Lecture 23: Deep Reinforcement Learning III: Policy Gradient

Xuming He
SIST, ShanghaiTech

Fall, 2019

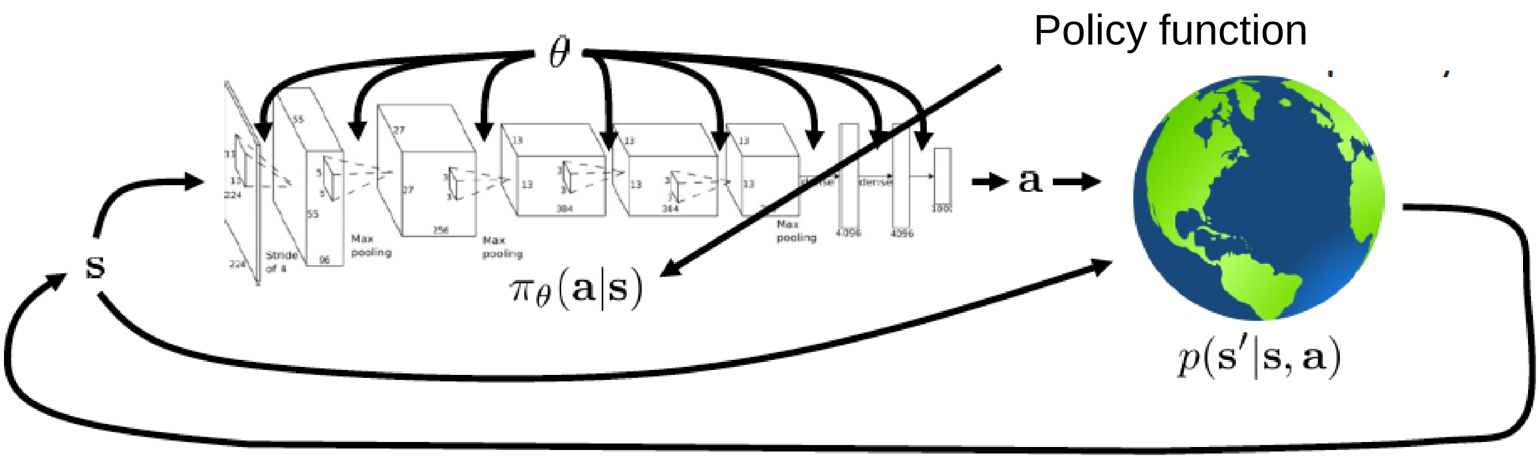
Outline

- Policy gradient method
- Reducing variance and Actor-critic

Acknowledgement: David Silver's, Bhiksha Raj's and Feifei Li et al's notes

Policy optimization

- Given sampled trajectories from an unknown MDP, we directly search for a parametrized policy that optimizes the expected return



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

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

REINFORCE algorithm

- An elegant algorithm for maximizing the expected return

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

- Intuition: trial and error
  - Sample a rollout τ. If you get a high reward, try to make it more likely. If you get a low reward, try to make it less likely.
- This can be seen/derived as stochastic gradient ascent on *J*

REINFORCE algorithm

- Take the gradient

a convenient identity

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

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

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

REINFORCE algorithm

- Plug in the MDP

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

log of both sides

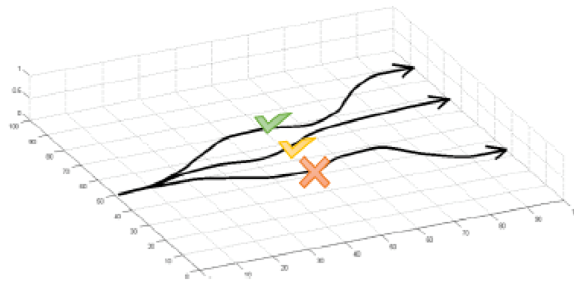
$$\log \pi_\theta(\tau) = \log p(\mathbf{s}_1) + \sum_{t=1}^{T} \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) + \log p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

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

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

Evaluating the gradient

- Using sample average

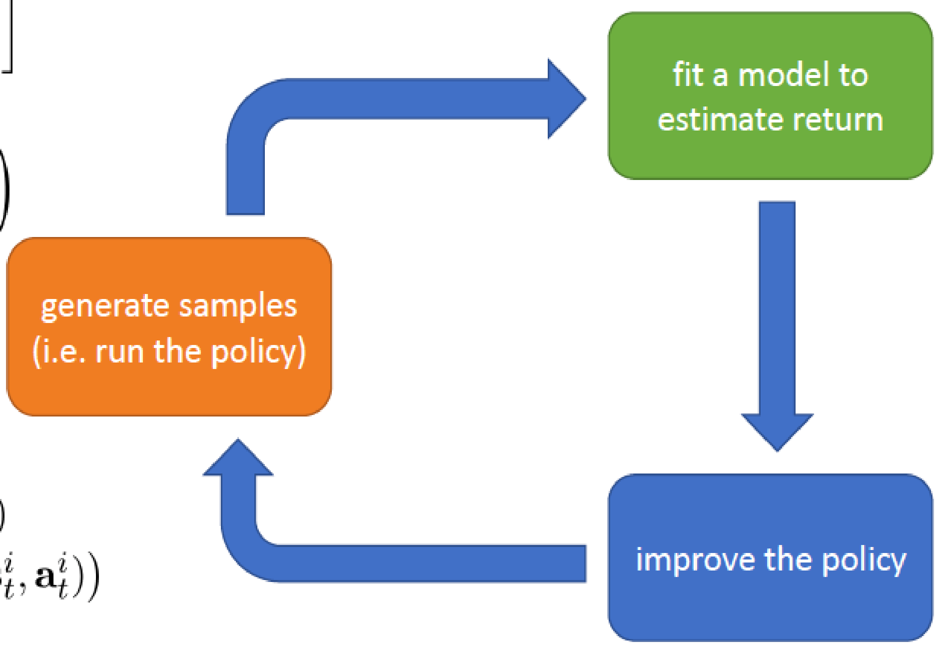


$$\text{recall: } J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right] \approx \frac{1}{N}\sum_i \sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

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

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

$$\underline{\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)}$$



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

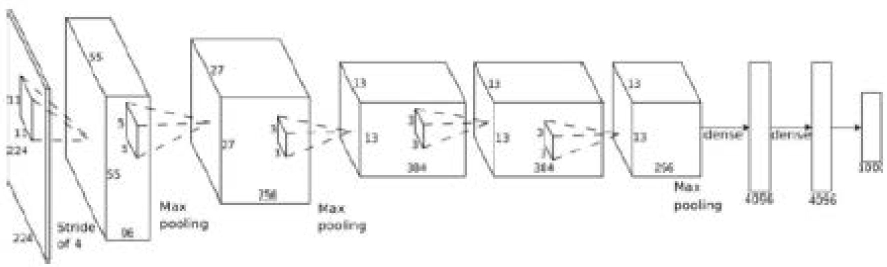
Examples of policy

- What is the policy function?
  - Discrete action space

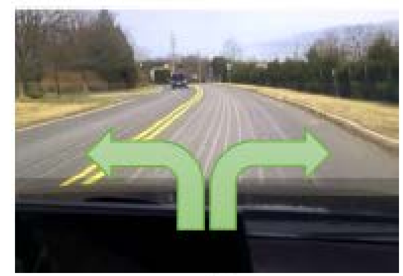
$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

what is this?

$\mathbf{s}_t$



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

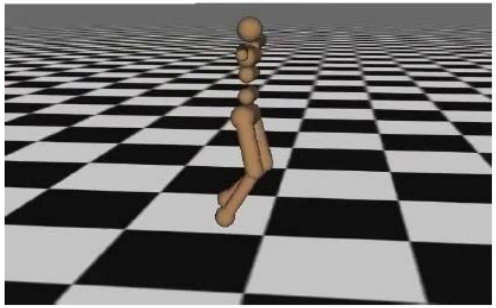


$\mathbf{a}_t$

Examples of policy

- What is the policy function?
  - Continuous action space

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



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

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

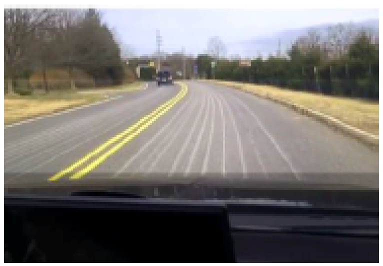
$$\nabla_\theta \log \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) = -\frac{1}{2} \Sigma^{-1} (f(\mathbf{s}_t) - \mathbf{a}_t) \frac{df}{d\theta}$$

$$\text{just backpropagate } -\tfrac{1}{2} \Sigma^{-1} (f(\mathbf{s}_t) - \mathbf{a}_t) \left(\textstyle\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right)$$

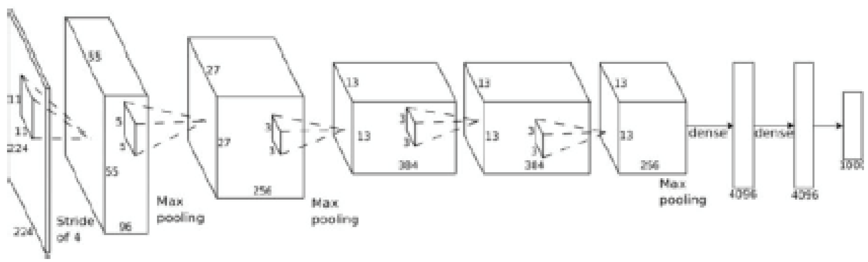
Comparison to Maximum Likelihood

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

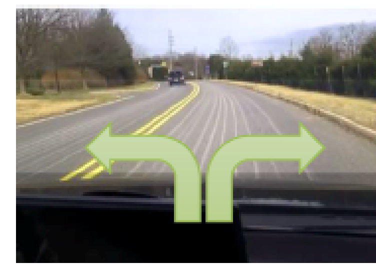
maximum likelihood: $$\nabla_\theta J_{\mathrm{ML}}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)$$



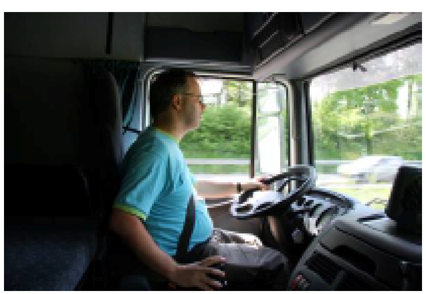
$\mathbf{s}_t$



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



$\mathbf{a}_t$



$\mathbf{s}_t$
$\mathbf{a}_t$ → training data → supervised learning → $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$

Intuition of REINFORCE

- REINFORCE vs ML

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

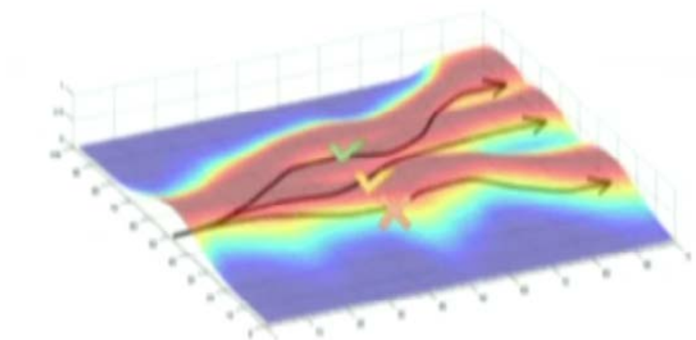
$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\underbrace{\nabla_\theta \log \pi_\theta(\tau_i)}_{\sum_{t=1}^{T}\nabla_\theta \log_\theta \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})} r(\tau_i)$$

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"!

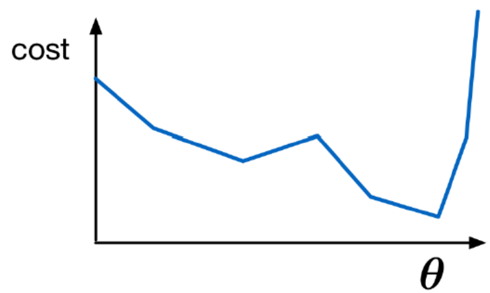


REINFORCE algorithm:

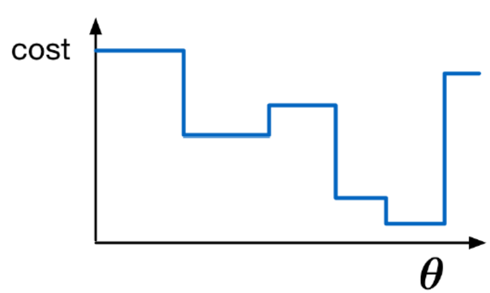
1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run it on the robot)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i,\mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha\nabla_\theta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

Example of REINFORCE learning

- Edge case of RL: handwritten digit classification, but maximizing accuracy (or minimizing 0–1 loss)
- Gradient descent completely fails if the cost function is discontinuous:



Non-differentiable: OK



Discontinuous: not OK

- Original solution: use a surrogate loss function, e.g. logistic-cross-entropy
- RL formulation: in each episode, the agent is shown an image, guesses a digit class, and receives a reward of 1 if it's right or 0 if it's wrong
- We'd never actually do it this way, but it will give us an interesting comparison with backprop

Example of REINFORCE learning

- Optimizing discontinuous objectives

- RL formulation
  - one time step
  - state $\mathbf{x}$: an image
  - action $\mathbf{a}$: a digit class
  - reward $r(\mathbf{x}, \mathbf{a})$: 1 if correct, 0 if wrong
  - policy $\pi(\mathbf{a} \mid \mathbf{x})$: a distribution over categories
    - Compute using an MLP with softmax outputs – this is a policy network

Example of REINFORCE learning

- Optimizing discontinuous objectives
  - Let z_k denote the logits, y_k denote the softmax output, t the integer target, and t_k the target one-hot representation.
  - To apply REINFORCE, we sample $\mathbf{a} \sim \pi_{\boldsymbol{\theta}}(\cdot \mid \mathbf{x})$ and apply:

$$
\begin{aligned}
\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \alpha r(\mathbf{a}, \mathbf{t}) \frac{\partial}{\partial \boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a} \mid \mathbf{x}) \\
&= \boldsymbol{\theta} + \alpha r(\mathbf{a}, \mathbf{t}) \frac{\partial}{\partial \boldsymbol{\theta}} \log y_a \\
&= \boldsymbol{\theta} + \alpha r(\mathbf{a}, \mathbf{t}) \sum_k (a_k - y_k) \frac{\partial}{\partial \boldsymbol{\theta}} z_k
\end{aligned}
$$

  - Compare with the logistic regression SGD update:

$$
\begin{aligned}
\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \alpha \frac{\partial}{\partial \boldsymbol{\theta}} \log y_t \\
&\leftarrow \boldsymbol{\theta} + \alpha \sum_k (t_k - y_k) \frac{\partial}{\partial \boldsymbol{\theta}} z_k
\end{aligned}
$$

Outline

- Policy gradient method
- Reducing variance and Actor-critic

Acknowledgement: David Silver's, Bhiksha Raj's and Feifei Li et al's notes

Problem with the gradient estimation

- High variance and slow convergence
- Hard to choose learning rate

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

- Solution:
  - Reducing variance by transforming the reward function

Reducing gradient variance

- I. Actions should only be reinforced based on future rewards, since they can't influence past rewards

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

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

  - Using "reward to go" $\hat{Q}_{i,t}$

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

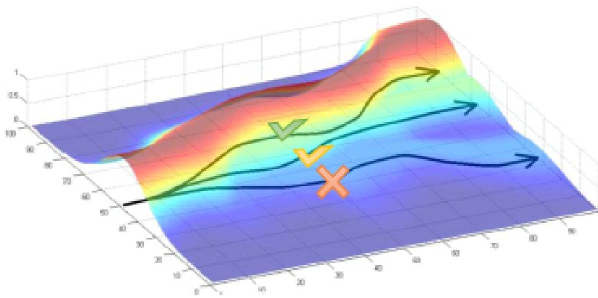
Reducing gradient variance

- II. Introducing "Baselines"

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

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

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!

Reducing gradient variance

- II. Introducing "Baselines"
  - Optimal baseline

$$\mathrm{Var}[x] = E[x^2] - E[x]^2$$

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

$$\mathrm{Var} = E_{\tau\sim\pi_\theta(\tau)}[(\nabla_\theta \log \pi_\theta(\tau)(r(\tau) - b))^2] - \underbrace{E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau)(r(\tau) - b)]^2}$$

this bit is just $E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]$

(baselines are unbiased in expectation)

$$\frac{d\mathrm{Var}}{db} = \frac{d}{db}E[g(\tau)^2(r(\tau) - b)^2] = \frac{d}{db}\left(\cancel{E[g(\tau)^2 r(\tau)^2]} - 2E[g(\tau)^2 r(\tau) b] + b^2 E[g(\tau)^2]\right)$$

$$= -2E[g(\tau)^2 r(\tau)] + 2bE[g(\tau)^2] = 0$$

$$b = \frac{E[g(\tau)^2 r(\tau)]}{E[g(\tau)^2]} \longleftarrow$$

This is just expected reward, but weighted by gradient magnitudes!

Implementing policy gradient ascent

- How do we implement the BP procedure?

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

pretty inefficient to compute these explicitly!

  - We need a computation graph that its gradient is the policy gradient

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

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

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

cross entropy (discrete) or squared error (Gaussian)

Implementing policy gradient ascent

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
# 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)
```

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

q_values

Policy gradient in practice

- 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
  - Need policy gradient-specific learning rate adjustment methods

- https://spinningup.openai.com/en/latest/index.html

Comparison with SL

- What's so great about backprop and gradient descent?
  - Backprop does credit assignment: it tells you exactly which activations and parameters should be adjusted upwards or downwards to decrease the loss on some training example.
  - REINFORCE doesn't do credit assignment. If a rollout happens to be good, all the actions get reinforced, even if some of them were bad.
  - Reinforcing all the actions as a group leads to random walk behavior.

Comparison with SL

- Why policy gradient?
  - Can handle discontinuous cost functions
  - Don't need an explicit model of the environment, i.e. rewards and dynamics are treated as black boxes
- Policy gradient is an example of model-free reinforcement learning, since the agent doesn't try to fit a model of the environment

Baseline in Policy Gradient

- Choose a better baseline?

A better baseline: Want to push up the probability of an action from a state, if this action was better than the **expected value of what we should get from that state**.

Q: What does this remind you of?

A: Q-function and value function!

Baseline in Policy Gradient

- Choose a better baseline?

A better baseline: Want to push up the probability of an action from a state, if this action was better than the **expected value of what we should get from that state**.

Intuitively, we are happy with an action a_t in a state s_t if $Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$ is large. On the contrary, we are unhappy with an action if it's small.

Using this, we get the estimator: $\nabla_\theta J(\theta) \approx \sum_{t \geq 0} (Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)) \nabla_\theta \log \pi_\theta(a_t | s_t)$

Actor-Critic Algorithm

- Computing the expected (optimal) value
  - Using Temporal-difference or Q-learning
  - Combining policy gradient and Q-learning by training both an **actor** (the policy) and a **critic** (the Q-function)

    - The actor decides which action to take, and the critic tells the actor how good its action was and how it should adjust
    - Also alleviates the task of the critic as it only has to learn the values of (state, action) pairs generated by the policy
    - Can also incorporate Q-learning tricks e.g. experience replay
    - **Remark:** we can define by the **advantage function** how much an action was better than expected

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

Actor-Critic Algorithm

- Algorithm summary

Initialize policy parameters θ, critic parameters ϕ
For iteration=1, 2 … **do**
    Sample m trajectories under the current policy
    $\Delta\theta \leftarrow 0$
    For i=1, …, m **do**
        For t=1, … , T **do**

$$A_t = \sum_{t' \geq t} \gamma^{t'-t} r_t^i - V_\phi(s_t^i)$$

$$\Delta\theta \leftarrow \Delta\theta + A_t \nabla_\theta \log(a_t^i | s_t^i)$$

$$\Delta\phi \leftarrow \sum_i \sum_t \nabla_\phi ||A_t^i||^2$$

$$\theta \leftarrow \alpha\Delta\theta$$

$$\phi \leftarrow \beta\Delta\phi$$

End for

Unroll for only a few steps, then compute the REINFORCE policy update using the expected returns estimated by the value network

Repeatedly update the value network to estimate V^π

The two networks adapt to each other, much like GAN training
Modern version: Asynchronous Advantage Actor-Critic (A3C)

Summary

- Deep Reinforcement Learning
  - Markov Decision Process
  - Q learning and DQN
  - Direct approach: Policy gradient method
- Last lecture
  - Recent progresses in deep learning