Policy Gradients

CS 185/285

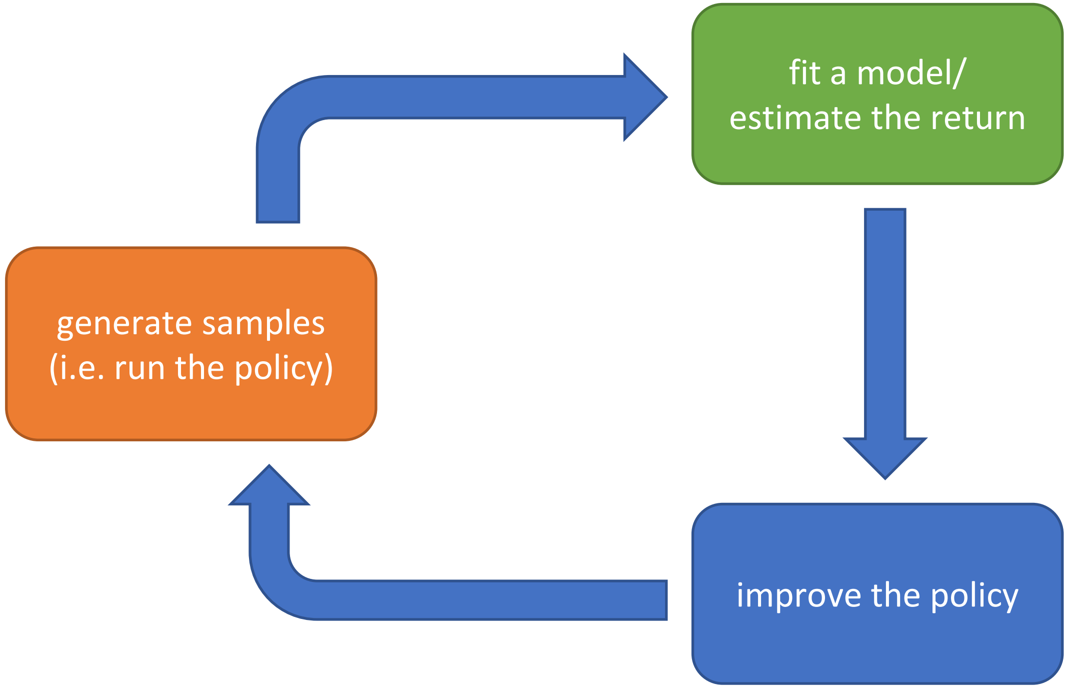
Instructor: Sergey Levine
UC Berkeley

Part 1:
REINFORCE: basic policy gradients

Optimizing the objective of RL

$$\theta^\star = \arg\max_\theta \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=1}^{H} r(\mathbf{s}_t,\mathbf{a}_t)\right]$$

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

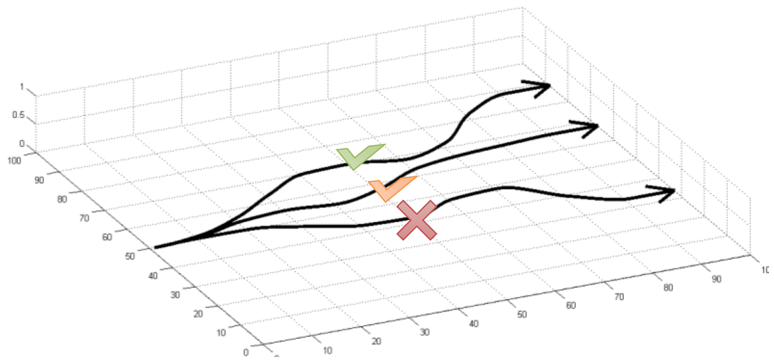


$$J(\theta) = E_\pi\left[\sum_t r_t\right] \approx \frac{1}{N}\sum_{i=1}^{N}\sum_t r_t^i$$

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

Evaluating the objective

$$\theta^\star = \arg\max_\theta \underbrace{\mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{H} r(\mathbf{s}_t, \mathbf{a}_t) \right]}_{J(\theta)}$$



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

sum over samples from π_θ

Direct policy differentiation

$$\theta^\star = \arg\max_\theta \underbrace{E_{\tau\sim p_\theta(\tau)}\left[\sum_{t=1}^{H} r(\mathbf{s}_t, \mathbf{a}_t)\right]}_{J(\theta)}$$

a convenient identity

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

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

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

Direct policy differentiation

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

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

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

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

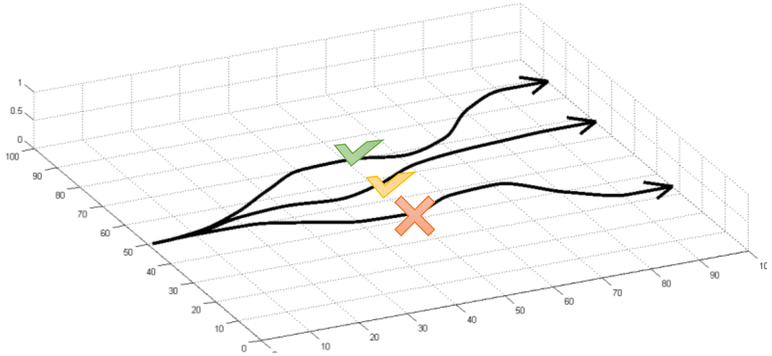
log of both sides

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

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

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

Evaluating the policy gradient

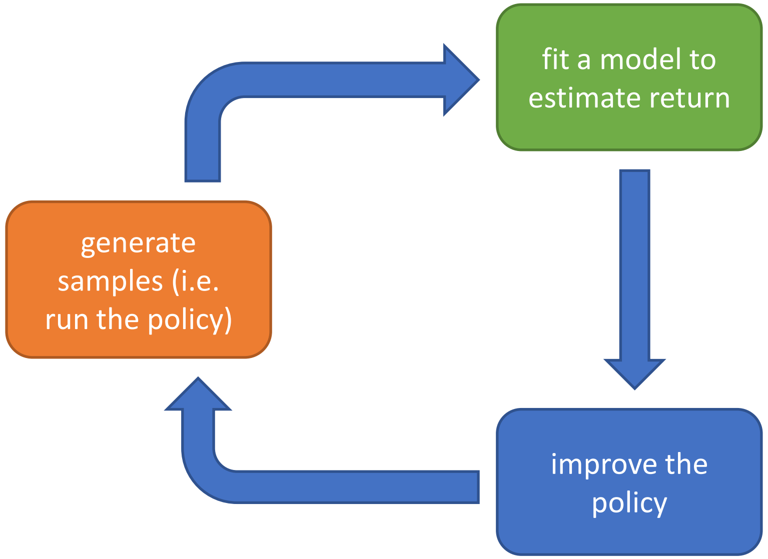


recall: $J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\sum_{t=1}^{H} r(\mathbf{s}_t, \mathbf{a}_t)\right] \approx \frac{1}{N}\sum_i \sum_t r(\mathbf{s}_t^{(i)}, \mathbf{a}_t^{(i)})$

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

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

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

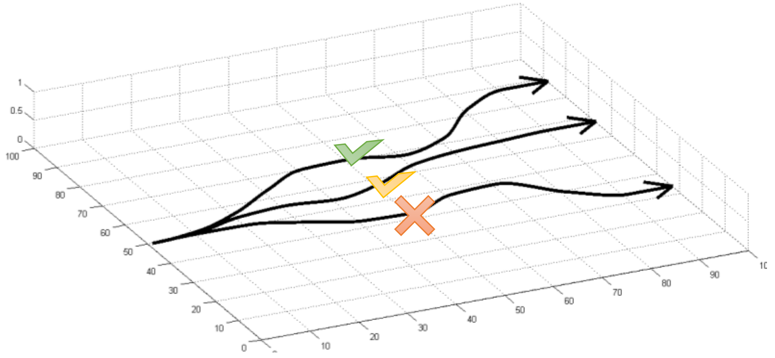


REINFORCE algorithm:

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

Part 2:
Understanding policy gradients

Evaluating the policy gradient

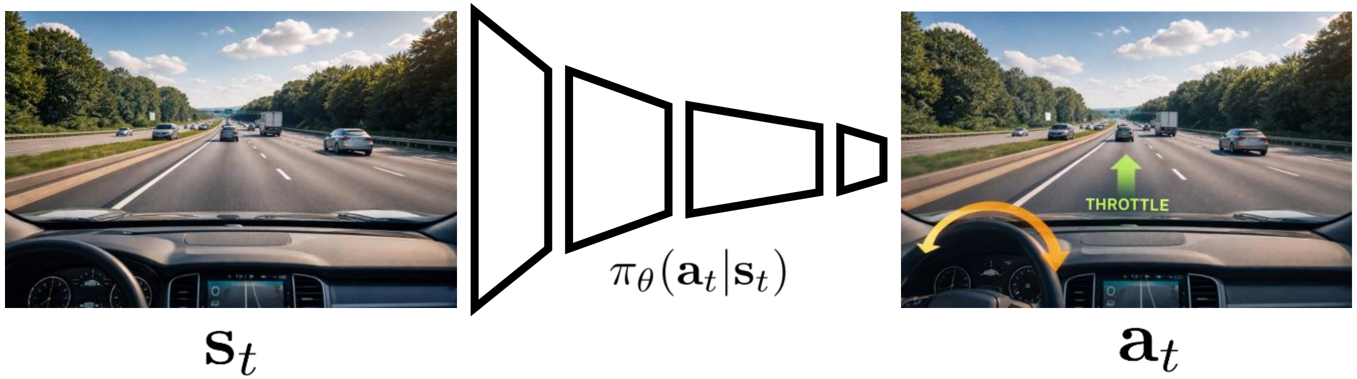


$$\text{recall:} \;\; J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}\left[\sum_{t=1}^{H} r(\mathbf{s}_t, \mathbf{a}_t)\right] \approx \frac{1}{N}\sum_i \sum_t r(\mathbf{s}_t^{(i)}, \mathbf{a}_t^{(i)})$$

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

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

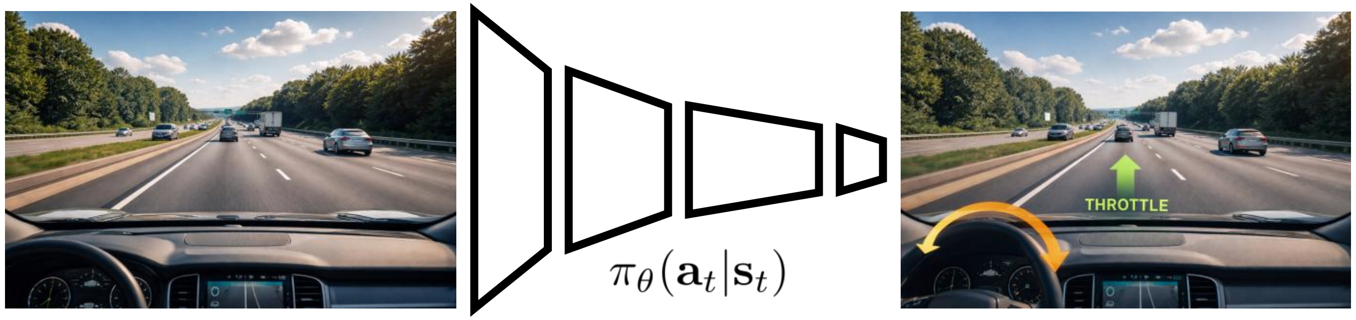
what is this?



Comparison to maximum likelihood

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

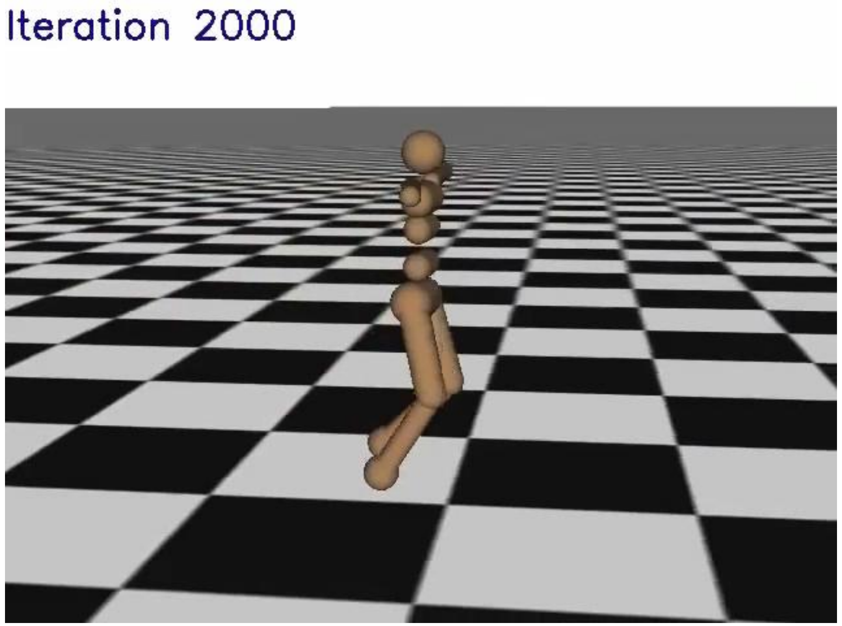
maximum likelihood: $$\nabla_\theta J_{\mathrm{ML}}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{H}\nabla_\theta \log \pi_\theta(\mathbf{a}_t^{(i)}|\mathbf{s}_t^{(i)})\right)$$



$\mathbf{s}_t$ $\mathbf{a}_t$

Example: Gaussian policies

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



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

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

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

REINFORCE algorithm:

1. sample $\{\tau^{(i)}\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run 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. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

What did we just do?

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

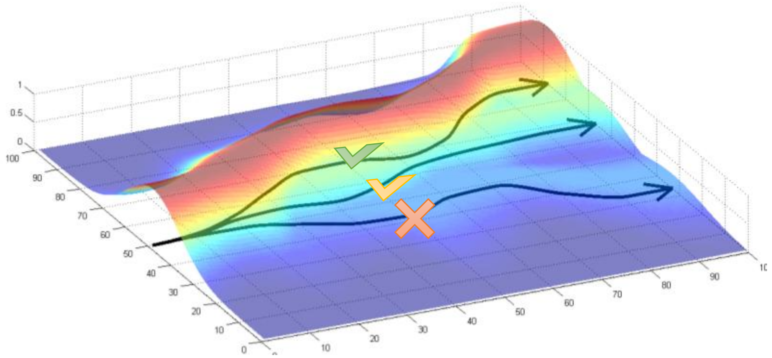
$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \underbrace{\nabla_\theta \log \pi_\theta(\tau^{(i)})}_{\sum_{t=1}^{H} \nabla_\theta \log_\theta \pi_\theta(\mathbf{a}_t^{(i)} | \mathbf{s}_t^{(i)})} r(\tau^{(i)})$$

maximum likelihood: $\nabla_\theta J_{\mathrm{ML}}(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau^{(i)})$

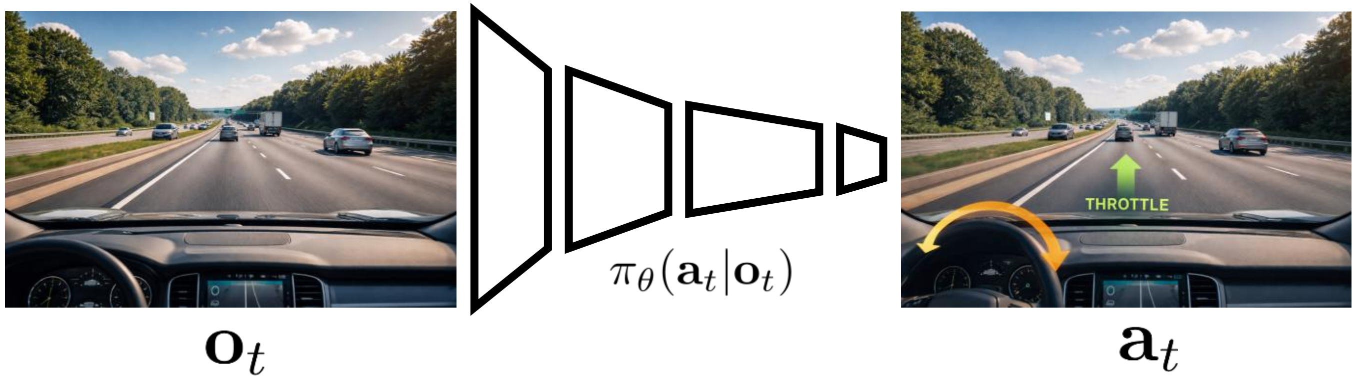
good stuff is made more likely

bad stuff is made less likely

simply formalizes the notion of "trial and error"!



Partial observability



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

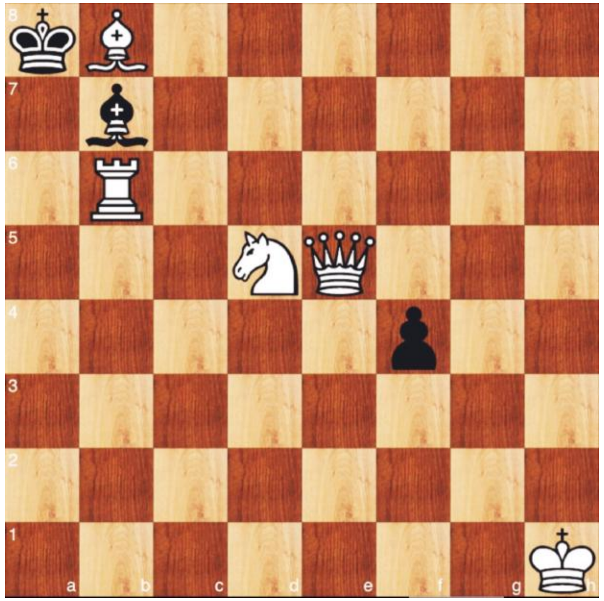
Markov property is not actually used!

Can use policy gradient in partially observed MDPs without modification

What is wrong with the policy gradient?

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

Example: preset position chess



R = +1 for winning, -1 for losing

What we want: positive multipliers on *good* moves, negative multipliers on *bad* moves

What we get:

positive multipliers on *lucky* starts, negative multipliers on *unlucky* starts

negative multipliers on good moves when you made a mistake later

positive multipliers on bad moves when your opponent randomly made a mistake later

these issues "average out" with enough samples, but we might need a very large number of samples to get there

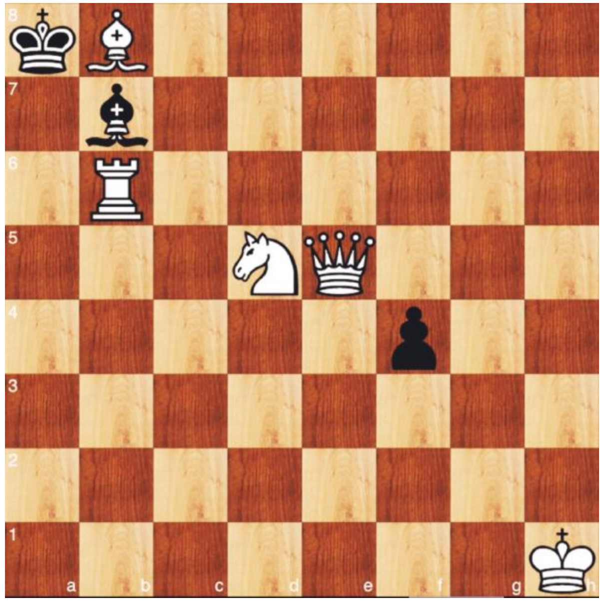
"high variance"

Part 3:
Variance reduction

What is wrong with the policy gradient?

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

Example: preset position chess



R = +1 for winning, -1 for losing

What we want: positive multipliers on *good* moves, negative multipliers on *bad* moves

What we get:

positive multipliers on *lucky* starts, negative multipliers on *unlucky* starts

negative multipliers on good moves when you made a mistake later

positive multipliers on bad moves when your opponent randomly made a mistake later

these issues "average out" with enough samples, but we might need a very large number of samples to get there

"high variance"

Baselines

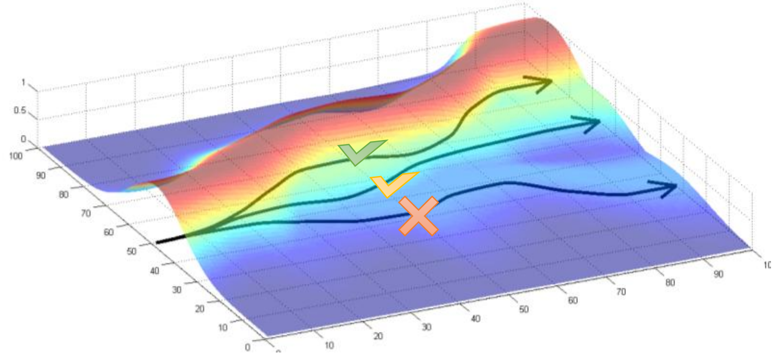
a convenient identity

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

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

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

but… are we *allowed* to do that??



$$\mathbb{E}[\nabla_\theta \log p_\theta(\tau) b] = \int p_\theta(\tau)\nabla_\theta \log p_\theta(\tau) b \, d\tau = \int \nabla_\theta p_\theta(\tau) b \, d\tau = b\nabla_\theta \int p_\theta(\tau) d\tau = b\nabla_\theta 1 = 0$$

subtracting a baseline is *unbiased* in expectation!

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

Causality

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

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

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{H} \nabla_\theta \log \pi_\theta(\mathbf{a}_t^{(i)} | \mathbf{s}_t^{(i)}) \underbrace{\left(\sum_{t'=t}^{H} r(\mathbf{s}_{t'}^{(i)}, \mathbf{a}_{t'}^{(i)}) - b_{t'} \right)}$$

"reward to go" $\hat{Q}_t^{(i)}$

Part 4:
Practical implementation

Policy gradient with automatic differentiation

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{H} \underline{\nabla_\theta \log \pi_\theta(\mathbf{a}_t^{(i)}|\mathbf{s}_t^{(i)})} \left(\overbrace{\sum_{t'=t}^{H} r(\mathbf{s}_{t'}^{(i)}, \mathbf{a}_{t'}^{(i)})}^{\hat{Q}_t^{(i)}} \right)$$

pretty inefficient to compute these explicitly!

How can we compute policy gradients with automatic differentiation?

We need a graph such that its gradient is the policy gradient!

maximum likelihood: $\nabla_\theta J_{\mathrm{ML}}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{H} \nabla_\theta \log \pi_\theta(\mathbf{a}_t^{(i)}|\mathbf{s}_t^{(i)})$ $\quad J_{\mathrm{ML}}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{H} \log \pi_\theta(\mathbf{a}_t^{(i)}|\mathbf{s}_t^{(i)})$

Just implement "pseudo-loss" as a weighted maximum likelihood:

$$\tilde{J}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{H} \log \pi_\theta(\mathbf{a}_t^{(i)}|\mathbf{s}_t^{(i)}) \hat{Q}_t^{(i)}$$

cross entropy (discrete) or squared error (Gaussian)

Policy gradient with automatic differentiation

Pseudocode example (with discrete actions):

Maximum likelihood:

```
# Given:
# actions - (N*T) x Da tensor of actions
# states - (N*T) x Ds tensor of states
# Build the graph:
logits = policy.predictions(states) # This should return (N*T) x Da tensor of action logits
negative_likelihoods = tf.nn.softmax_cross_entropy_with_logits(labels=actions, logits=logits)
loss = tf.reduce_mean(negative_likelihoods)
gradients = loss.gradients(loss, variables)
```

Policy gradient with automatic differentiation

Pseudocode example (with discrete actions):

Policy gradient:

```
# Given:
# actions - (N*T) x Da tensor of actions
# states - (N*T) x Ds tensor of states
# q_values – (N*T) x 1 tensor of estimated state-action values (with baseline subtracted)
# Build the graph:
logits = policy.predictions(states) # This should return (N*T) x Da tensor of action logits
negative_likelihoods = tf.nn.softmax_cross_entropy_with_logits(labels=actions, logits=logits)
weighted_negative_likelihoods = tf.multiply(negative_likelihoods, q_values)
loss = tf.reduce_mean(weighted_negative_likelihoods)
gradients = loss.gradients(loss, variables)
```

$$\tilde{J}(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{H} \log \pi_\theta(\mathbf{a}_t^{(i)} | \mathbf{s}_t^{(i)}) \hat{Q}_t^{(i)}$$

q_values

Policy gradient in practice

- Remember that the gradient has high variance
  - This isn't the same as supervised learning!
  - Gradients will be really noisy!
- Consider using much larger batches
- Tweaking learning rates is very hard
  - Adaptive step size rules like ADAM can be OK-ish
  - We'll learn about policy gradient-specific learning rate adjustment methods later!