Sequential Decision Making

WARNING 1

Do not use this stuff because it's cool, or it seems fancy or because you think it works better.

It doesn't work better.

It's worse. It's provably worse.

<u>You use these methods when you don't have any other options.</u>

WARNING 2

We only have time to cover the high level intuition of these methods in one lecture, so a lot of this will be incomplete.

The goal is to give you a flavor for the kinds of things you need to pay attention to when using RL.

Sequential Decision Making

Environment

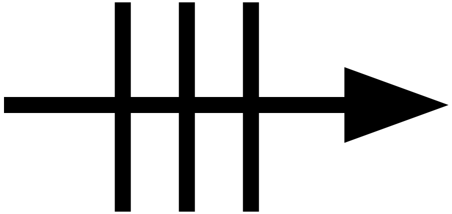
Sequential Decision Making

Observation (x)

Environment

Sequential Decision Making

Observation (x)

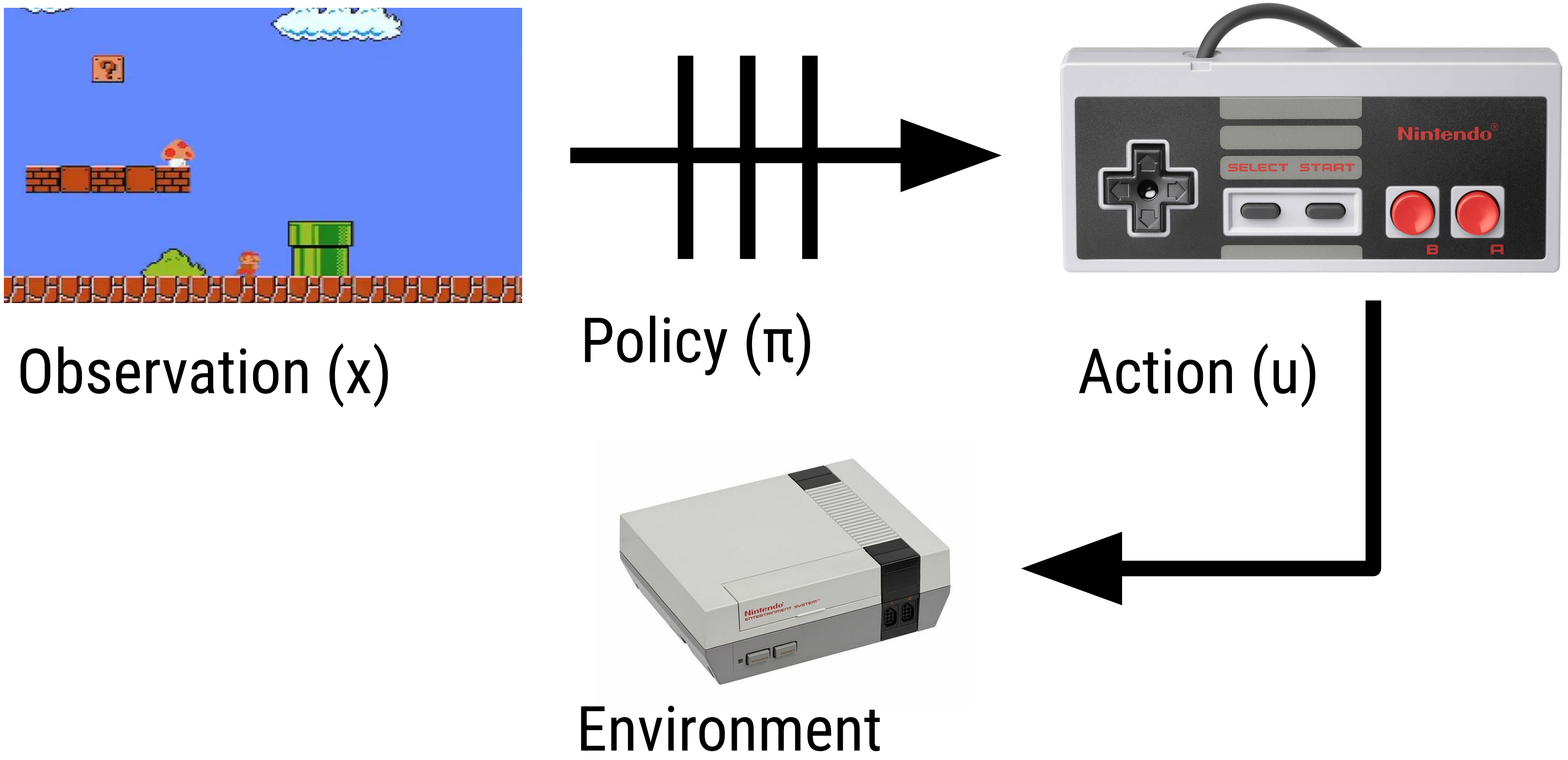


Policy (π)

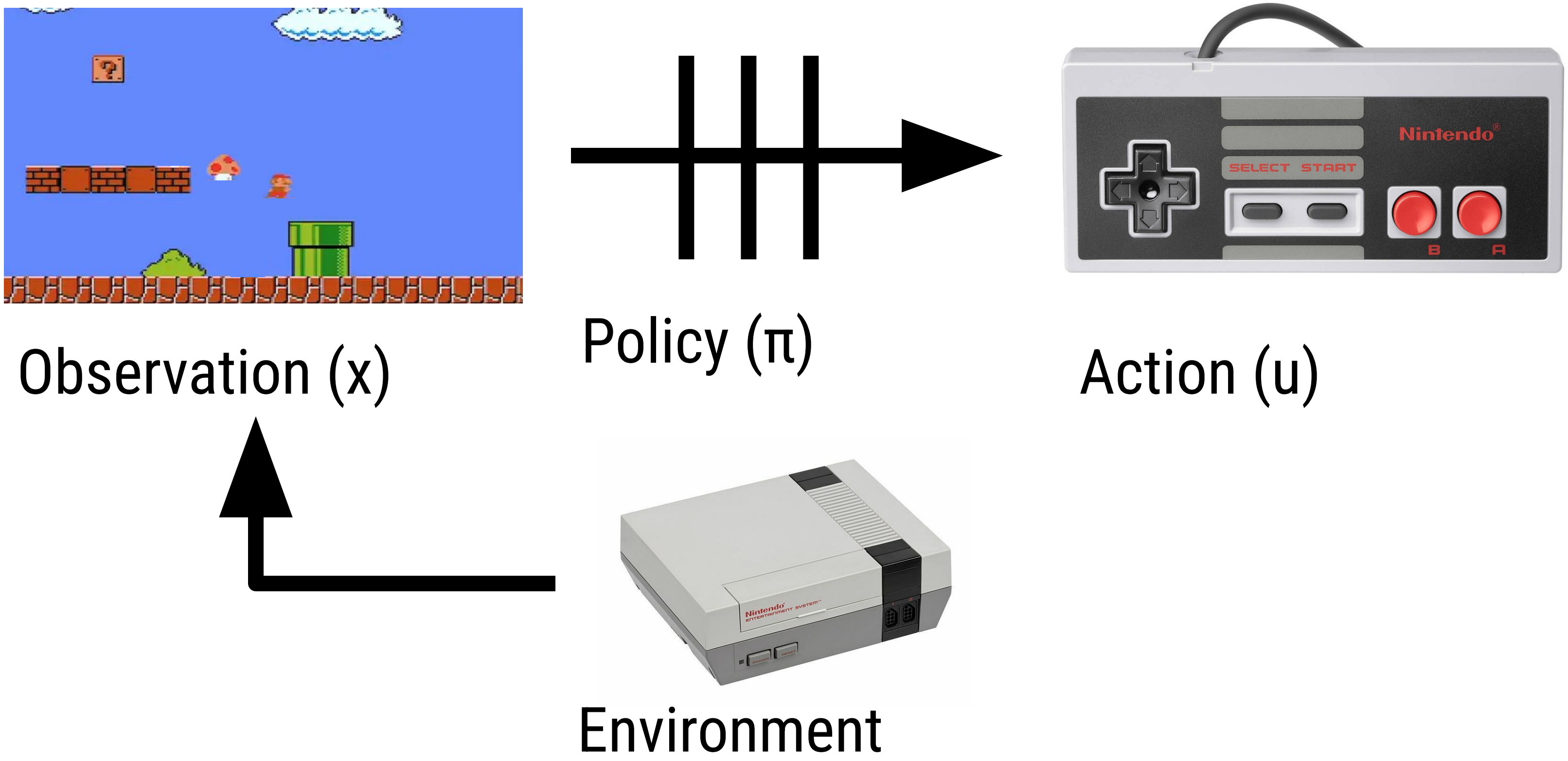
Action (u)

Environment

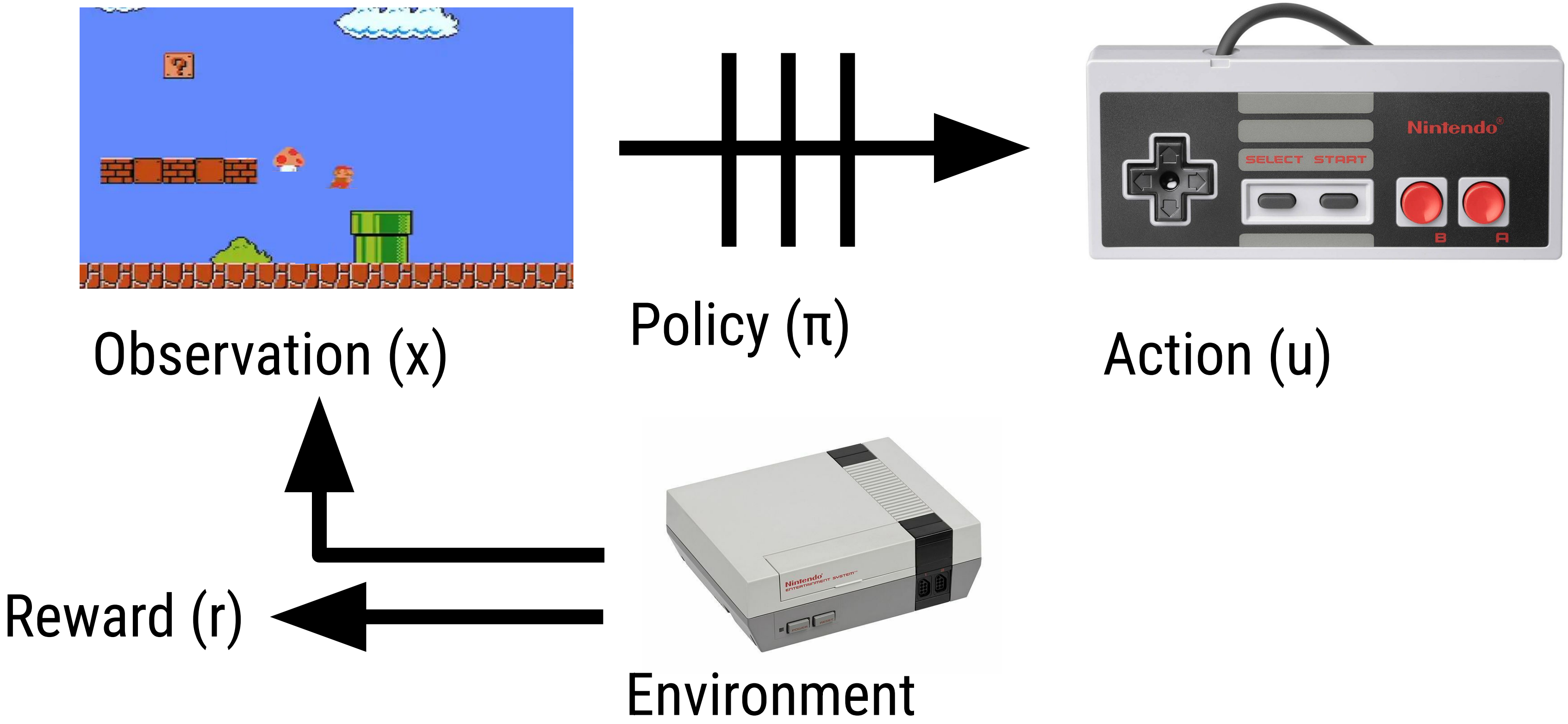
Sequential Decision Making



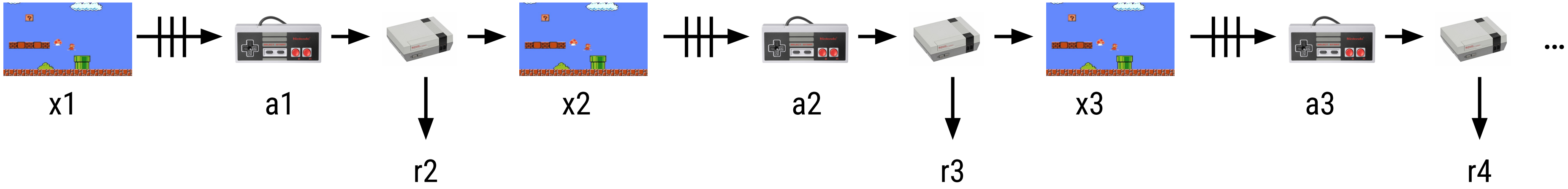
Sequential Decision Making



Sequential Decision Making



Sequential Decision Making



Two High Level Categories

Imitation Learning (covered briefly last time)

- Learn from advice/instructions
- Learn from a knowledgeable teacher

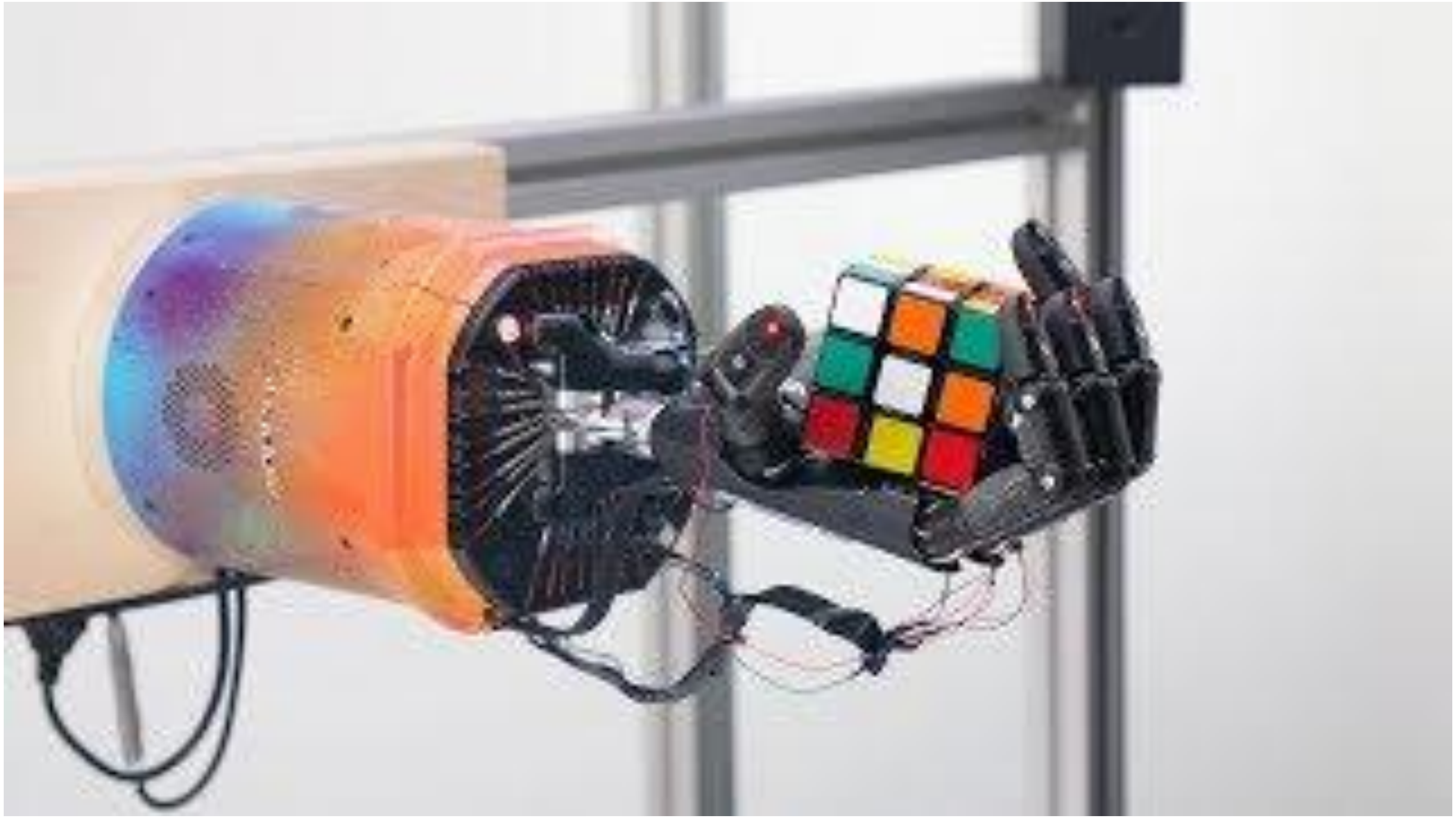
Reinforcement Learning (today)

- Learn from a reward signal that tells us how well we did
- Learn by trial and error

Reinforcement Learning

Mnih et al. 2013 “Playing Atari with Deep Reinforcement Learning”

Silver et al. 2016 “Mastering the game of Go with deep neural networks and tree search”



OpenAI et al. 2019 “Solving Rubik's Cube with a Robot Hand”

Vinyals et al. 2019 “Grandmaster level in StarCraft II using multi-agent reinforcement learning”
Berner et al. 2019 “Dota 2 with large scale deep reinforcement learning”

Haarnoja et al. 2023 “Learning Agile Soccer Skills for a Bipedal Robot with Deep Reinforcement Learning”

What is Common About These Domains?

What is Common About These Domains?

1. Difficult to provide direct instruction

What is Common About These Domains?

1. Difficult to provide direct instruction
2. Search methods may not be tractable

What is Common About These Domains?

1. Difficult to provide direct instruction
2. Search methods may not be tractable
3. May not even know how good optimal performance can be (AlphaGo)

Policy Gradient

(REINFORCE aka Vanilla Policy Gradient)
(Ancestor of A2C, A3C, TRPO, PPO)

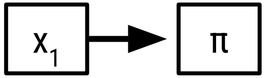
Policy Gradient

- States/Observations : x

x_1

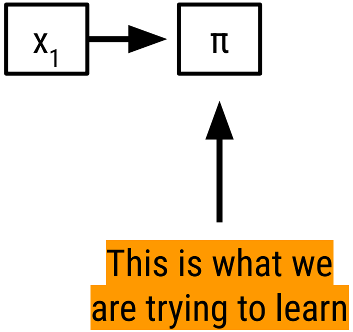
Policy Gradient

- States/Observations : x
- Policy : π



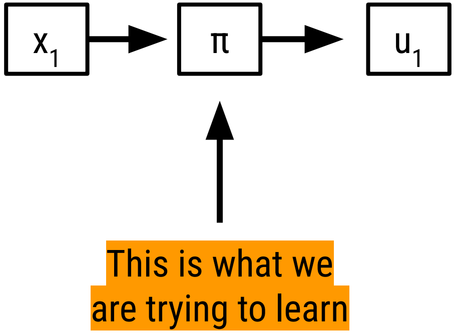
Policy Gradient

- States/Observations : x
- Policy : π



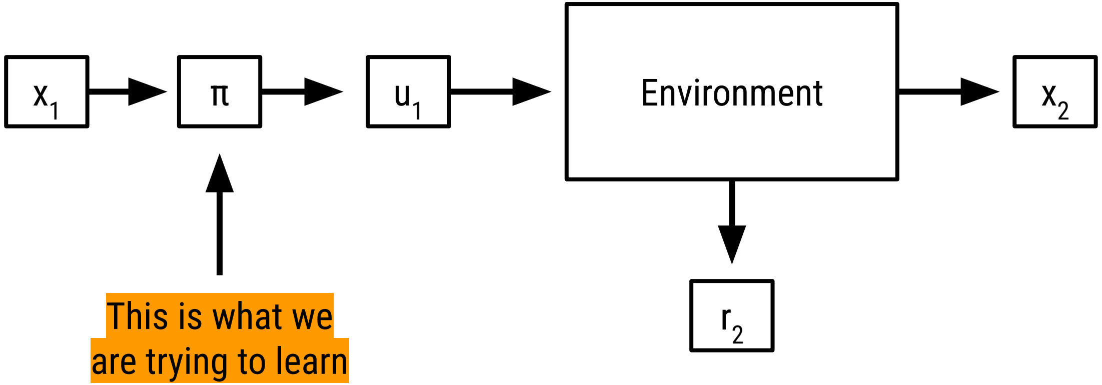
Policy Gradient

- States/Observations : x
- Policy : π
- Actions Space : u



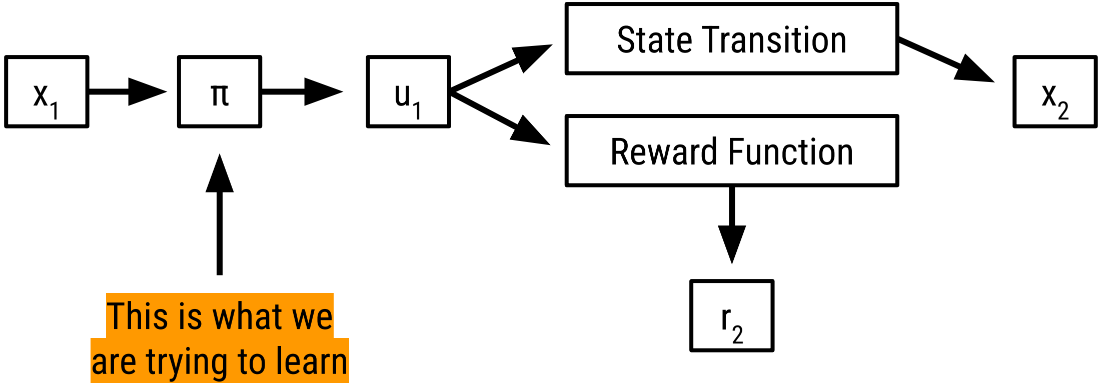
Policy Gradient

- States/Observations : x
- Policy : π
- Actions Space : u
- Transition Probabilities : p(x'|u,x)
- Reward Function : r(x,u)



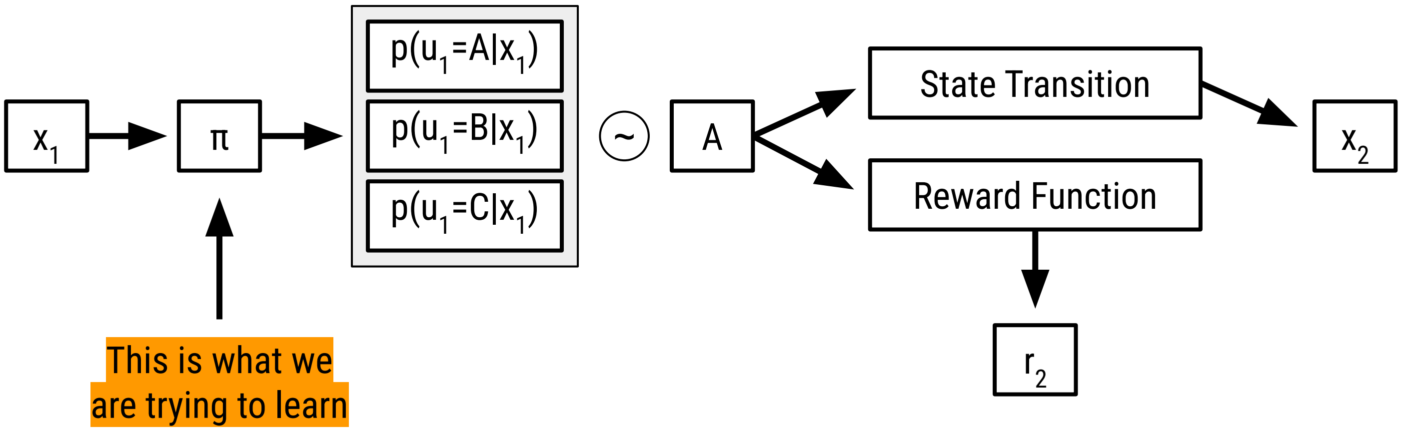
Policy Gradient

- States/Observations : x
- Policy : π
- Actions Space : u
- Transition Probabilities : p(x'|u,x)
- Reward Function : r(x,u)



Policy Gradient

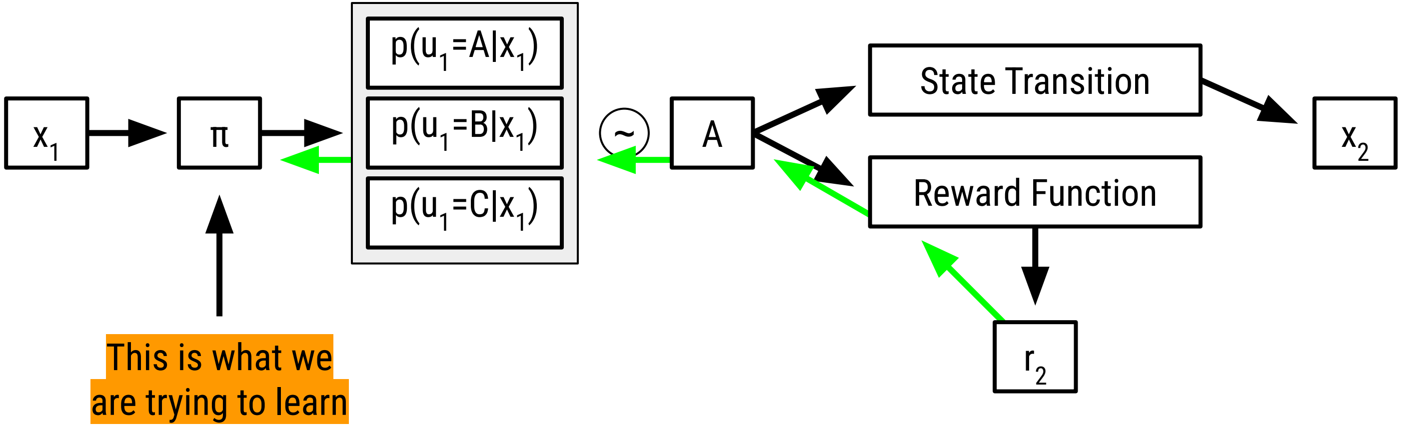
- States/Observations : x
- Policy : π
- Actions Space : u
- Transition Probabilities : $p(x'|u,x)$
- Reward Function : $r(x,u)$



Policy Gradient

- States/Observations : x
- Policy : π
- Actions Space : u
- Transition Probabilities : p(x'|u,x)
- Reward Function : r(x,u)

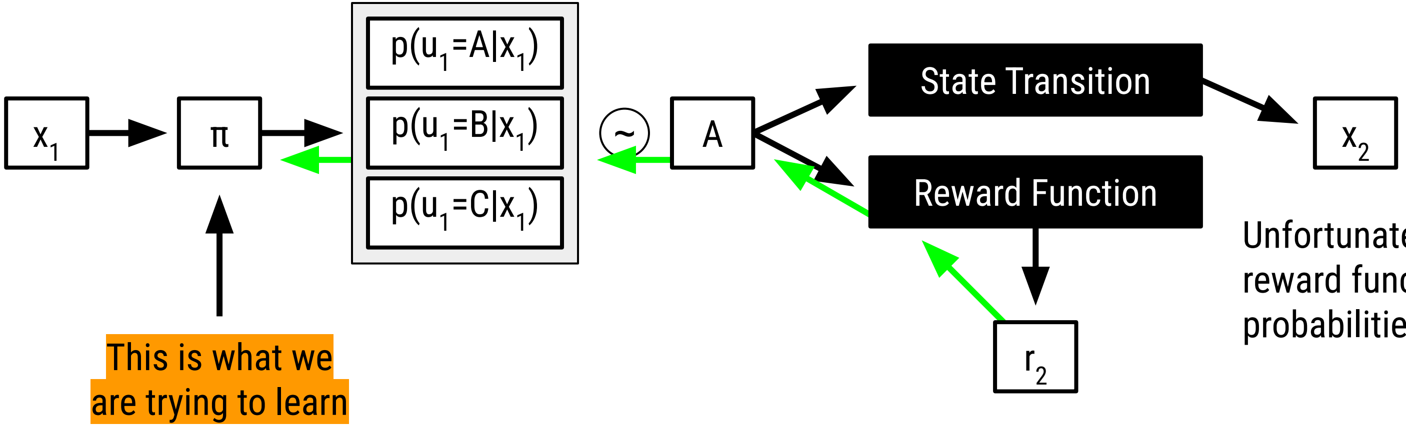
Idea 1: If r_2 is kind of the opposite of a loss function (something we want to maximize rather than minimize), can we just backprop through this chain of operations?



Policy Gradient

- States/Observations : x
- Policy : π
- Actions Space : u
- Transition Probabilities : $p(x'|u,x)$
- Reward Function : $r(x,u)$

Idea 1: If r_2 is kind of the opposite of a loss function (something we want to maximize rather than minimize), can we just backprop through this chain of operations?

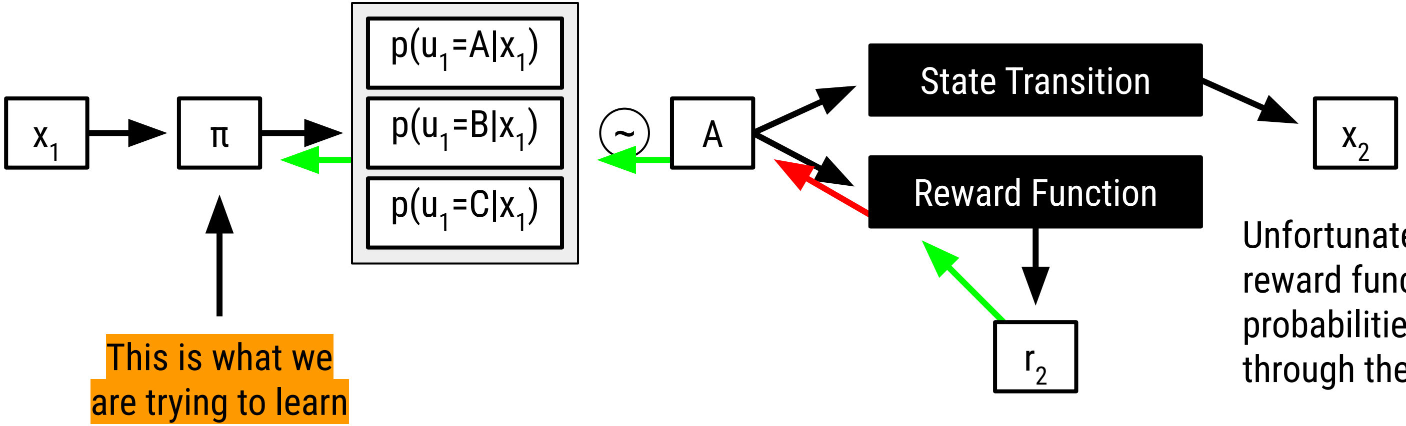


Unfortunately, we don't know the reward function or state-transition probabilities…

Policy Gradient

- States/Observations : x
- Policy : π
- Actions Space : u
- Transition Probabilities : $p(x'|u,x)$
- Reward Function : $r(x,u)$

Idea 1: If r_2 is kind of the opposite of a loss function (something we want to maximize rather than minimize), can we just backprop through this chain of operations?

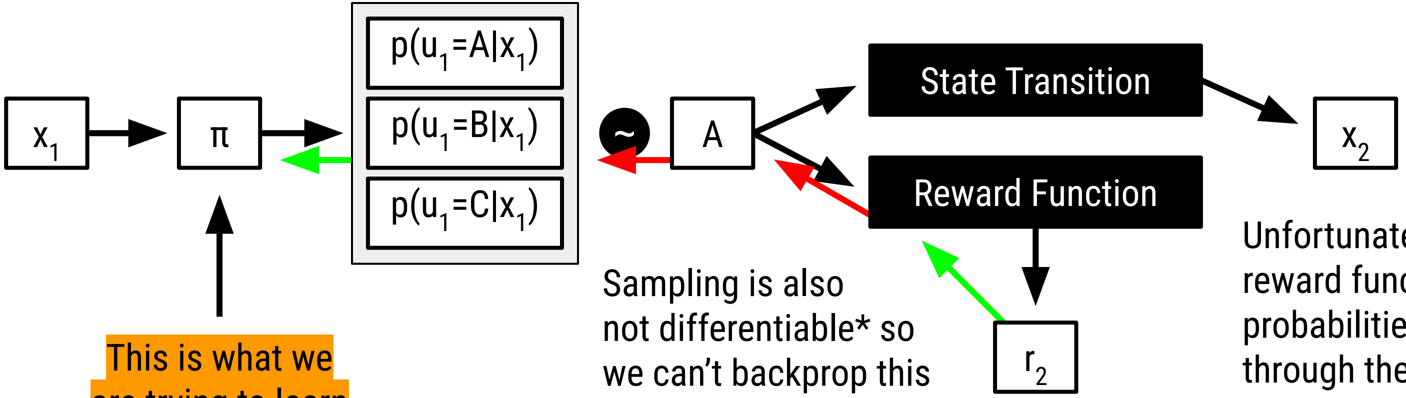


Unfortunately, we don't know the reward function or state-transition probabilities, so we can't backprop through the reward function.

Policy Gradient

- States/Observations : x
- Policy : π
- Actions Space : u
- Transition Probabilities : p(x’|u,x)
- Reward Function : r(x,u)

Idea 1: If r_2 is kind of the opposite of a loss function (something we want to maximize rather than minimize), can we just backprop through this chain of operations?



This is what we are trying to learn

Sampling is also not differentiable* so we can’t backprop this step either.

Unfortunately, we don’t know the reward function or state-transition probabilities, so we can’t backprop through the reward function.

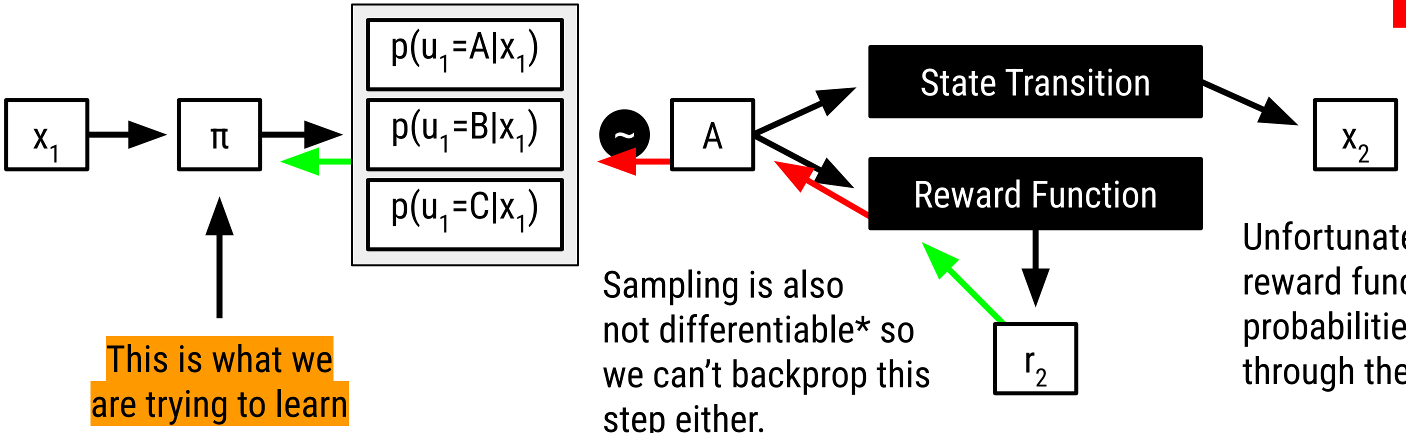
*there are hacks around this though

Policy Gradient

- States/Observations : x
- Policy : π
- Actions Space : u
- Transition Probabilities : p(x'|u,x)
- Reward Function : r(x,u)

Idea 1: If r_2 is kind of the opposite of a loss function (something we want to maximize rather than minimize), can we just backprop through this chain of operations?

NO



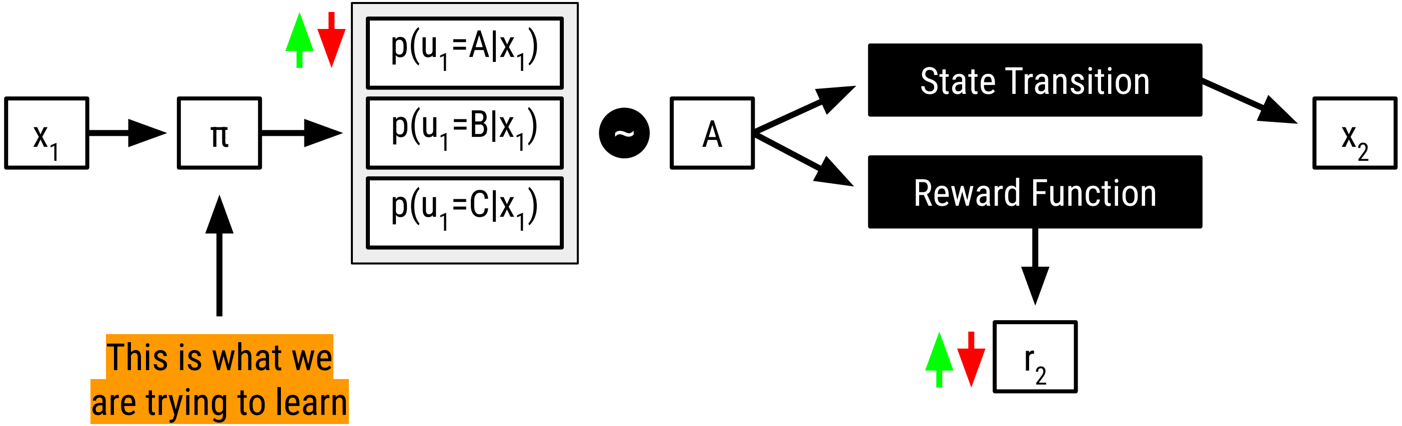
Unfortunately, we don't know the reward function or state-transition probabilities, so we can't backprop through the reward function.

*we may see a way around this in a later lecture

Policy Gradient

- States/Observations : x
- Policy : π
- Actions Space : u
- Transition Probabilities : p(x'|u,x)
- Reward Function : r(x,u)

Idea 2: Can we just increase or decrease the probability of the sampled action based on how good r_2 is?

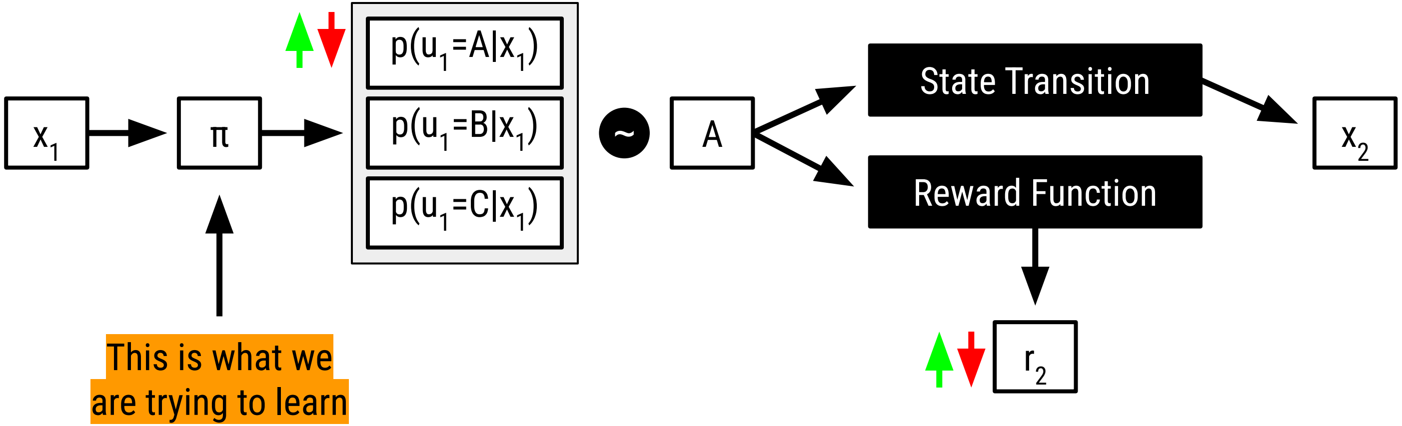


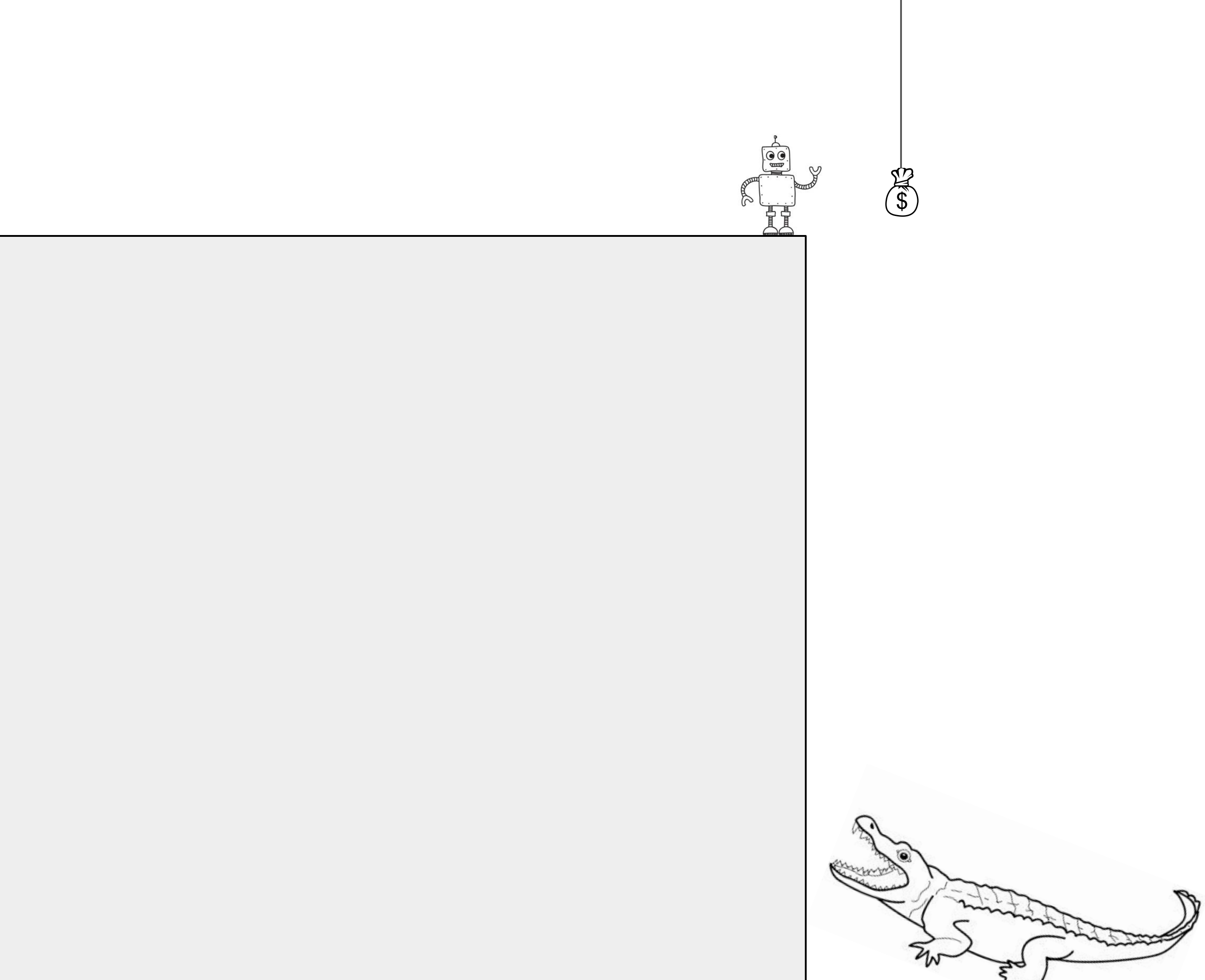
Policy Gradient

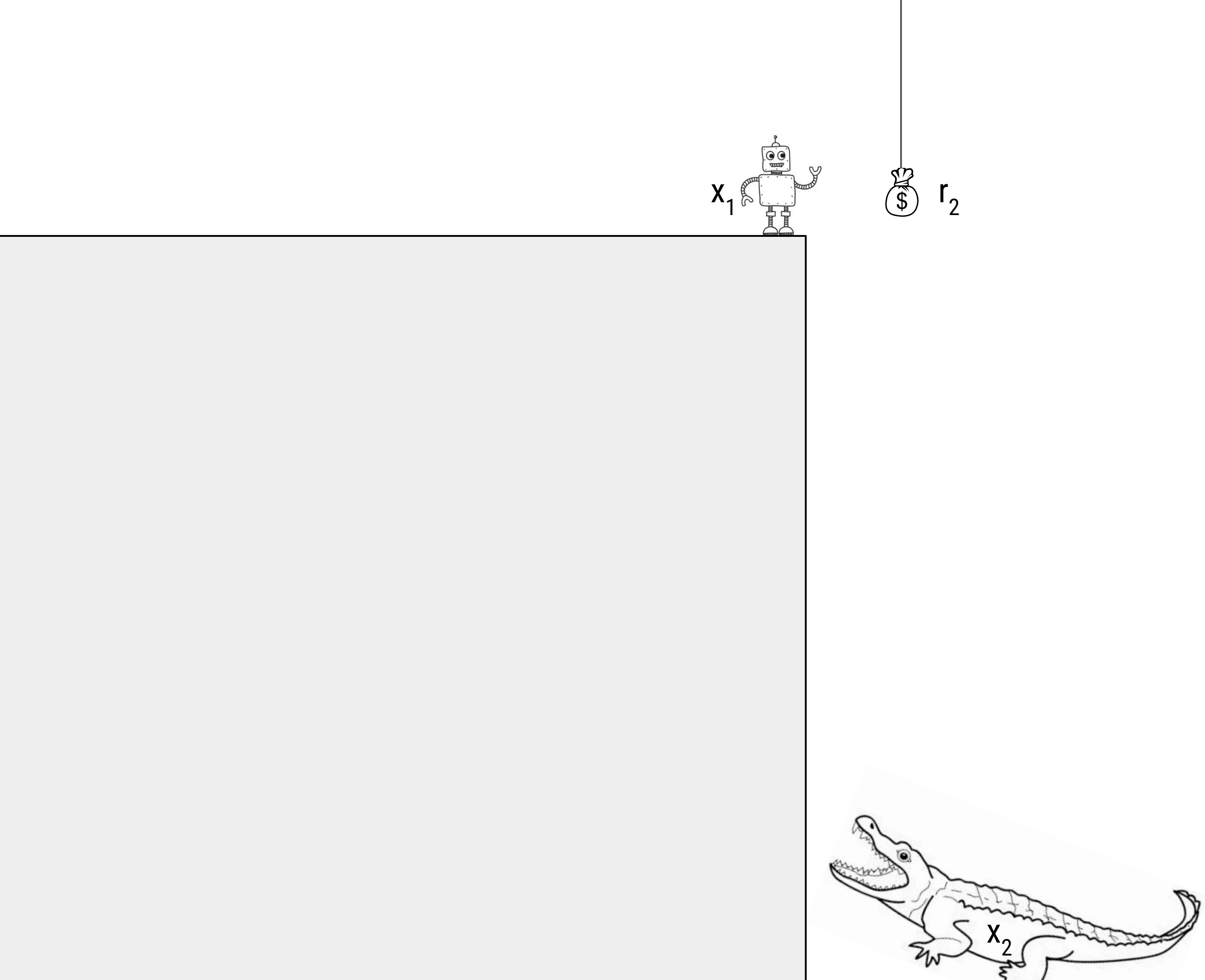
- States/Observations : x
- Policy : π
- Actions Space : u
- Transition Probabilities : $p(x'|u,x)$
- Reward Function : $r(x,u)$

Idea 2: Can we just increase or decrease the probability of the sampled action based on how good r_2 is?

Remember that this is one action in a large sequence though! What if x_2 is terrible?





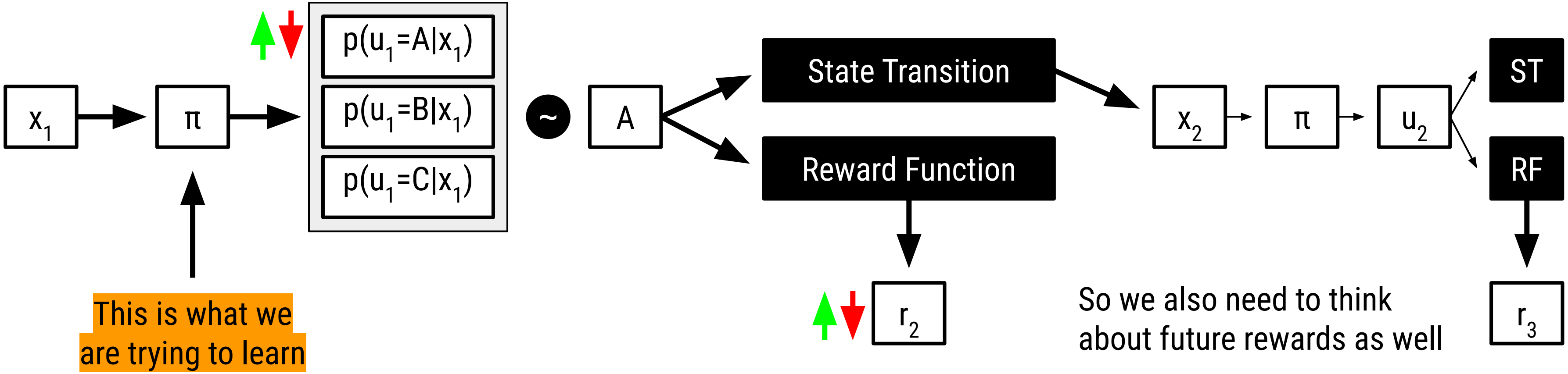

x1
r2
x2

Policy Gradient

- States/Observations : x
- Policy : π
- Actions Space : u
- Transition Probabilities : p(x'|u,x)
- Reward Function : r(x,u)

Idea 2: Can we just increase or decrease the probability of the sampled action based on how good r_2 is?

Remember that this is one action in a large sequence though! What if x_2 is terrible?

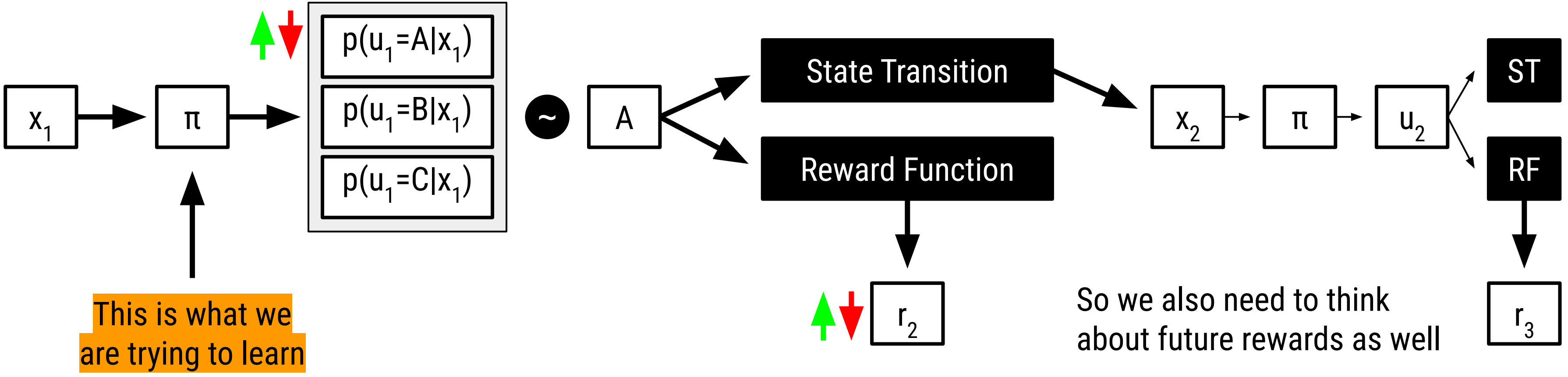


Policy Gradient

- States/Observations : x
- Policy : π
- Actions Space : u
- Transition Probabilities : p(x'|u,x)
- Reward Function : r(x,u)

Idea 2: Can we just increase or decrease the probability of the sampled action based on how good r_2 is? **NO… but**

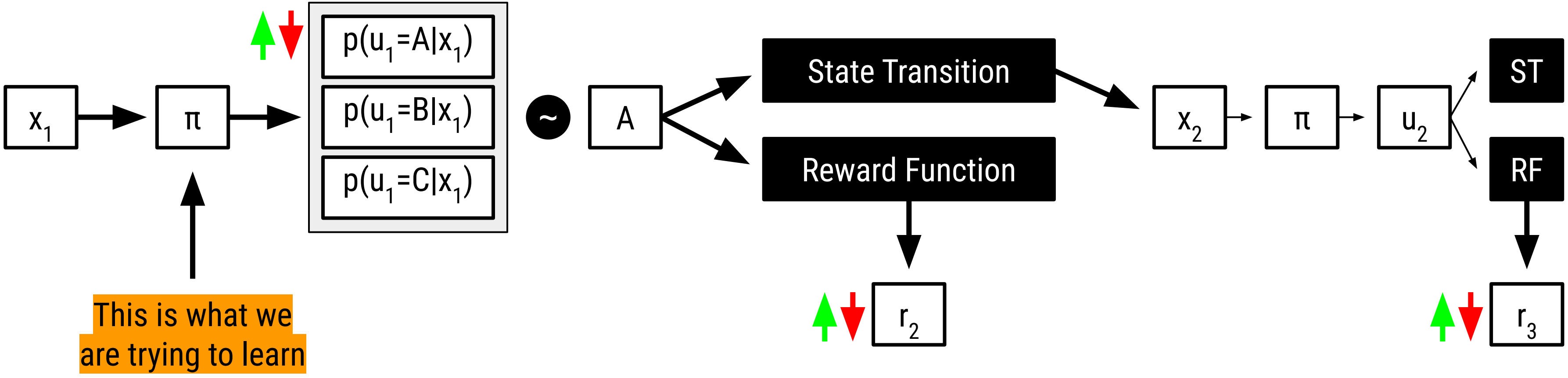
Remember that this is one action in a large sequence though! What if x_2 is terrible?



Policy Gradient

- States/Observations : x
- Policy : π
- Actions Space : u
- Transition Probabilities : p(x'|u,x)
- Reward Function : r(x,u)

Idea 2.5: Can we just increase or decrease the probability of the sampled action based on how good the sum of future rewards is?

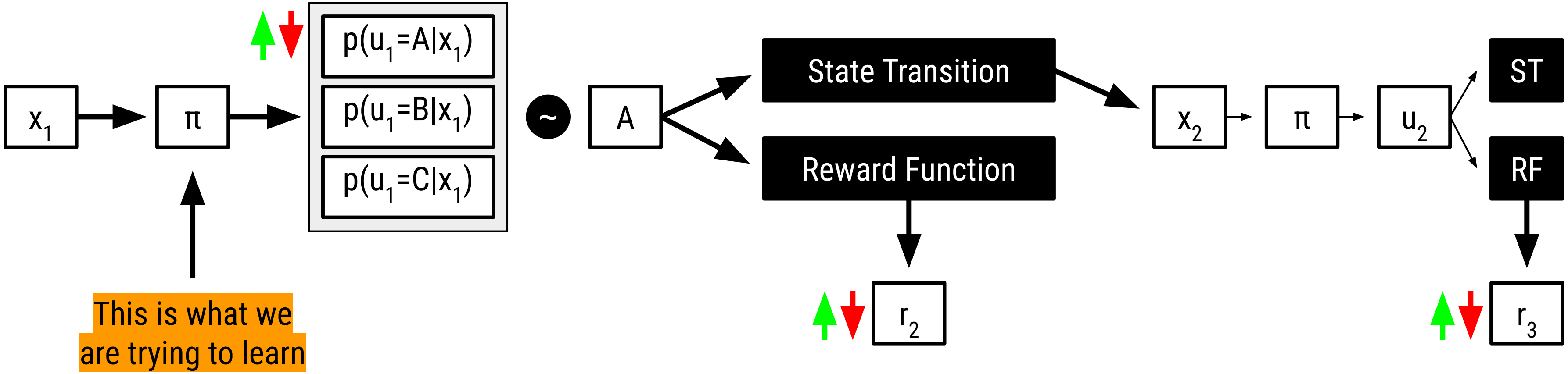


Policy Gradient

- States/Observations : x
- Policy : π
- Actions Space : u
- Transition Probabilities : p(x'|u,x)
- Reward Function : r(x,u)

Idea 2.5: Can we just increase or decrease the probability of the sampled action based on how good the sum of future rewards is?

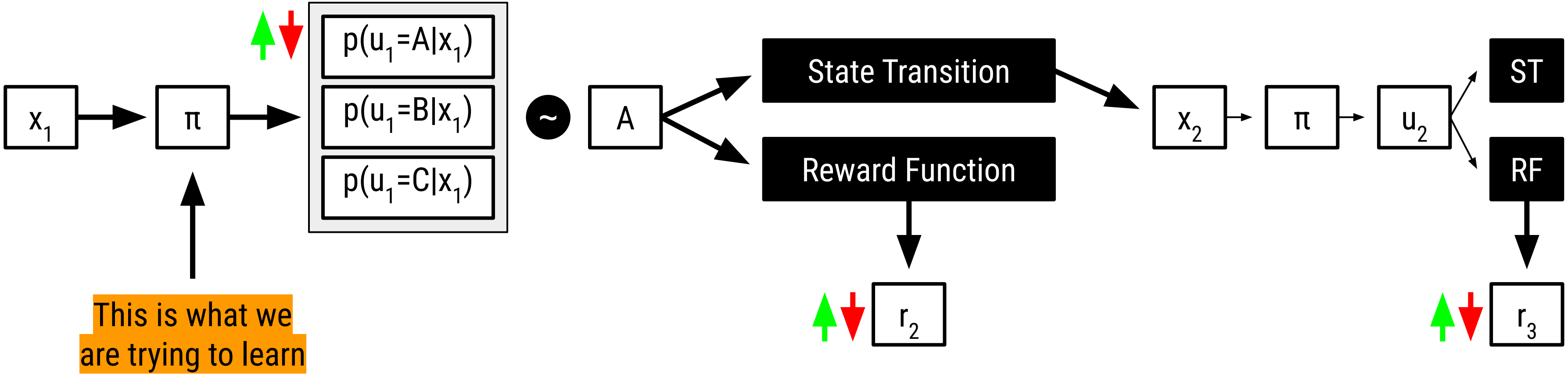
YES! This is the rough intuition behind policy gradients.



Policy Gradient

- States/Observations : x
- Policy : π
- Actions Space : u
- Transition Probabilities : p(x'|u,x)
- Reward Function : r(x,u)

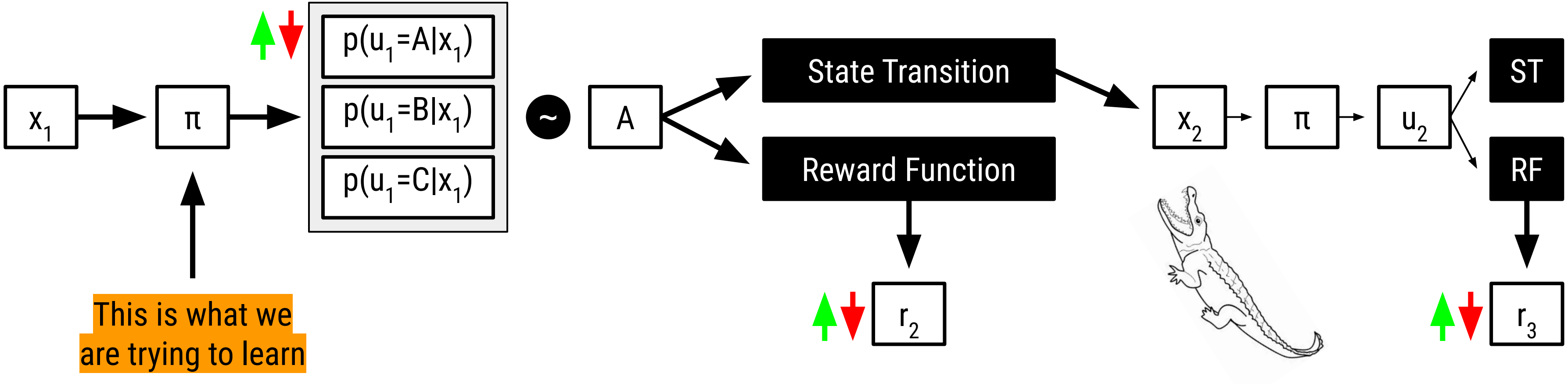
One more thing to say about this though.



Policy Gradient

- States/Observations : x
- Policy : π
- Actions Space : u
- Transition Probabilities : p(x’|u,x)
- Reward Function : r(x,u)

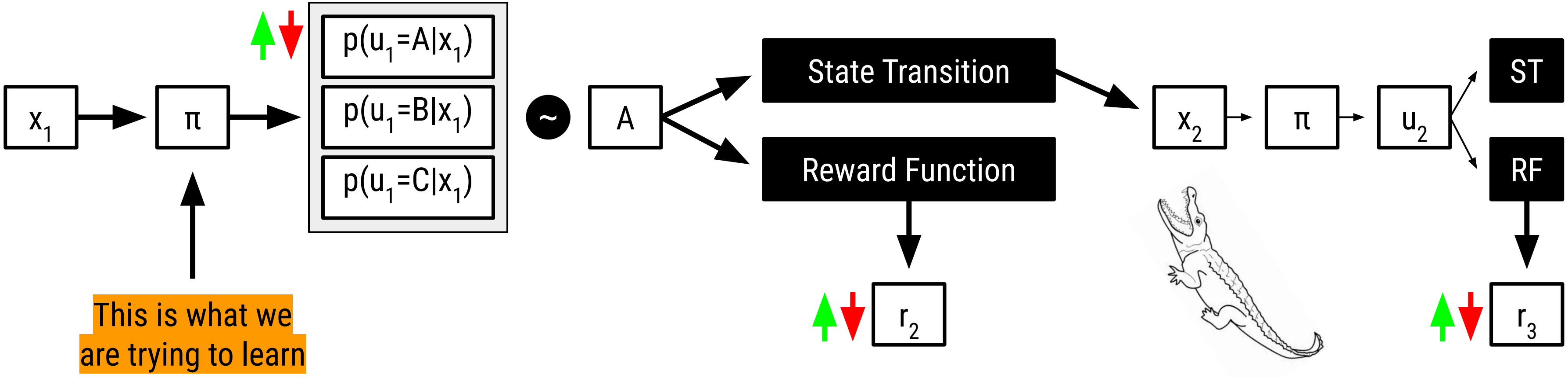
One more thing to say about this though. We just saw what can happen if x_2 is bad…



Policy Gradient

- States/Observations : x
- Policy : π
- Actions Space : u
- Transition Probabilities : p(x'|u,x)
- Reward Function : r(x,u)

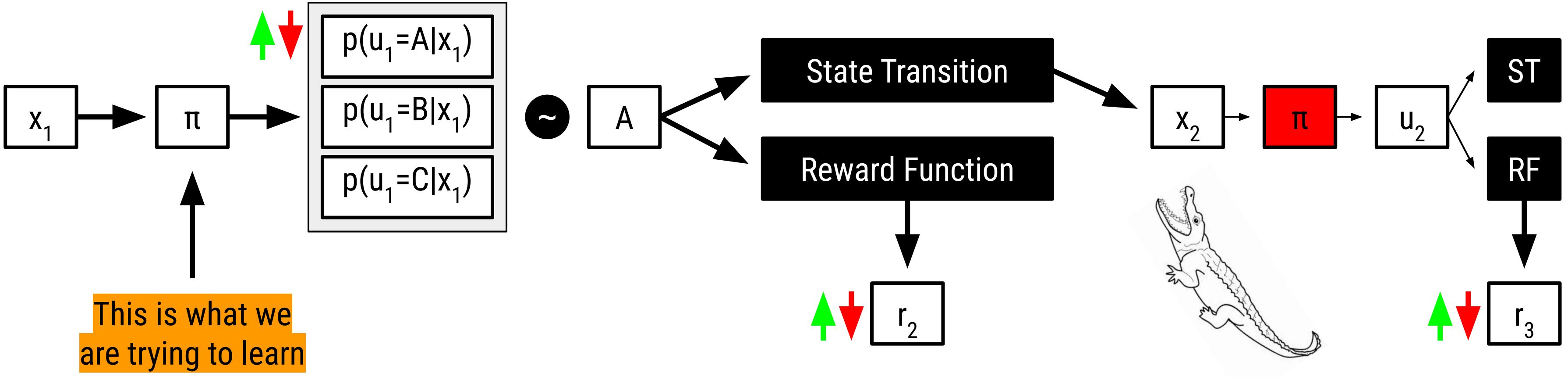
One more thing to say about this though. We just saw what can happen if x_2 is bad, but there's another thing that can cause r_3 to be bad…



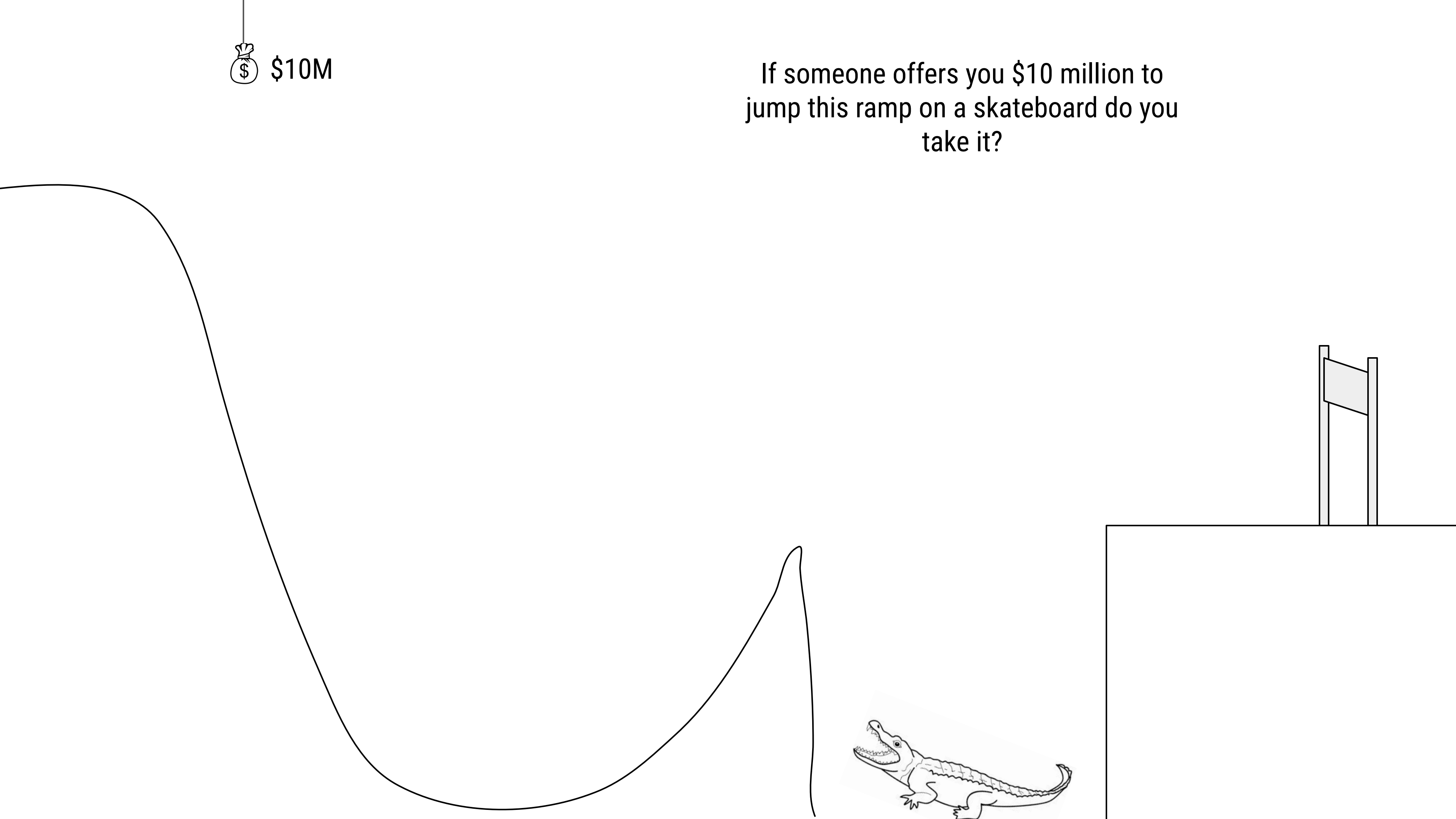
Policy Gradient

- States/Observations : x
- Policy : π
- Actions Space : u
- Transition Probabilities : p(x’|u,x)
- Reward Function : r(x,u)

One more thing to say about this though. We just saw what can happen if x_2 is bad, but there’s another thing that can cause r_3 to be bad and it’s us!



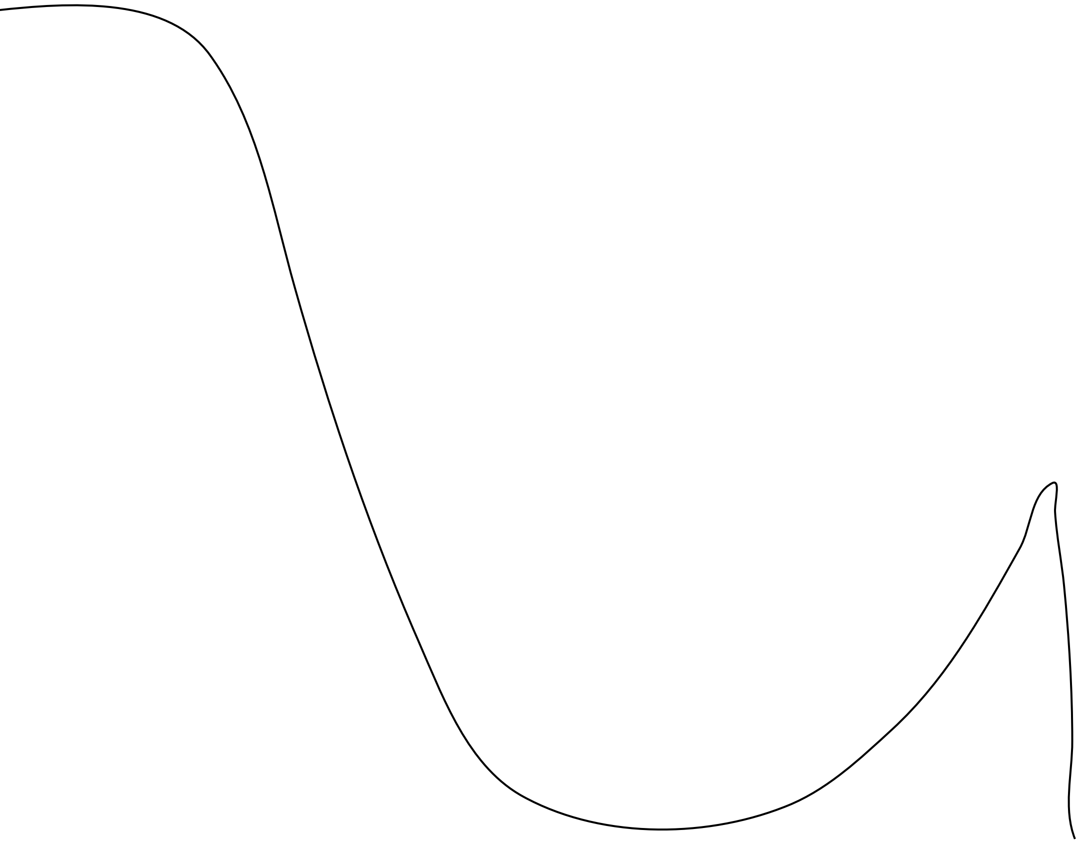
If someone offers you $10 million to jump this ramp on a skateboard do you take it?

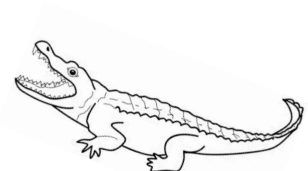


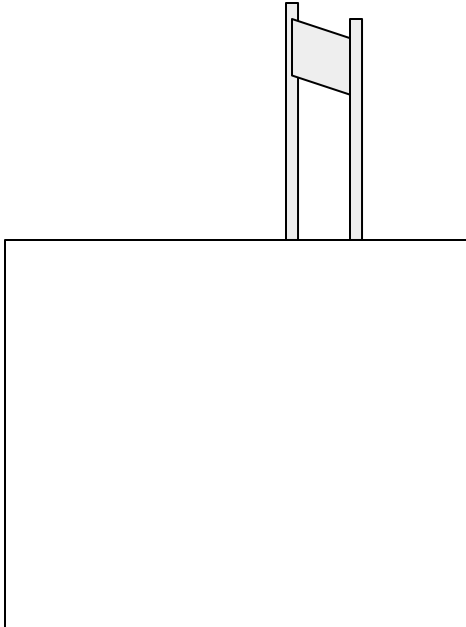
If someone offers you $10 million to jump this ramp on a skateboard do you take it?

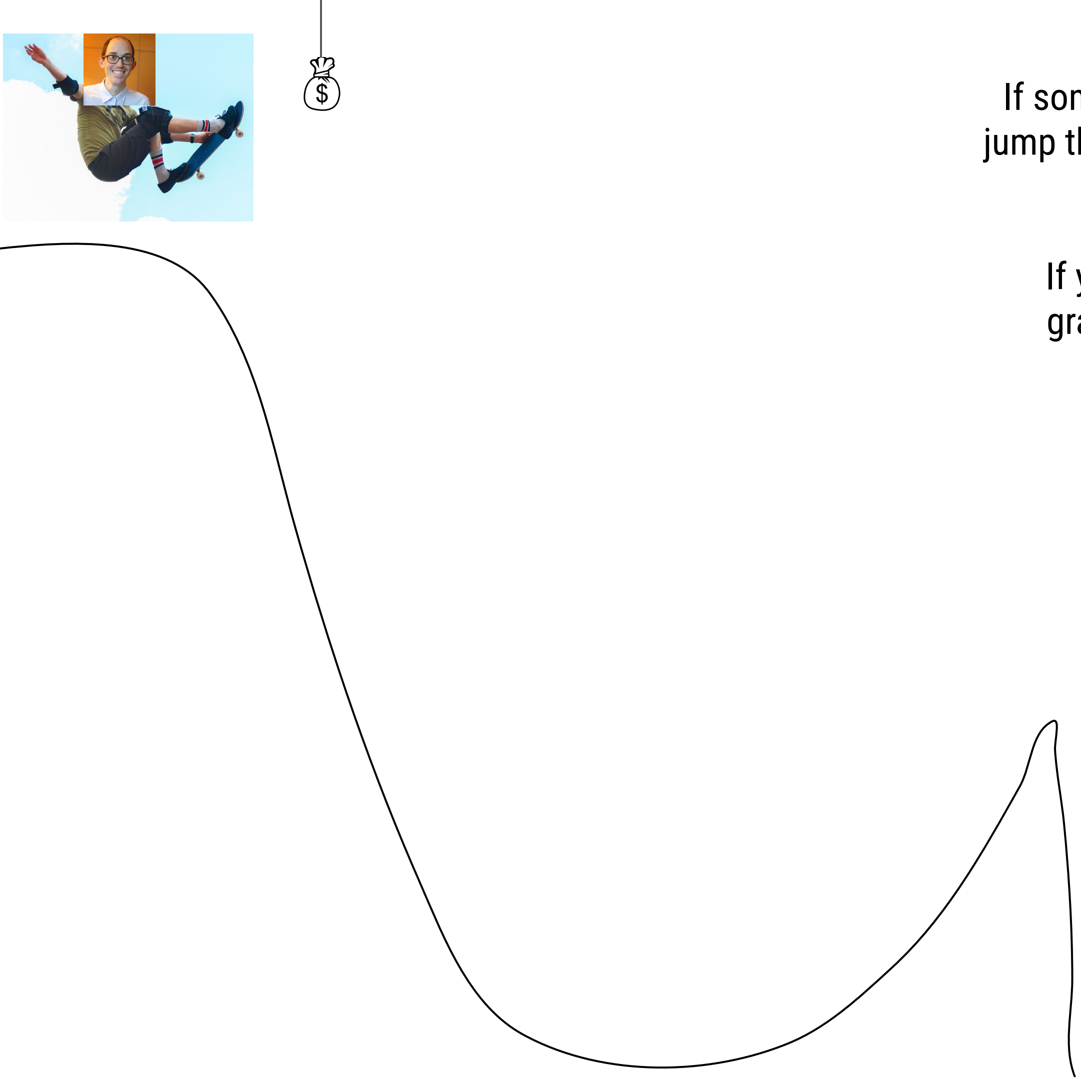
If you are Tony Hawk:

YES!





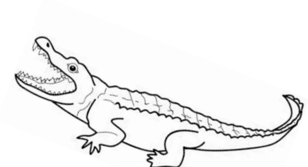


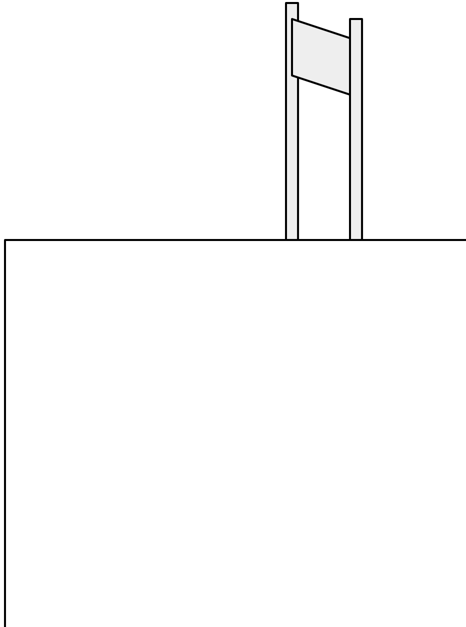


If someone offers you $10 million to jump this ramp on a skateboard do you take it?

If you are Aaron Walsman with grad student health insurance:

NO!

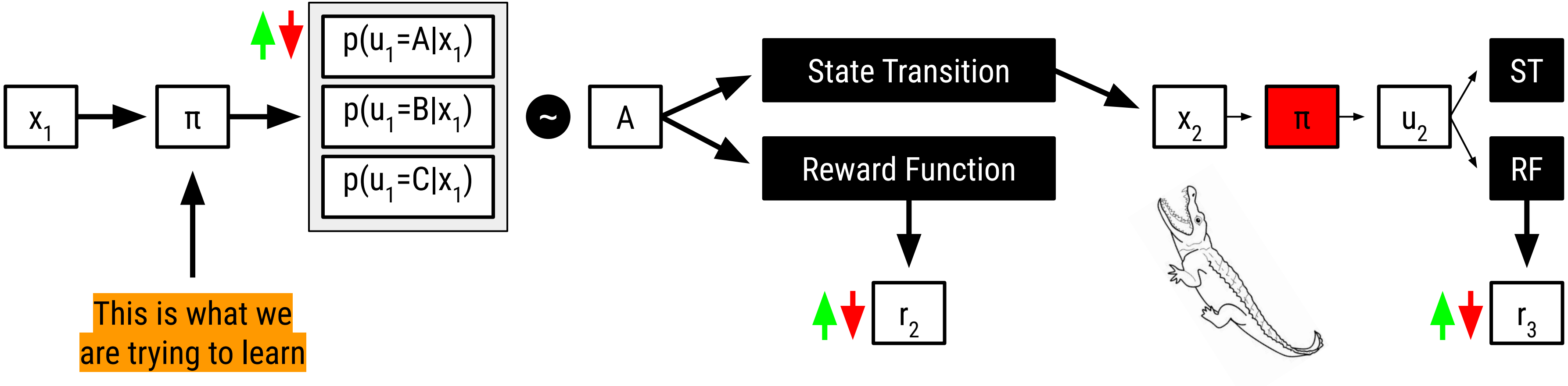




Policy Gradient

- States/Observations : x
- Policy : π
- Actions Space : u
- Transition Probabilities : $p(x'|u,x)$
- Reward Function : $r(x,u)$

The point is that r_3 depends not only on the physical environment, but also the capability of our current model π! This will come back to haunt us later!



Policy Gradient

How do we adjust the probability based on the sum of rewards?

Policy Gradient

How do we adjust the probability based on the sum of rewards?

We will look at two ways:

Policy Gradient

How do we adjust the probability based on the sum of rewards?

We will look at two ways:

1. Derivation!

Policy Gradient

How do we adjust the probability based on the sum of rewards?

We will look at two ways:

1. Derivation!
2. Simple-ish Intuition!

Policy Gradient

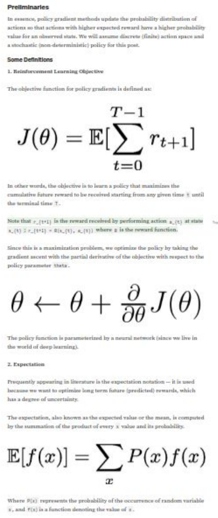
How do we adjust the probability based on the sum of rewards?

We will look at two ways:

1. Derivation!
2. Simple-ish Intuition!

Policy Gradient

So the derivation… I was actually going to walk through all of this, but then realized we wouldn't have time.



Preliminaries

Some Definitions

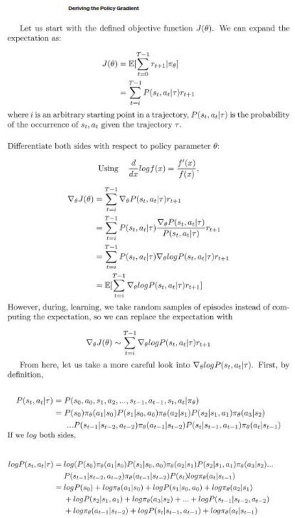
1. Reinforcement Learning Objective

$$J(\theta) = \mathbb{E}[\sum_{t=0}^{T-1} r_{t+1}]$$

$$\theta \leftarrow \theta + \frac{\partial}{\partial \theta} J(\theta)$$

2. Expectation

$$\mathbb{E}[f(x)] = \sum_x P(x) f(x)$$



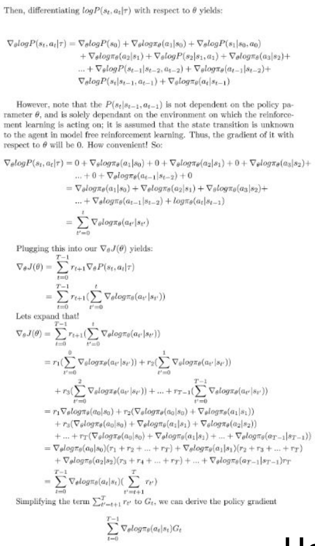
Deriving the Policy Gradient

Let us start with the defined objective function $J(\theta)$. We can expand the expectation as:

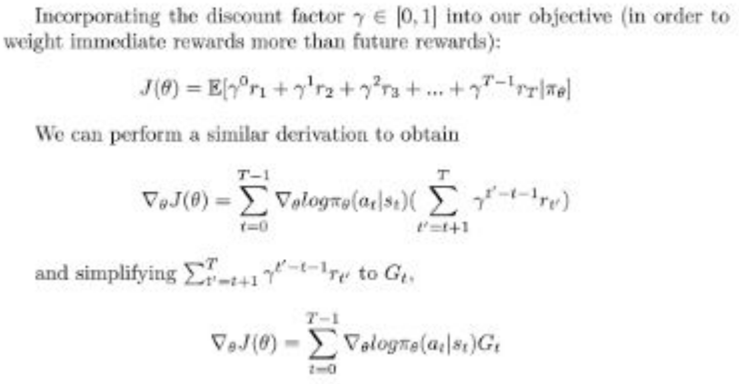
$$J(\theta) = \mathbb{E}[\sum_{t=0}^{T-1} r_{t+1} | \pi_\theta] = \sum_{t=i}^{T-1} P(s_t, a_t|\tau) r_{t+1}$$

where i is an arbitrary starting point in a trajectory, $P(s_t, a_t|\tau)$ is the probability of the occurrence of s_t, a_t given the trajectory τ.

Differentiate both sides with respect to policy parameter θ:



Then, differentiating $logP(s_t, a_t|\tau)$ with respect to θ yields:



Incorporating the discount factor $\gamma \in [0, 1]$ into our objective (in order to weight immediate rewards more than future rewards):

$$J(\theta) = \mathbb{E}[\gamma^0 r_1 + \gamma^1 r_2 + \gamma^2 r_3 + ... + \gamma^{T-1} r_T | \pi_\theta]$$

We can perform a similar derivation to obtain

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t|s_t)(\sum_{t'=t+1}^{T} \gamma^{t'-t-1} r_{t'})$$

and simplifying $\sum_{t'=t+1}^{T} \gamma^{t'-t-1} r_{t'}$ to G_t,

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t|s_t) G_t$$

Here's a blog post on it though, it's actually not that bad:
https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63

Policy Gradient

So the derivation… I was actually going to walk through all of this, but then realized we wouldn't have time. This is the important part though, this is the "answer."

Preliminaries

In essence, policy gradient methods update the probability distribution of actions so that actions with higher expected reward have a higher probability value for an observed state. We will assume discrete (finite) action space and a stochastic (non-deterministic) policy for this post.

Some Definitions

1. Reinforcement Learning Objective

The objective function for policy gradients is defined as:

$$J(\theta) = \mathbb{E}[\sum_{t=0}^{T-1} r_{t+1}]$$

In other words, the objective is to learn a policy that maximizes the cumulative future reward to be received starting from any given time `t` until the terminal time `T`.

Note that `r_{t+1}` is the reward received by performing action `a_{t}` at state `s_{t}` ; `r_{t+1} = R(s_{t}, a_{t})` where `R` is the reward function.

Since this is a maximization problem, we optimize the policy by taking the gradient ascent with the partial derivative of the objective with respect to the policy parameter `theta`.

$$\theta \leftarrow \theta + \frac{\partial}{\partial \theta} J(\theta)$$

The policy function is parameterized by a neural network (since we live in the world of deep learning).

2. Expectation

Frequently appearing in literature is the expectation notation — it is used because we want to optimize long term future (predicted) rewards, which has a degree of uncertainty.

The expectation, also known as the expected value or the mean, is computed by the summation of the product of every `x` value and its probability.

$$\mathbb{E}[f(x)] = \sum_{x} P(x) f(x)$$

Where `P(x)` represents the probability of the occurrence of random variable `x`, and `f(x)` is a function denoting the value of `x`.

Deriving the Policy Gradient

Let us start with the defined objective function $J(\theta)$. We can expand the expectation as:

$$\begin{aligned} J(\theta) &= \mathbb{E}[\sum_{t=0}^{T-1} r_{t+1} | \pi_\theta] \\ &= \sum_{t=i}^{T-1} P(s_t, a_t|\tau) r_{t+1} \end{aligned}$$

where i is an arbitrary starting point in a trajectory, $P(s_t, a_t|\tau)$ is the probability of the occurrence of s_t, a_t given the trajectory τ.

Differentiate both sides with respect to policy parameter θ:

$$\text{Using} \quad \frac{d}{dx} log f(x) = \frac{f'(x)}{f(x)},$$

$$\begin{aligned} \nabla_\theta J(\theta) &= \sum_{t=i}^{T-1} \nabla_\theta P(s_t, a_t|\tau) r_{t+1} \\ &= \sum_{t=i}^{T-1} P(s_t, a_t|\tau) \frac{\nabla_\theta P(s_t, a_t|\tau)}{P(s_t, a_t|\tau)} r_{t+1} \\ &= \sum_{t=i}^{T-1} P(s_t, a_t|\tau) \nabla_\theta log P(s_t, a_t|\tau) r_{t+1} \\ &= \mathbb{E}[\sum_{t=i}^{T-1} \nabla_\theta log P(s_t, a_t|\tau) r_{t+1}] \end{aligned}$$

However, during, learning, we take random samples of episodes instead of computing the expectation, so we can replace the expectation with

$$\nabla_\theta J(\theta) \sim \sum_{t=i}^{T-1} \nabla_\theta log P(s_t, a_t|\tau) r_{t+1}$$

From here, let us take a more careful look into $\nabla_\theta log P(s_t, a_t|\tau)$. First, by definition,

$$\begin{aligned} P(s_t, a_t|\tau) &= P(s_0, a_0, s_1, a_2, ..., s_{t-1}, a_{t-1}, s_t, a_t|\pi_\theta) \\ &= P(s_0)\pi_\theta(a_1|s_0)P(s_1|s_0, a_0)\pi_\theta(a_2|s_1)P(s_2|s_1, a_1)\pi_\theta(a_3|s_2) \\ &\quad ...P(s_{t-1}|s_{t-2}, a_{t-2})\pi_\theta(a_{t-1}|s_{t-2})P(s_t|s_{t-1}, a_{t-1})\pi_\theta(a_t|s_{t-1}) \end{aligned}$$

If we log both sides,

$$\begin{aligned} log P(s_t, a_t|\tau) &= log(P(s_0)\pi_\theta(a_1|s_0)P(s_1|s_0, a_0)\pi_\theta(a_2|s_1)P(s_2|s_1, a_1)\pi_\theta(a_3|s_2)... \\ &\quad P(s_{t-1}|s_{t-2}, a_{t-2})\pi_\theta(a_{t-1}|s_{t-2})P(s_t) log \pi_\theta(a_t|s_{t-1}) \\ &= log P(s_0) + log \pi_\theta(a_1|s_0) + log P(s_1|s_0, a_0) + log \pi_\theta(a_2|s_1) \\ &\quad + log P(s_2|s_1, a_1) + log \pi_\theta(a_3|s_2) + ... + log P(s_{t-1}|s_{t-2}, a_{t-2}) \\ &\quad + log \pi_\theta(a_{t-1}|s_{t-2}) + log P(s_t|s_{t-1}, a_{t-1}) + log \pi_\theta(a_t|s_{t-1}) \end{aligned}$$

Then, differentiating $log P(s_t, a_t|\tau)$ with respect to θ yields:

$$\begin{aligned} \nabla_\theta log P(s_t, a_t|\tau) &= \nabla_\theta log P(s_0) + \nabla_\theta log \pi_\theta(a_1|s_0) + \nabla_\theta log P(s_1|s_0, a_0) \\ &\quad + \nabla_\theta log \pi_\theta(a_2|s_1) + \nabla_\theta log P(s_2|s_1, a_1) + \nabla_\theta log \pi_\theta(a_3|s_2) + \\ &\quad ... + \nabla_\theta log P(s_{t-1}|s_{t-2}, a_{t-2}) + \nabla_\theta log \pi_\theta(a_{t-1}|s_{t-2}) + \\ &\quad \nabla_\theta log P(s_t|s_{t-1}, a_{t-1}) + \nabla_\theta log \pi_\theta(a_t|s_{t-1}) \end{aligned}$$

However, note that the $P(s_t|s_{t-1}, a_{t-1})$ is not dependent on the policy parameter θ, and is solely dependant on the environment on which the reinforcement learning is acting on; it is assumed that the state transition is unknown to the agent in model free reinforcement learning. Thus, the gradient of it with respect to θ will be 0. How convenient! So:

$$\begin{aligned} \nabla_\theta log P(s_t, a_t|\tau) &= 0 + \nabla_\theta log \pi_\theta(a_1|s_0) + 0 + \nabla_\theta log \pi_\theta(a_2|s_1) + 0 + \nabla_\theta log \pi_\theta(a_3|s_2) + \\ &\quad ... + 0 + \nabla_\theta log \pi_\theta(a_{t-1}|s_{t-2}) + 0 \\ &= \nabla_\theta log \pi_\theta(a_1|s_0) + \nabla_\theta log \pi_\theta(a_2|s_1) + \nabla_\theta log \pi_\theta(a_3|s_2) + \\ &\quad ... + \nabla_\theta log \pi_\theta(a_{t-1}|s_{t-2}) + log \pi_\theta(a_t|s_{t-1}) \\ &= \sum_{t'=0}^{t} \nabla_\theta log \pi_\theta(a_{t'}|s_{t'}) \end{aligned}$$

Plugging this into our $\nabla_\theta J(\theta)$ yields:

$$\begin{aligned} \nabla_\theta J(\theta) &= \sum_{t=0}^{T-1} r_{t+1} \nabla_\theta P(s_t, a_t|\tau) \\ &= \sum_{t=0}^{T-1} r_{t+1} (\sum_{t'=0}^{t} \nabla_\theta log \pi_\theta(a_{t'}|s_{t'})) \end{aligned}$$

Lets expand that!

$$\begin{aligned} \nabla_\theta J(\theta) &= \sum_{t=0}^{T-1} r_{t+1} (\sum_{t'=0}^{t} \nabla_\theta log \pi_\theta(a_{t'}|s_{t'})) \\ &= r_1 (\sum_{t'=0}^{0} \nabla_\theta log \pi_\theta(a_{t'}|s_{t'})) + r_2 (\sum_{t'=0}^{1} \nabla_\theta log \pi_\theta(a_{t'}|s_{t'})) \\ &\quad + r_3 (\sum_{t'=0}^{2} \nabla_\theta log \pi_\theta(a_{t'}|s_{t'})) + ... + r_{T-1} (\sum_{t'=0}^{T-1} \nabla_\theta log \pi_\theta(a_{t'}|s_{t'})) \\ &= r_1 \nabla_\theta log \pi_\theta(a_0|s_0) + r_2 (\nabla_\theta log \pi_\theta(a_0|s_0) + \nabla_\theta log \pi_\theta(a_1|s_1)) \\ &\quad + r_3 (\nabla_\theta log \pi_\theta(a_0|s_0) + \nabla_\theta log \pi_\theta(a_1|s_1) + \nabla_\theta log \pi_\theta(a_2|s_2)) \\ &\quad + ... + r_T (\nabla_\theta log \pi_\theta(a_0|s_0) + \nabla_\theta log \pi_\theta(a_1|s_1) + ... + \nabla_\theta log \pi_\theta(a_{T-1}|s_{T-1})) \\ &= \nabla_\theta log \pi_\theta(a_0|s_0)(r_1 + r_2 + ... + r_T) + \nabla_\theta log \pi_\theta(a_1|s_1)(r_2 + r_3 + ... + r_T) \\ &\quad + \nabla_\theta log \pi_\theta(a_2|s_2)(r_3 + r_4 + ... + r_T) + ... + \nabla_\theta log \pi_\theta(a_{T-1}|s_{T-1}) r_T \\ &= \sum_{t=0}^{T-1} \nabla_\theta log \pi_\theta(a_t|s_t) (\sum_{t'=t+1}^{T} r_{t'}) \end{aligned}$$

Simplifying the term $\sum_{t'=t+1}^{T} r_{t'}$ to G_t, we can derive the policy gradient

$$\sum_{t=0}^{T-1} \nabla_\theta log \pi_\theta(a_t|s_t) G_t$$

Incorporating the discount factor $\gamma \in [0, 1]$ into our objective (in order to weight immediate rewards more than future rewards):

$$J(\theta) = \mathbb{E}[\gamma^0 r_1 + \gamma^1 r_2 + \gamma^2 r_3 + ... + \gamma^{T-1} r_T | \pi_\theta]$$

We can perform a similar derivation to obtain

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T-1} \nabla_\theta log \pi_\theta(a_t|s_t) (\sum_{t'=t+1}^{T} \gamma^{t'-t-1} r_{t'})$$

and simplifying $\sum_{t'=t+1}^{T} \gamma^{t'-t-1} r_{t'}$ to G_t,

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T-1} \nabla_\theta log \pi_\theta(a_t|s_t) G_t$$

Here's a blog post on it though, it's actually not that bad:
https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63

Policy Gradient

What does it mean?

$$J(\theta) = \mathbb{E}[\sum_{t=0}^{T-1} r_{t+1}]$$

Here's a blog post on it though, it's actually not that bad:
https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63

Policy Gradient

What does it mean?

$$J(\theta) = \mathbb{E}[\sum_{t=0}^{T-1} r_{t+1}]$$

The objective function

Here's a blog post on it though, it's actually not that bad:
https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63

Policy Gradient

What does it mean?

$$J(\theta) = \mathbb{E}[\sum_{t=0}^{T-1} r_{t+1}]$$

The objective function

Model parameters (NN weights)

Here's a blog post on it though, it's actually not that bad:
https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63

Policy Gradient

What does it mean?

$$J(\theta) = \mathbb{E}[\sum_{t=0}^{T-1} r_{t+1}]$$

The objective function

Model parameters (NN weights)

Expectation is with respect to unknown transition dynamics and our own action distribution

Here's a blog post on it though, it's actually not that bad:
https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63

Policy Gradient

What does it mean?

The sum of future rewards

$$J(\theta) = \mathbb{E}[\sum_{t=0}^{T-1} r_{t+1}]$$

The objective function

Model parameters (NN weights)

Expectation is with respect to unknown transition dynamics and our own action distribution

Here's a blog post on it though, it's actually not that bad:
https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63

Policy Gradient

What does it mean?

There is often a discount factor in here, but we will ignore it for now

The sum of future rewards

$$J(\theta) = \mathbb{E}[\sum_{t=0}^{T-1} r_{t+1}]$$

The objective function

Model parameters (NN weights)

Expectation is with respect to unknown transition dynamics and our own action distribution

Here's a blog post on it though, it's actually not that bad:
https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63

Policy Gradient

What does it mean?

$$J(\theta) = \mathbb{E}[\sum_{t=0}^{T-1} r_{t+1}]$$

So what we want is to maximize this thing, which is the expected sum of future rewards

Here's a blog post on it though, it's actually not that bad:
https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63

Policy Gradient

What does it mean?

$$J(\theta) = \mathbb{E}[\sum_{t=0}^{T-1} r_{t+1}]$$

$$\nabla_\theta J(\theta)$$

$$\theta \leftarrow \theta + \frac{\partial}{\partial \theta} J(\theta)$$

And what we want is the gradient of this objective function…

…so we can adjust our network parameters in the direction that increases this objective.

Here's a blog post on it though, it's actually not that bad:
https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63

Policy Gradient

What does it mean?

$$J(\theta) = \mathbb{E}[\sum_{t=0}^{T-1} r_{t+1}]$$

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T-1} \nabla_\theta log \pi_\theta(a_t|s_t) G_t$$

This is the “answer” that we highlighted earlier

Here’s a blog post on it though, it’s actually not that bad:
https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63

Policy Gradient

What does it mean?

$$J(\theta) = \mathbb{E}[\sum_{t=0}^{T-1} r_{t+1}]$$

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T-1} \nabla_\theta log \pi_\theta(a_t|s_t) G_t$$

What it says is that the gradient is the sum over all steps in a trajectory…

…of the log of the probability of taking whichever action was taken…

…times the empirical return (the sum of future rewards).

Here's a blog post on it though, it's actually not that bad:
https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63

Policy Gradient

How do we adjust the probability based on the sum of rewards?

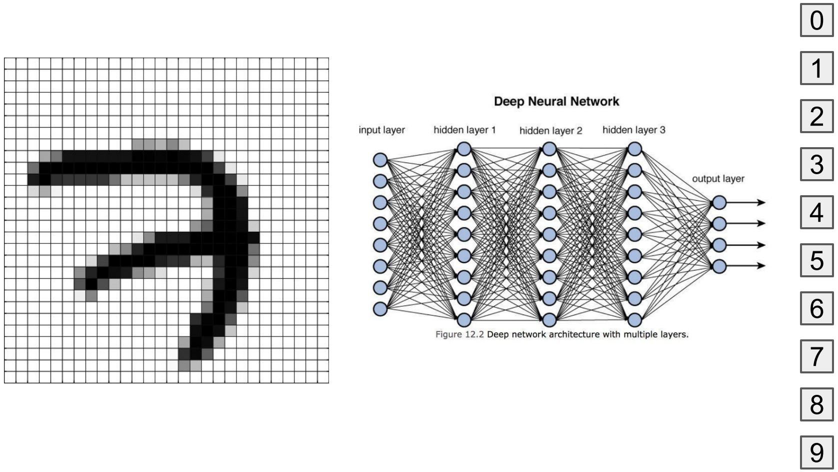
We will show this two ways:

1. Derivation!
2. <u>Simple-ish Intuition!</u>

Policy Gradient

How do we adjust the probability based on the sum of rewards?

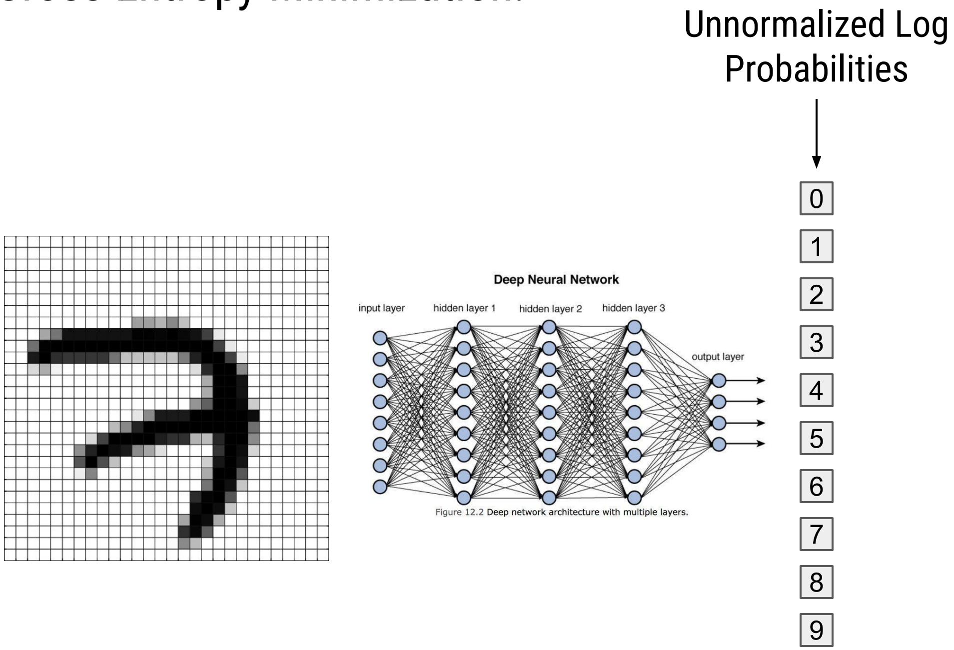
Cross Entropy Minimization:



Policy Gradient

How do we adjust the probability based on the sum of rewards?

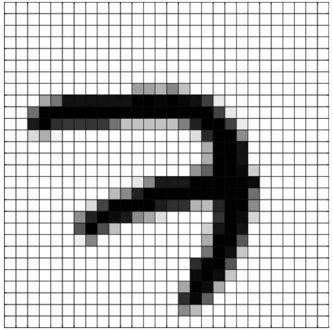
Cross Entropy Minimization:



Policy Gradient

How do we adjust the probability based on the sum of rewards?

Cross Entropy Minimization:



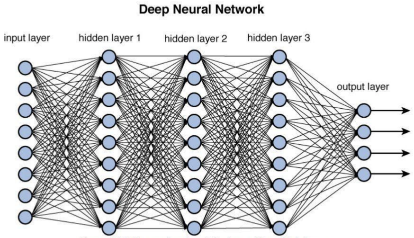


Figure 12.2 Deep network architecture with multiple layers.

Unnormalized Log Probabilities

0
1
2
3
4
5
6
7
8
9

$$\frac{e^{x_i}}{\sum_{i=0\ldots9} e^{x_i}}$$

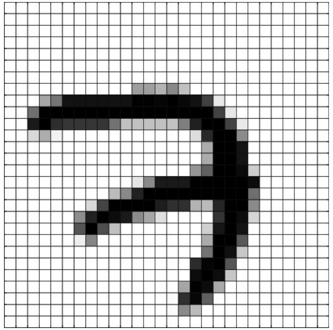
Normalized Probabilities (p_{xi})

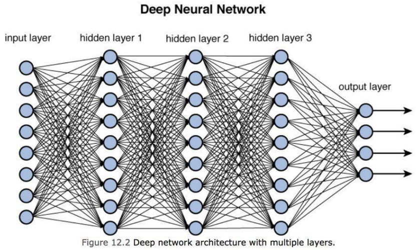
0
1
2
3
4
5
6
7
8
9

Policy Gradient

How do we adjust the probability based on the sum of rewards?

Cross Entropy Minimization:





Unnormalized Log Probabilities

0 1 2 3 4 5 6 7 8 9

$$\frac{e^{xi}}{\sum_{i=0\ldots9} e^{xi}}$$

Normalized Probabilities (p_{xi})

0 1 2 3 4 5 6 7 8 9

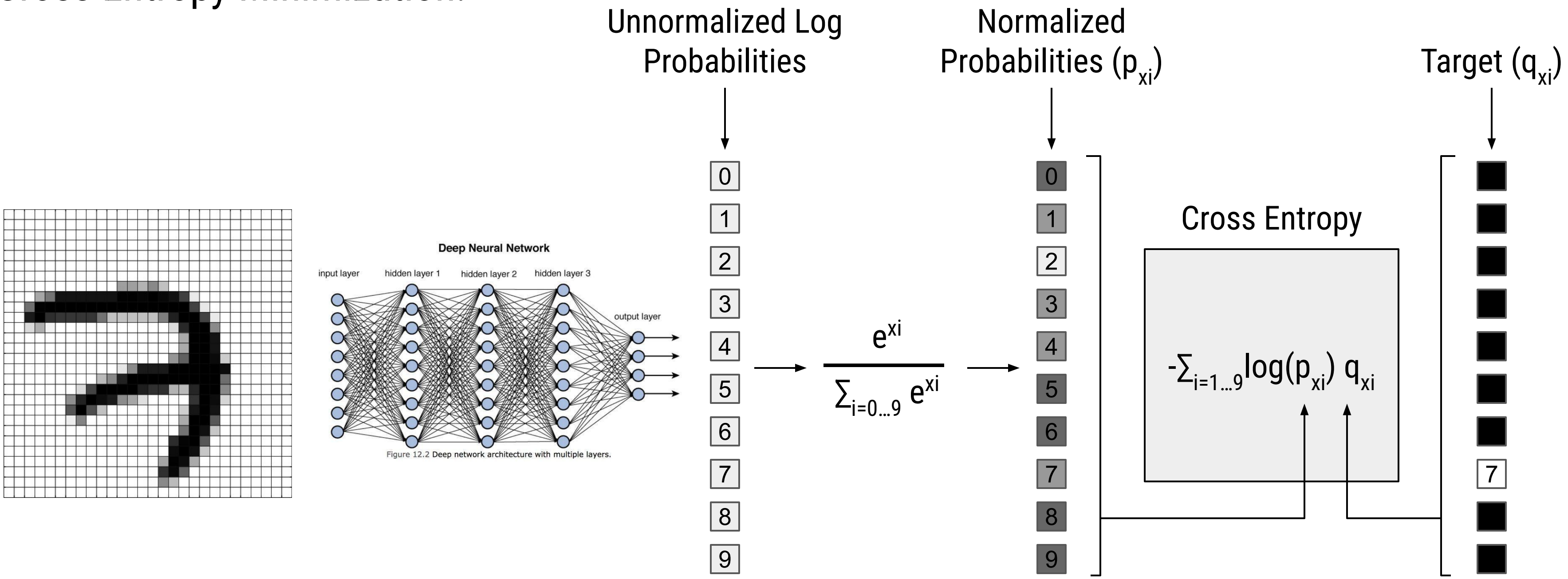
Target (q_{xi})

7

Policy Gradient

How do we adjust the probability based on the sum of rewards?

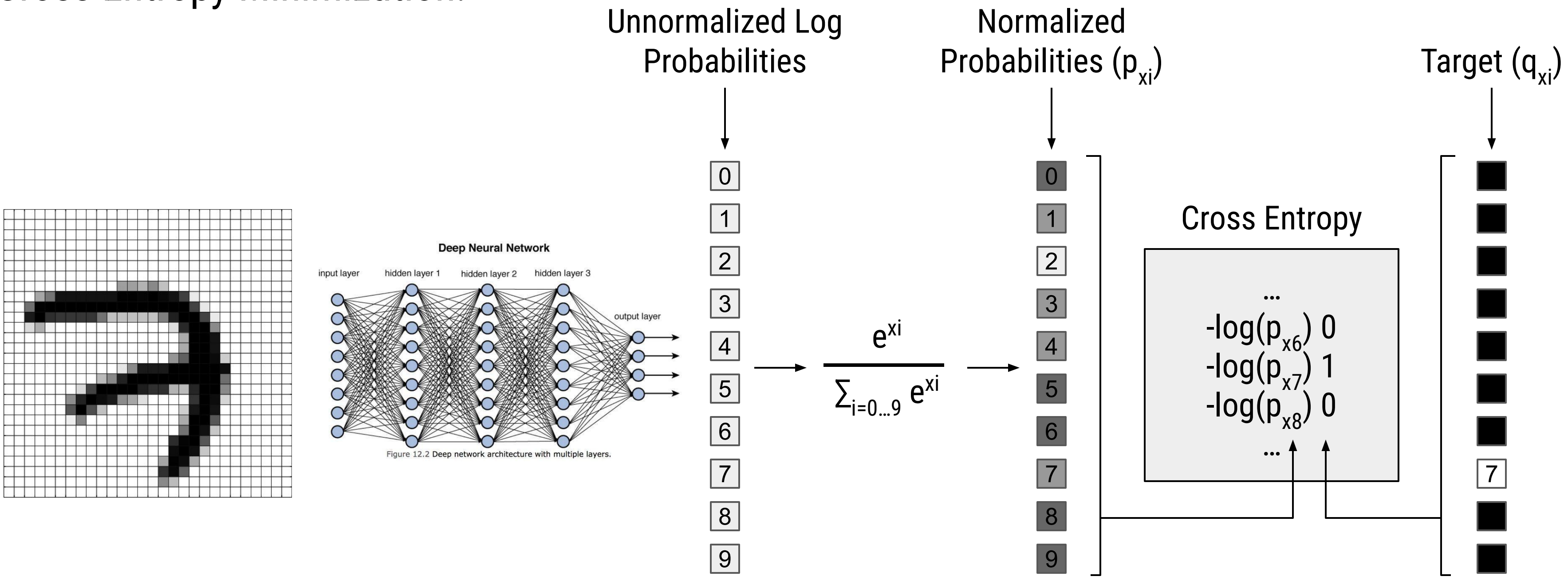
Cross Entropy Minimization:



Policy Gradient

How do we adjust the probability based on the sum of rewards?

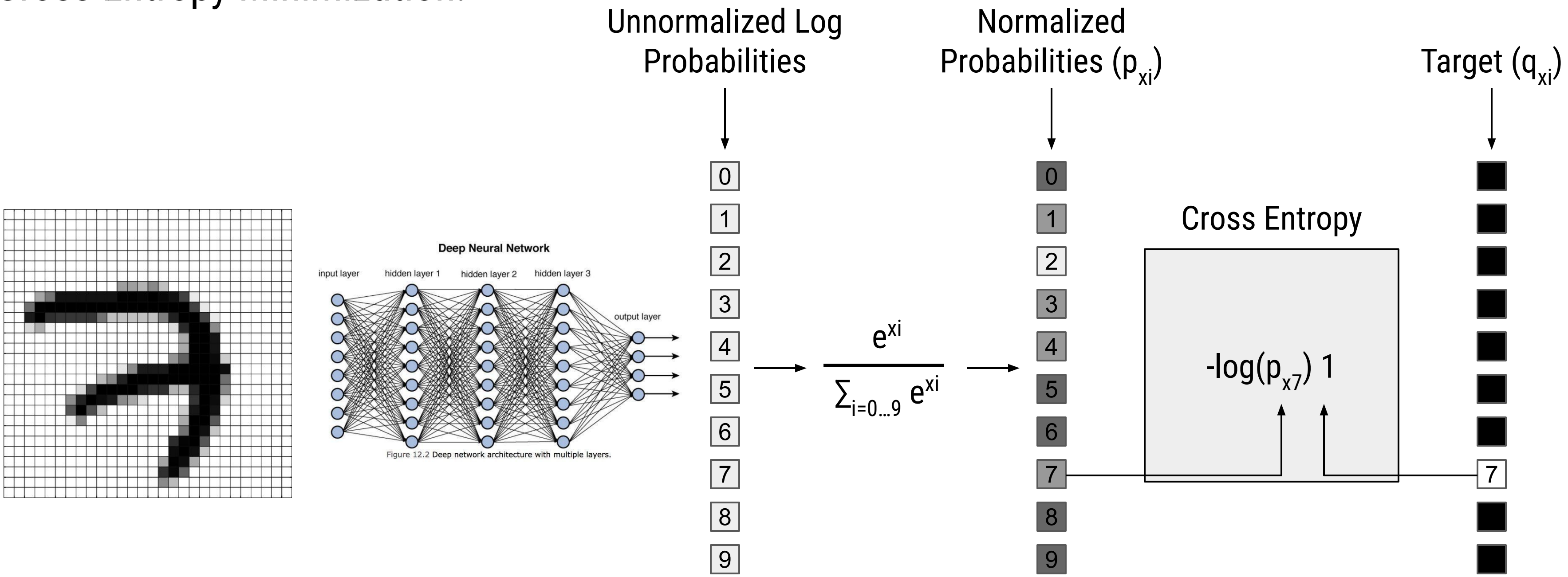
Cross Entropy Minimization:



Policy Gradient

How do we adjust the probability based on the sum of rewards?

Cross Entropy Minimization:



Policy Gradient

How do we adjust the probability based on the sum of rewards?

Cross Entropy Minimization:

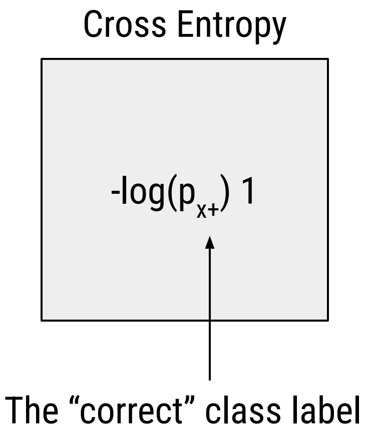
Cross Entropy

$-\log(p_{x7})$ 1

Policy Gradient

How do we adjust the probability based on the sum of rewards?

Cross Entropy Minimization:



Policy Gradient

How do we adjust the probability based on the sum of rewards?

Cross Entropy Minimization:

Policy Gradient

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T-1} \nabla_\theta log \pi_\theta(a_t|s_t) G_t$$

Cross Entropy

$-log(p_{x+})\ 1$

Policy Gradient

How do we adjust the probability based on the sum of rewards?

Cross Entropy Minimization:

Policy Gradient

$$\nabla_\theta log \pi_\theta(a_t|s_t)G_t$$

Cross Entropy

-log(p_{x+}) 1

Policy Gradient

How do we adjust the probability based on the sum of rewards?

Cross Entropy Minimization:

Policy Gradient

$$\nabla_\theta log \pi_\theta(a_t|s_t) G_t$$

Cross Entropy Gradient

$$\nabla_\theta -\log(p_{x+})\ 1$$

Policy Gradient

How do we adjust the probability based on the sum of rewards?

Cross Entropy Minimization:

This is the cross entropy gradient scaled by the return

Policy Gradient

$$\nabla_\theta log \pi_\theta(a_t|s_t)G_t$$

Policy Distribution

Cross Entropy Gradient

$$\nabla_\theta -\log(p_{x+})\ 1$$

Class Distribution

Policy Gradient

How do we adjust the probability based on the sum of rewards?

Take the update you would make to the network if the action you took was "correct" according to a standard classification objective and scale it by the return

Policy Gradient

$$\nabla_\theta log \pi_\theta(a_t|s_t) G_t$$

Policy Distribution

Cross Entropy Gradient

$$\nabla_\theta -\log(p_{x+})\ 1$$

Class Distribution

Policy Gradient

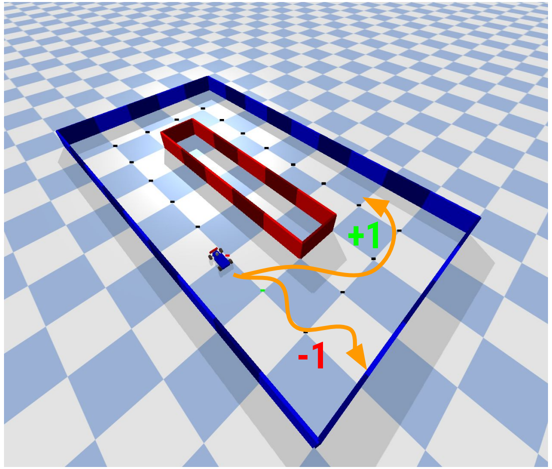
One Caveat:

What is the optimal policy for both of these environments?

Reward:

Crashing: -1

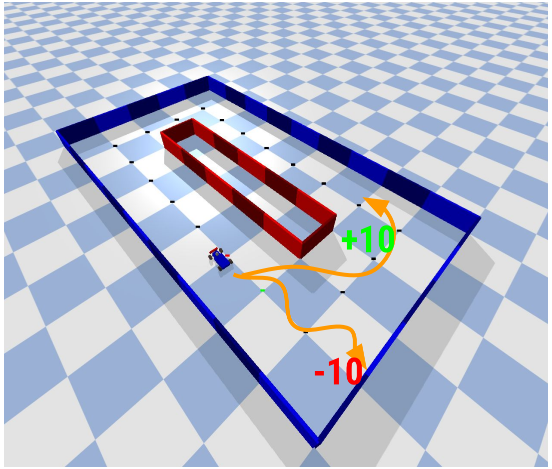
Making the first turn: +1



Reward:

Crashing: -10

Making the first turn: +10



Policy Gradient

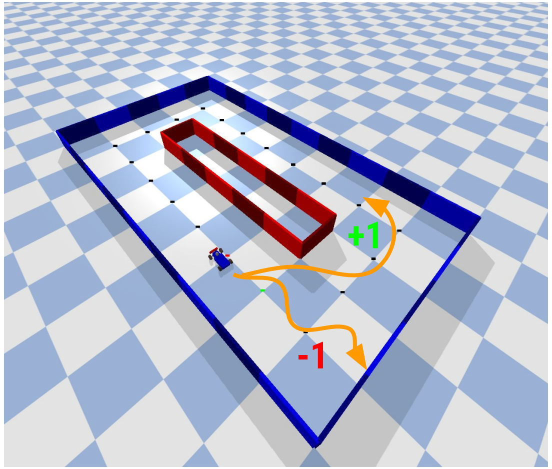
One Caveat:

What about these two?

Reward:

Crashing: -1

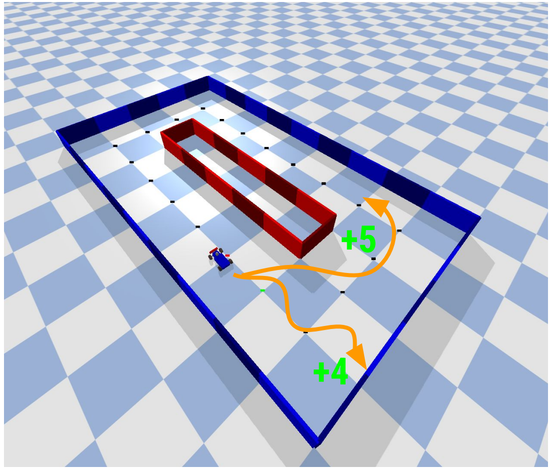
Making the first turn: +1



Reward:

Crashing: +4

Making the first turn: +5



Policy Gradient

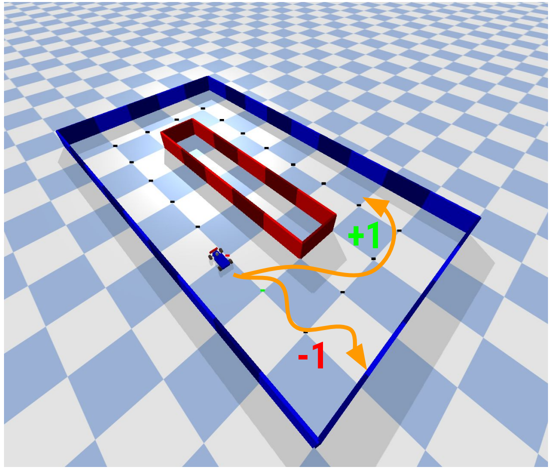
One Caveat:

The “ordering” of policies is invariant to linear transformations of reward!

Reward:

Crashing: -1

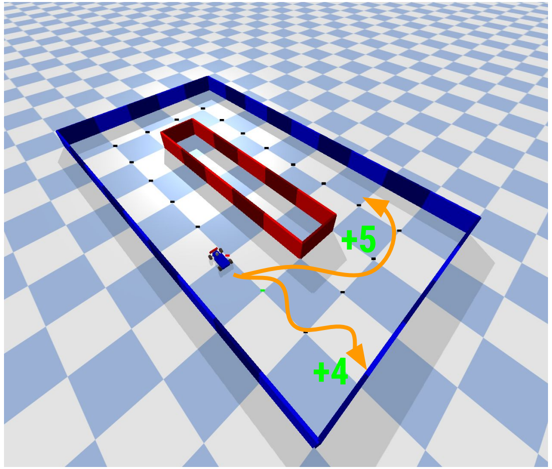
Making the first turn: +1



Reward:

Crashing: +4

Making the first turn: +5



Policy Gradient

One Caveat:

The “ordering” of policies is invariant to linear transformations of reward!

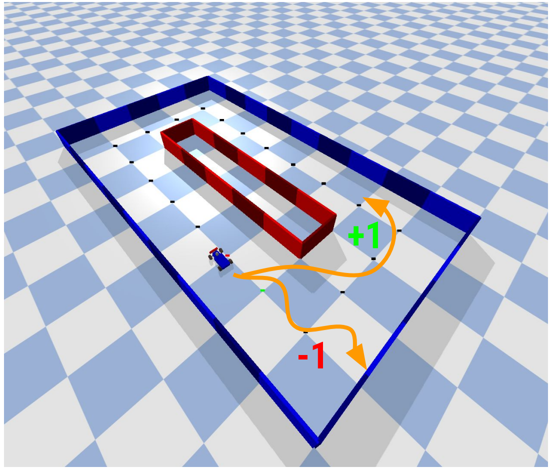
<u>But our learning rule is definitely sensitive to these transformations!</u>

$$\nabla_\theta log \pi_\theta(a_t|s_t) G_t$$

Reward:

Crashing: -1

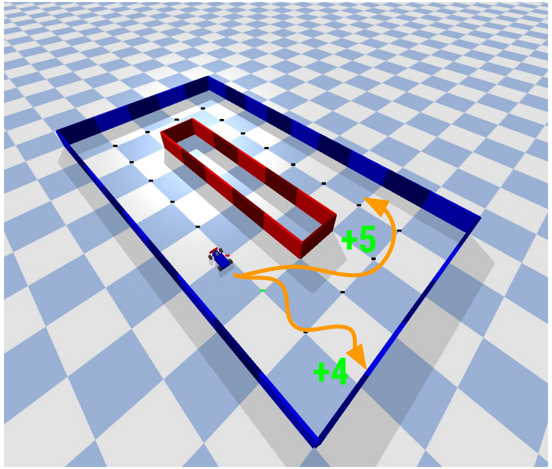
Making the first turn: +1



Reward:

Crashing: +4

Making the first turn: +5



Policy Gradient

The simplest fix:

Subtract the mean and divide by the standard deviation before using returns for training.

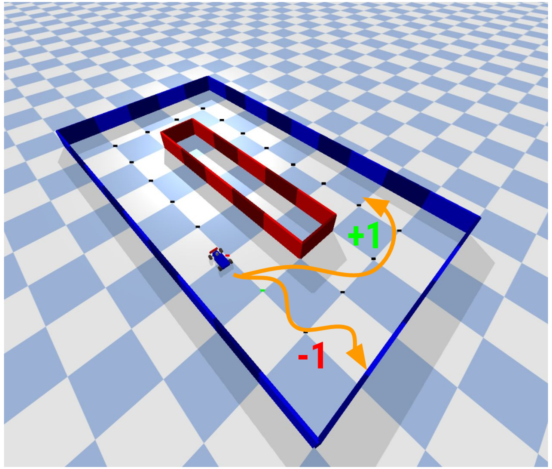
Encourages above average actions while discouraging below-average actions.

$$\nabla_\theta log \pi_\theta(a_t|s_t)\left(\frac{G_t \text{ - mean}}{\text{std}}\right)$$

Reward:

Crashing: -0.707

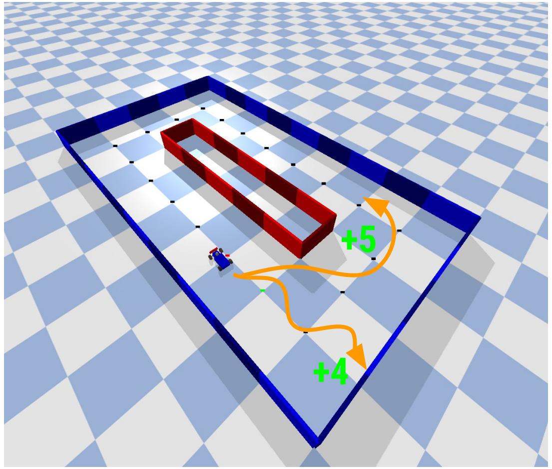
Making the first turn: +0.707



Reward:

Crashing: -0.707

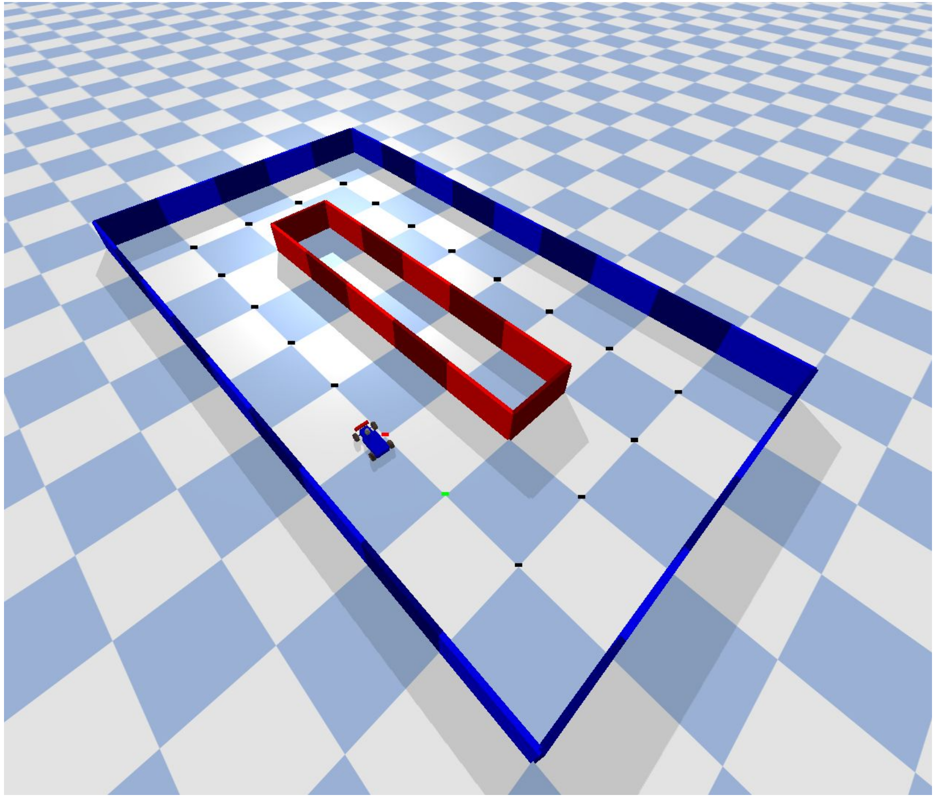
Making the first turn: +0.707



Policy Gradient

Second Caveat:

Some states are actually better than others.

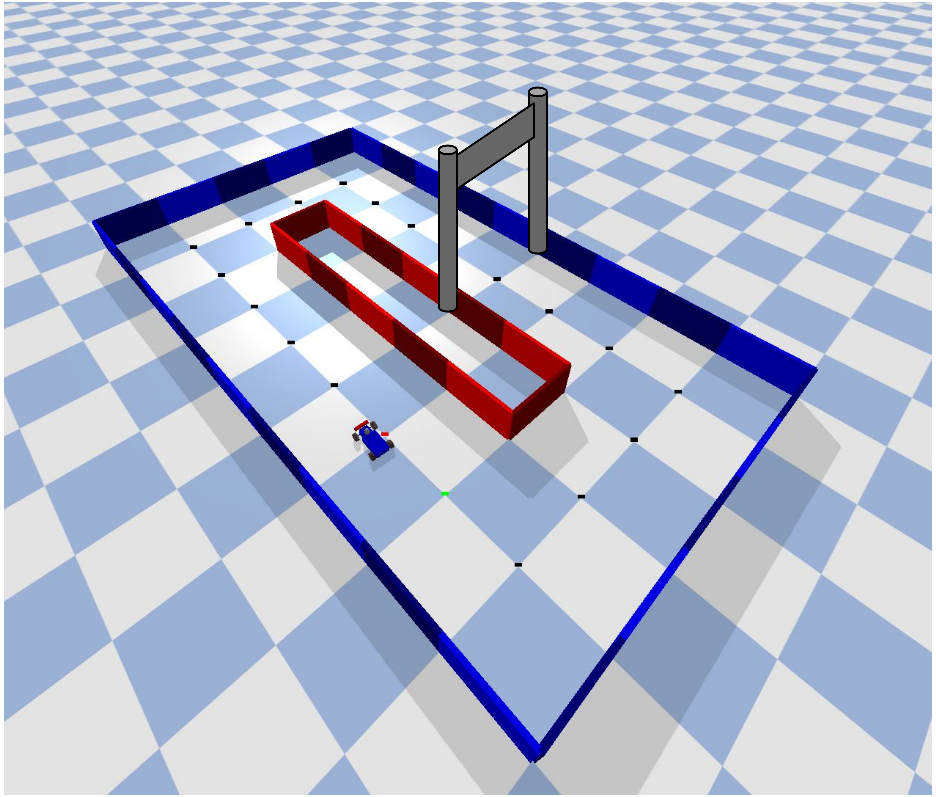


Policy Gradient

Second Caveat:

Some states are actually better than others.

Imagine our episode only lasts for a certain number of steps.



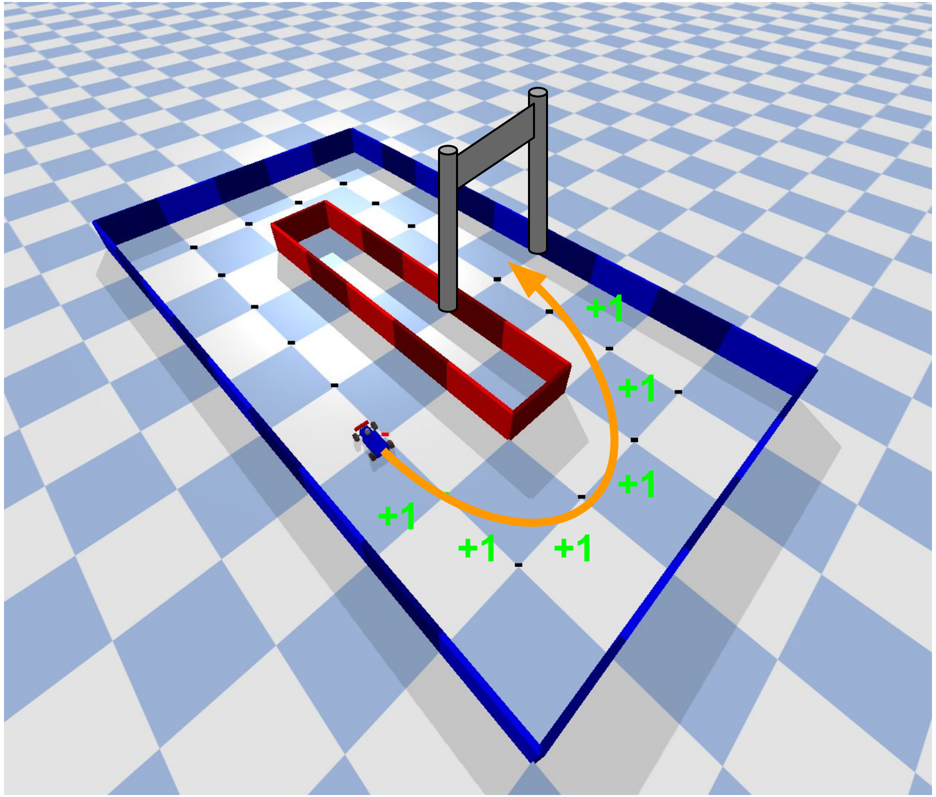
Policy Gradient

Second Caveat:

Some states are actually better than others.

Imagine our episode only lasts for a certain number of steps.

And you get reward for making progress along the way.



Policy Gradient

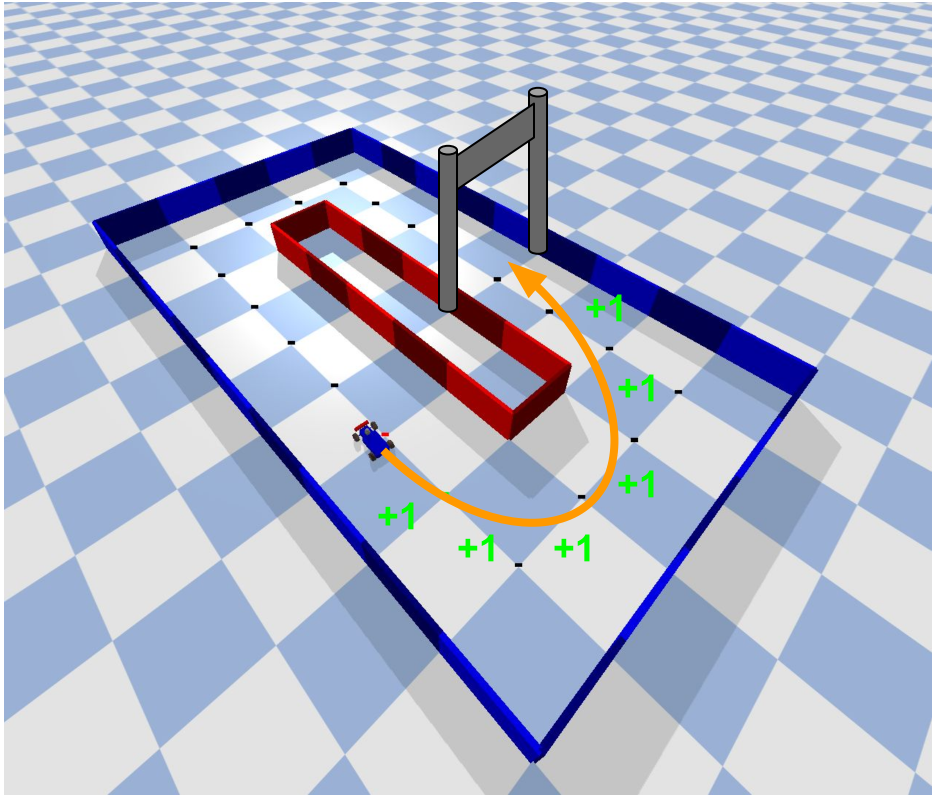
Second Caveat:

Some states are actually better than others.

Imagine our episode only lasts for a certain number of steps.

And you get reward for making progress along the way.

What happens?



Policy Gradient

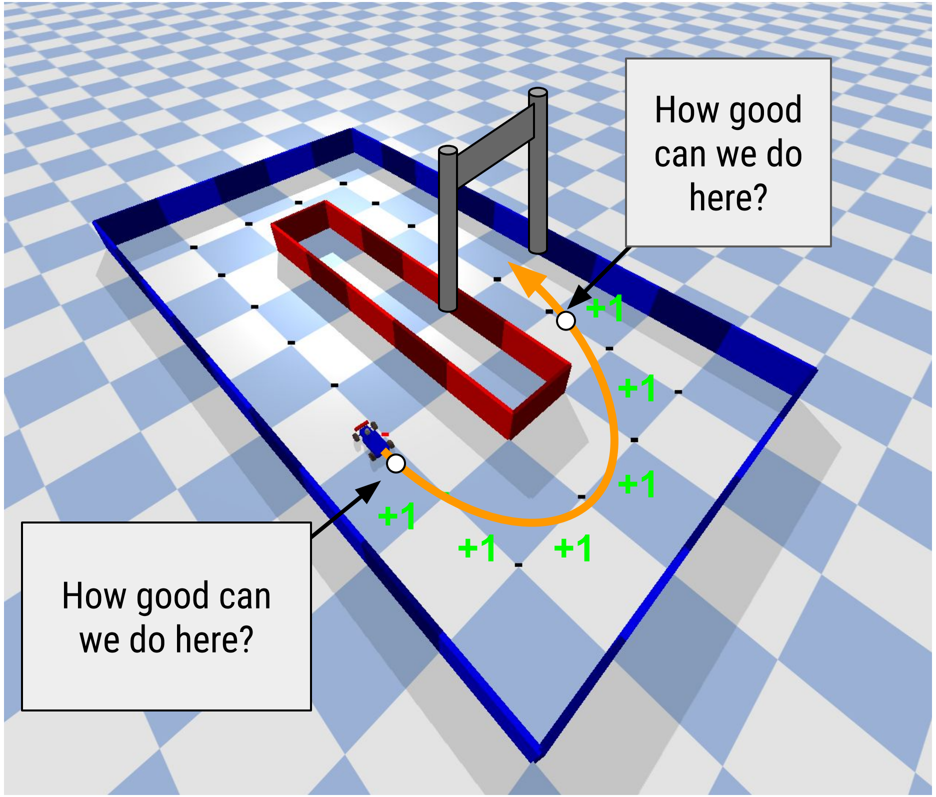
Second Caveat:

Some states are actually better than others.

Imagine our episode only lasts for a certain number of steps.

And you get reward for making progress along the way.

What happens?



Policy Gradient

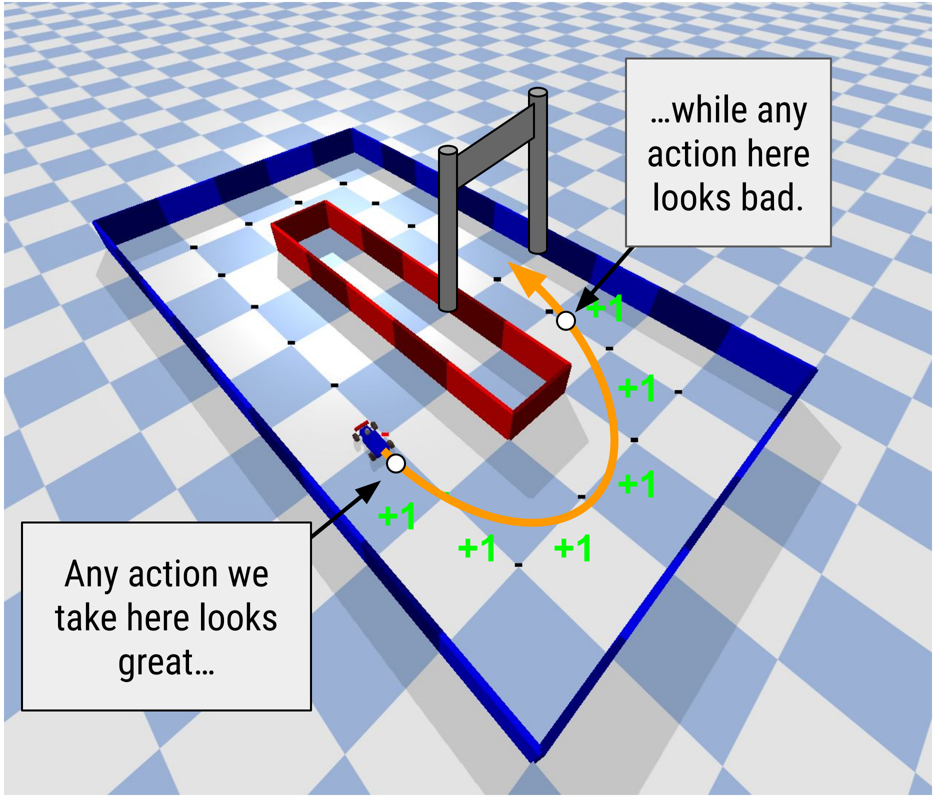
Second Caveat:

Some states are actually better than others.

Imagine our episode only lasts for a certain number of steps.

And you get reward for making progress along the way.

What happens?



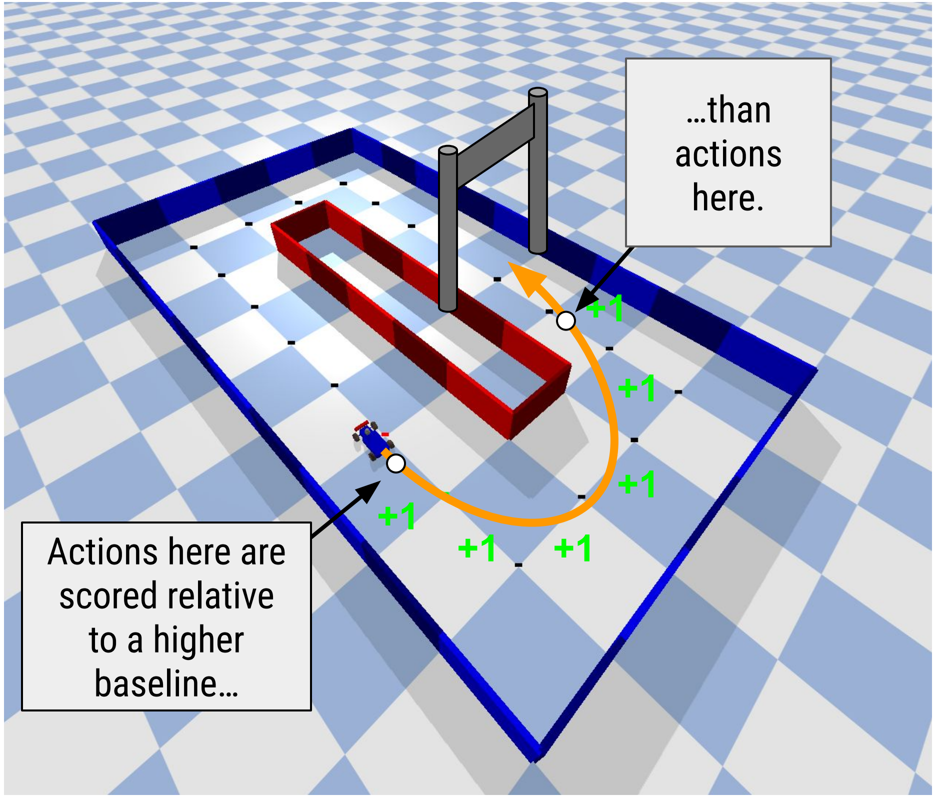
Policy Gradient

The simplest fix:

Train a second “baseline” network to estimate future returns of each state.

Subtract the baseline from returns.

$$\nabla_\theta log \pi_\theta(a_t|s_t)\left(\frac{G_t - \text{mean}}{\text{std}} - \text{baseline}(s_t)\right)$$

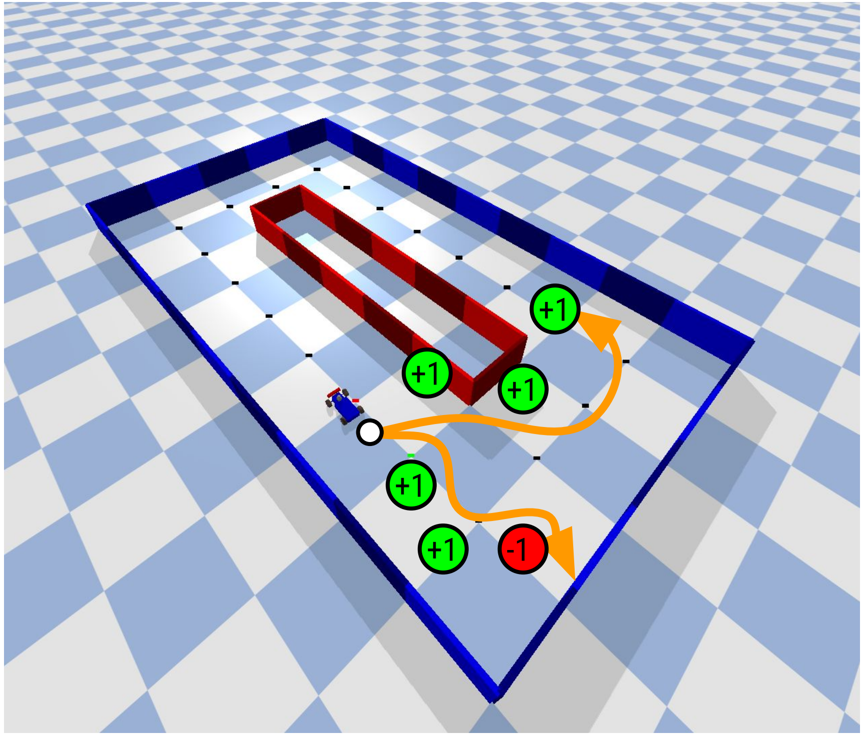


Policy Gradient

Ok, so everything we’ve done so far has only shown us how to do a single update. How do we turn this into an entire algorithm?

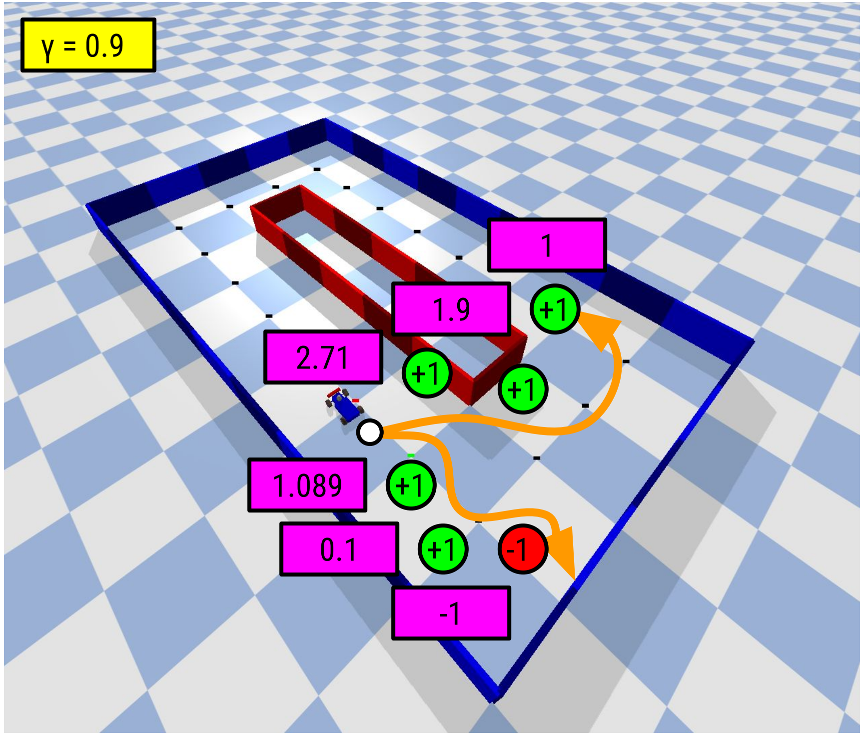
Policy Gradient

1. Collect data by letting the agent drive in the environment



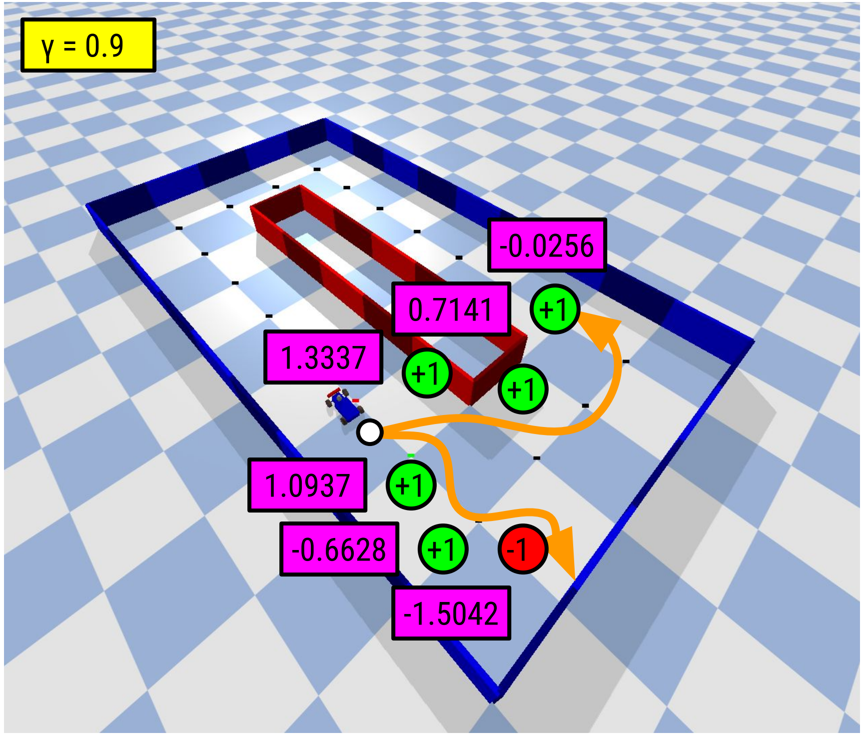
Policy Gradient

1. Collect data by letting the agent drive in the environment
2. Compute returns from the rewards in the trajectories



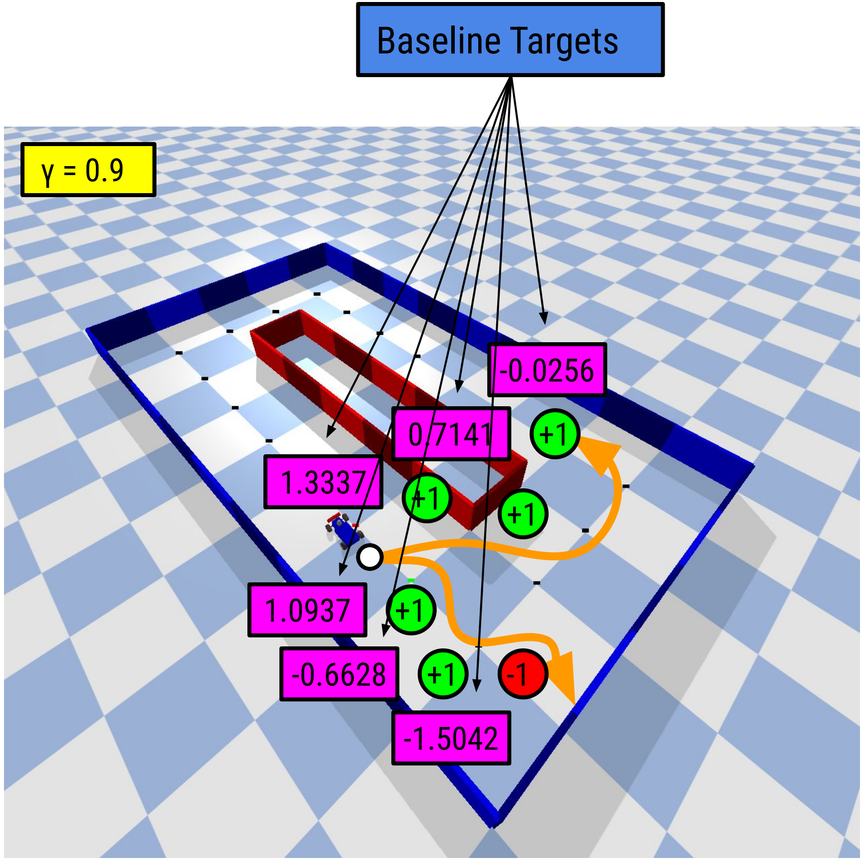
Policy Gradient

1. Collect data by letting the agent drive in the environment
2. Compute returns from the rewards in the trajectories
3. Normalize the returns using the mean, and std



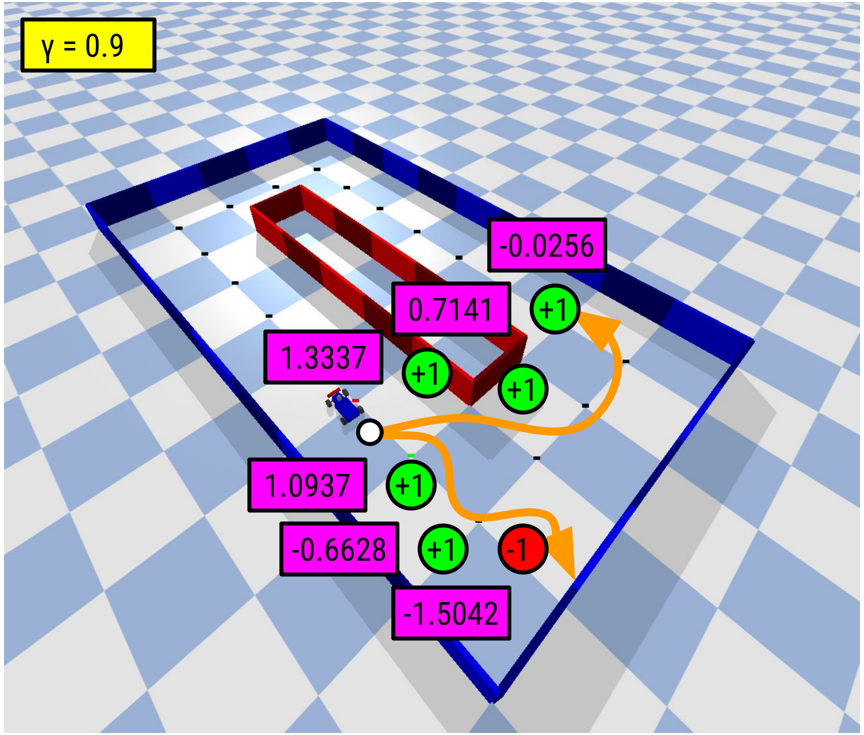
Policy Gradient

1. Collect data by letting the agent drive in the environment
2. Compute returns from the rewards in the trajectories
3. Normalize the returns using the mean, and std
4. Update the baseline by training it to match the current returns at each of the states visited



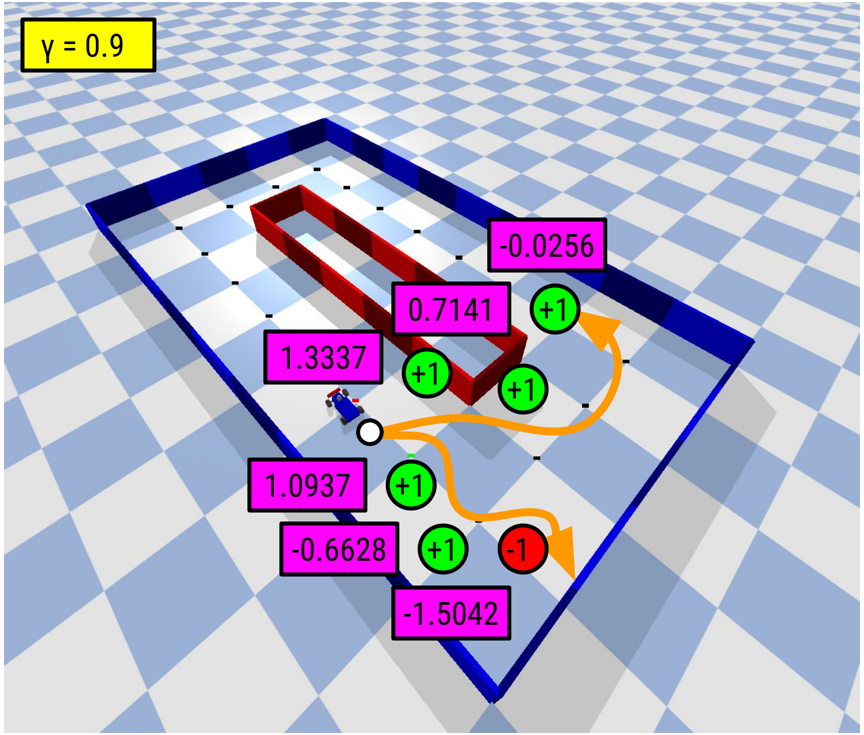
Policy Gradient

1. Collect data by letting the agent drive in the environment
2. Compute returns from the rewards in the trajectories
3. Normalize the returns using the mean, and std
4. Update the baseline by training it to match the current returns at each of the states visited
5. Update the Policy using the policy gradient and the returns from step 3 offset by the baseline



Policy Gradient

1. Collect data by letting the agent drive in the environment
2. Compute returns from the rewards in the trajectories
3. Normalize the returns using the mean, and std
4. Update the baseline by training it to match the current returns at each of the states visited
5. Update the Policy using the policy gradient and the returns from step 3 offset by the baseline
6. Repeat



Policy Gradient

1. Collect data by letting the agent drive in the environment
2. Compute returns from the rewards in the trajectories
3. Normalize the returns using the mean, and std
4. Update the baseline by training it to match the current returns at each of the states visited
5. Update the Policy using the policy gradient and the returns from step 3 offset by the baseline
6. Repeat

In the basic policy gradient algorithm, these two parts only train on the most recent data, can we somehow keep data around from the past like we did with DAgger and keep training on that too?

Policy Gradient

1. Collect data by letting the agent drive in the environment
2. Compute returns from the rewards in the trajectories
3. Normalize the returns using the mean, and std
4. Update the baseline by training it to match the current returns at each of the states visited
5. Update the Policy using the policy gradient and the returns from step 3 offset by the baseline
6. Repeat

In the basic policy gradient algorithm, these two parts operate on the most recent data, can we somehow keep data around from the past like we did with DQN and keep training on that too?

NO!

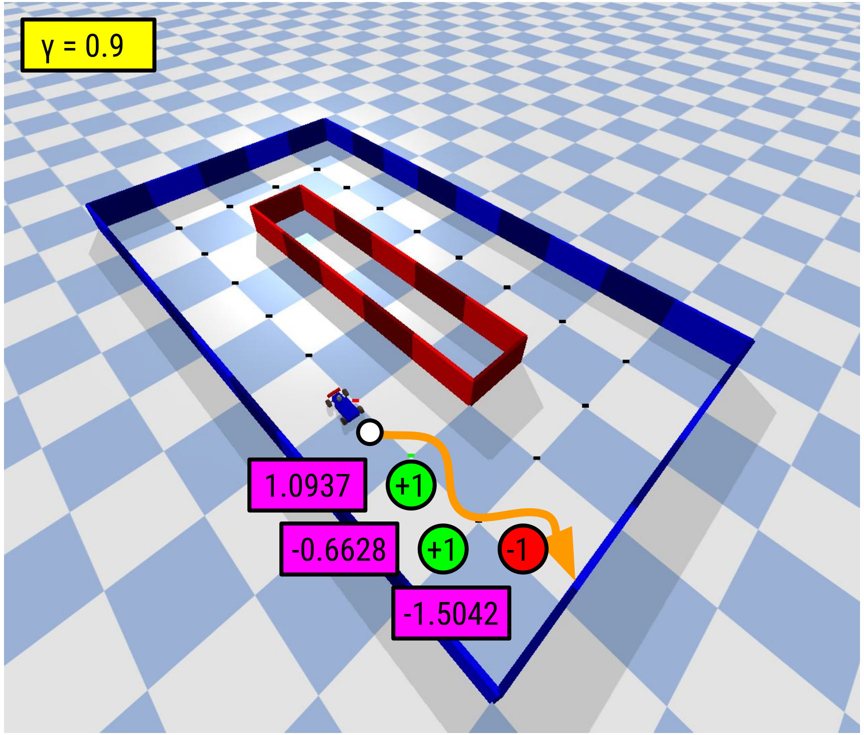
Policy Gradient

Why not?

Policy Gradient

Why not?

Let's consider the trajectory that crashed.

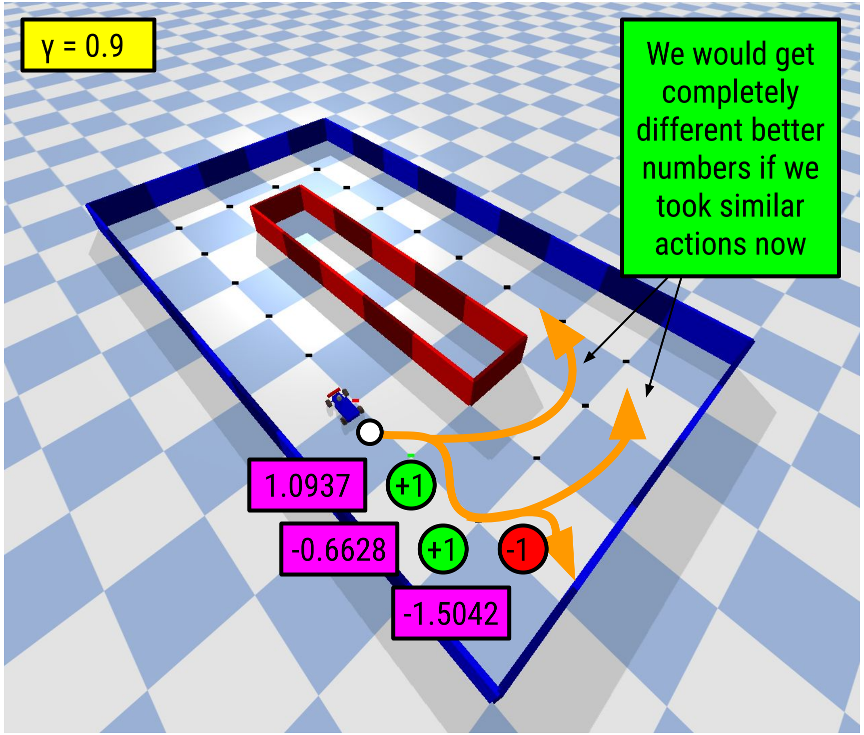


Policy Gradient

Why not?

Let's consider the trajectory that crashed.

And let's assume that after training for a while, we would learn from our mistakes and perform better from these intermediate states.



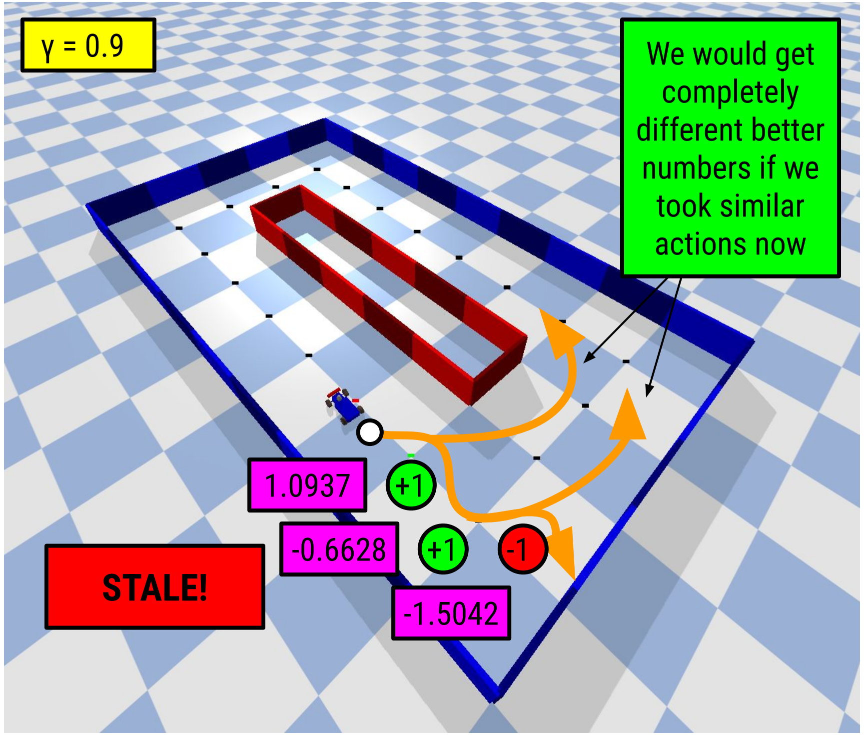
Policy Gradient

Why not?

Let's consider the trajectory that crashed.

And let's assume that after training for a while, we would learn from our mistakes and perform better from these intermediate states.

But if we keep around the old data and keep training on it, the return values no longer reflect how well we would do if we take this action.



Policy Gradient

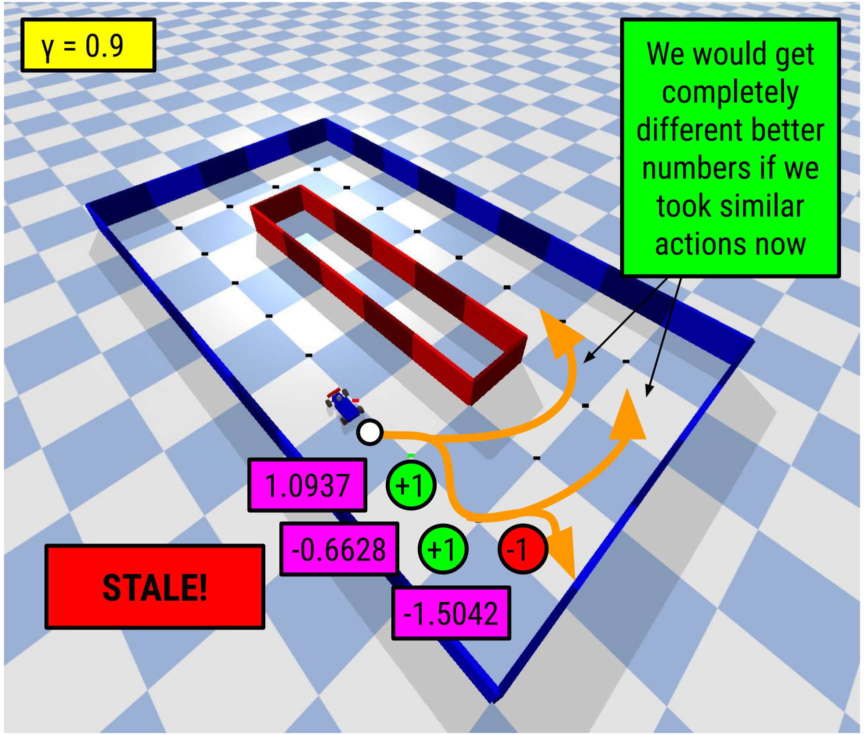
Why not?

Let's consider the trajectory that crashed.

And let's assume that after training for a while, we would learn from our mistakes and perform better from these intermediate states.

But if we keep around the old data and keep training on it, the return values no longer reflect how well we would do if we take this action.

For this reason, we call these algorithms "On Policy" because they only work when training from data generated by the CURRENT policy.



Correlated Data!

Policy Gradient

What is good about this?

Policy Gradient

What is good about this?

- Doesn’t require expert advice!

Policy Gradient

What is good about this?

- Doesn’t require expert advice!
- Can potentially learn a model better than any performance level you’re aware of

Policy Gradient

What is bad about this?

Policy Gradient

What is bad about this?

- So slow!

Policy Gradient

What is bad about this?

- So slow!
    - Only get feedback on one action at a time (scales with size of action space)

Policy Gradient

What is bad about this?

- So slow!
  - Only get feedback on one action at a time (scales with size of action space)
  - Combining feedback from the future may be confusing (depends on horizon)

Policy Gradient

What is bad about this?

- So slow!
    - Only get feedback on one action at a time (scales with size of action space)
    - Combining feedback from the future may be confusing (depends on horizon)
    - Have to constantly throw away your data (reuse data 100x in other settings)

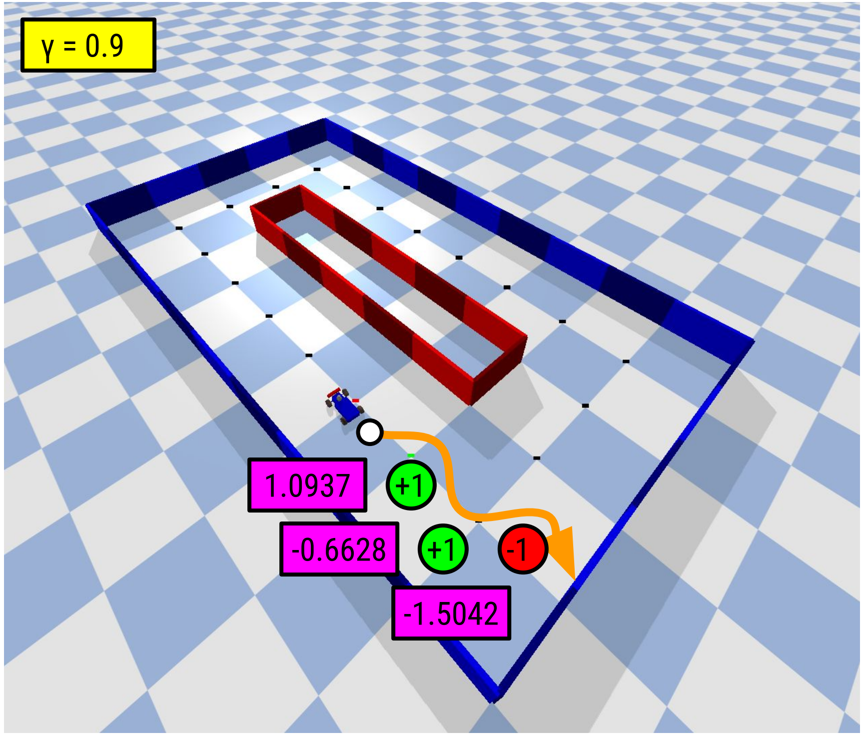
Off-Policy Methods

(DQN, DDPG, SAC)

The Problem With On-Policy Methods

We discovered that we cannot train on old data when using policy gradient.

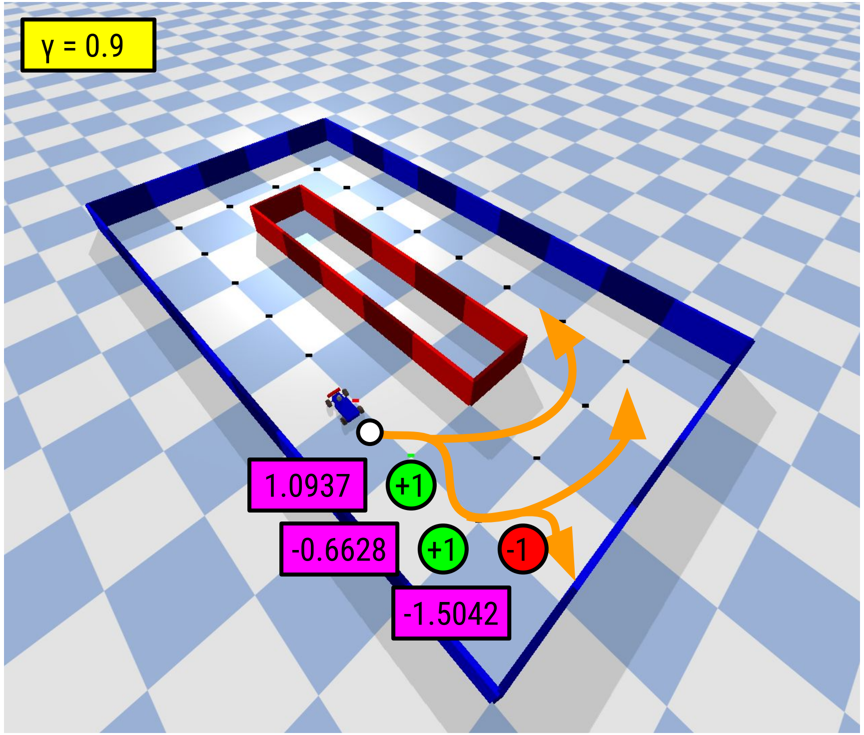
If we have some data from the beginning of training…



The Problem With On-Policy Methods

We discovered that we cannot train on old data when using policy gradient.

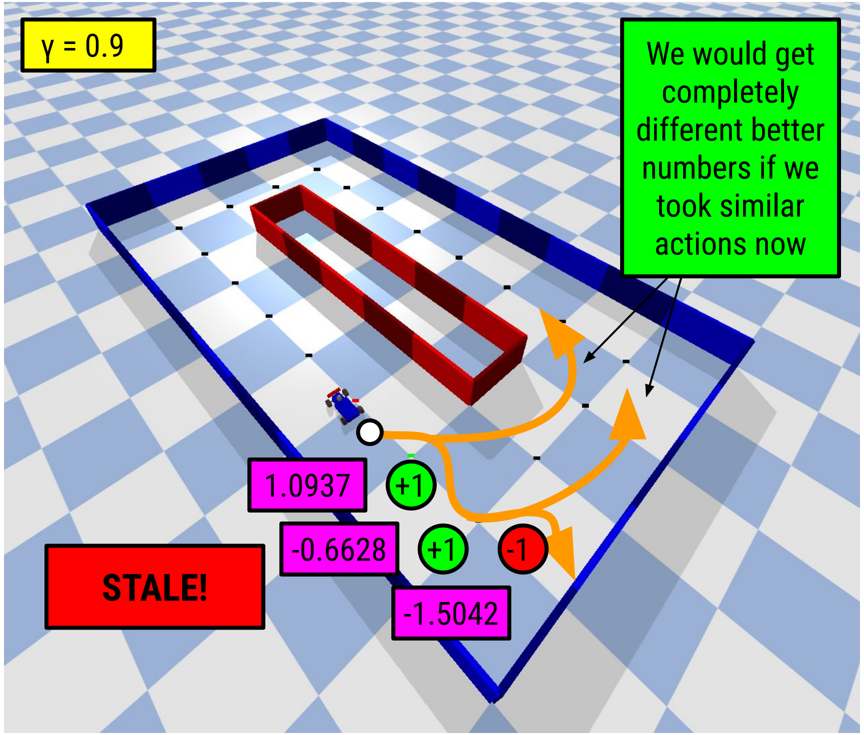
If we have some data from the beginning of training and then improve our performance…



The Problem With On-Policy Methods

We discovered that we cannot train on old data when using policy gradient.

If we have some data from the beginning of training and then improve our performance, the return information that we collected when we gathered the data is no longer relevant. The data is stale.

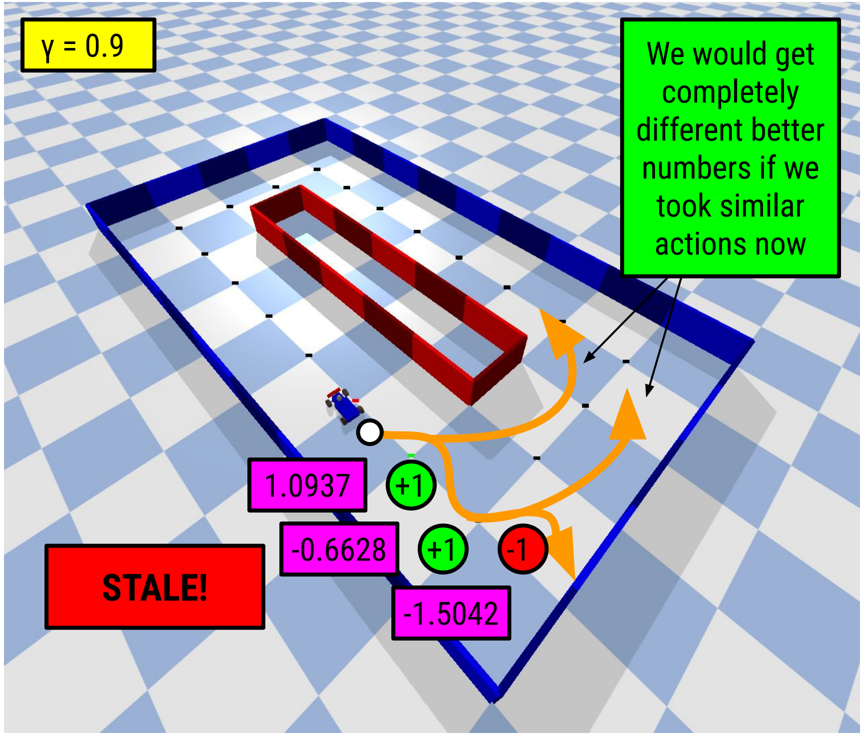


The Problem With On-Policy Methods

We discovered that we cannot train on old data when using policy gradient.

If we have some data from the beginning of training and then improve our performance, the return information that we collected when we gathered the data is no longer relevant. The data is stale.

We said that this is called “On-Policy” because we can only train on data collected with the current policy.



The Problem With On-Policy Methods

Why is this bad?

If your environment is a really fast simulator that's very cheap to run, it's not that bad.

The Problem With On-Policy Methods

Why is this bad?

If your environment is a really fast simulator that's very cheap to run, it's not that bad.

But if your environment is an expensive robot that can break if you do something wrong, then it's a huge burden.

The Problem With On-Policy Methods

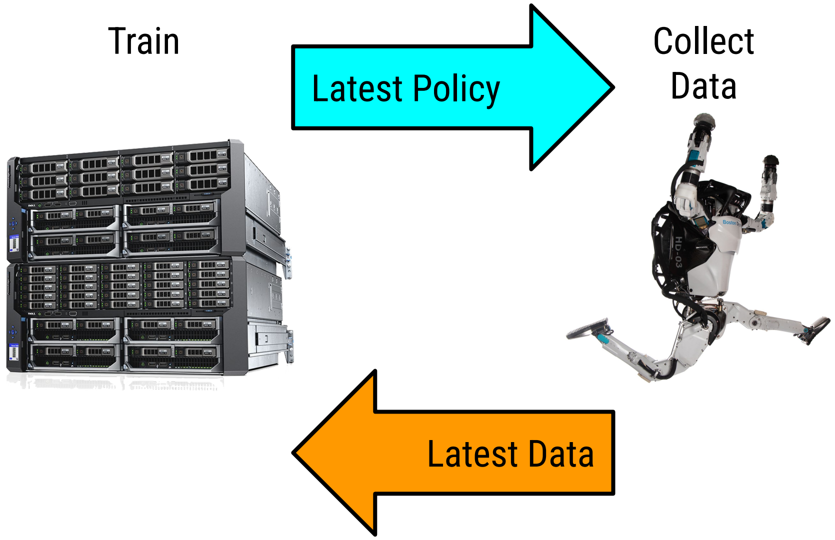
Why is this bad?

If your environment is a really fast simulator that's very cheap to run, it's not that bad.

But if your environment is an expensive robot that can break if you do something wrong, then it's a huge burden.

Your main loop is:

1. Collect data on the robot **(Manual labor!)**
2. Train the robot using the data



The Problem With On-Policy Methods

Why is this bad?

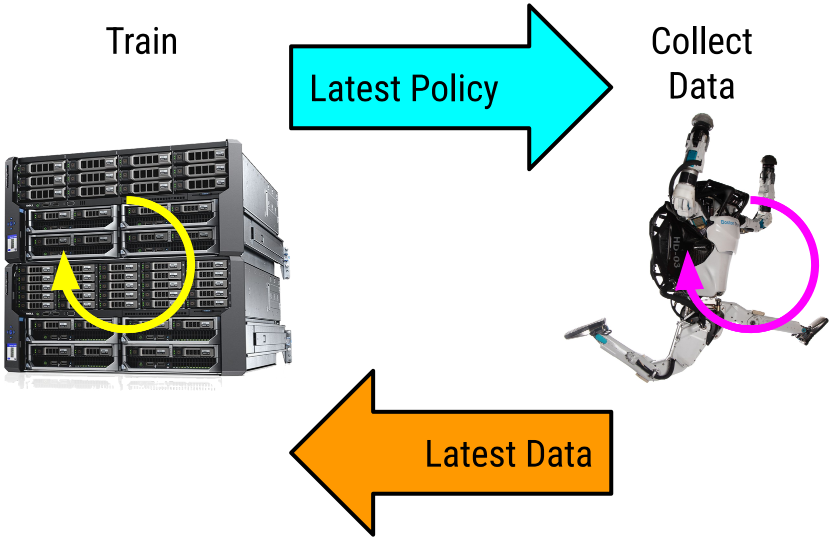
If your environment is a really fast simulator that's very cheap to run, it's not that bad.

But if your environment is an expensive robot that can break if you do something wrong, then it's a huge burden.

Your main loop is:

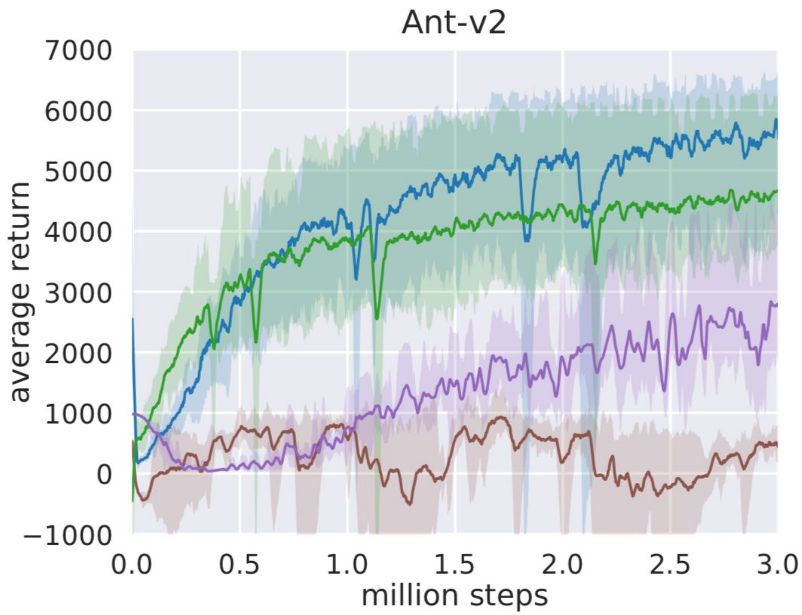
1. Collect data on the robot **(Manual labor!)**
2. Train the robot using the data

It's also very inconvenient if you have to keep switching back and forth frequently.



The Problem With On-Policy Methods

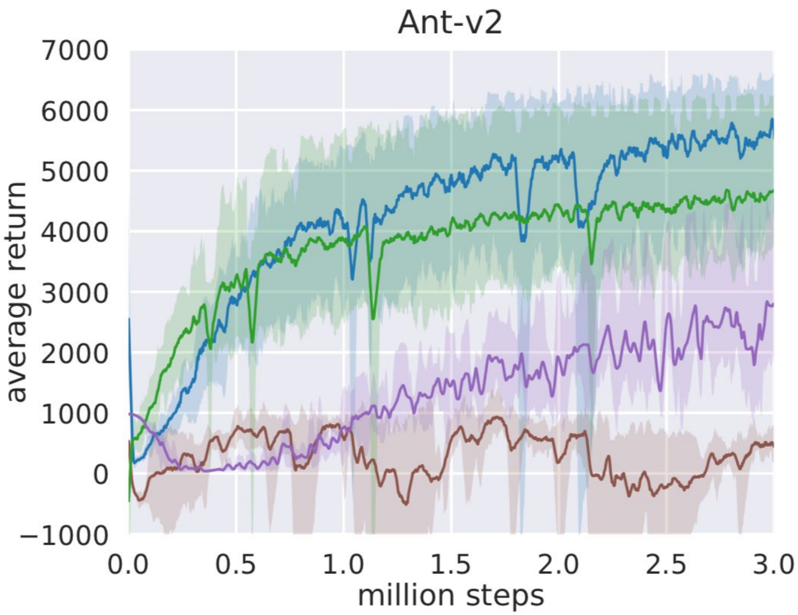
It is common to measure the performance of RL algorithms using “Sample Complexity” or the amount of interactions you have need to have with an environment in order to reach a certain performance level.



The Problem With On-Policy Methods

It is common to measure the performance of RL algorithms using "Sample Complexity" or the amount of interactions you have need to have with an environment in order to reach a certain performance level.

On-Policy methods usually have very high Sample Complexity because you need to interact with the environment every time you want to improve your model.

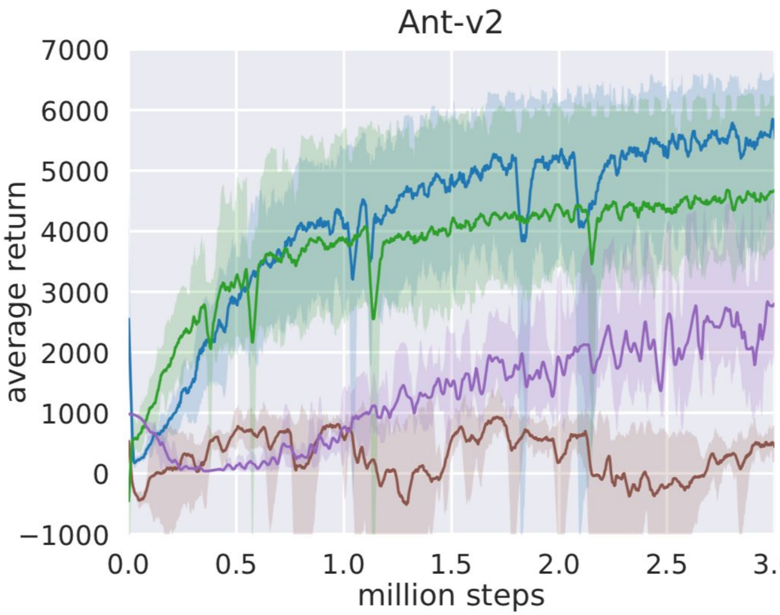


The Problem With On-Policy Methods

It is common to measure the performance of RL algorithms using “Sample Complexity” or the amount of interactions you have need to have with an environment in order to reach a certain performance level.

On-Policy methods usually have very high Sample Complexity because you need to interact with the environment every time you want to improve your model.

We can also measure how many training steps we need, but in almost all applications, training steps are much cheaper than interacting with the environment to collect data.



The x-axis is environment steps, not training steps!

What We Would Like

Program Sketch:

What We Would Like

Program Sketch:

1. initialize an empty dataset

What We Would Like

Program Sketch:

1. initialize an empty dataset
2. for some number of rounds:

What We Would Like

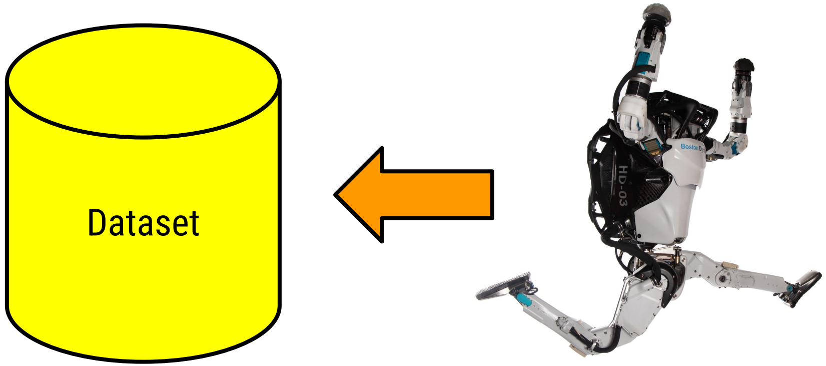
Program Sketch:

1. initialize an empty dataset
2. for some number of rounds:
    a. for m steps:
        i. Do one step of interaction with the environment

What We Would Like

Program Sketch:

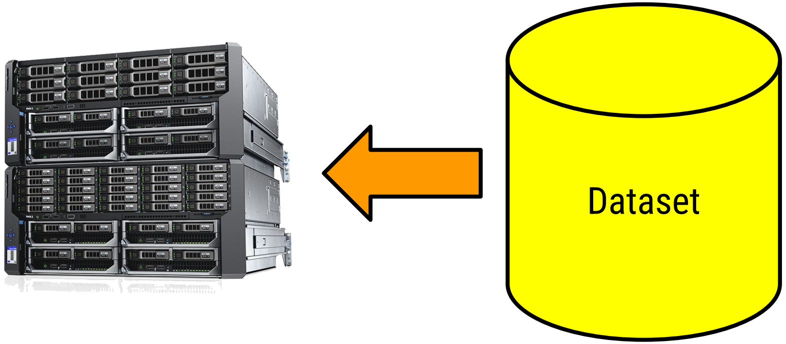
1. initialize an empty dataset
2. for some number of rounds:
    a. for m steps:
        i. Do one step of interaction with the environment
    b. Add the data to a growing dataset (like DAgger)



What We Would Like

Program Sketch:

1. initialize an empty dataset
2. for some number of rounds:
    a. for m steps:
        i. Do one step of interaction with the environment
    b. Add the data to a growing dataset (like DAgger)
    c. for n steps:
        i. Do one training step on a randomly sampled batch from the dataset

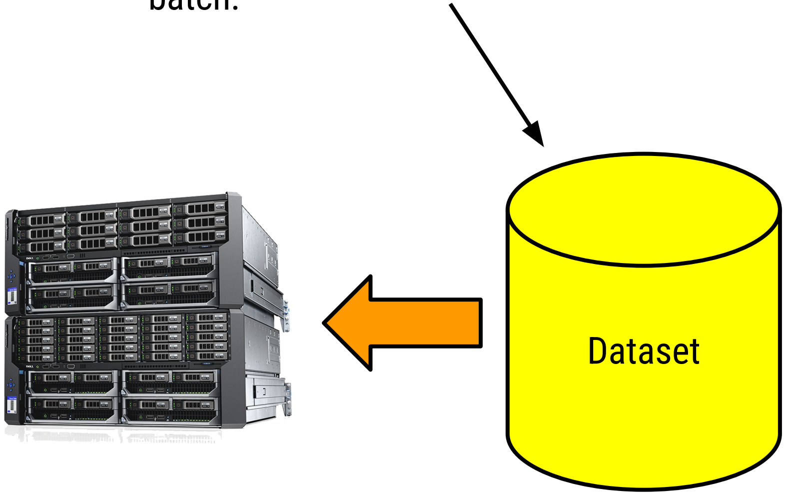


What We Would Like

Program Sketch:

1. initialize an empty dataset
2. for some number of rounds:
   a. for m steps:
      i. Do one step of interaction with the environment
   b. Add the data to a growing dataset (like DAgger)
   c. for n steps:
      i. Do one training step on a randomly sampled batch from the dataset

Saving old data into a large dataset and sampling random batches has the additional advantage of providing data diversity in each batch.



What We Would Like

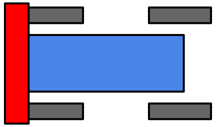
Program Sketch:

1. ~~initialize an empty dataset~~
2. for some number of rounds:
   a. for m steps:
      i. Do one step of interaction with the environment
   b. ~~Add the data to a growing dataset (like DAgger)~~
   c. for n steps:
      i. Do one training step on a ~~randomly sampled batch from the dataset~~

And just to reiterate, in Policy Gradient and other On-Policy methods, we can't store a large dataset and have to train only on the most recent data.

How do we get what we want?

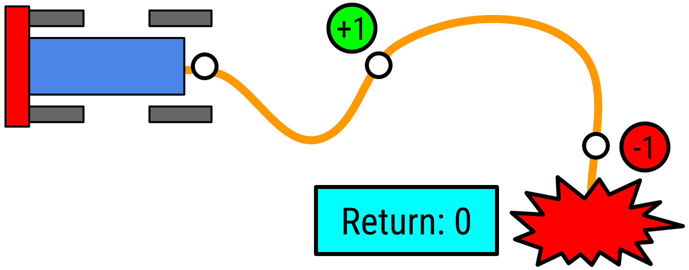
Let's unpack our previous illustration:



How do we get what we want?

Let's unpack our previous illustration:

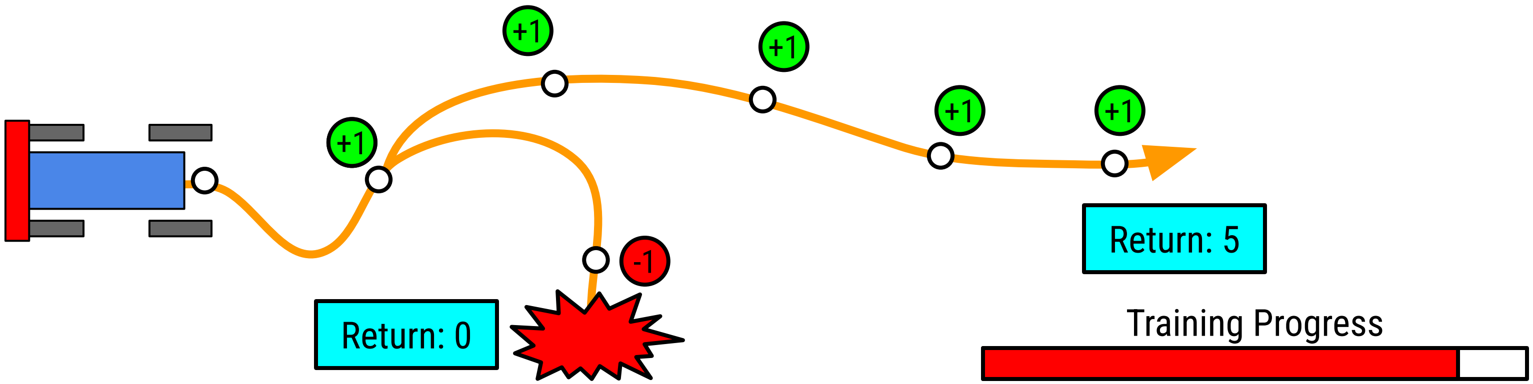
We showed that the returns captured early on…



How do we get what we want?

Let's unpack our previous illustration:

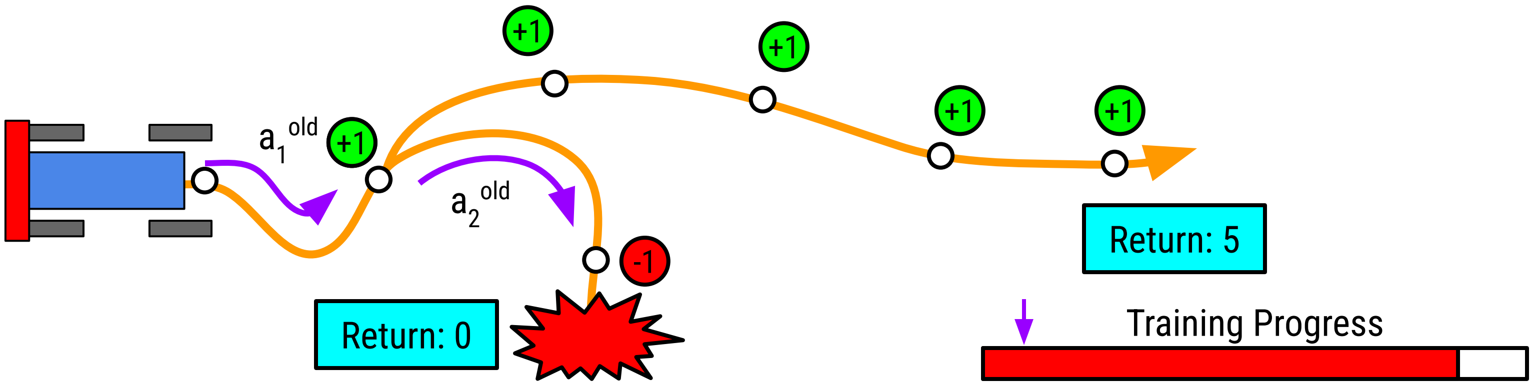
We showed that the returns captured early on may not reflect the returns we will see after our policy has improved if we were to visit a similar state.



How do we get what we want?

Let's unpack our previous illustration:

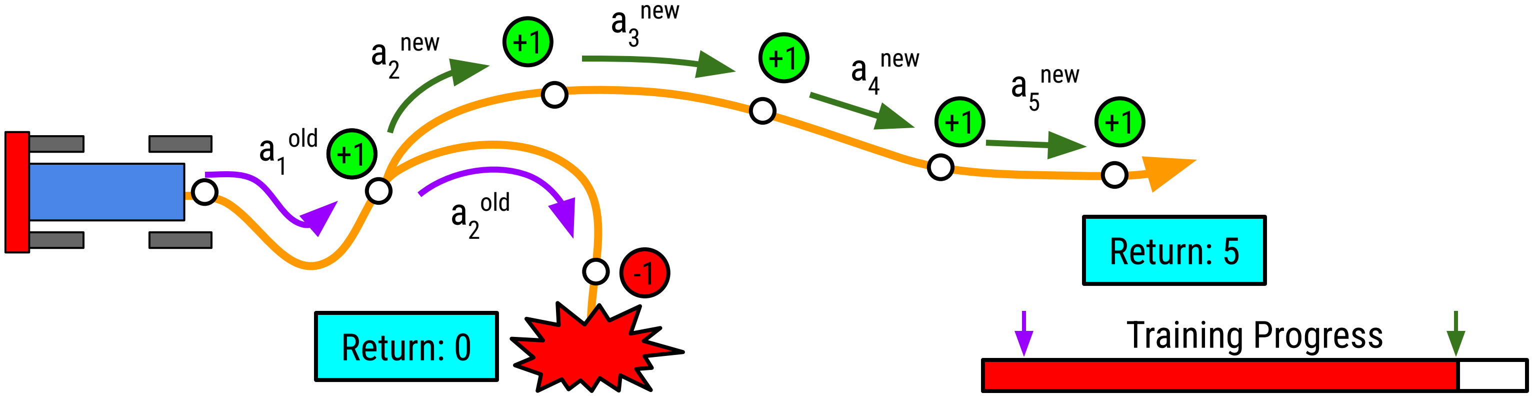
Let's label the actions that we took originally as a_i^{old}...



How do we get what we want?

Let's unpack our previous illustration:

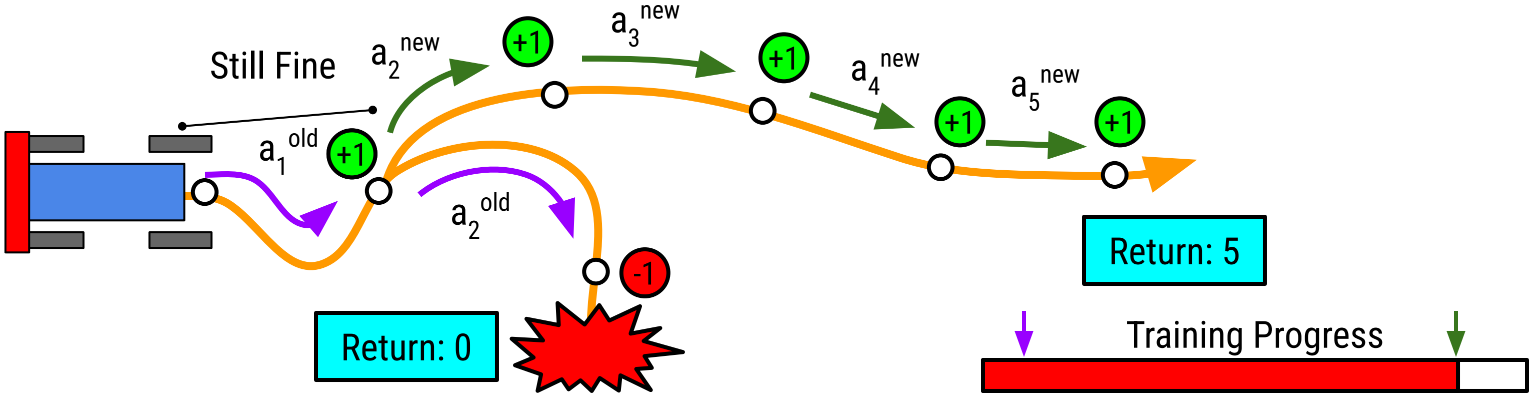
Let's label the actions that we took originally as a_i^{old} and the actions we would take in the future as a_i^{new}.



How do we get what we want?

Let's unpack our previous illustration:

Next note that the first step of this experience is actually still fine. If I take action a_1^{old} the reward that I originally got does not depend on the shift in policy distributions.

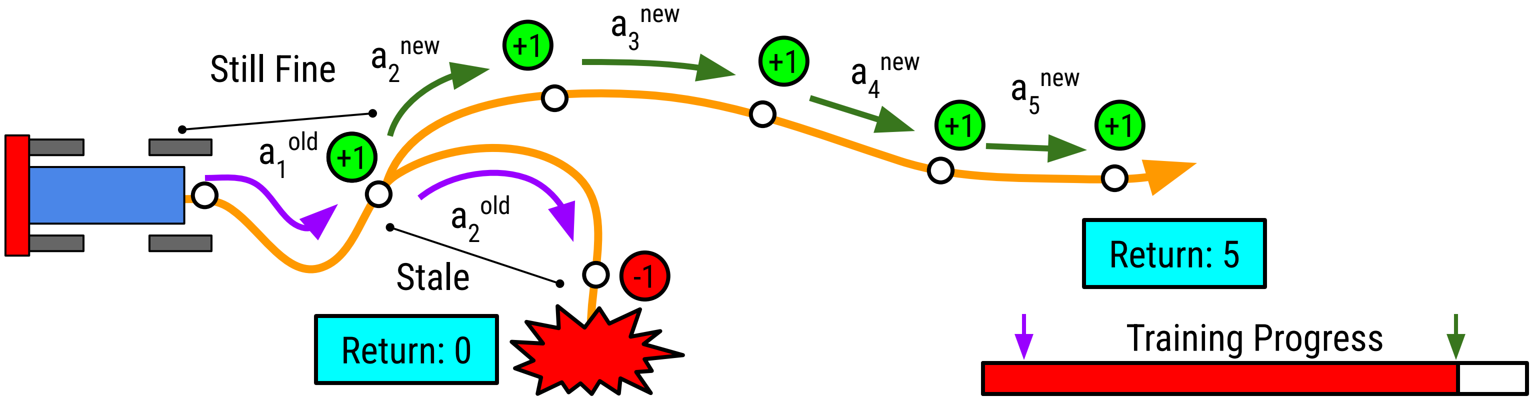


How do we get what we want?

Let's unpack our previous illustration:

Next note that the first step of this experience is actually still fine. If I take action a_1^{old} the <u>reward</u> that I originally got does not depend on the shift in policy distributions.

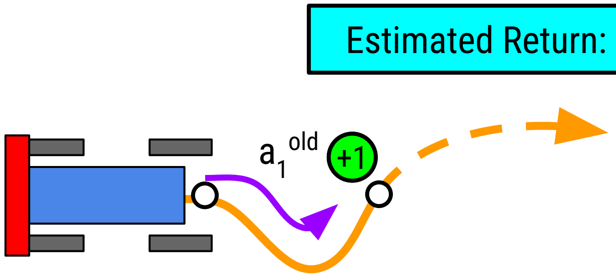
It might be true that even the first step would be different under the new policy, BUT if I take the a_1^{old} I can expect similar results to what I saw last time. The problem is with the part that comes afterward.



How do we get what we want?

Let’s unpack our previous illustration:

But what if we replaced the empirical returns with an estimate of my value in this second state?

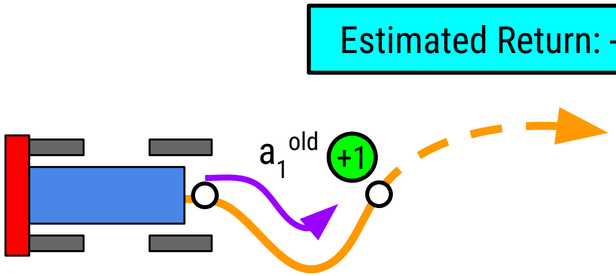


How do we get what we want?

Let’s unpack our previous illustration:

But what if we replaced the empirical returns with an estimate of my value in this second state?

If that estimate is -1 early in training…

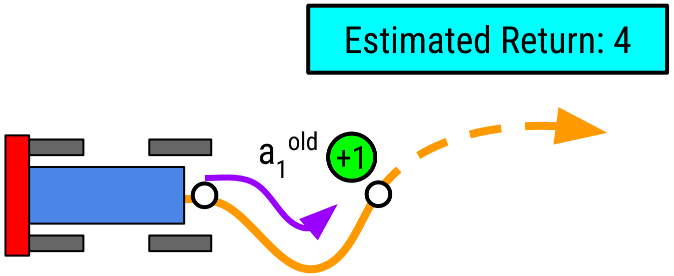


How do we get what we want?

Let's unpack our previous illustration:

But what if we replaced the empirical returns with an estimate of my value in this second state?

If that estimate is -1 early in training, but 4 later then we can train on $r + \gamma$ estimate(s_{i+1}) and everything is fine.



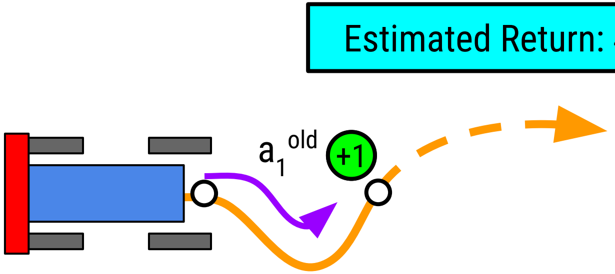
How do we get what we want?

Let’s unpack our previous illustration:

But what if we replaced the empirical returns with an estimate of my value in this second state?

If that estimate is -1 early in training, but 4 later then we can train on $r + \gamma \text{ estimate}(s_{i+1})$ and everything is fine.

So what we can do is keep the old data around, but use NEW estimates of future return.



How do we get what we want?

We could use these estimated returns ($r + \gamma$ estimate(s_{i+1})) in a policy gradient framework, which would lead to something like the actor-critic framework we talked about last time. We're actually going to go a bit further, but to do so, we need some new tools.

How do we get what we want?

We could use these estimated returns ($r + \gamma$ estimate(s_{i+1})) in a policy gradient framework, which would lead to something like the actor-critic framework we talked about last time. We're actually going to go a bit further, but to do so, we need some new tools.

Also, I am so sorry guys, I really tried to avoid this, but I'm going to pass along some generational trauma in the form of grinding through some math over the next few slides. This is how most people teach RL and I hate it, but it's kind of necessary to get where we need to be.

How do we get what we want?

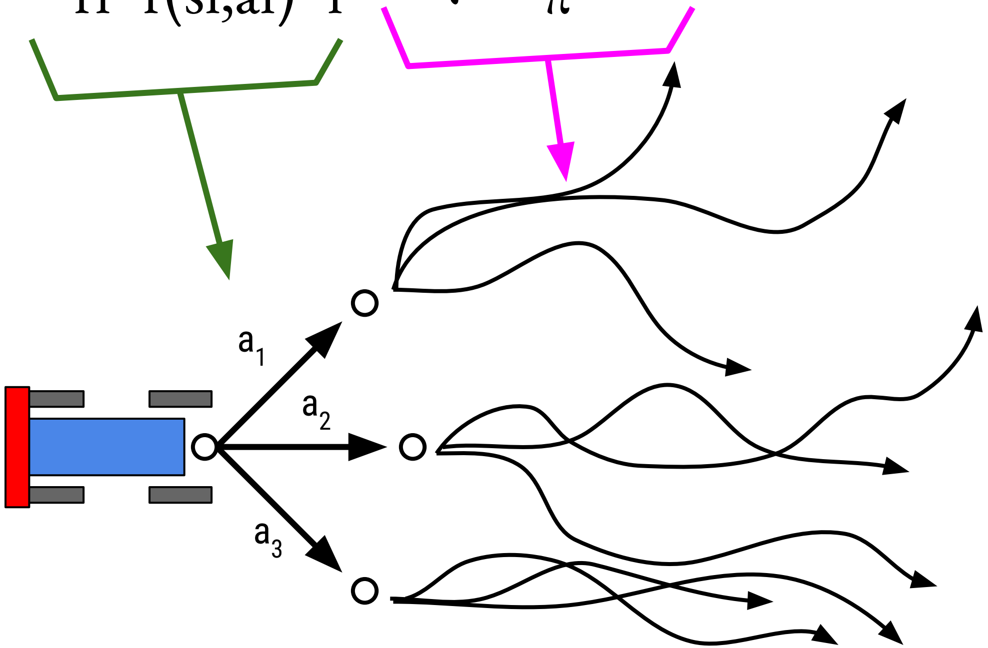
New Tool 1:

1. policy (π) : some agent capable of acting in the environment. Often written as π_θ when it is a network with parameters θ.
2. states/observations (s_i or x_i or occasionally o_i) : the states or observations used to make decisions at step i.
3. actions (a_i or u_i) : the actions an agent takes at step i
4. reward (r_i) : the reward returned from the environment at step i
5. discount (γ) : a scalar constant describing how much we care about short term vs. long term reward
6. return ($g_i = \sum_{t=i...T} \gamma^{(t-i)} r_t$) : the discounted empirical sum of future rewards after taking an action
7. value ($v_\pi(s_i) = E_{ai...T\sim\pi}\, g_i$) : the expected return of being in state s_i and acting using the policy π until the end of an episode
8. action value ($q_\pi(s_i , a_i) = \mathbf{E}_{ri\sim r(si,ai)} r_i + \gamma\, v_\pi(s_{i+1})$): the expected value of taking action a_i in state s_i then following π until the end

$$q_\pi(s_i , a_i) = \mathbf{E}_{ri\sim r(si,ai)} r_i + \gamma\, v_\pi (s_{i+1})$$

How do we get what we want?

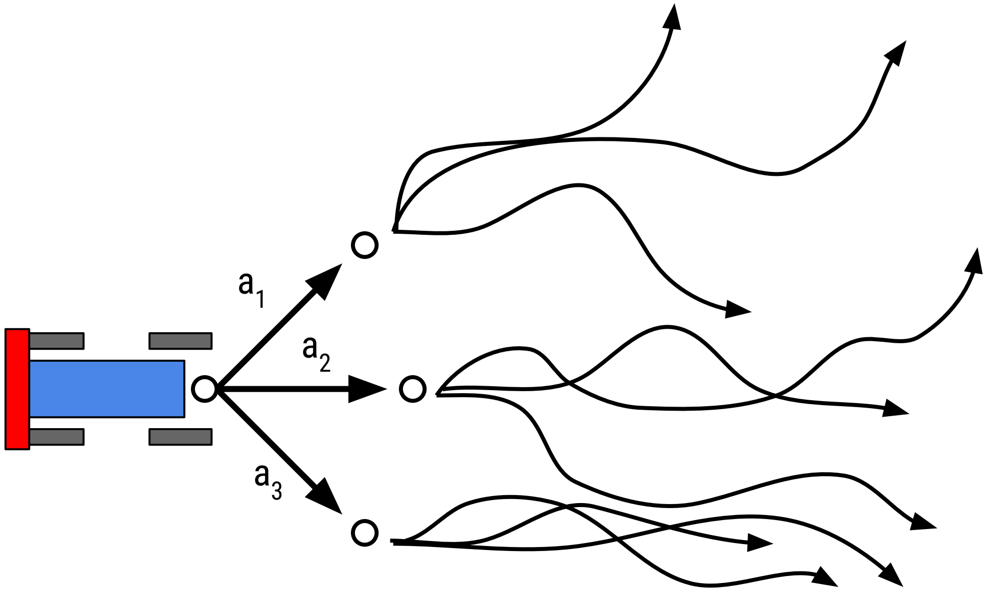
$$q_\pi(s_i, a_i) = \mathbf{E}_{ri\sim r(si,ai)} r_i + \gamma v_\pi(s_{i+1})$$



How do we get what we want?

$$q_\pi(s_i, a_i) = \mathbf{E}_{ri \sim r(si,ai)} r_i + \gamma v_\pi(s_{i+1})$$

In our next algorithm DQN, we're going to be learning this



How do we get what we want?

New Tool 2: Bellman Equation:

$v_\pi(s_i) = \mathbf{E}_{ai...T\sim\pi}\, g_i$ ← expand

How do we get what we want?

New Tool 2: Bellman Equation:

$v_\pi(s_i) = \mathbf{E}_{ai\ldots T\sim\pi}\, g_i$ ← expand

$v_\pi(s_i) = \mathbf{E}_{ai\ldots T\sim\pi}\, r_i + \gamma\, r_{i+1} + \gamma^2\, r_{i+2} \ldots$ ← Factor out γ

How do we get what we want?

New Tool 2: Bellman Equation:

$v_\pi(s_i) = \mathbf{E}_{ai\ldots T\sim\pi}\, g_i$ ← expand

$v_\pi(s_i) = \mathbf{E}_{ai\ldots T\sim\pi}\, r_i + \gamma\, r_{i+1} + \gamma^2\, r_{i+2} \ldots$ ← Factor out γ

$v_\pi(s_i) = \mathbf{E}_{ai\ldots T\sim\pi}\, r_i + \gamma\, (r_{i+1} + \gamma\, r_{i+2} \ldots)$ ← Substitute

How do we get what we want?

New Tool 2: Bellman Equation:

$v_\pi(s_i) = \mathbf{E}_{a_i \ldots T \sim \pi}\, g_i$ ← expand

$v_\pi(s_i) = \mathbf{E}_{a_i \ldots T \sim \pi}\, r_i + \gamma\, r_{i+1} + \gamma^2\, r_{i+2} \ldots$ ← Factor out γ

$v_\pi(s_i) = \mathbf{E}_{a_i \ldots T \sim \pi}\, r_i + \gamma\, (r_{i+1} + \gamma\, r_{i+2} \ldots)$ ← Substitute

$v_\pi(s_i) = \mathbf{E}_{a_i \ldots T \sim \pi}\, r_i + \gamma\, g_{i+1}$ ← Separate the Expectation

How do we get what we want?

New Tool 2: Bellman Equation:

$v_\pi(s_i) = \mathbf{E}_{ai...T\sim\pi}\, g_i$ ← expand

$v_\pi(s_i) = \mathbf{E}_{ai...T\sim\pi}\, r_i + \gamma\, r_{i+1} + \gamma^2\, r_{i+2} ...$ ← Factor out γ

$v_\pi(s_i) = \mathbf{E}_{ai...T\sim\pi}\, r_i + \gamma\, (r_{i+1} + \gamma\, r_{i+2} ...)$ ← Substitute

$v_\pi(s_i) = \mathbf{E}_{ai...T\sim\pi}\, r_i + \gamma\, g_{i+1}$ ← Separate the Expectation

$v_\pi(s_i) = \mathbf{E}_{ai\sim\pi}\, r_i + \mathbf{E}_{ai+1...T}\, \gamma\, g_{i+1}$ ← Move γ outside the expectation

How do we get what we want?

New Tool 2: Bellman Equation:

$v_\pi(s_i) = \mathbf{E}_{ai...T\sim\pi}\, g_i$ ← expand

$v_\pi(s_i) = \mathbf{E}_{ai...T\sim\pi}\, r_i + \gamma\, r_{i+1} + \gamma^2\, r_{i+2} \ldots$ ← Factor out γ

$v_\pi(s_i) = \mathbf{E}_{ai...T\sim\pi}\, r_i + \gamma\, (r_{i+1} + \gamma\, r_{i+2} \ldots)$ ← Substitute

$v_\pi(s_i) = \mathbf{E}_{ai...T\sim\pi}\, r_i + \gamma\, g_{i+1}$ ← Separate the Expectation

$v_\pi(s_i) = \mathbf{E}_{ai\sim\pi}\, r_i + \mathbf{E}_{ai+1...T}\, \gamma\, g_{i+1}$ ← Move γ outside the expectation

$v_\pi(s_i) = \mathbf{E}_{ai\sim\pi}\, r_i + \gamma \mathbf{E}_{ai+1...T}\, g_{i+1}$ ← Substitute

How do we get what we want?

New Tool 2: Bellman Equation:

$$v_\pi(s_i) = \mathbf{E}_{ai...T\sim\pi}\, g_i$$

$$v_\pi(s_i) = \mathbf{E}_{ai...T\sim\pi}\, r_i + \gamma\, r_{i+1} + \gamma^2\, r_{i+2} \dots$$

$$v_\pi(s_i) = \mathbf{E}_{ai...T\sim\pi}\, r_i + \gamma\, (r_{i+1} + \gamma\, r_{i+2} \dots)$$

$$v_\pi(s_i) = \mathbf{E}_{ai...T\sim\pi}\, r_i + \gamma\, g_{i+1}$$

$$v_\pi(s_i) = \mathbf{E}_{ai\sim\pi}\, r_i + \mathbf{E}_{ai+1...T}\gamma\, g_{i+1}$$

$$v_\pi(s_i) = \mathbf{E}_{ai\sim\pi}\, r_i + \gamma\mathbf{E}_{ai+1...T}\, g_{i+1}$$

$$v_\pi(s_i) = \mathbf{E}_{ai\sim\pi}\, r_i + \gamma\, v_\pi(s_{i+1})$$ ← The Bellman Equation

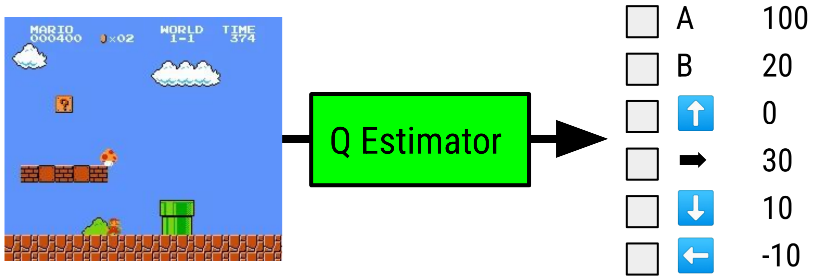
$$v_\pi(s_i) = \mathbf{E}_{ai\sim\pi}\, r_i + \gamma\, v_\pi(s_{i+1})$$

How do we get what we want?

Let's say we're trying to play Mario, and we already have a really good Q estimator.

$$q_\pi(s_i,a_i) = \mathbf{E}_{ri\sim r(si,ai)} r_i + \gamma\, v_\pi(s_{i+1})$$

Our Q estimator takes an observation in and produces q values for all possible actions.

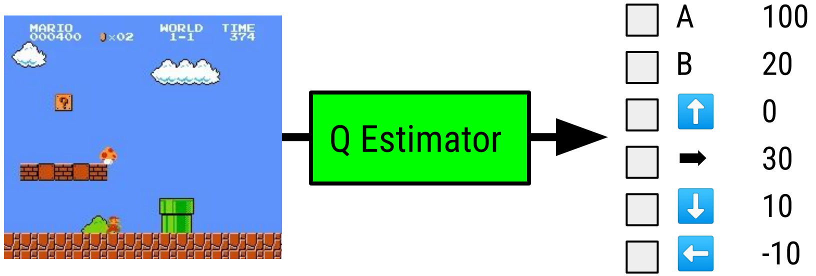


How do we get what we want?

Let's say we're trying to play Mario, and we already have a really good Q estimator.

$$q_\pi(s_i,a_i) = \mathbf{E}_{ri\sim r(si,ai)} r_i + \gamma\, v_\pi(s_{i+1})$$

If I have this information, how should I act? (What should my policy be?)



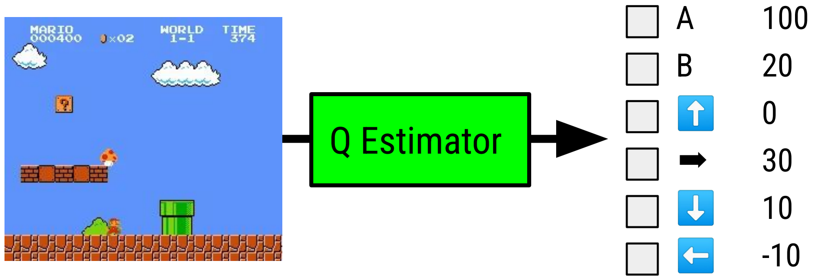
How do we get what we want?

Let's say we're trying to play Mario, and we already have a really good Q estimator.

$$q_\pi(s_i,a_i) = \mathbf{E}_{ri\sim r(si,ai)} r_i + \gamma \, v_\pi(s_{i+1})$$

If I have this information, how should I act? (What should my policy be?)

$$\pi(s_i) = \text{argmax}_{ai}(q_\pi(s_i, a_i))$$



How do we get what we want?

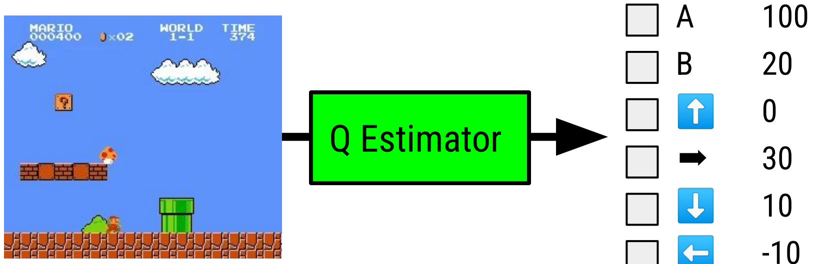
Let's say we're trying to play Mario, and we already have a really good Q estimator.

$$q_\pi(s_i,a_i) = \mathbf{E}_{ri\sim r(si,ai)} r_i + \gamma\, v_\pi(s_{i+1})$$

If I have this information, how should I act? (What should my policy be?)

$$\pi(s_i) = \text{argmax}_{ai}(q_\pi(s_i, a_i))$$

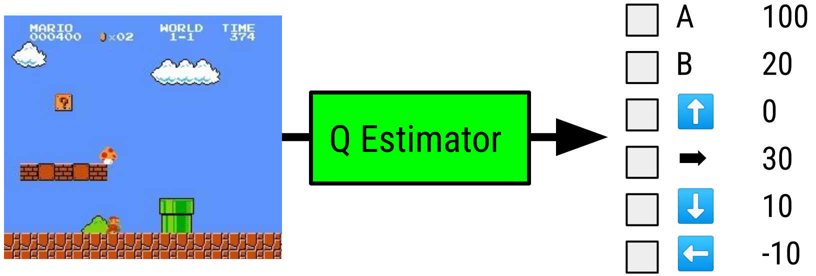
So if my policy is to take the maximum q, what is $v_\pi(s_{i+1}) = \mathbf{E}_{ai+1\ldots T\sim\pi}\, g_{i+1}$?



How do we get what we want?

Let's say we're trying to play Mario, and we already have a really good Q estimator.

$q_\pi(s_i,a_i) = \mathbf{E}_{r_i \sim r(s_i,a_i)} r_i + \gamma\, v_\pi(s_{i+1})$



If I have this information, how should I act? (What should my policy be?)

$\pi(s_i) = \text{argmax}_{a_i}(q_\pi(s_i, a_i))$

So if my policy is to take the maximum q, what is $v_\pi(s_{i+1}) = \mathbf{E}_{a_{i+1 \ldots T} \sim \pi}\, g_{i+1}$?

$v_\pi(s_{i+1}) = \mathbf{E}_{a_{i+1 \ldots T} \sim \pi}\, g_{i+1}$

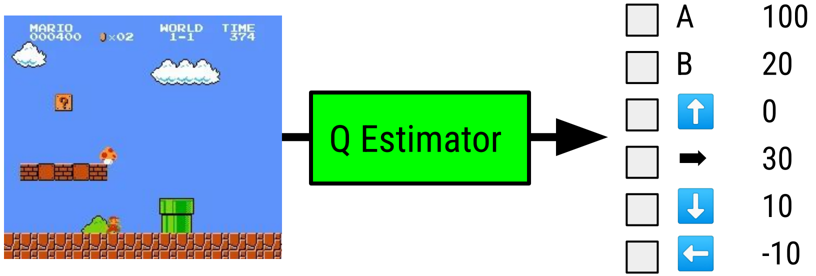
Bellman

$v_\pi(s_{i+1}) = \mathbf{E}_{a_{i+1} \sim \pi}\, r_{i+1} + \gamma\, v_\pi(s_{i+2})$

Expand Expectation

How do we get what we want?

Let's say we're trying to play Mario, and we already have a really good Q estimator.



$q_\pi(s_i,a_i) = \mathbf{E}_{ri\sim r(si,ai)} r_i + \gamma\, v_\pi(s_{i+1})$

If I have this information, how should I act? (What should my policy be?)

$\pi(s_i) = \mathrm{argmax}_{ai}(q_\pi(s_i, a_i))$

So if my policy is to take the maximum q, what is $v_\pi(s_{i+1}) = \mathbf{E}_{ai+1\ldots T\sim\pi}\, g_{i+1}$?

$v_\pi(s_{i+1}) = \mathbf{E}_{ai+1\ldots T\sim\pi}\, g_{i+1}$

Bellman

$v_\pi(s_{i+1}) = \mathbf{E}_{ai+1\sim\pi}\, r_{i+1} + \gamma\, v_\pi(s_{i+2})$

Expand Expectation

$v_\pi(s_{i+1}) = \Sigma_{ai+1}\pi(a_{i+1})\, \mathbf{E}_{ri+1\sim r(si+1,ai+1)} r_{i+1} + \gamma\, v_\pi(s_{i+2})$

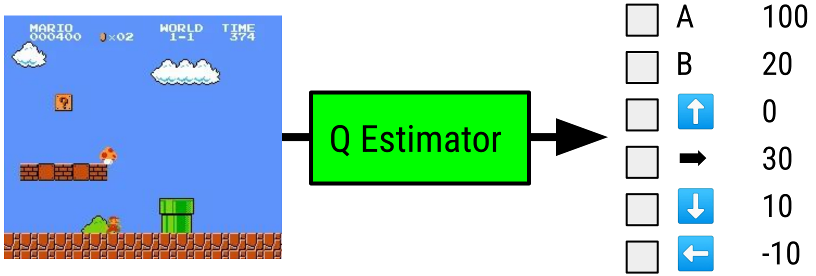
Sub q

How do we get what we want?

Let's say we're trying to play Mario, and we already have a really good Q estimator.

$$q_\pi(s_i,a_i) = \mathbf{E}_{r_i \sim r(s_i,a_i)} r_i + \gamma\, v_\pi(s_{i+1})$$

If I have this information, how should I act? (What should my policy be?)



$$\pi(s_i) = \text{argmax}_{a_i}(q_\pi(s_i, a_i))$$

So if my policy is to take the maximum q, what is $v_\pi(s_{i+1}) = \mathbf{E}_{a_{i+1} \ldots T \sim \pi}\, g_{i+1}$?

$$v_\pi(s_{i+1}) = \mathbf{E}_{a_{i+1} \ldots T \sim \pi}\, g_{i+1}$$

Bellman

$$v_\pi(s_{i+1}) = \mathbf{E}_{a_{i+1} \sim \pi}\, r_{i+1} + \gamma\, v_\pi(s_{i+2})$$

Expand Expectation

$$v_\pi(s_{i+1}) = \Sigma_{a_{i+1}} \pi(a_{i+1})\, \mathbf{E}_{r_{i+1} \sim r(s_{i+1},a_{i+1})} r_{i+1} + \gamma\, v_\pi(s_{i+2})$$

Sub q

$$v_\pi(s_{i+1}) = \Sigma_{a_{i+1}} \pi(a_{i+1})\, q_\pi(s_{i+1},a_{i+1})$$

Deterministic

How do we get what we want?

Let's say we're trying to play Mario, and we already have a really good Q estimator.

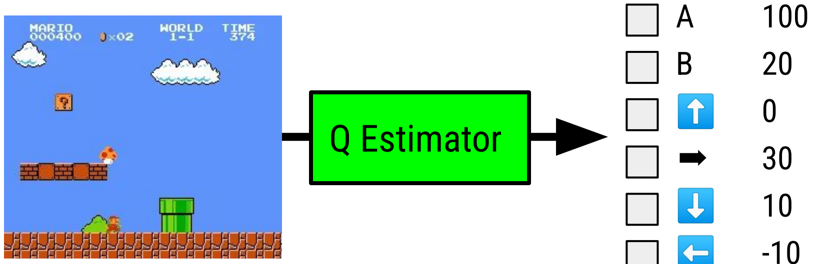
$q_\pi(s_i,a_i) = \mathbf{E}_{r_i \sim r(s_i,a_i)} r_i + \gamma\, v_\pi(s_{i+1})$

If I have this information, how should I act? (What should my policy be?)

$\pi(s_i) = \mathrm{argmax}_{a_i}(q_\pi(s_i, a_i))$

So if my policy is to take the maximum q, what is $v_\pi(s_{i+1}) = \mathbf{E}_{a_{i+1 \ldots T} \sim \pi}\, g_{i+1}$?

$v_\pi(s_{i+1}) = \mathbf{E}_{a_{i+1 \ldots T} \sim \pi}\, g_{i+1}$



Bellman

$v_\pi(s_{i+1}) = \mathbf{E}_{a_{i+1} \sim \pi}\, r_{i+1} + \gamma\, v_\pi(s_{i+2})$ ← Expand Expectation

$v_\pi(s_{i+1}) = \Sigma_{a_{i+1}} \pi(a_{i+1})\ \mathbf{E}_{r_{i+1} \sim r(s_{i+1},a_{i+1})} r_{i+1} + \gamma\, v_\pi(s_{i+2})$

Sub q

$v_\pi(s_{i+1}) = \Sigma_{a_{i+1}} \pi(a_{i+1})\ q_\pi(s_{i+1},a_{i+1})$

Deterministic

$v_\pi(s_{i+1}) = \max_{a_{i+1}} q_\pi(s_{i+1}, a_{i+1})$

How do we get what we want?

$q_\pi(s_i,a_i) = \mathbf{E}_{ri\sim r(si,ai)} r_i + \gamma\, v_\pi(s_{i+1})$ ← From our definition

$v_\pi(s_{i+1}) = \max_{ai+1} q_\pi(s_{i+1},a_{i+1})$ ← We just showed

How do we get what we want?

$q_\pi(s_i,a_i) = \mathbf{E}_{ri\sim r(si,ai)} r_i + \gamma\, v_\pi(s_{i+1})$ ← From our definition

$v_\pi(s_{i+1}) = \max_{ai+1} q_\pi(s_{i+1},a_{i+1})$ ← We just showed

$q_\pi(s_i,a_i) = \mathbf{E}_{ri\sim r(si,ai)} r_i + \gamma \max_{ai+1} q_\pi(s_{i+1},a_{i+1})$ ← Substitution

Now we have a recursive definition of Q.

What if our Q function is bad and we want to improve it?

How do we get what we want?

$$q_\pi(s_i,a_i) = \mathbf{E}_{ri\sim r(si,ai)} r_i + \gamma \max_{ai+1} q_\pi(s_{i+1},a_{i+1})$$

Now we have a recursive definition of Q.

What if our Q function is bad and we want to improve it?

How do we get what we want?

$q_\pi(s_i,a_i) = \mathbf{E}_{ri\sim r(si,ai)} r_i + \gamma \max_{ai+1} q_\pi(s_{i+1},a_{i+1})$

Now we have a recursive definition of Q.

What if our Q function is bad and we want to improve it?

We can act in the environment and collect (s_i,a_i,r_i,s_{i+1}) tuples for each step.

How do we get what we want?

$$q_\pi(s_i,a_i) = \mathbf{E}_{ri\sim r(si,ai)} r_i + \gamma \max_{ai+1} q_\pi(s_{i+1},a_{i+1})$$

Now we have a recursive definition of Q.
What if our Q function is bad and we want to improve it?

We can act in the environment and collect (s_i,a_i,r_i,s_{i+1}) tuples for each step.

Then we update $q_\pi(s_i,a_i)$ in the direction of:

$$q_\pi(s_i,a_i) := r_i + \gamma \max_{ai+1} q_\pi(s_{i+1},a_{i+1}) \longleftarrow$$

Our current best estimate of the future

Finally we got what we want!

DQN:

1. initialize an empty dataset
2. for some number of rounds:
    a. for m steps:
        i. Do one step of interaction with the environment using ε-greedy
    b. Add (s,a,r,s') to a growing dataset (like DAgger)
    c. for n steps:
        i. Do one training step on a randomly sampled batch from the dataset according to:

$$q_\pi(s_i,a_i) := r_i + \gamma \max_{ai+1} q_\pi(s_{i+1},a_{i+1})$$

ε-greedy:

- Sample a random number between 0 and 1.
- If the number is less than ε take a random action
- Otherwise take the max q action

Finally we got what we want!

Caveats!

- We need to explore, so when generating data we use ε-greedy:
  - Sample a random number between 0 and 1.
  - If the number is less than ε take a random action
  - Otherwise take the max q action

Finally we got what we want!

Caveats!

- We need to explore, so when generating data we use ε-greedy:
  - $q_\pi(s_i,a_i) = \mathbf{E}_{ri\sim r(si,ai)} r_i + \gamma \max_{ai+1} q_\pi(s_{i+1},a_{i+1})$

This is super biased! If our q function starts wrong, it can really screw up our learning.

Furthermore the max, makes this even worse

- Use "Double-Q" trick

- Also use slowly moving target network for the second part of the equation

$$q_\pi^{target}(s_{i+1},a_{i+1}) = \alpha\, q_\pi^{target}(s_{i+1},a_{i+1}) + (1-\alpha)q_\pi(s_{i+1},a_{i+1})$$

Finally we got what we want!

Caveats!

- We need to explore, so when generating data we use ε-greedy:
  - $q_\pi(s_i,a_i) = \mathbf{E}_{ri\sim r(si,ai)} r_i + \gamma \max_{ai+1} q_\pi(s_{i+1},a_{i+1})$

This is super biased! If our q function starts wrong, it can really screw up our learning.

Furthermore the max, makes this even worse

- The max means we can only do this for discrete action spaces!

Continuous Action Spaces

Idea:

- Before our network produced q estimates for all actions $q(s) \rightarrow [q_{s1}, q_{s2}, q_{s3}, \ldots]$
- Now our q network will take a state and action and produce a single estimate $q(s,a) \rightarrow q_{sa}$
- We will also add an “actor” network that produces an estimate of the current best action
- We train our q network using the actor: $q(s,a) = r + q(s, actor(s'))$
- We train our actor using a gradient that tries to increase the q values. Compute: $q(s, actor(s)) \rightarrow q_{s,actor(s)}$ and use $-q_{s,actor(s)}$ as a loss.
- DDPG/SAC

References

DQN:
Playing Atari with Deep Reinforcement Learning [Mnih et al. ‘13]

DPG:
Deterministic Policy Gradient Algorithms [Sliver et al. ‘14]

DDPG:
Continuous control with deep reinforcement learning [Lillicrap et al. ‘15]

SAC:
Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor [Haarnoja et al. ‘18]

Last Thoughts

Imitation Learning:

- Online data helps!
- DAgger will perform better than Behavior Cloning if you can afford it

Reinforcement Learning:

- Learning from rewards can be very powerful
- But is hard to get right
- On-Policy methods are not very data efficient
- Off-Policy methods are better but can have a lot of moving parts and require care to get right