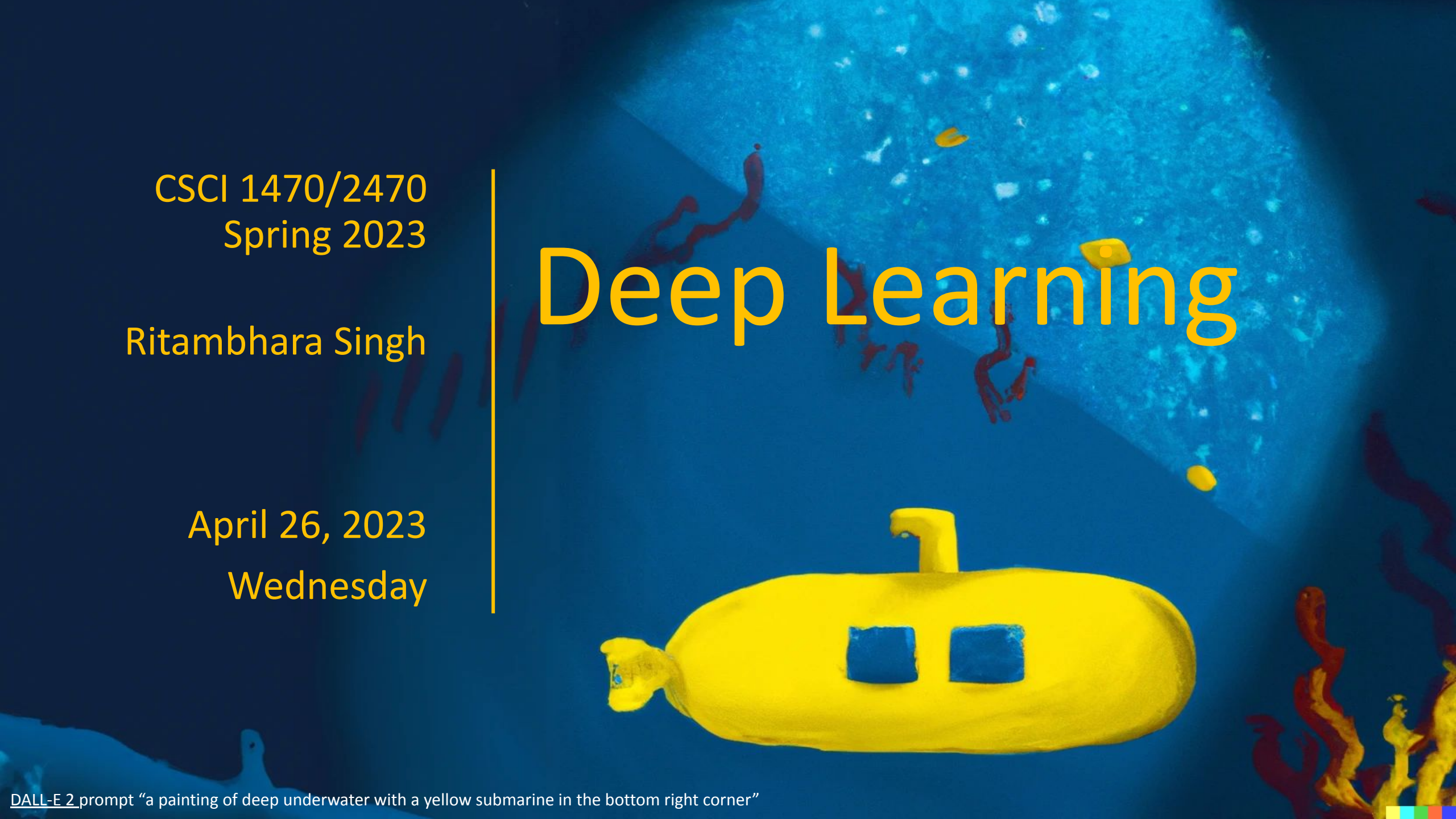
CSCI 1470/2470
Spring 2023

Ritambhara Singh

April 26, 2023

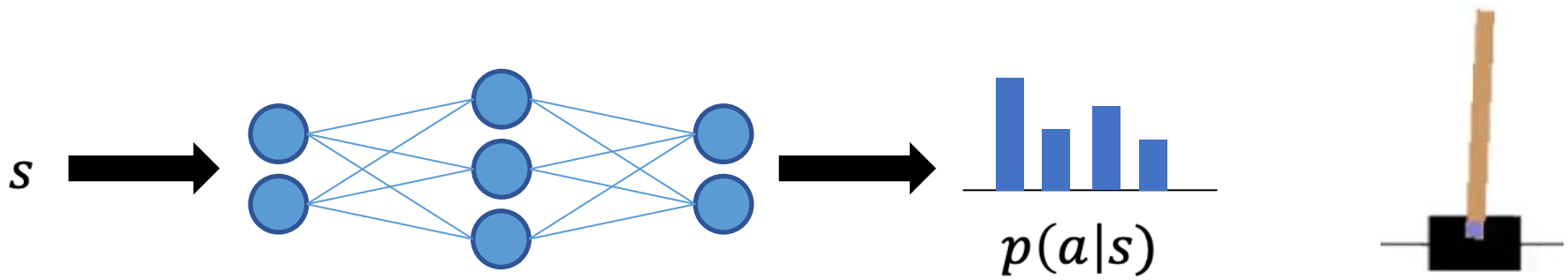
Wednesday

Deep Learning



DALL-E 2 prompt “a painting of deep underwater with a yellow submarine in the bottom right corner”

Review: Policy Network for Cart Pole



Policy Gradient: $-\sum_{t=1}^{T} \nabla \log p(a_t|s_t)\, D(s_t, a_t)$

Review: REINFORCE: Pseudo Code

Initialize model weights θ

Repeat until done (converge, time limit expired, etc.):

Run N episodes of environment simulation, each for T timesteps

For each episode

For $t = 1$ to $t = T$

$$\theta \leftarrow \theta + \text{OptimizerStep}(\nabla \log p(a_t|s_t) D\ (s_t, a_t))$$

Your favorite optimizer (SGD, Adam, ...)

Return θ

Reinforce vs DQN

Pros

- Policy often easier to learn than Q function
- Automates explore vs. exploit tradeoff
  - Policy network starts off random and gradually becomes better as it is trained for more and more episodes
- Can learn stochastic policies
  - More naturalistic behavior
- In practice, can converge faster than DQN

Cons

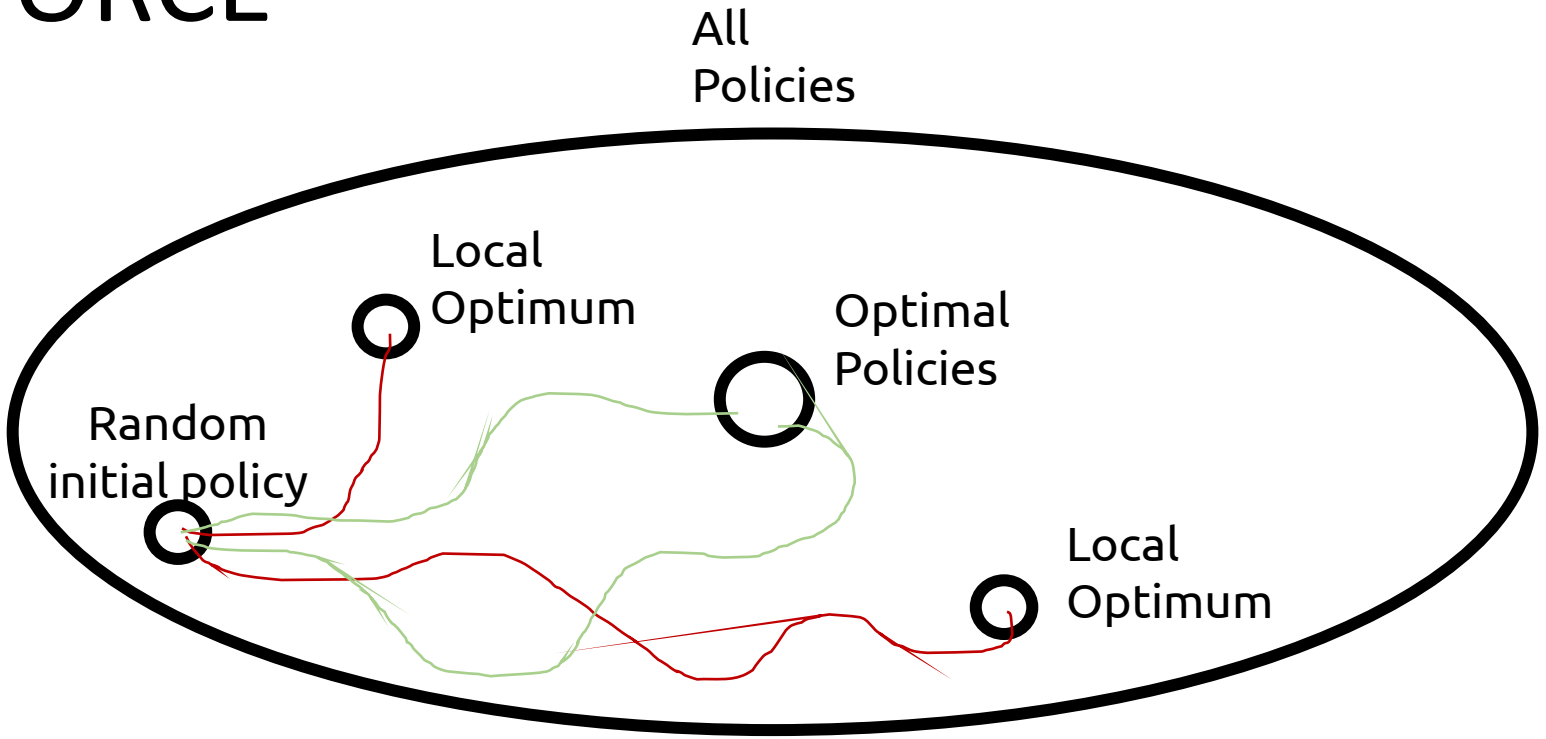
- Finds local optima more often than DQN...
- Unstable training
- Gradient updates only at end of each game (DQN updates after every step)

We'll see how to fix these two issues in the next lecture...

Issues with REINFORCE

- **High Variance**
  - Multiple runs of training an agent with REINFORCE can yield very different results
  - Susceptible to local optima

First, going to address this problem…

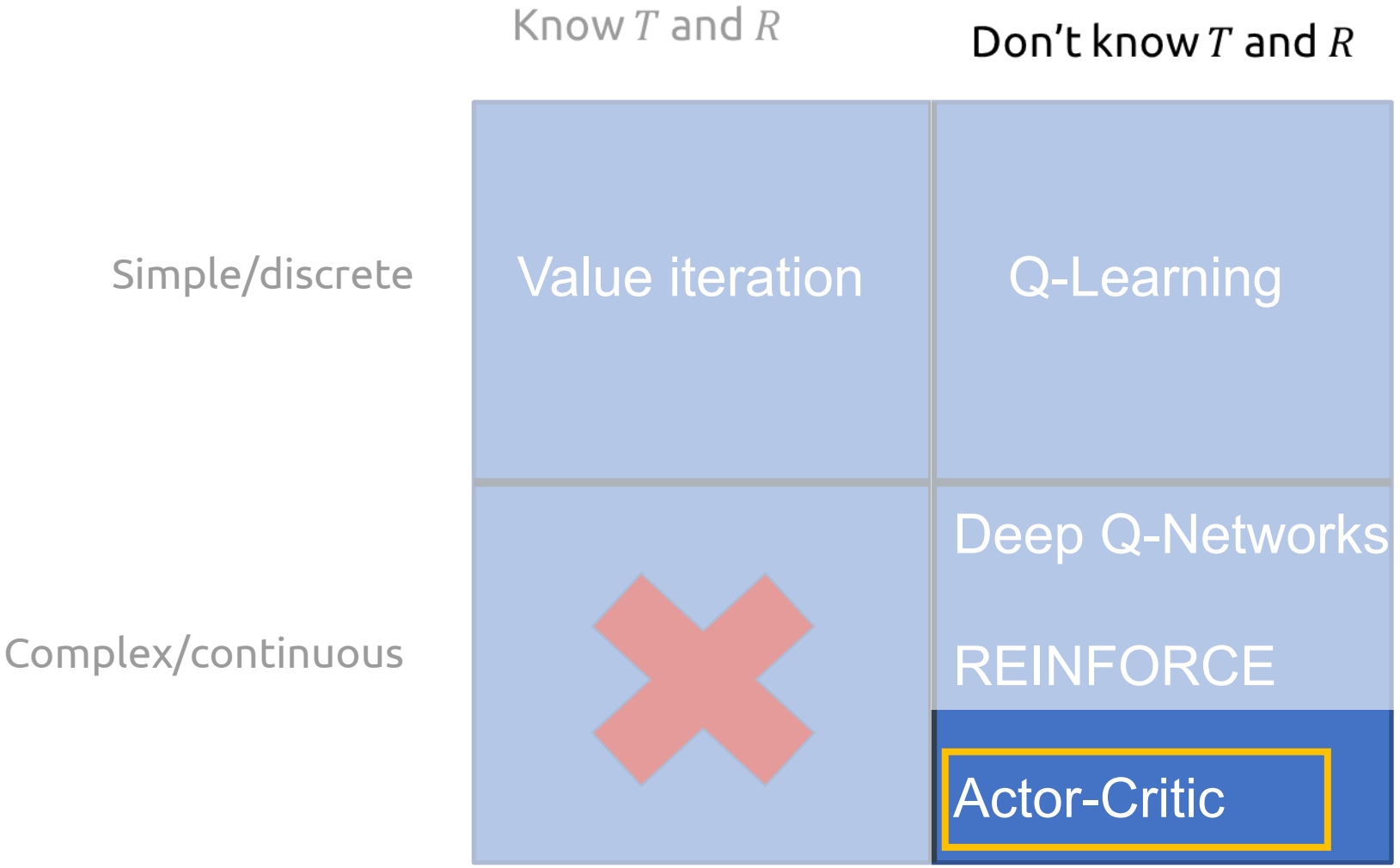


- **High "sample complexity"**
  - Must play an entire episode to get gradient, takes many episodes to learn

…which will actually give us a solution for this problem via a simple extension

The Solution: Looking at Policy Gradient through a different “lens”

Organizing RL problems/algorithms



For a more complete taxonomy of RL algorithms, see https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html#citations-below

The “Actor Critic” Framework

- Consider the REINFORCE gradient:

$$-\sum_{t=1}^{T} \nabla \log p(a_t|s_t) D(s_t, a_t)$$

The “Actor Critic” Framework

- Consider the REINFORCE gradient:

$$-\sum_{t=1}^{T} \nabla \log p(a_t|s_t) D(s_t, a_t)$$

Actor:
Specifies how to (probabilistically) choose actions for a given state

The “Actor Critic” Framework

- Consider the REINFORCE gradient:

$$-\sum_{t=1}^{T} \nabla \log p(a_t|s_t) D(s_t, a_t)$$

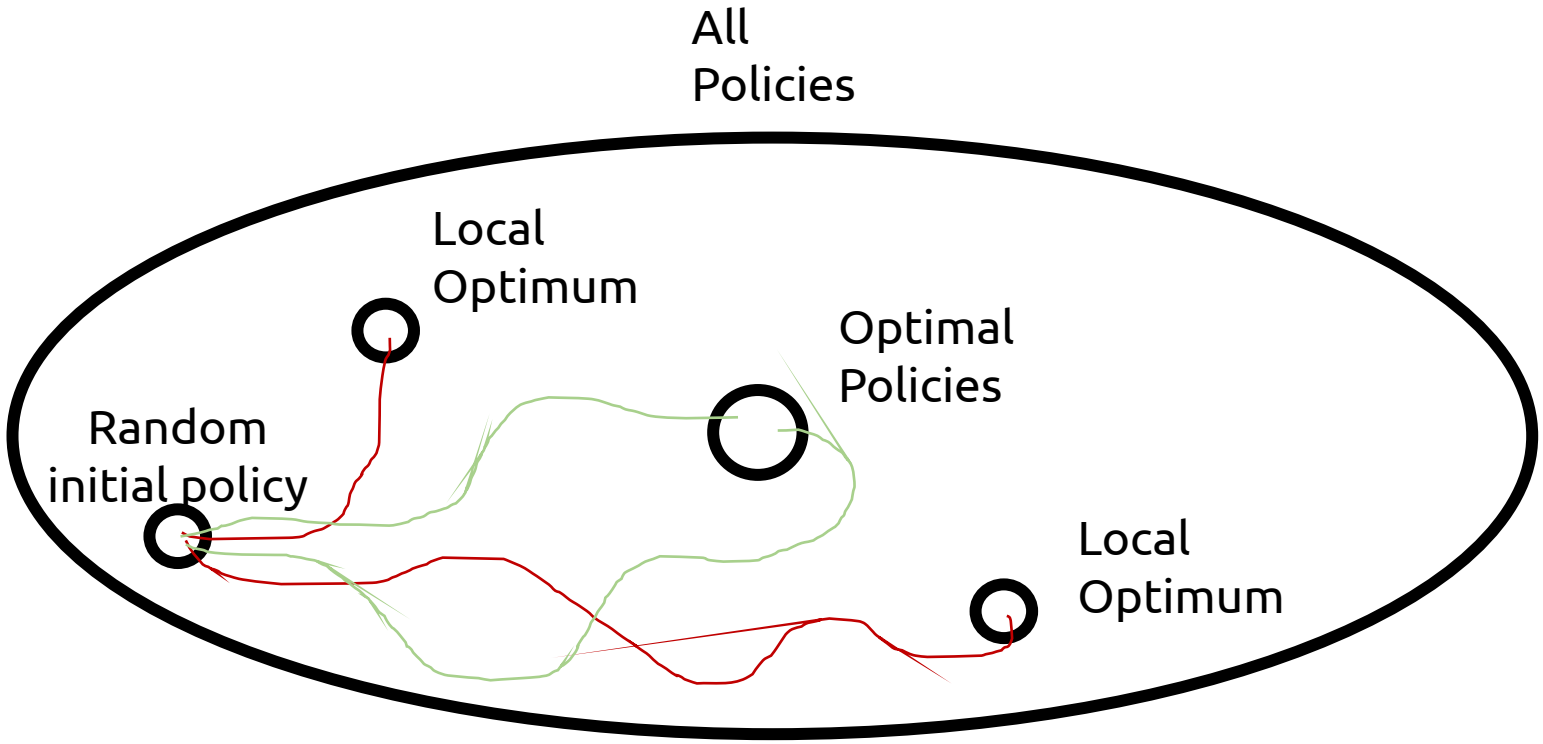
Actor:
Specifies how to (probabilistically) choose actions for a given state

Critic:
“Scores” the goodness/badness of taking an action

Different critics are possible
- E.g. $Q(s_t, a_t)$, if we knew it
- **Recall that $D(s_t, a_t)$ is our single-episode estimate of $Q(s_t, a_t)$**

The trick to solving our problems:
Coming up with a better critic function…

The Problem: High Variance

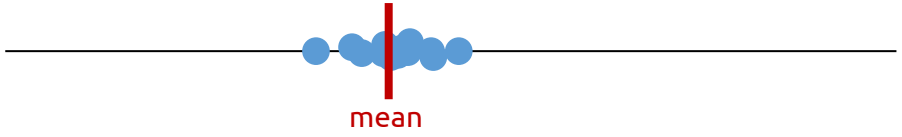


- To understand why this happens, have to dig into the math a little bit

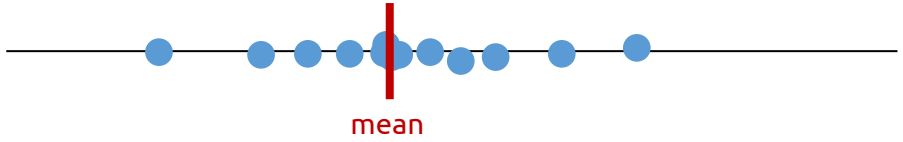
High Variance

- For a random variable X: $Var[X] = E[(X - E[X])^2]$
  - The expected squared difference from the expected value
  - "The average distance from the average"

Low Variance



High Variance



High Variance in REINFORCE

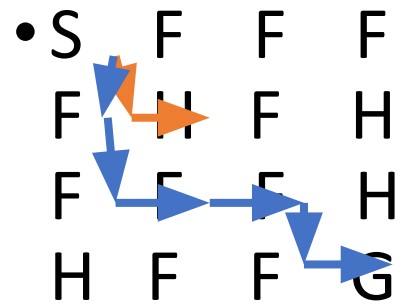
Out of the two terms, which is more likely produce high variance?

- The REINFORCE gradient: $-\sum_{t=1}^{T} \nabla \log p(a_t|s_t) D(s_t, a_t)$
- $D(s_t, a_t)$ is a random variable, because it depends on following a stochastic policy (and a potentially stochastic transition function)
- Assertion: $D(s_t, a_t)$ is high variance
- ***Why?***
  - $D(s_t, a_t)$ can be very low ***or*** very high depending on subsequent actions
  - Especially for actions taken early in the episode
  - Especially early in training, when the policy is mostly random

High Variance Rewards: Frozen Lake

- Frozen Lake
  - S: Starting Point
  - F: Frozen
  - H: Hole, ends episode
  - G: Goal

- Look at $D(s_0 = S, a_0 = \text{down})$

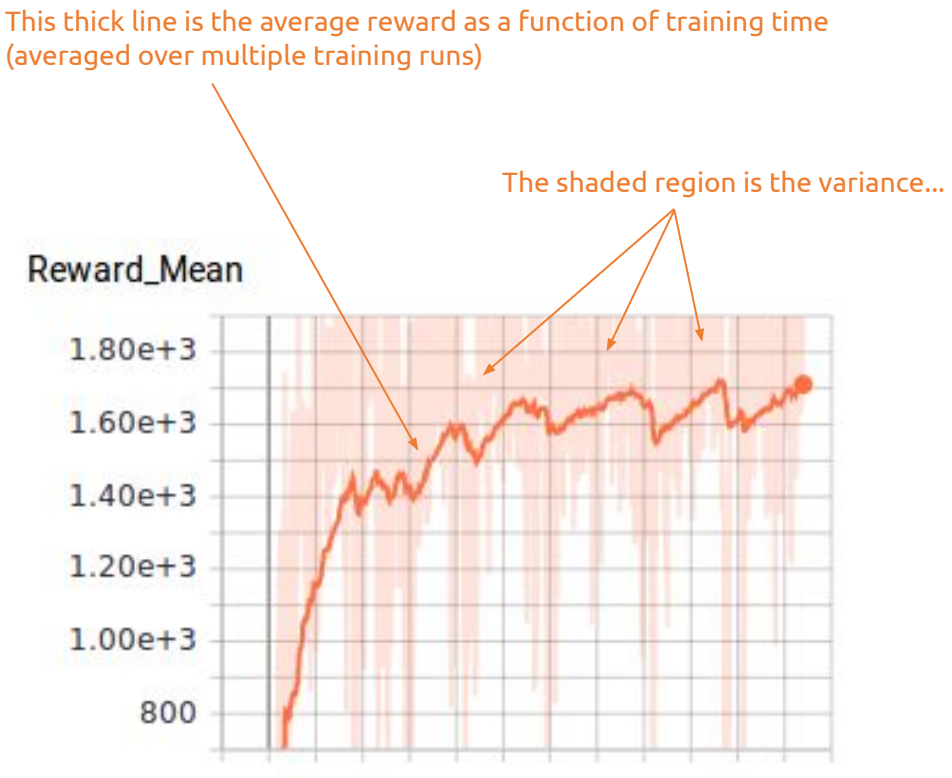


Very Low Reward

Very High Reward

High Variance Rewards: VizDoom

https://www.youtube.com/watch?v=93TrfMZ2Dqs



https://www.oreilly.com/ideas/reinforcement-learning-with-tensorflow

High Variance in REINFORCE

- $D(s_t, a_t)$ is high variance
- This in turn makes $-\sum_{t=1}^{T} \nabla \log p(a_t|s_t) D(s_t, a_t)$ have high variance
- ***What's the consequence of high variance gradients?***
  - Magnitude and direction of gradients is unstable
  - Need very low learning rate to keep training from blowing up
  - Low learning rate → need ***lots*** of training episodes to converge

High Variance in REINFORCE

- Naïve solution: if the gradients fluctuate too much, just scale them so they don't fluctuate as much
  - $-\sum_{t=1}^{T} \nabla \log p(a_t|s_t) D(s_t, a_t) \rightarrow -\boldsymbol{\beta} \sum_{t=1}^{T} \nabla \log p(a_t|s_t) D(s_t, a_t)$

- ***Why won't this work?***
  - Scaling the gradients is equivalent to scaling the learning rate, which is exactly what we're trying to avoid!

Solving High Variance: A Better Critic

- A better critic function:

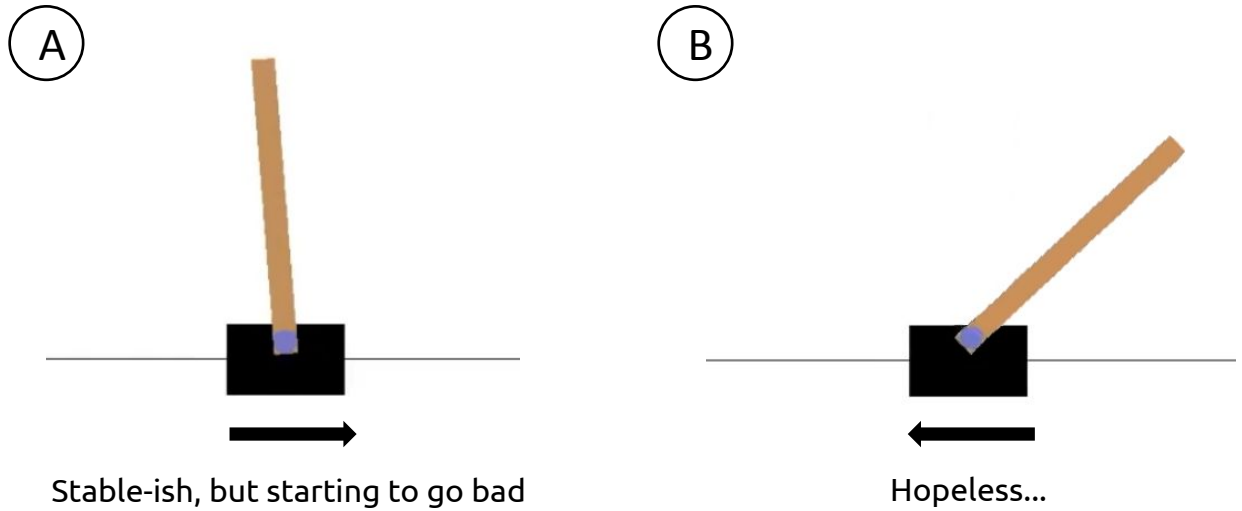
$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$$
$$= Q(s_t, a_t) - \max_{a_t} Q(s_t, a_t)$$

- This is called the ***advantage function***
  - The "advantage" of taking action a_t vs. taking the best possible action in state s_t, under the current policy

- **Claim:** $A(s_t, a_t)$ has lower variance than $Q(s_t, a_t)$ (or $D(s_t, a_t)$)

Why $A(s_t, a_t)$ has lower variance

- Consider these two states in Cart Pole



Why $A(s_t, a_t)$ has lower variance

- What would be the Q value of taking the □ action in either state?

Stable-ish, but starting to go bad

Hopeless...

$Q(S_A, \leftarrow)$: Good! (helps stat

$Q(S_B, \leftarrow)$: Bad... (tipping over and this isn't helping)

$Q(S_A, \leftarrow) - Q(S_B, \leftarrow)$: **Large, i.e. high variance...**

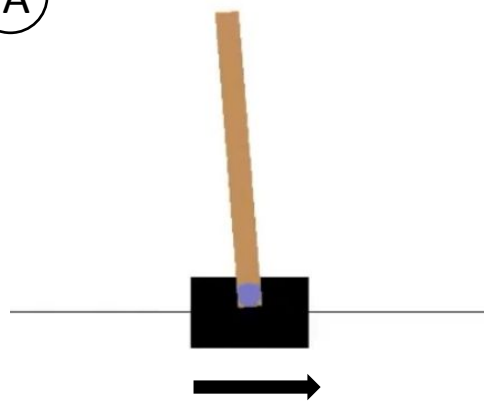
Why $A(s_t, a_t)$ has lower variance

What will be the A values for situation A and B (conceptually)?

- Now consider the A value of taking the ← action in either state:

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$$
$$= Q(s_t, a_t) - \max_{a_t} Q(s_t, a_t)$$

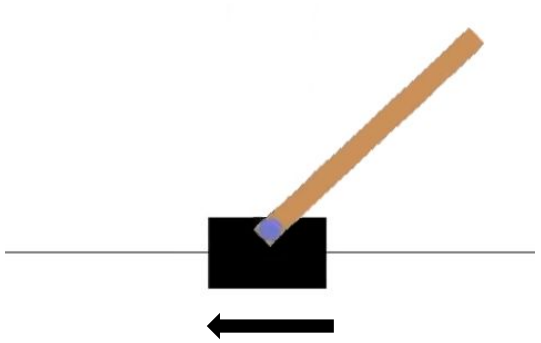
A



Stable-ish, but starting to go bad

$A(S_A, \leftarrow)$: Zero (this is the best

B

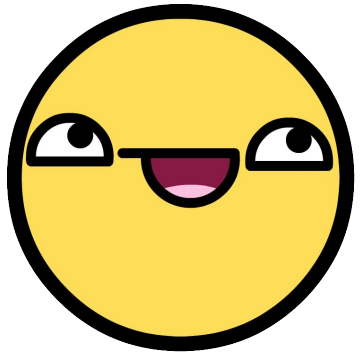


Hopeless...

$A(S_B, \leftarrow)$: Tiny negative number (not the best action, but state is already bad)

$A(S_A, \leftarrow) - A(S_B, \leftarrow)$: **Small difference! Lower variance**

The Advantage of Using Advantage



- **The main idea:** to learn a policy, it doesn't matter whether some states are better than others. All that matters is which *actions* are better for a given state.
- ***Factor out*** the difference in state value and just look at the difference in action value
- $|A(s_1, a_1) - A(s_2, a_2)| < |Q(s_1, a_1) - Q(s_2, a_2)|$

Using Advantage in REINFORCE

- Substitute in the advantage function for the critic:

$$-\sum_{t=1}^{T} \nabla \log p(a_t|s_t) A(s_t, a_t)$$

- Of course, in practice, we don't have Q, so we use D instead:

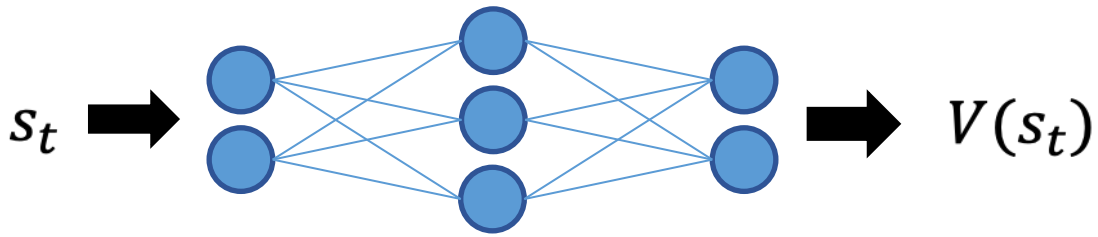
$$-\sum_{t=1}^{T} \nabla \log p(a_t|s_t)\big(D(s_t, a_t) - V(s_t)\big)$$

Are we done?

Wait a minute…we *also* don't have V…

Value Networks

- Use another neural net, the ***Value Network***, to learn an approx. of V
- Like Policy Network, architecture depends on the MDP being solved (i.e. what data representation its state uses)
- For Cart Pole, state is a 4D vector of real numbers
  - [Cart pos, cart vel, pole ang, pole tip vel]
- Fully connected net is appropriate here:



How to Train a Value Network

- Recall the definition of the Value Function
  - Expected future return from being in a given state
  - $V(s_t) = \max_{a_t} Q(s_t, a_t)$
- ***What data do we get from each episode that we might use for training?***
  - The discounted future reward that we got in that episode, from each timestep
  - $D(s_t, a_t) = \sum_{i=t}^{T} \gamma^{i-1} r(s_i, a_i, s_{i+1})$
- Idea: just as we used D as an approximation for Q, let's also use it to approximate V
  - i.e. train the Value Network with (input, output) pairs of the form $\left(s_t, D(s_t, a_t)\right)$
  - When trained with many such pairs obtained over many episodes, the network will learn to output a good estimate of the future reward that can be expected starting from the input state—**in other words, the Value Function!**

Any questions?

How to Train a Value Network

- Training loss: $L(s_t) = \big(D(s_t, a_t) - V(s_t)\big)^2$, where V is the Value Network
- ***Does this look familiar?***
- This is exactly the "advantage" term in our gradient update!

$$-\sum_{t=1}^{T} \nabla \log p(a_t|s_t)\big(D(s_t, a_t) - V(s_t)\big)$$

- In other words: training the value network == minimizing the advantage
  - Which is exactly what we want in order to lower the variance!
- The $V(s_t)$ term in the gradient update is also called a "baseline," which gives this algorithm the name ***REINFORCE with Baseline (RwB)***

REINFORCE: Pseudo Code

Modify the code to RL w/ baseline

Initialize policy net weights θ

Repeat until done (converge, time limit expired, etc.):

Run N episodes of environment simulation, each for T timesteps

For each episode

For $t = 1$ to $t = T$

$$\theta \leftarrow \theta + \text{OptimizerStep}(\nabla \log p(a_t|s_t) D\ (s_t, a_t))$$

Return θ

RwB: Pseudo Code

Initialize policy net ***and*** value net weights θ

Repeat until done (converge, time limit expired, etc.):

Run N episodes of environment simulation, each for T timesteps

For each episode

For $t = 1$ to $t = T$

$$L_{\text{actor}} = -\log p(a_t|s_t)(D(s_t, a_t) - V(s_t))$$

$$L_{\text{critic}} = \left(D(s_t, a_t) - V(s_t)\right)^2$$

$$\theta \leftarrow \theta + \text{OptimizerStep}(\nabla(L_{\text{actor}} + L_{\text{critic}}))$$

Return θ

RwB: Pseudo Code

Initialize policy net ***and*** value net weights θ

Repeat until done (converge, time limit expired, etc.):

Run N episodes of environment simulation, each for T timesteps

For each episode

For $t = 1$ to $t = T$

In practice, batch episodes and/or timesteps rather than looping over them

$$L_{\text{actor}} = -\log p(a_t|s_t)(D(s_t, a_t) - V(s_t))$$

$$L_{\text{critic}} = \left(D(s_t, a_t) - V(s_t)\right)^2$$

$$\theta \leftarrow \theta + \text{OptimizerStep}(\nabla(L_{\text{actor}} + L_{\text{critic}}))$$

Return θ

Cart pole with Actor-Critic

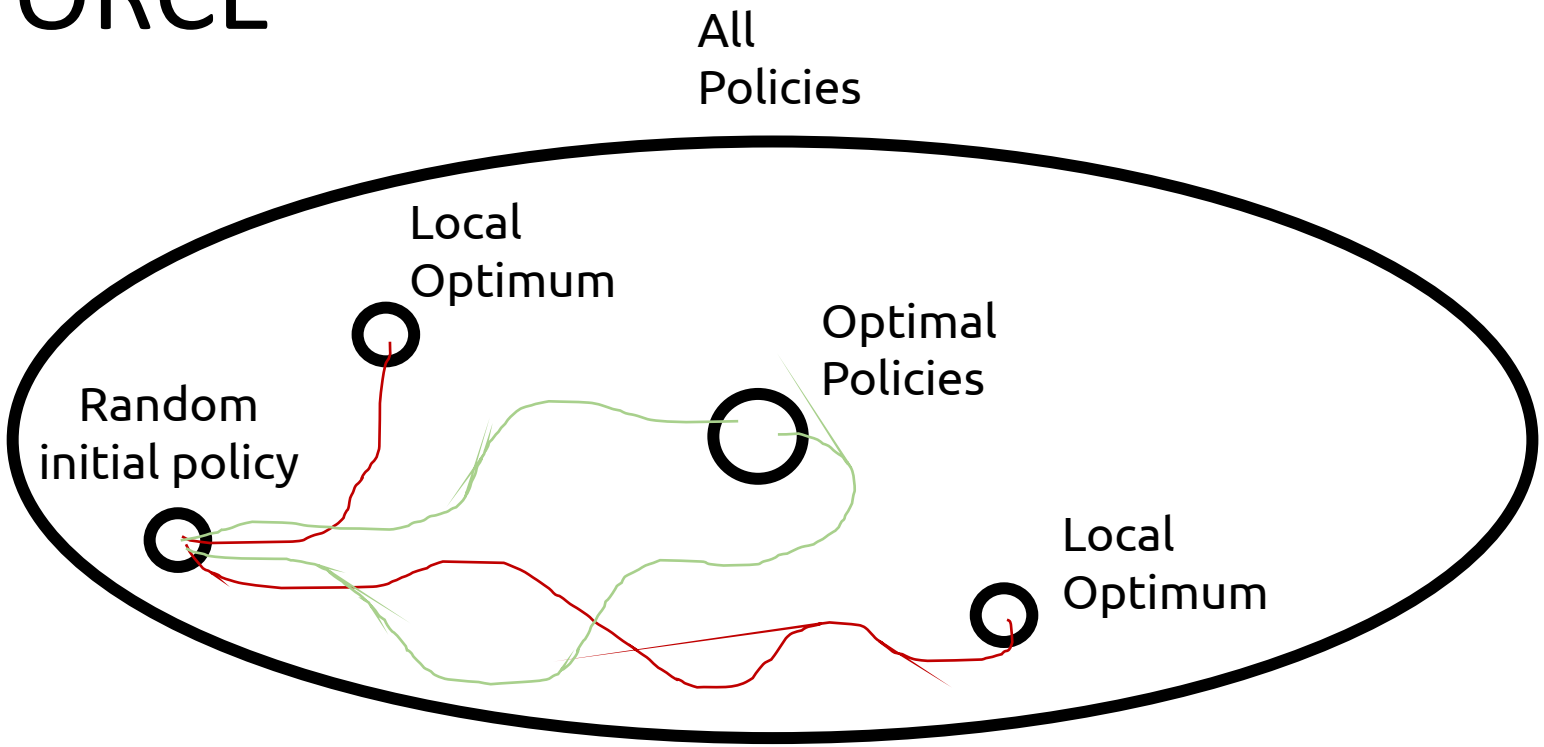
The number of episodes needed to obtain maximum reward << than that for REINFORCE

https://medium.com/nerd-for-tech/policy-gradients-reinforce-with-baseline-6c871a3a068

Issues with REINFORCE

- **High Variance**
  - Multiple runs training an agent with REINFORCE can yield very different results based on luck to local optima

First, go… this problem…



- **High "sample complexity"**
  - Must play an entire episode to get gradient, takes many episodes to learn

…which will actually give us a solution for this problem via a simple extension

Dealing with Sample Complexity

Enabling more frequent gradient updates

- In REINFORCE, we have to wait until the end of an episode to make a gradient update
- That's because our gradient is $-\sum_{t=1}^{T} \nabla \log p(a_t|s_t) D(s_t, a_t)$
  - To calculate $D(s_t, a_t)$, we need to know everything that happens up to the end of the episode
- You could cut the episode off early, but then $D(s_t, a_t)$ would be biased
- ***What would happen if we did this while training on Frozen Lake?***
  - The only nonzero reward comes on the very last step of the episode
  - If we cut the episode off early, we'd never see this reward, and the agent would never learn anything!

Enabling more frequent gradient updates

- RwB gives us a way to cut the episode off early (or pause it) and take a gradient update *without introducing bias*
- Recall the definition of $D(s_t, a_t)$:

$$D(s_t, a_t) = \sum_{i=t}^{T} \gamma^{i-1} r(s_i, a_i, s_{i+1})$$

- Write it as a recurrence relation:

$$D(s_T, a_T) = 0$$

$$D(s_t, a_t) = r(s_t, a_t, s_{t+1}) + \gamma D(s_{t+1}, a_{t+1})$$

Enabling more frequent gradient updates

- At any point, we can choose to stop expanding this recurrent relation and instead use our value network to estimate the remainder of D:

$$D(s_t, a_t) = r(s_t, a_t, s_{t+1}) + \gamma D(s_{t+1}, a_{t+1})$$

$$\Downarrow$$

$$D(s_t, a_t) = r(s_t, a_t, s_{t+1}) + \gamma V(s_{t+1})$$

- **Fun fact:** this strategy is how AlphaGo trained itself to play Go without having to explore the (massive) search tree of a Go game: it could terminate the search early and use a trained value network to estimate the value of being in a particular board state

Reducing the number of episodes needed

- Simulating episodes can be very compute-intensive
- More intensive, in fact, than training the networks!
  - AlphaGo used 64 GPUs and 19 CPUs for its model updates...
  - ...but it used ~5,500 TPUs for its Go simulations to create training episodes
- Idea: get more out of the training episodes we've already simulated by periodically re-using them
  - Not a crazy idea: we iterate multiple epochs over the same training set in supervised learning, after all
- Known as ***Experience Replay***

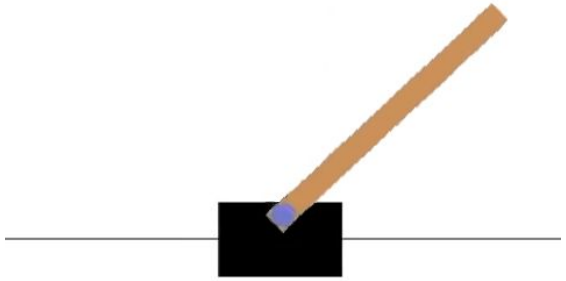
Experience Replay can also stabilize training!

- Recall from way back at the beginning of the class: SGD assumes that the training data is **IID (independent, identically distributed)**
- Time steps taken from a simulated MDP episode are definitely ***not*** independent
  - $(s_0, a_0), (s_1, a_1), \ldots, (s_n, a_n)$ → successive time steps are highly correlated
- By training on this data, agent could overfit to patterns in one episode, then have to un-learn when presented with a different episode
- Experience replay mixes up timesteps from past/present episodes, making the data "more IID"
  - Think of it like the agent 'teleporting' around to different timesteps of different episodes during training

Any questions?

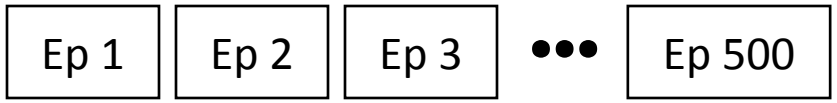
Experience Replay: Caveat

- As the agent gets better over time, episodes from earlier in training become less valuable (even useless)
- ***Why?***
  - Those mostly explore bad parts of the state space from when the agent was flailing around randomly
  - Now it knows not to go to those states anymore, so why bother learning what to do in them?



Experience Replay: Caveat

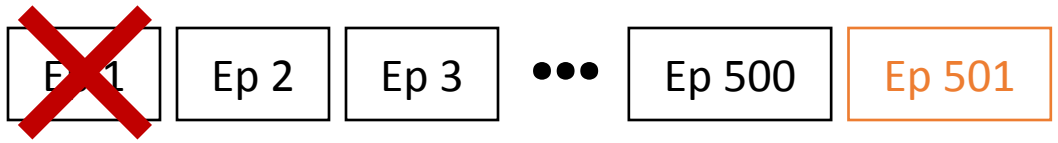
- Solution: only keep a limited-size buffer of episodes



- Train on randomly-sampled timesteps from these episodes, for some number of training steps

Experience Replay: Caveat

- Solution: only keep a limited-size buffer of episodes

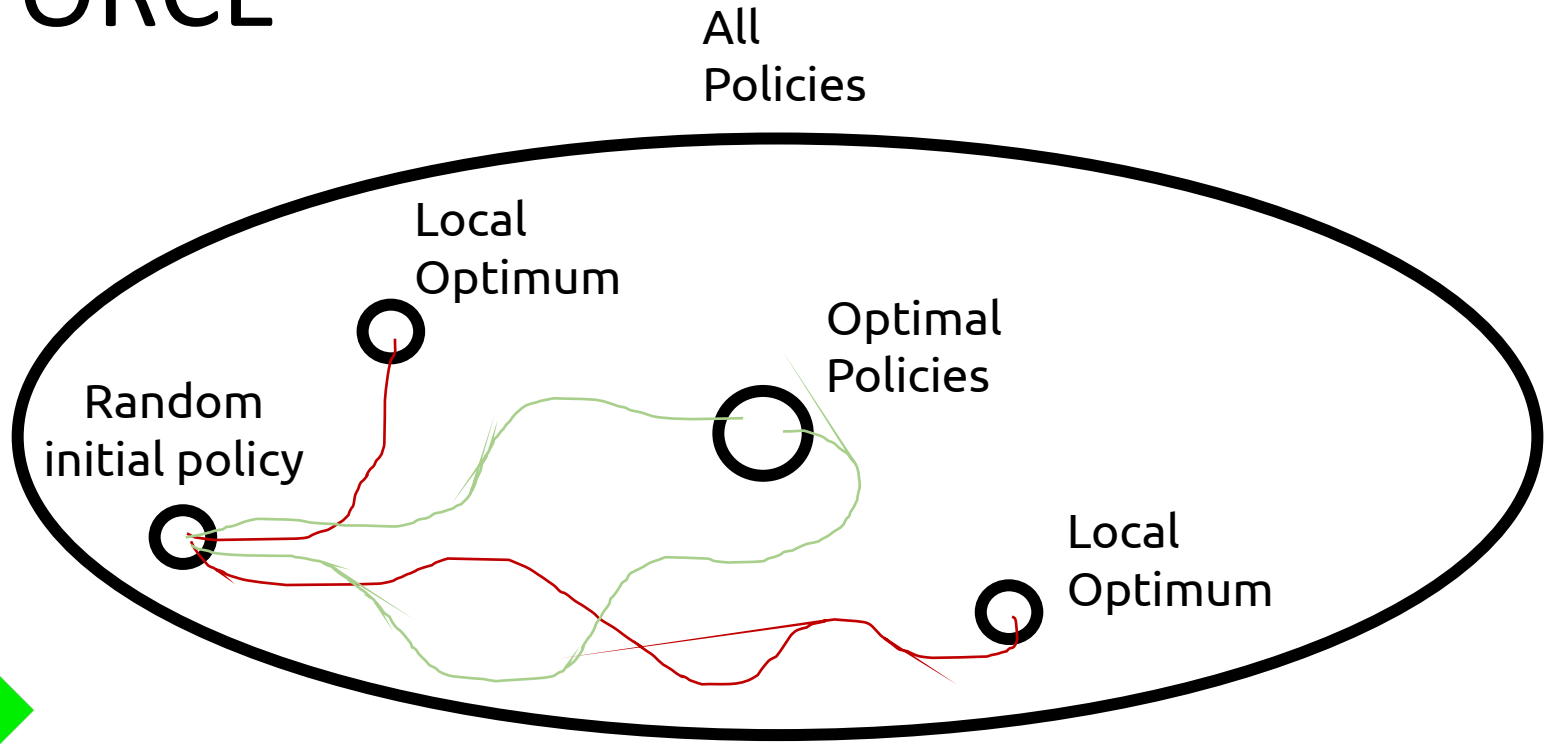


- Train on randomly-sampled timesteps from these episodes, for some number of training steps
- Then, sample a new episode, add to the buffer, and remove the oldest episode in the buffer
  - i.e. the buffer is a **queue**

Issues with REINFORCE

- **High Variance**
  - Multiple runs training an agent with REINFORCE can yield very different results
  - Susceptible to local optima

First, go… this problem…

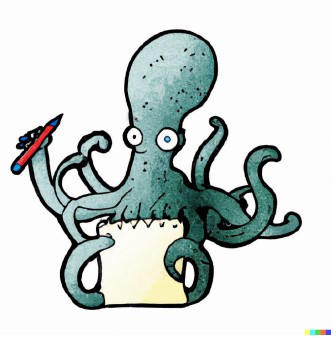


- **High "sample complexity"**
  - Many episode to get gradient, takes many episodes to learn

…which will actually… solution for this problem via a simple extension

Organizing RL problems/algorithms

	Know T and R	Don't know T and R
Simple/discrete	Value iteration	Q-Learning
Complex/continuous		Deep Q-Networks REINFORCE Actor-Critic



For a more complete taxonomy of RL algorithms, see https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html#citations-below