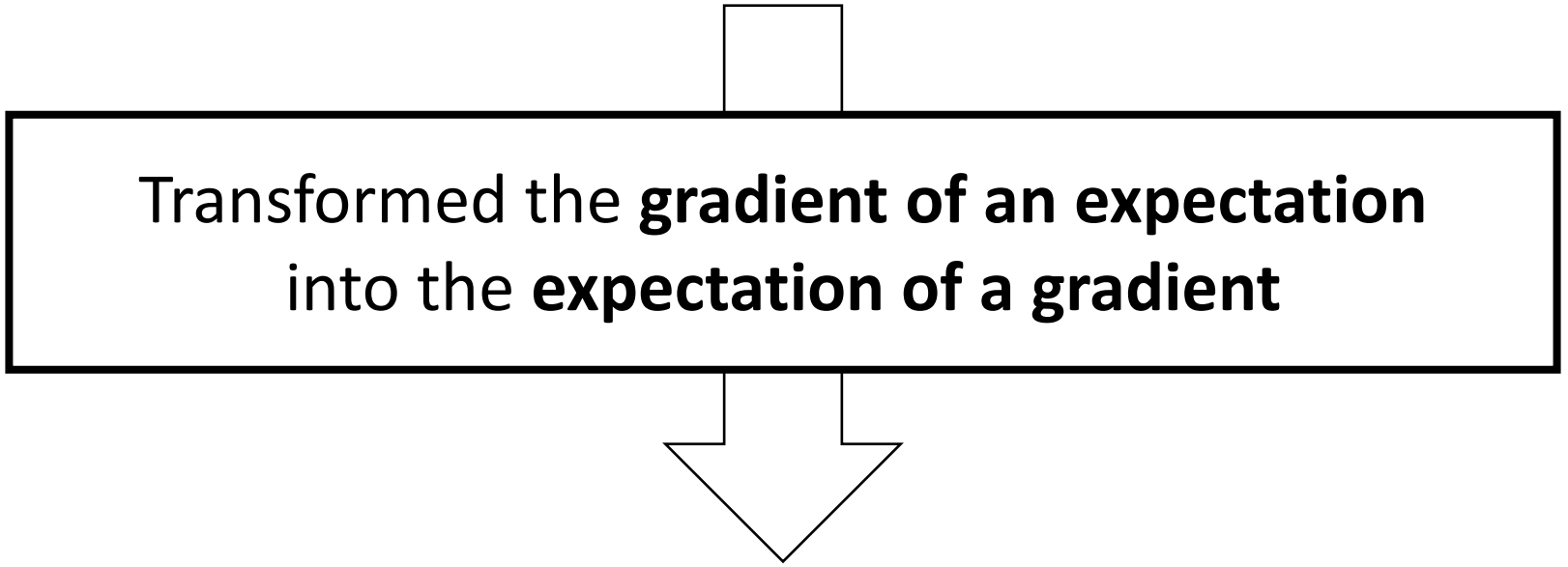
$$\nabla_\theta J(\theta) = \int_\tau \nabla_\theta p(\tau;\theta)\, r(\tau)\ d\tau$$

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

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

Slide adapted from Dhruv Batra

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



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

Slide adapted from Dhruv Batra

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

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

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

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

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

No model needed! Model-free RL.

Slide adapted from Dhruv Batra

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

Monte Carlo Approximation:

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

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.

Slide adapted from Dhruv Batra

REINFORCE Algorithm

(Williams, 1992)

While not converged:

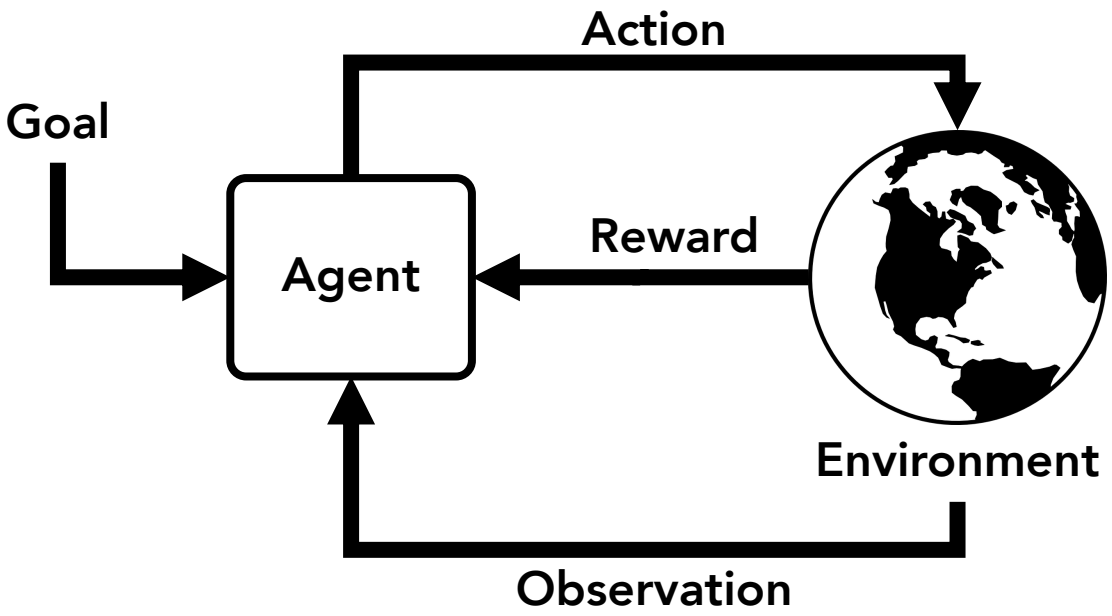
1. Run policy to collect trajectories $\tau_j = (\mathrm{s_{j0}}, \mathrm{a_{j0}}, s_{j1}, a_{j1}, s_{j2}, \dots)$ and reward $\mathrm{r}(\tau_j)$
2. Compute gradient estimate $\nabla_\theta J(\theta) \approx \sum_j \sum_{s_{ji}, a_{ji} \in \tau_j} \nabla_\theta \log \pi(a_{ji} | s_{ji}; \theta)\, r(\tau_j)$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

Slide adapted from Dhruv Batra

https://colab.research.google.com/drive/1rVZfMPmCU7xnhaWjzBP3FNRX4FNuTEQ9?usp=sharing

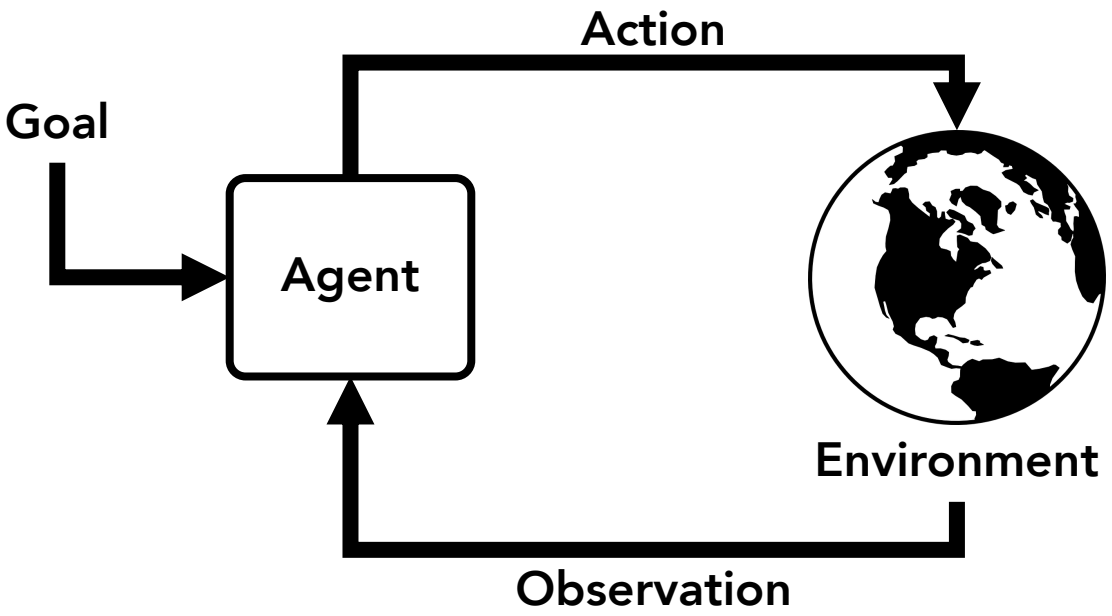
Reinforcement Learning

- Environment provides feedback
- No examples of optimal policy



Imitation Learning

- Have expert demonstrations (possibly interactive)



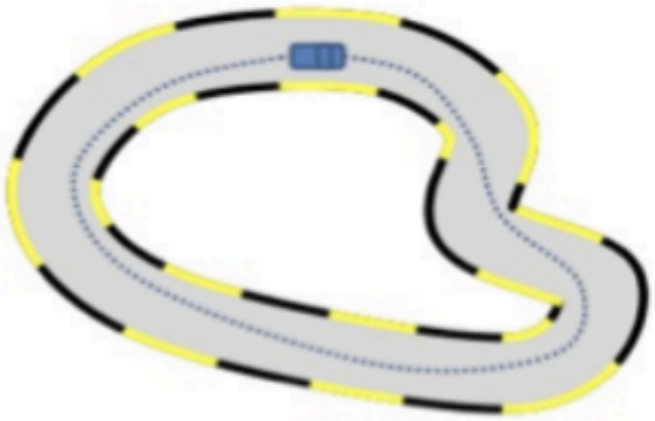
Imitation Learning

- Assume access to an expert demonstrator π^* at some point or another and to varying levels of interactivity. **No reward signals provided.**

Imitation Learning

Example #1: Racing Game

(Super Tux Kart)



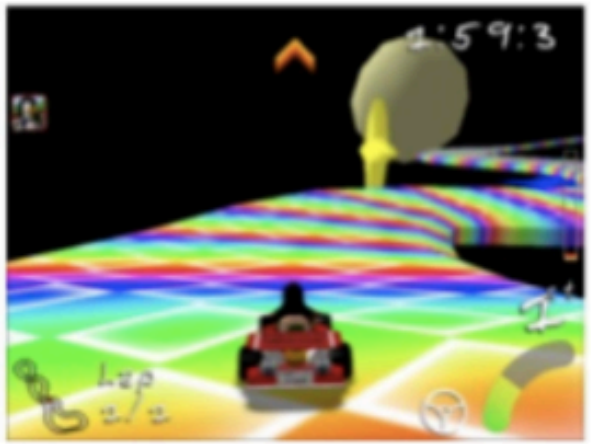
s = game screen

a = turning angle

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

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

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



Images from Stephane Ross

Slides Credit: Yisong Yue

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

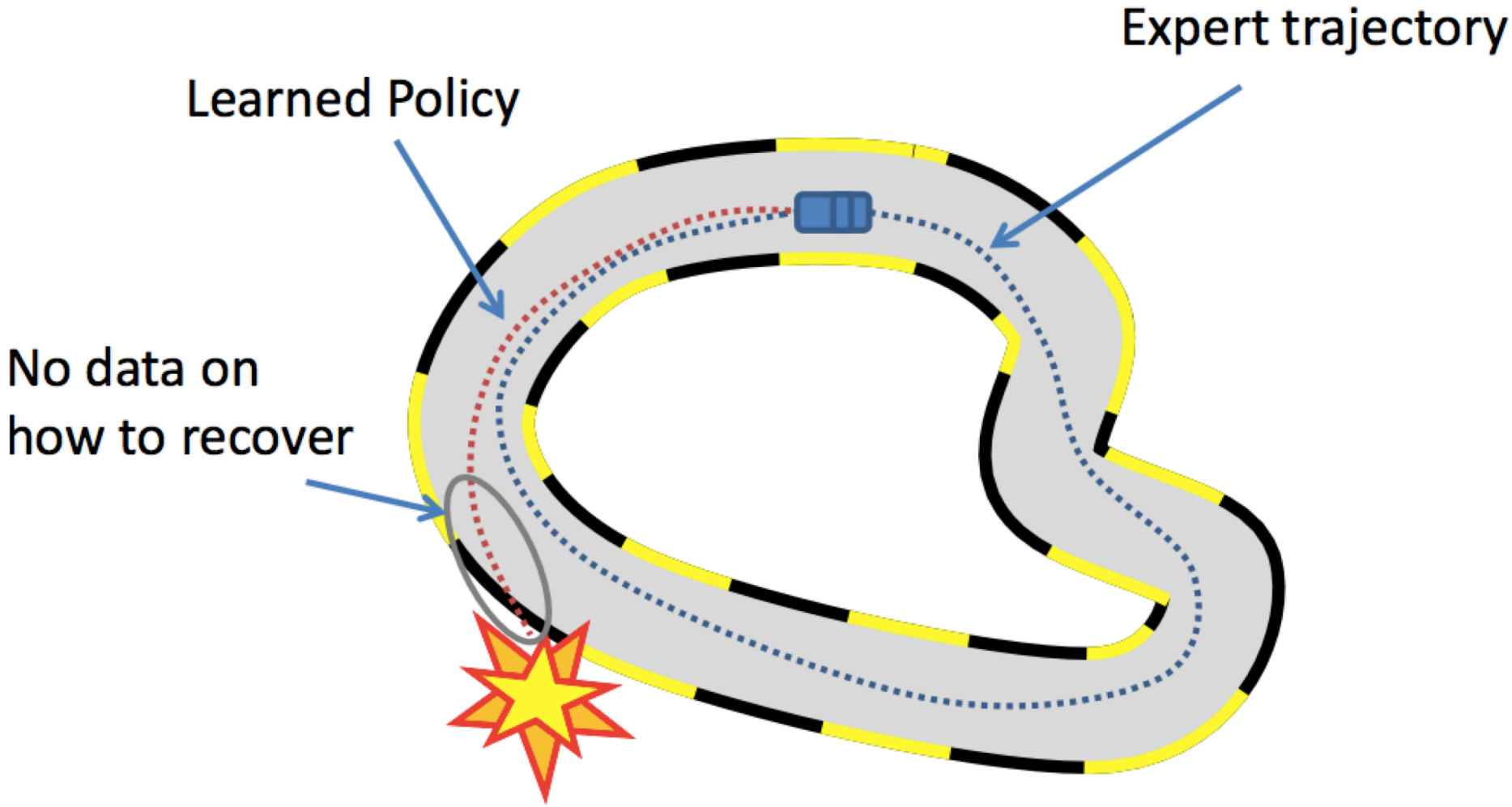
Slide adapted from Yisong Yue

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.

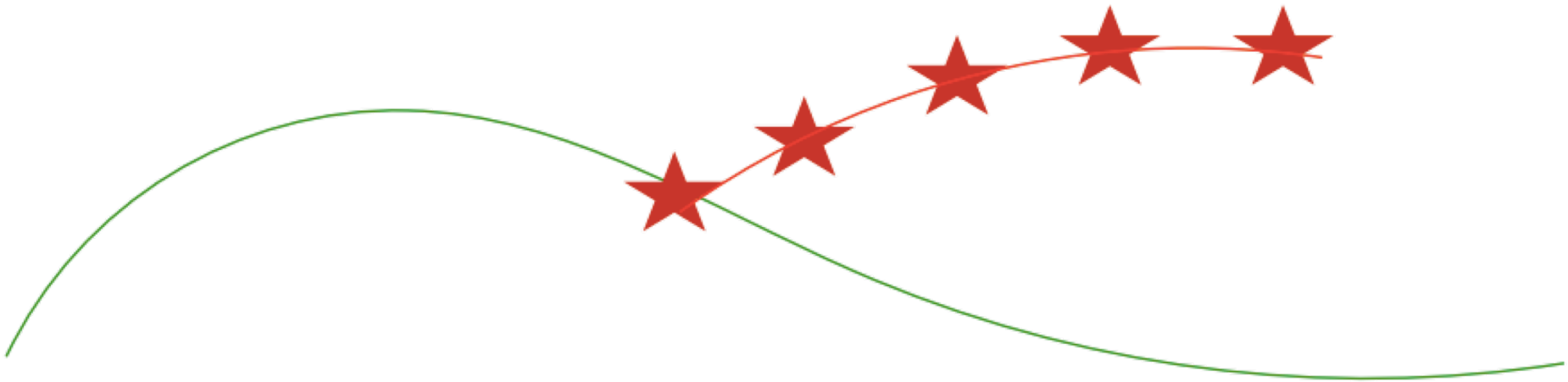
Data Distribution Miss-match



Slides Credit: Yisong Yue

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.

Today's Learning Objectives

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?~~

Next time

Next Time: Review for the final exam.