CS182/282A: Designing, Visualizing and Understanding Deep Neural Networks

John Canny

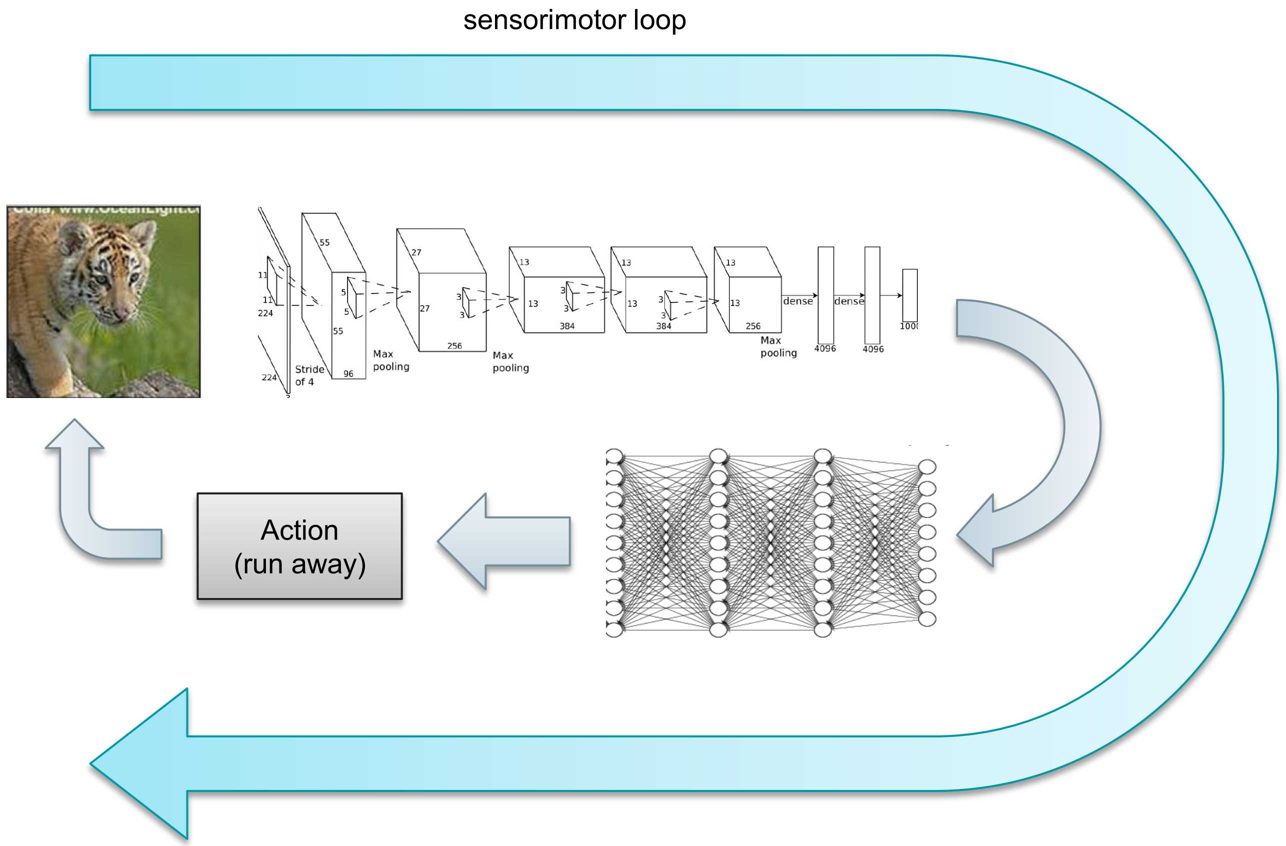
Spring 2020

Lecture 21: Deep Reinforcement Learning

Policy Gradients

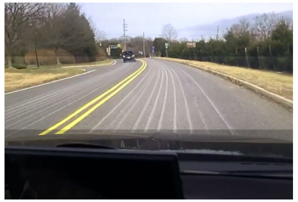
Some slides borrowed from S. Levine et al. “Deep Reinforcement Learning”

Last Time: Sensorimotor Learning

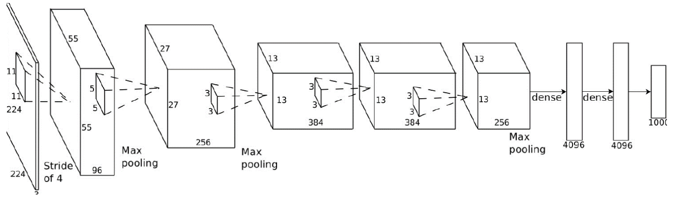


Slide from “Deep Reinforcement Learning” CS285, Levine and Finn

Last Time: Imitation Learning



s_t



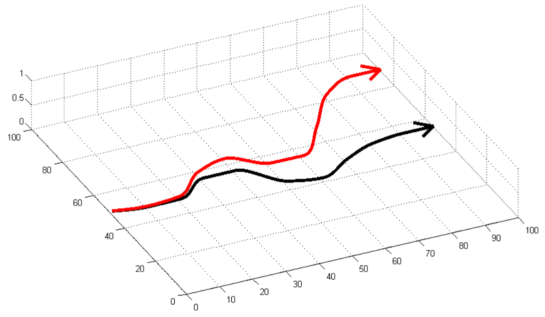
$\pi_\theta(a_t|s_t)$

a_t

s_t
a_t

$\rightarrow$ training data $\rightarrow$ supervised learning

$\pi_\theta(a_t|s_t)$

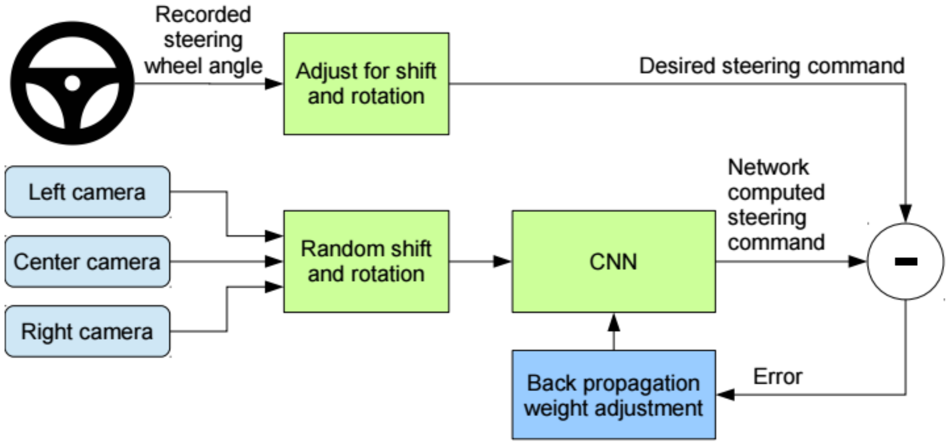


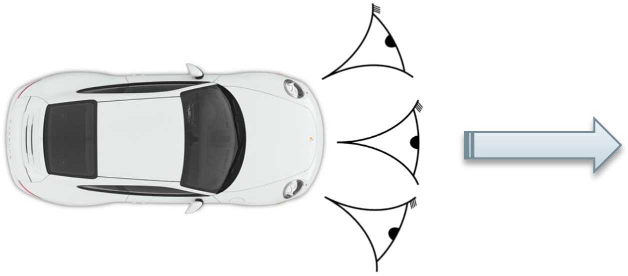
Images: Bojarski et al. '16, NVIDIA

Last Time: Data Augmentation Heuristics

At training time:

3 camera views are used to simulate deviations from the human trajectory.


Recorded steering wheel angle
Adjust for shift and rotation
Desired steering command
Left camera
Center camera
Right camera
Random shift and rotation
CNN
Network computed steering command
-
Back propagation weight adjustment
Error



Bojarski et al. ‘16, NVIDIA

Last Time: DAGGer

Can we make $p_{data}(s_t) = p_{\pi_\theta}(s_t)$?

Idea: instead of being clever about $p_{\pi_\theta}(s_t)$, be clever about $p_{data}(s_t)$!

DAGGer: Dataset AGGregation

Goal: collect training data from $p_{\pi_\theta}(s_t)$ instead of $p_{data}(s_t)$

How? Just run $\pi_\theta(a_t|s_t)$

But we need "labels" a_t

1. Train $\pi_\theta(a_t|s_t)$ from human data $\mathcal{D} = \{s_1, a_1, \dots, s_N, a_N\}$
2. Run $\pi_\theta(a_t|s_t)$ to get a dataset $\mathcal{D}_\pi = \{s_1, \dots, s_N\}$
3. Ask human to label $\mathcal{D}_\pi$ with actions a_t
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

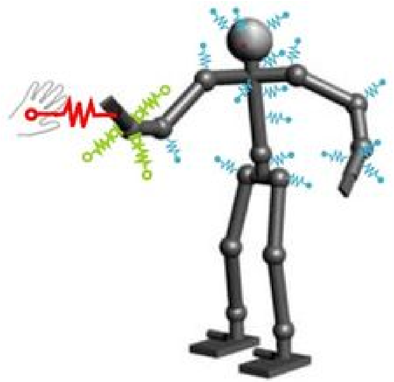
Current policy $\pi_\theta(a_t|s_t)$

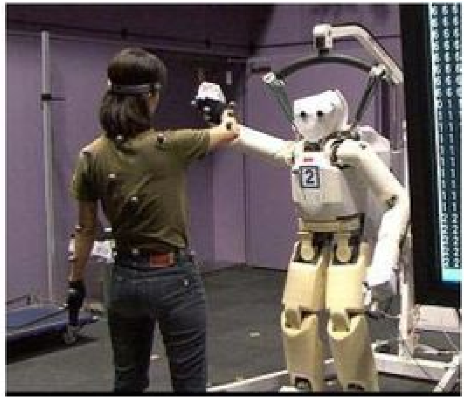
"Expert" actions a_t

Ross et al. '11

Last Time: GAIL: Generative Adversarial Imitation Learning

Rather than trying to mimic the user, try to solve the same control problem that the user is solving. i.e. estimate the user's cost function, and then optimize the cost by training. This is called Inverse Reinforcement Learning (IRL).





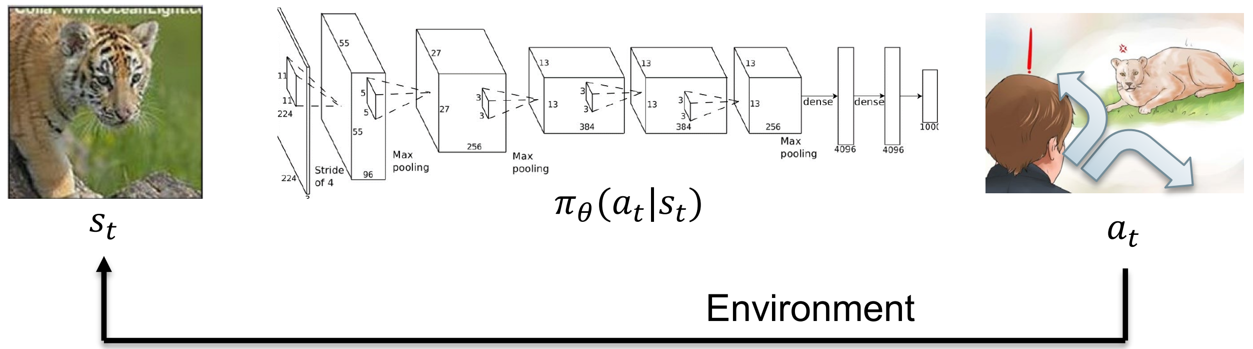
Estimating the cost function: $\underset{c \in \mathcal{C}}{\text{maximize}} \left(\min_{\pi \in \Pi} -H(\pi) + \mathbb{E}_{\pi}[c(s,a)] \right) - \mathbb{E}_{\pi_E}[c(s,a)]$

Jonathan Ho and Stefano Ermon, "Generative Adversarial Imitation Learning"

Outline

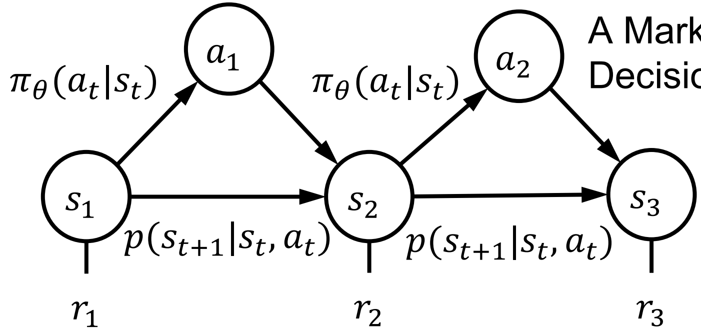
- Markov Decision Processes
- Policy Gradients
- Reducing Variance – Baselines
- Off-policy learning
- Trust-Region Policy Optimization (TRPO) + Proximal Policy Optimization (PPO)

Terminology & notation



s_t - state
a_t - action
r_t - reward

$\pi_\theta(a_t|s_t)$ = policy = probability of doing action a_t in state s_t



A Markov Decision Process

high reward

low reward

Definitions

Markov decision process $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r\}$

$\mathcal{S}$ – state space states $s \in \mathcal{S}$ (discrete or continuous)

$\mathcal{A}$ – action space actions $a \in \mathcal{A}$ (discrete or continuous)

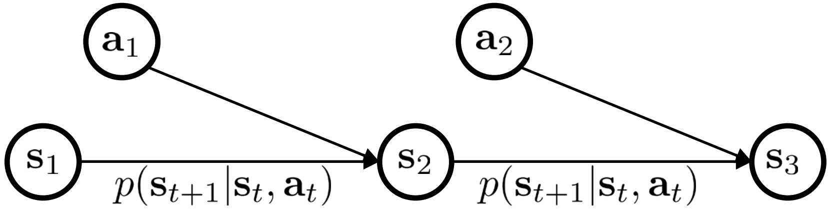
$\mathcal{T}$ – transition operator (now a tensor!)

let $\mu_{t,j} = p(s_t = j)$

let $\xi_{t,k} = p(a_t = k)$

let $\mathcal{T}_{i,j,k} = p(s_{t+1} = i | s_t = j, a_t = k)$

$$\mu_{t,i} = \sum_{j,k} \mathcal{T}_{i,j,k} \mu_{t,j} \xi_{t,k}$$


$\mathbf{a}_1$
$\mathbf{a}_2$
$\mathbf{s}_1$
$\mathbf{s}_2$
$\mathbf{s}_3$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$

Andrey Markov

Richard Bellman

Definitions

Markov decision process $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r\}$

$\mathcal{S}$ – state space states $s \in \mathcal{S}$ (discrete or continuous)

$\mathcal{A}$ – action space actions $a \in \mathcal{A}$ (discrete or continuous)

$\mathcal{T}$ – transition operator (now a tensor!)

r – reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

$r(s_t, a_t)$ – reward

$\tau = (s_0, a_0, s_1, a_1, \ldots, s_T, a_T)$ – trajectory – a sequence of states and actions

Andrey Markov

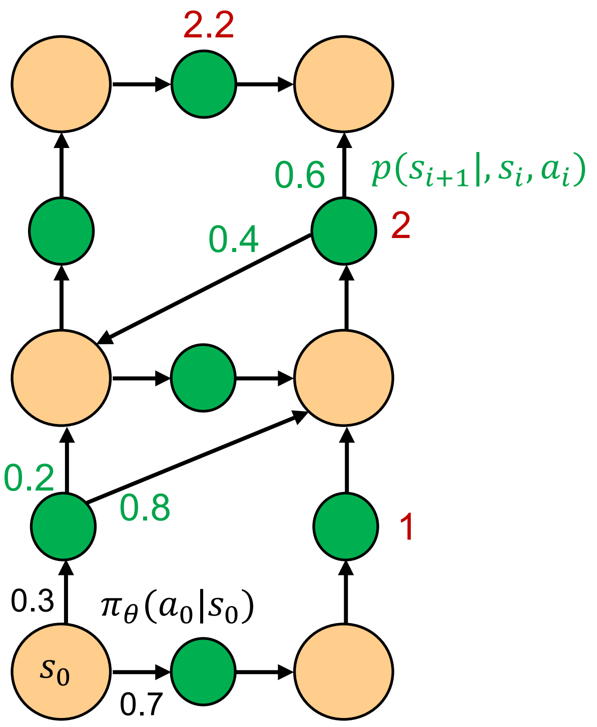
Richard Bellman

Sample MDP

States are orange circles.

Actions are green circles, with transition probabilities in green for actions with multiple successor states.

Rewards are shown in red.



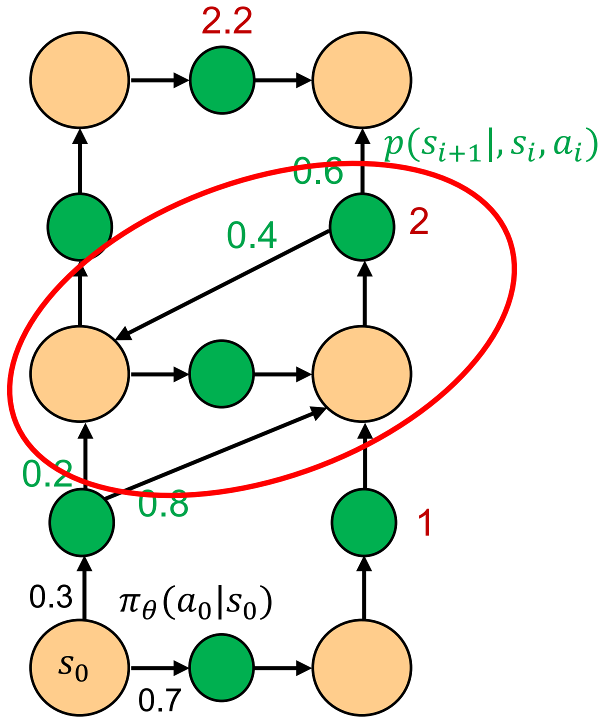
Sample MDP

States are orange circles.

Actions are green circles, with transition probabilities in green for actions with multiple successor states.

Rewards are shown in red.

Note that the graph may have cycles…



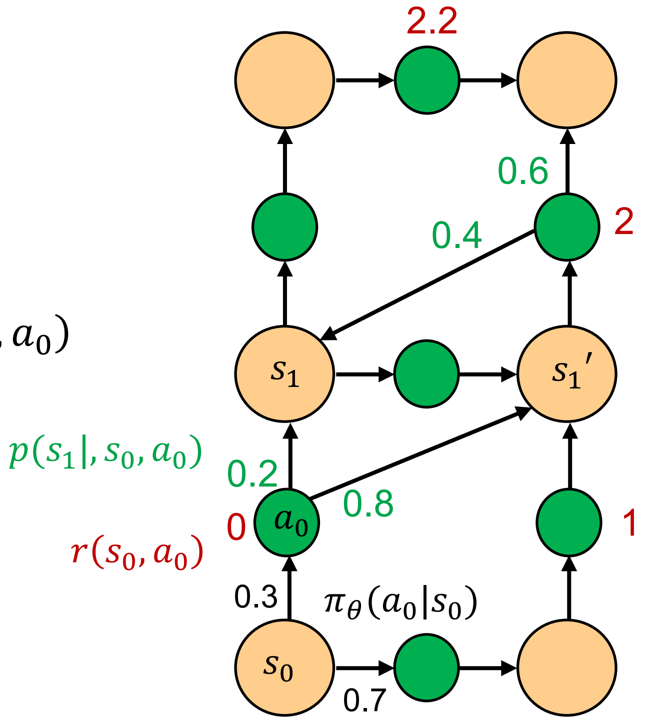
Sample MDP

You can think of the MDP has a two-player game.

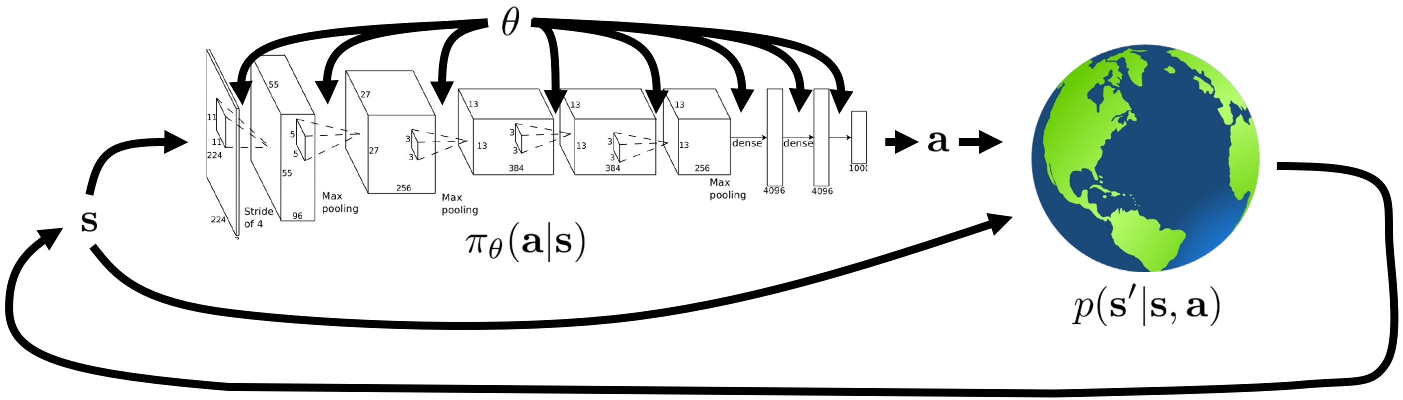
Start at state s_0

Policy plays a_0 according to $\pi_\theta(a_0|s_0)$,
receives reward $r(s_0, a_0)$

Environment plays s_1 or $s_1{}'$ according to $\mathrm{p}(s_1|s_0, a_0)$



The goal of reinforcement learning



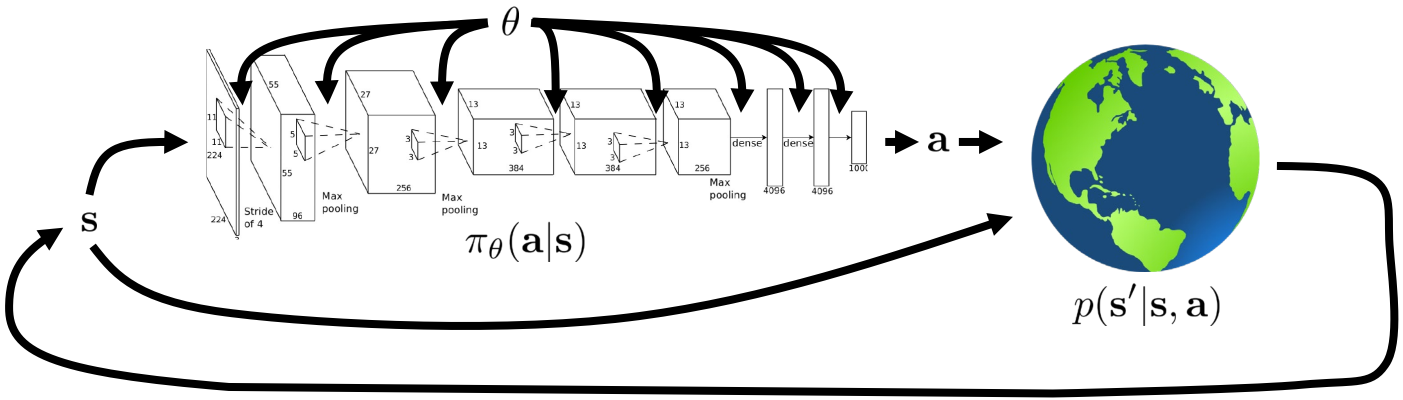
θ
$\mathbf{s}$
$\mathbf{a}$
$\pi_\theta(\mathbf{a}|\mathbf{s})$
$p(\mathbf{s}'|\mathbf{s},\mathbf{a})$
Stride of 4
Max pooling
dense

$$\underbrace{p_\theta(\mathbf{s}_1,\mathbf{a}_1,\ldots,\mathbf{s}_T,\mathbf{a}_T)}_{\pi_\theta(\tau)} = p(\mathbf{s}_1)\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$$

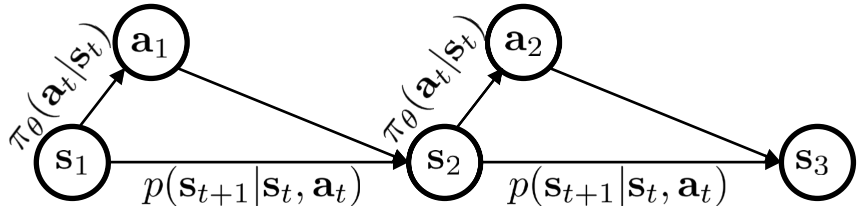
$$\theta^\star = \arg\max_\theta \underbrace{E_{\tau\sim p_\theta(\tau)}}\left[\sum_t r(\mathbf{s}_t,\mathbf{a}_t)\right]$$

Expectation is computed by sampling τ according to the distribution $p_\theta(\tau)$

The goal of reinforcement learning

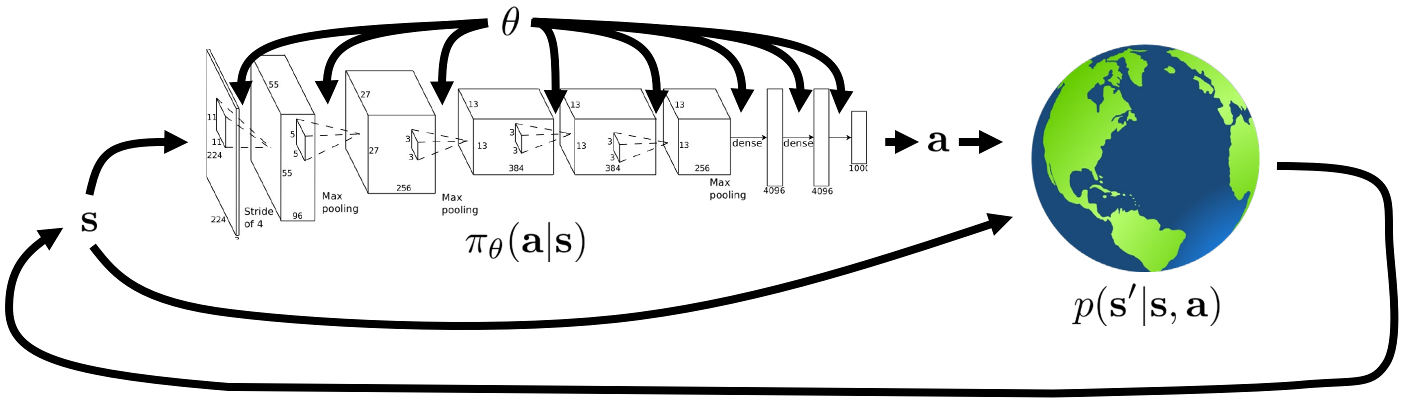


$$\underbrace{p_\theta(\mathbf{s}_1,\mathbf{a}_1,\ldots,\mathbf{s}_T,\mathbf{a}_T)}_{\pi_\theta(\tau)} = p(\mathbf{s}_1)\prod_{t=1}^{T}\underbrace{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}_{\text{Markov chain on } (\mathbf{s},\mathbf{a})}$$



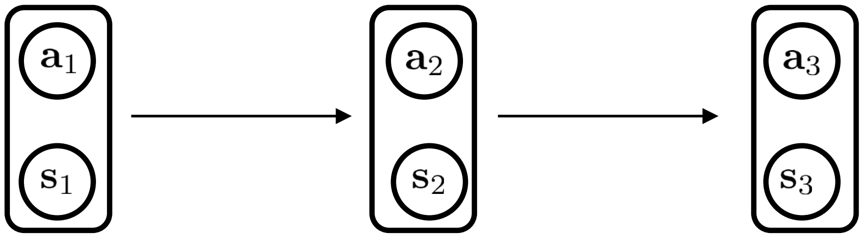
$\mathbf{a}_1$
$\mathbf{a}_2$
$\mathbf{s}_1$
$\mathbf{s}_2$
$\mathbf{s}_3$
$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$
$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$

The goal of reinforcement learning

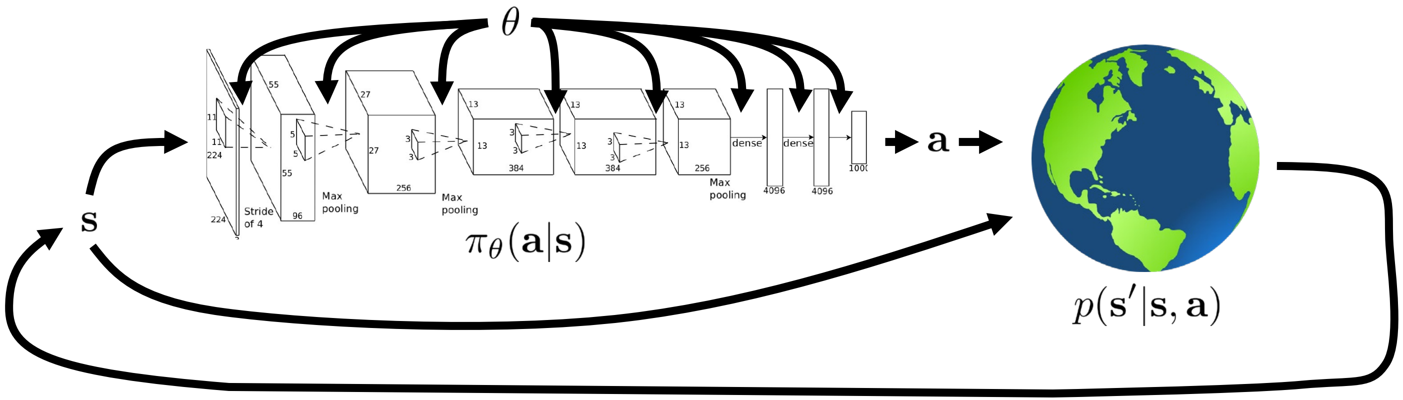


$$\underbrace{p_\theta(\mathbf{s}_1,\mathbf{a}_1,\dots,\mathbf{s}_T,\mathbf{a}_T)}_{\pi_\theta(\tau)} = p(\mathbf{s}_1)\prod_{t=1}^{T}\underbrace{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}_{\text{Markov chain on }(\mathbf{s},\mathbf{a})}$$

$$p((\mathbf{s}_{t+1},\mathbf{a}_{t+1})|(\mathbf{s}_t,\mathbf{a}_t)) =$$
$$p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)\pi_\theta(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})$$



The goal of reinforcement learning



$$\underbrace{p_\theta(\mathbf{s}_1,\mathbf{a}_1,\ldots,\mathbf{s}_T,\mathbf{a}_T)}_{\pi_\theta(\tau)} = p(\mathbf{s}_1)\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$$

Optimal Policy parameters $\theta^\star = \arg\max_\theta \underbrace{E_{\tau\sim p_\theta(\tau)}}_{} \left[\sum_t r(\mathbf{s}_t,\mathbf{a}_t)\right]$

Expectation is computed by sampling τ according to the distribution $p_\theta(\tau)$

Value Functions

We can define the value of a state s_i under a given policy π, $V(s_i)$ as the (discounted) *reward-to-go* from that state:

$$V(s_i) = \sum_{a_i} \overbrace{\pi(a_i|s_i)}^{\text{Probability of taking action } a_i} (\underbrace{r(s_i, a_i)}_{\text{Actual reward at current step}} + \underbrace{\gamma}_{\text{“discount factor” } \gamma} \overbrace{\sum_{s_{i+1}} V(s_{i+1})\, p(s_{i+1}|s_i, a_i)}^{\text{Expected total future rewards}})$$

$0 < \lambda \leq 1$ is typically close to 1.

$\lambda < 1$ favors short-term rewards, and causes the recurrence to converge on all MDPs.

Bellman Update

We can maximize the expected total reward directly in the value recurrence by taking the best (maximum reward) action:

$$V(s_i) = \overbrace{\max_{a_i}}^{\text{Take best action } a_i} r(s_i|a_i) + \gamma \sum_{s_{i+1}} V(s_{i+1})\, p(s_{i+1}|s_i, a_i)$$

If the state space is small enough to fit in memory, we can solve this recurrence directly using iterative calculation.

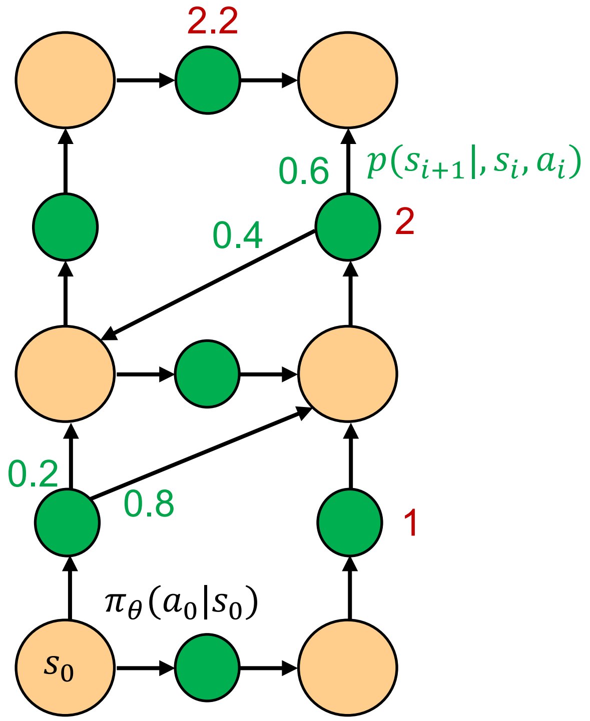
Sample MDP

States are orange circles.

Actions are green circles, with transition probabilities in green for actions with multiple successor states.

Rewards are shown in red.

Note that the graph may have cycles…



Bellman updates

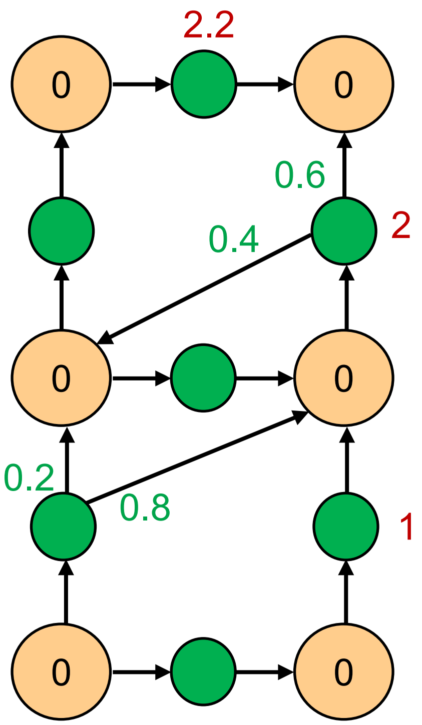
States are orange circles.

Actions are green circles, with transition probabilities in green for actions with multiple successor states.

Rewards are shown in red.

Since the graph has cycles, repeated updates may be necessary to each node (so this is not a dynamic programming problem).

Initialize all node values to 0 (rewards are positive so this is a lower bound on the values).



Bellman updates

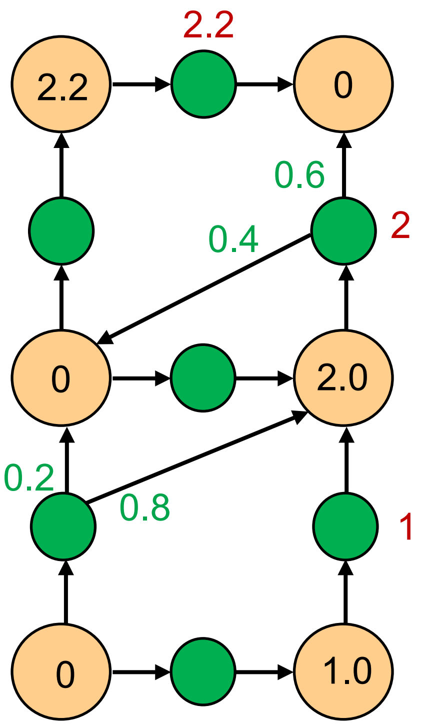
States are orange circles.

Actions are green circles, with transition probabilities in green for actions with multiple successor states.

Rewards are shown in red.

Since the graph has cycles, repeated updates may be necessary to each node (so this is not a dynamic programming problem).

Assume $\gamma = 0.9$ when propagating values.



Bellman updates

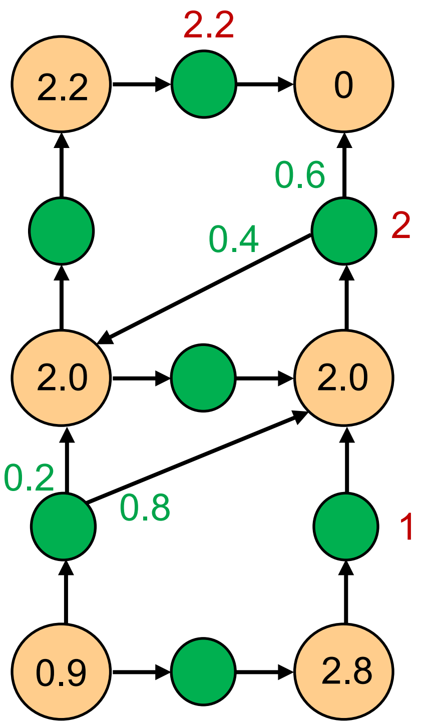
States are orange circles.

Actions are green circles, with transition probabilities in green for actions with multiple successor states.

Rewards are shown in red.

Since the graph has cycles, repeated updates may be necessary to each node (so this is not a dynamic programming problem).

Assume $\gamma = 0.9$ when propagating values.



Bellman updates

States are orange circles.

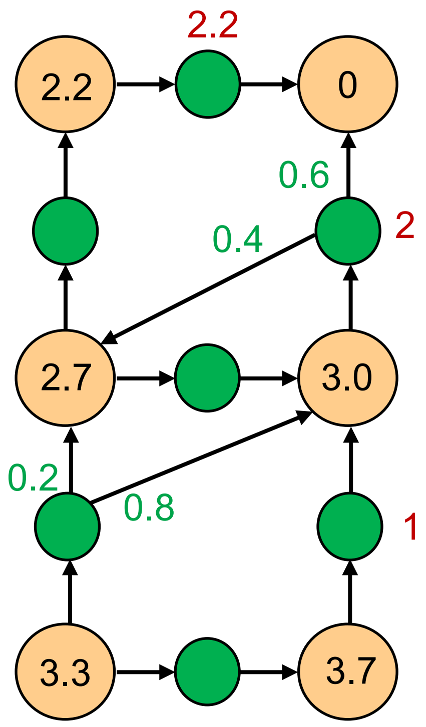
Actions are green circles, with transition probabilities in green for actions with multiple successor states.

Rewards are shown in red.

Since the graph has cycles, repeated updates may be necessary to each node (so this is not a dynamic programming problem).

Assume $\gamma = 0.9$ when propagating values.

Eventually the recurrence yields stationary values. (one significant digit only!)



Bellman updates

States are orange circles.

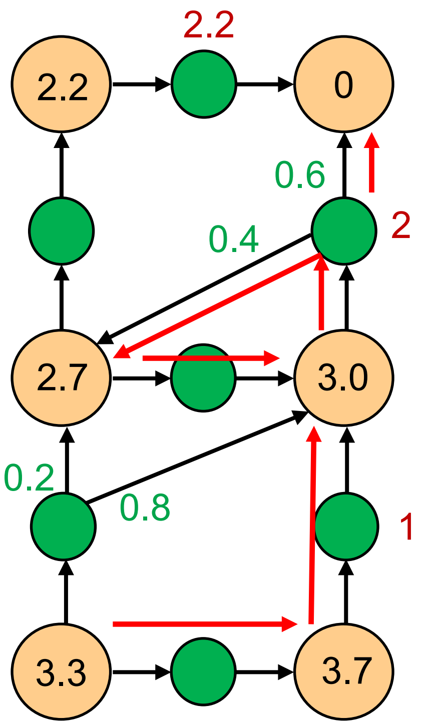
Actions are green circles, with transition probabilities in green for actions with multiple successor states.

Rewards are shown in red.

Since the graph has cycles, repeated updates may be necessary to each node (so this is not a dynamic programming problem).

Assume $\gamma = 0.9$ when propagating values.

Note that the maximum reward policy can yield trajectories with cycles!

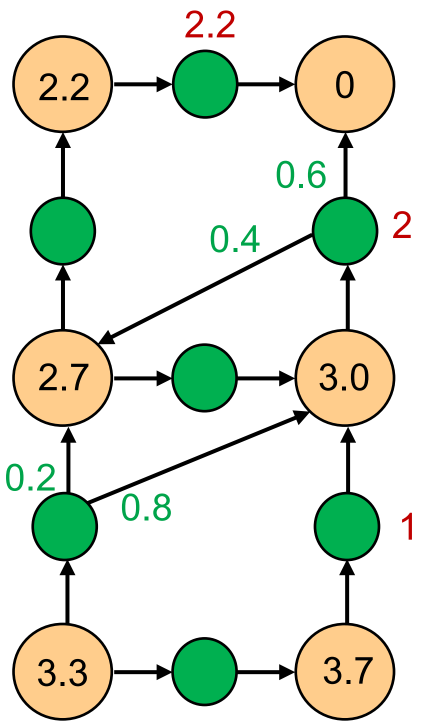


Bellman updates

Aside: if the transition graph is acyclic, then the value function can be computed with dynamic programming.

This requires only O(SA) steps, where S is the number of states, and A is the number of actions.

But graphs with cycles may take longer and the number of iterations depends on the precision of the result.



The goal of reinforcement learning

$$\theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

$$\theta^\star = \arg\max_\theta \overbrace{E_{(\mathbf{s},\mathbf{a}) \sim p_\theta(\mathbf{s},\mathbf{a})} [r(\mathbf{s},\mathbf{a})]}^{\text{Expected one-step reward}}$$

infinite horizon case

$$\theta^\star = \arg\max_\theta \sum_{t=1}^{T} E_{(\mathbf{s}_t,\mathbf{a}_t) \sim p_\theta(\mathbf{s}_t,\mathbf{a}_t)} [r(\mathbf{s}_t,\mathbf{a}_t)]$$

finite horizon case

Challenges of Reinforcement Learning

For supervised learning problems, we trained end-to-end by minimizing a differentiable loss attached to the output of our network.

Assuming we have a differentiable policy $\pi_\theta(a_i|s_i)$ such as a deep network, why cant we differentiate the reward wrt θ to optimize the policy?

Challenges of Reinforcement Learning

Assuming we have a differentiable policy $\pi_\theta(a_i|s_i)$ such as a deep network, why cant we differentiate the reward wrt θ to optimize the policy?

- The reward $r(s_i, a_i)$ is a *function* of the action a_i selected by the policy, and the action set is discrete for many problems. We can’t directly differentiate through this discrete set.

Challenges of Reinforcement Learning

Assuming we have a differentiable policy $\pi_\theta(a_i|s_i)$ such as a deep network, why cant we differentiate the reward wrt θ to optimize the policy?

- The reward $r(s_i, a_i)$ is a *function* of the action a_i selected by the policy, and the action set is discrete for many problems. We can't directly differentiate through this discrete set.

- We don't know the reward function – it's a black box. We cant differentiate through it.

Challenges of Reinforcement Learning

Assuming we have a differentiable policy $\pi_\theta(a_i|s_i)$ such as a deep network, why cant we differentiate the reward wrt θ to optimize the policy?

- The reward $r(s_i, a_i)$ is a *function* of the action a_i selected by the policy, and the action set is discrete for many problems. We can't directly differentiate through this discrete set.

- We don't know the reward function – it's a black box. We cant differentiate through it.

Rewards also depend on the current state s_i, which depends on *previous* actions. Some earlier action a_j which led us to s_i may have been more important. Assigning appropriate weight to earlier actions is the *Temporal Credit Assignment Problem*.

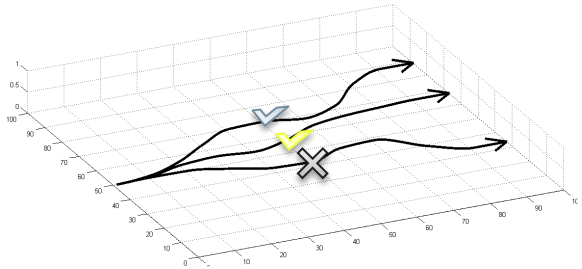
Policy Gradient Approaches

We can’t differentiate the loss end-to-end via a “reward network,” but we can estimate the gradient by enumerating trajectories, and computing the gradients along them. i.e. we can marginalize (average) over states and actions without knowing how they depend on the policy.

This is the policy gradient approach.

Evaluating the objective

$$\theta^\star = \underbrace{\arg\max_\theta E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t,\mathbf{a}_t)\right]}_{J(\theta)}$$



$$J(\theta) = E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t,\mathbf{a}_t)\right] \approx \frac{1}{N}\sum_i \sum_t r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})$$

Sum over sample trajectories from π_θ

Direct policy differentiation

$$\theta^\star = \arg\max_\theta \underbrace{E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t,\mathbf{a}_t)\right]}_{J(\theta)}$$

a convenient identity

$$\underline{\pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau)} = \pi_\theta(\tau)\frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} = \underline{\nabla_\theta \pi_\theta(\tau)}$$

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[\underbrace{r(\tau)}_{\sum_{t=1}^{T} r(\mathbf{s}_t,\mathbf{a}_t)}] = \int \pi_\theta(\tau) r(\tau) d\tau$$

$$\nabla_\theta J(\theta) = \int \underline{\nabla_\theta \pi_\theta(\tau)} r(\tau) d\tau = \int \underline{\pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau)} r(\tau) d\tau = E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]$$

Direct policy differentiation

$$\theta^\star = \arg\max_\theta J(\theta)$$

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[r(\tau)]$$

$$\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \underline{\log \pi_\theta(\tau)} r(\tau)]$$

$$\underbrace{\pi_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{\pi_\theta(\tau)} = p(\mathbf{s}_1)\prod_{t=1}^T \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

log of both sides

$$\underline{\log \pi_\theta(\tau) = \log p(\mathbf{s}_1) + \sum_{t=1}^T \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) + \log p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}$$

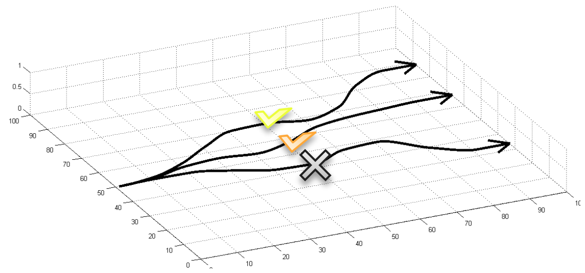
$$\nabla_\theta \left[\cancel{\log p(\mathbf{s}_1)} + \sum_{t=1}^T \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) + \cancel{\log p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)} \right]$$

Eliminate terms independent of θ

$$\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta(\tau)}\left[\left(\sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\right)\left(\sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t)\right)\right]$$

Optimizing the Model

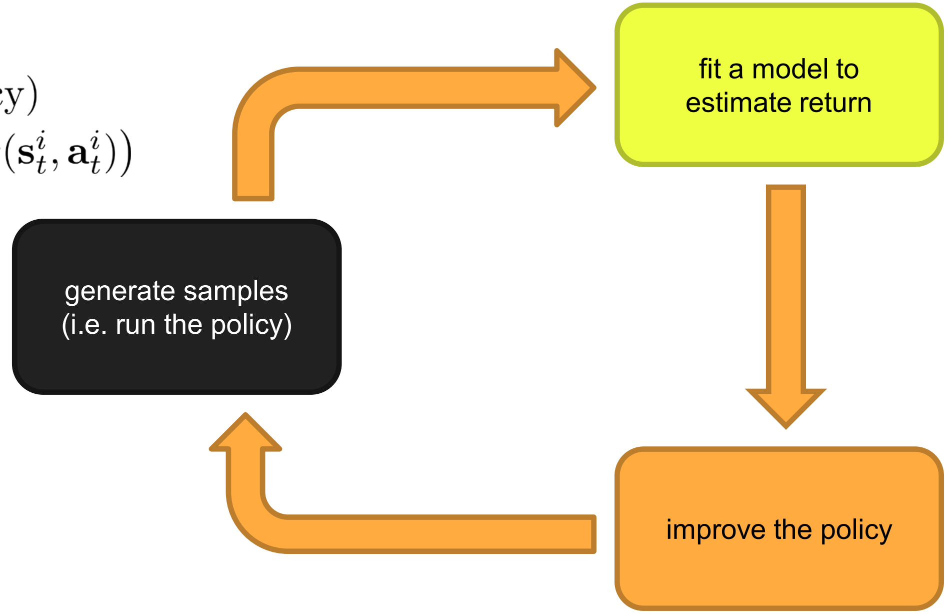
$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})\right)$$



To optimize, simply iterate: $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

REINFORCE algorithm:

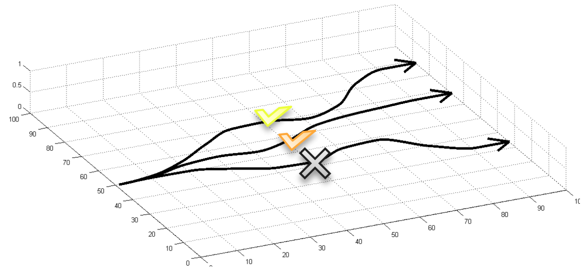
1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i,\mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

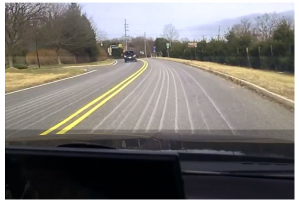

fit a model to estimate return
improve the policy
generate samples (i.e. run the policy)

Evaluating the policy gradient

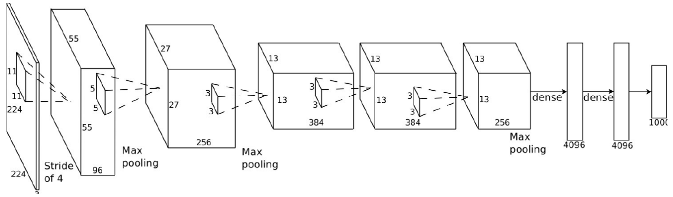
$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})\right)$$

what is this?

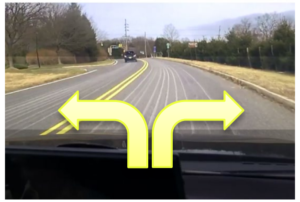




$\mathbf{s}_t$



$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$



$\mathbf{a}_t$

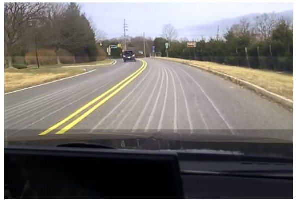
Discrete Action Spaces

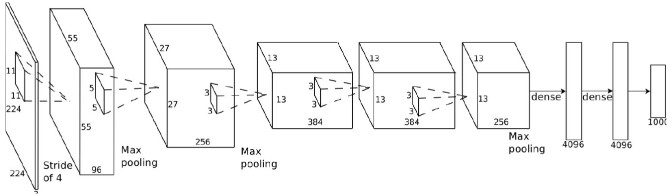
policy gradient: $$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})\right)$$

$\boldsymbol{a}_{i,t}$ is the action taken by the policy

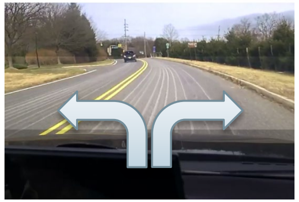
Gradient of the cross-entropy action prediction loss: $$\nabla_\theta J_{\mathrm{ML}}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)$$

$\boldsymbol{a}_{i,t}$ is the action taken by the teacher


$\mathbf{s}_t$

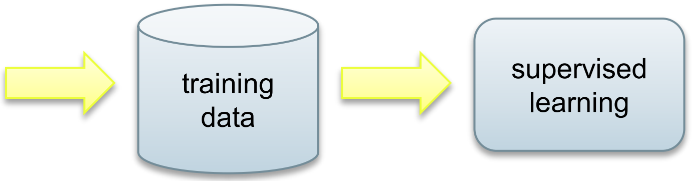


$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$


$\mathbf{a}_t$

$\mathbf{s}_t$

$\mathbf{a}_t$



$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$

Discrete Action Spaces

policy gradient: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})\right)$

Gradient of the cross-entropy action prediction loss: $\nabla_\theta J_{\mathrm{ML}}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)$

So we end up solving a familiar supervised learning problem after all – predicting actions along sample trajectories. we can use standard deep network optimization methods… But:

- We need to average over many trajectories to get a reasonable estimate of the reward $J(\theta)$

- The loss for each trajectory is weighted by the trajectory's reward.

Continuous Actions with Gaussian policies

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})\right)$$

Deep net predicts the mean of a gaussian distribution of actions

$$\text{example:}\ \ \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = \overbrace{\mathcal{N}(f_{\text{neural network}}(\mathbf{s}_t);\Sigma)}$$

$$\log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = -\frac{1}{2}\|f(\mathbf{s}_t)-\mathbf{a}_t\|_\Sigma^2 + \text{const}$$

$$\nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = -\frac{1}{2}\Sigma^{-1}(f(\mathbf{s}_t)-\mathbf{a}_t)\frac{df}{d\theta}$$

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run it on the robot)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i,\mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$

Continuous Actions with Gaussian policies

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run it on the robot)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Outline

- Markov Decision Processes
- Policy Gradients
- Reducing Variance – Baselines
- Off-policy learning
- Trust-Region Policy Optimization (TRPO) + Proximal Policy Optimization (PPO)

Reducing variance

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})\right)$$

Causality: policy at time t' cannot affect reward at time t when $t < t'$

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\underbrace{\left(\sum_{t'=\boxed{t}}^{T} r(\mathbf{s}_{i,t'},\mathbf{a}_{i,t'})\right)}_{\text{“reward to go”}}$$

$$\hat{Q}_{i,t}$$

Baselines

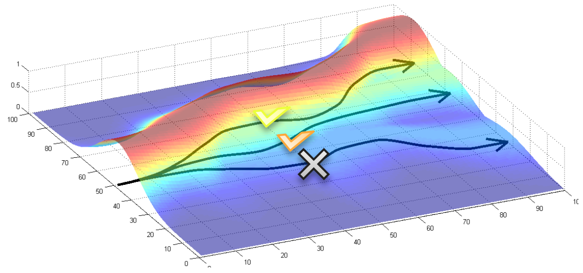
a convenient identity

$$\pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau) = \nabla_\theta \pi_\theta(\tau)$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau)[r(\tau) - b]$$

$$b = \frac{1}{N}\sum_{i=1}^{N} r(\tau)$$

but… are we *allowed* to do that??

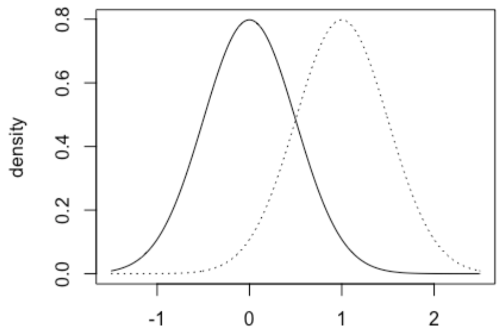


$$E[\nabla_\theta \log \pi_\theta(\tau) b] = \int \pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau) b \, d\tau = \int \nabla_\theta \pi_\theta(\tau) b \, d\tau = b\nabla_\theta \int \pi_\theta(\tau) d\tau = b\nabla_\theta 1 = 0$$

subtracting a baseline is *unbiased* in expectation!

average reward is *not* the best baseline, but it’s pretty good!

Review: Importance Sampling



To estimate an expected value over a distribution $p(x)$ given samples from another distribution $q(x)$.

An importance sampling estimator is:

$$E_{x\sim p(x)}[V(x)] = E_{x\sim q(x)}[V(x)L(x)] \quad \text{where} \quad E_{x\sim q(x)}[L(x)] = 1$$

A simple choice for L is $L(x) = \frac{p(x)}{q(x)}$ since

$$E_{x\sim q(x)}[V(x)L(x)] = \int q(x)\frac{p(x)}{q(x)}V(x)dx = \int p(x)V(x)dx = E_{x\sim p(x)}\ [V(x)]$$

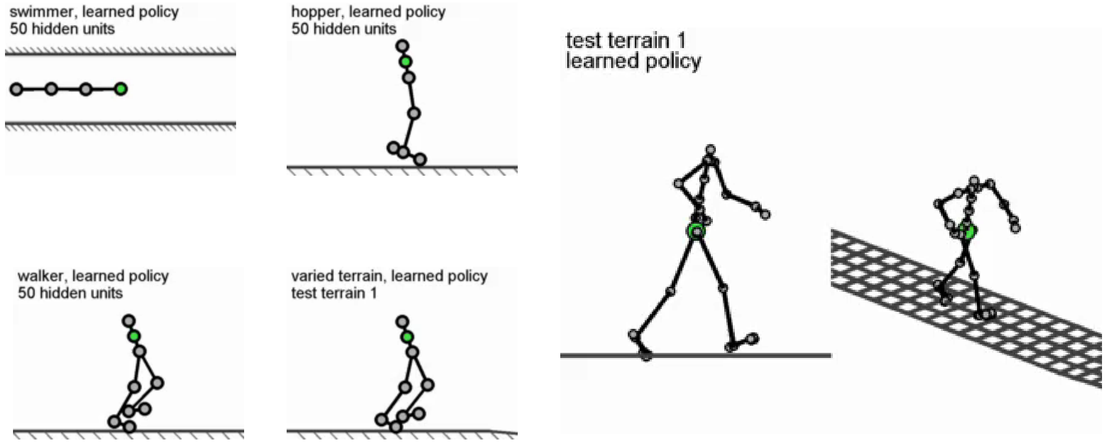
Off-policy policy gradient with importance sampling

$$\nabla_{\theta'} J(\theta') = E_{\tau\sim\pi_\theta(\tau)}\left[\sum_{t=1}^{T} \nabla_{\theta'}\log\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)\boxed{\left(\prod_{t'=1}^{t} \frac{\pi_{\theta'}(\mathbf{a}_{t'}|\mathbf{s}_{t'})}{\pi_\theta(\mathbf{a}_{t'}|\mathbf{s}_{t'})}\right)}\left(\sum_{t'=t}^{T} r(\mathbf{s}_{t'},\mathbf{a}_{t'})\right)\right]$$

Importance Sampling correction for samples from policy $\pi_{\theta'}$ instead of π_θ

Incorporate example demonstrations using importance sampling

Neural network policies



Levine, Koltun ‘13

Challenges with Policy Gradients

Our gradient estimate is:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

1. Requires O(NT) environment actions to get a single gradient estimate.

Many states (and some trajectories) have 0 reward, others have high reward.

2. The gradient estimate has high variance because of this.

Challenges with Policy Gradients

Our model update is:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

1. Each gradient step is very expensive, so we want to minimize the number of steps, i.e. use a high learning rate α.

2. But the gradients are very noisy, so we can't make too large a step or we risk instability.

Methods like TRPO and PPO were developed to take safe steps while still maximizing reward gain.

Challenges with Policy Gradients

But what is a “large” gradient step?

The policy parametrization θ is arbitrary, so we shouldn’t care how much θ changes.

On the other hand, we care very much when the action distributions change, i.e. we care about changes in π_θ as a probability distribution over actions.

Approach: Maximize the reward with a penalty for large changes in π_θ

Trust Region Policy Optimization

We maximize the objective:

$$L(\theta') - c\, KL(\pi_\theta, \pi_{\theta'})$$

Where $$L(\theta') = E_{\tau \sim \pi_\theta}\left[\underbrace{\frac{\pi_{\theta'}(a|s)}{\pi_\theta(a|s)}}_{\text{Importance sample to get the reward at } \pi_{\theta'} \text{ using samples from } \pi_\theta} \overbrace{A(s,a)}^{\text{Advantage of the action } a \text{ (next time) just think of it as reward for now}}\right]$$

and the other term is the KL-divergence:

$$KL(\pi_\theta, \pi_{\theta'}) = E_s \sum_a \pi_\theta(a|s) \log \frac{\pi_{\theta'}(a|s)}{\pi_\theta(a|s)}$$

Trust Region Policy Optimization

Aside KL-divergence:

$$KL(\pi_\theta, \pi_{\theta'}) = E_s \sum_a \pi_\theta(a|s) \log \frac{\pi_{\theta'}(a|s)}{\pi_\theta(a|s)}$$

KL divergence measures the difference between the distribution of action probabilities, averaged across states.

The idea of using the TRPO objective is to maximize reward while not moving $\pi_{\theta'}$ “too far” from π_θ.

Trust Region Policy Optimization

We maximize the objective:

$$L(\theta') - c\ KL(\pi_\theta, \pi_{\theta'})$$

Using a first-order expansion of $L(\theta')$ and a second-order expansion of $KL(\pi_\theta, \pi_{\theta'})$.

Denote $g = \nabla_\theta L(\theta)$ and $F = \nabla^2 KL(\pi_\theta, \pi_{\theta'})$

F is called the Fisher Information Matrix. Unlike the Hessian of a general function such as $L(\theta')$, F is positive semi-definite, so has no saddles and no spurious minima.

Then the parameter update is $\theta' - \theta = \frac{1}{c} F^{-1} g$

Natural Gradient

The quantity $F^{-1}g$ is called the Natural Gradient.

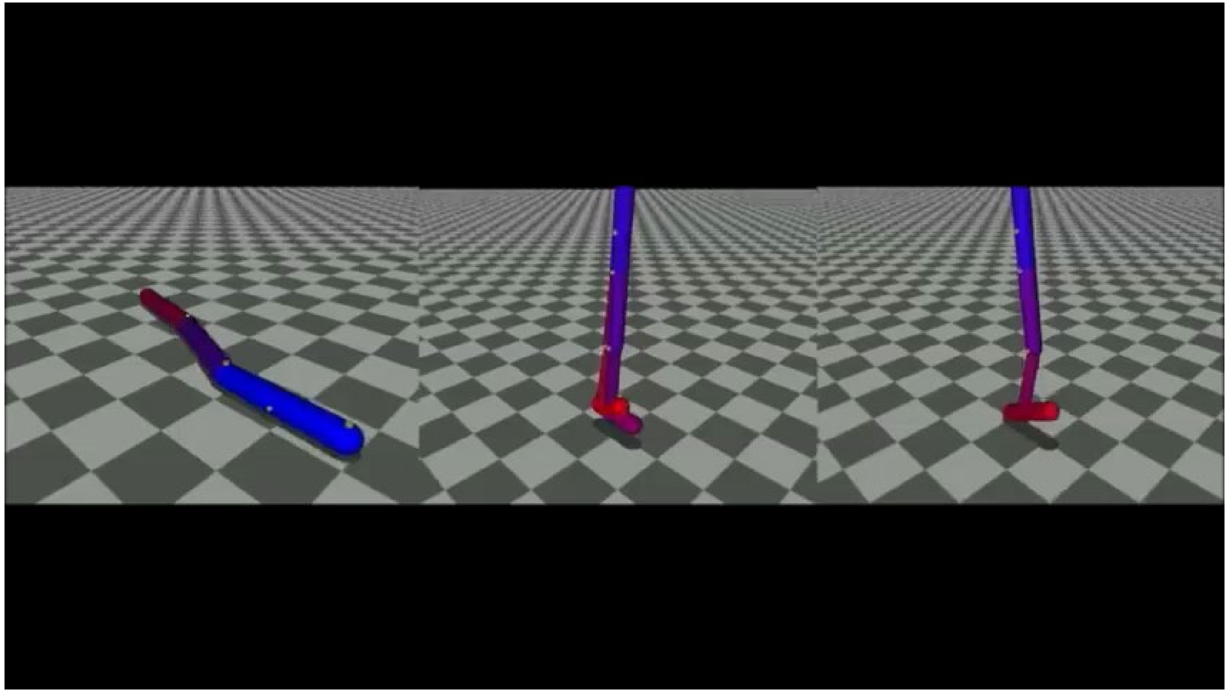
Natural gradient was proposed before for reinforcement learning by Kakade.

TRPO uses a fairly complex algorithm to compute the natural gradient via conjugate gradients without explicitly constructing F.

TRPO examples

Natural gradient with automatic step size.

Discrete and continuous actions.



Proximal Policy Optimization (PPO)

TRPO optimizes: $L(\theta') = E\left[\frac{\pi_{\theta'}(a|s)}{\pi_\theta(a|s)} A(s,a)\right]$ with a KL-divergence regularization loss to avoid large changes in π_θ.

PPO combines the main and regularization loss into a single formula. First define

$$r_t(\theta) = \frac{\pi_{\theta'}(a|s)}{\pi_\theta(a|s)}$$

the PPO objective is

$$L(\theta') = E[\min(r_t A_t, \mathrm{clip}(r_t, 1-\epsilon, 1+\epsilon) A_t)]$$

where

$$\mathrm{clip}(x,a,b) = \begin{cases} a & \text{if} \quad x < a \\ b & \text{if} \quad x > b \\ x & \text{otherwise} \end{cases}$$

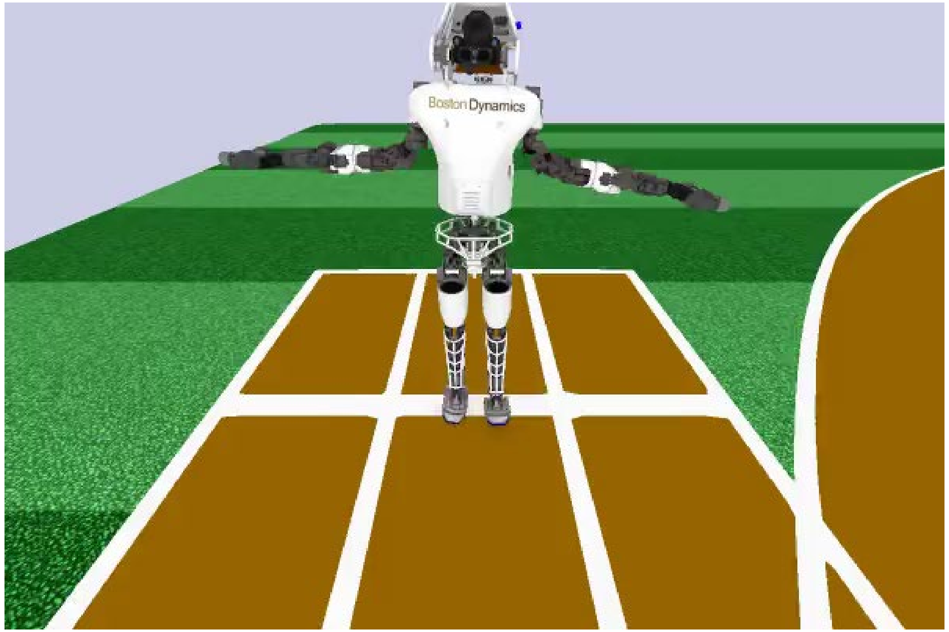
Proximal Policy Optimization (PPO)

PPO is much simpler, and generally performs better than TRPO.

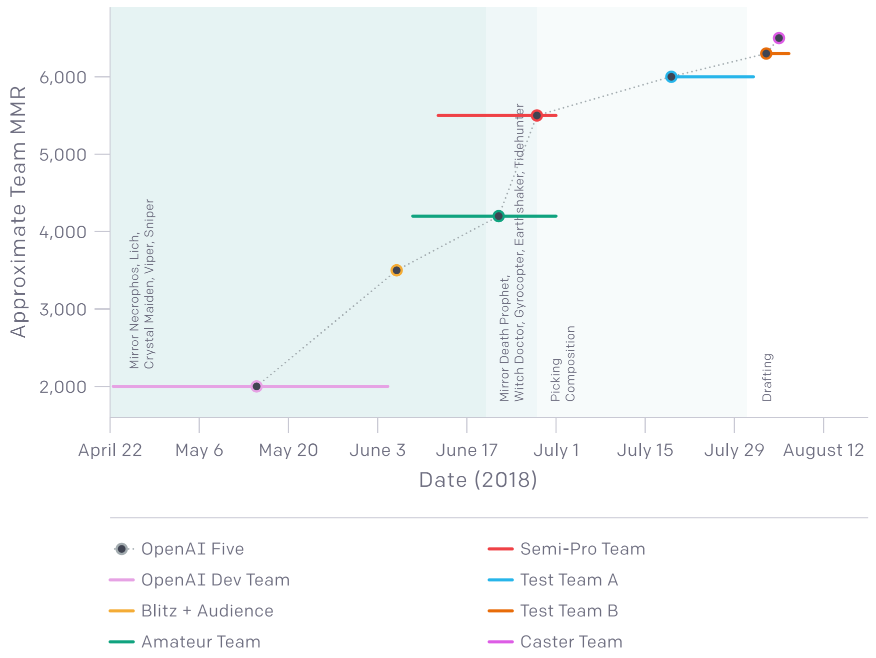
Optimization is separate from the objective.

PPO objective can be optimized with symbolic differentiation software and SGD.

PPO aims for $\pi_{\theta'}$ to be not worse than, and often slightly better than π_θ.



Open-AI Five (PPO)



Summary

- Markov Decision Processes
- Policy Gradients
- Reducing Variance – Baselines
- Off-policy learning
- Trust-Region Policy Optimization (TRPO) + Proximal Policy Optimization (PPO)