Deep Reinforcement Learning and Control

Policy gradients

CMU 10-403

Katerina Fragkiadaki
Revision
Value-Based and Policy-Based RL

- **Value Based**
  - Learned Value Function
  - Implicit policy (e.g. $\epsilon$-greedy)

- **Policy Based**
  - No Value Function
  - Learned Policy

- **Actor-Critic**
  - Learned Value Function
  - Learned Policy
Advantages of Policy-Based RL

- Effective in high-dimensional or continuous action spaces
- Can learn stochastic policies
Policy function approximators

\[ o_t \quad \pi_\theta(u_t | o_t) \quad u_t \]
Policy function approximators

**Deterministic continuous policy**

\[ a = \pi_\theta(s) \]

- Outputs a steering angle directly

**Stochastic continuous policy**

\[ a \sim \mathcal{N}(\mu_\theta(s), \sigma_\theta^2(s)) \]

**Stochastic discrete actions**

- Go left
- Go right
- Press brake

Outputs a distribution over a discrete set of actions
Policy function approximators - this lecture

deterministic continuous policy

\[ a = \pi_\theta(s) \]

- e.g. outputs a steering angle directly

stochastic continuous policy

\[ a \sim \mathcal{N}(\mu_\theta(s), \sigma_\theta^2(s)) \]

(stochastic) discrete actions

- go left
- go right
- press brake

Outputs a distribution over a discrete set of actions

Imitation Learning

Images: Bojarski et al. '16, NVIDIA training data supervised learning
Policy Optimization

- Let $U(\theta)$ be any policy **objective function**
- Policy based reinforcement learning is an **optimization problem**
  - Find $\theta$ that maximizes $U(\theta)$
- Some approaches do not use gradient
  - Hill climbing
  - Genetic algorithms
- Greater efficiency often possible using **gradient**
Policy Gradient

- Let $U(\theta)$ be any policy **objective function**

- Policy gradient algorithms search for a local maximum in $U(\theta)$ by ascending the gradient of the policy, w.r.t. parameters $\theta$

\[
\theta_{\text{new}} = \theta_{\text{old}} + \Delta \theta
\]

\[
\Delta \theta = \alpha \nabla_{\theta} U(\theta)
\]

$\alpha$ is a step-size parameter (learning rate)

is the **policy gradient**

\[
\nabla_{\theta} U(\theta) = \left( \begin{array}{c} \frac{\partial U(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial U(\theta)}{\partial \theta_n} \end{array} \right)
\]

Policy gradient: the gradient of the policy objective w.r.t. the parameters of the policy
Computing Gradients By Finite Differences

- Numerically approximating the policy gradient of $\pi_\theta(s, a)$
- For each dimension $k$ in $[1, n]$
  - Estimate $k^{th}$ partial derivative of objective function w.r.t. $\theta$
  - By perturbing $\theta$ by small amount $\epsilon$ in $k^{th}$ dimension
    \[
    \frac{\partial U(\theta)}{\partial \theta_k} \approx \frac{U(\theta + \epsilon u_k) - U(\theta)}{\epsilon}
    \]
    where $u_k$ is a unit vector with 1 in $k^{th}$ component, 0 elsewhere
- Uses $n$ evaluations to compute policy gradient in $n$ dimensions
- Simple, noisy, inefficient - but sometimes effective
- Works for arbitrary policies, even if policy is not differentiable
Learning an AIBO running policy

Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion, Kohl and Stone, 2004

\[ \pi \leftarrow \text{Initial Policy} \]
\[ \text{while } \neg \text{done do} \]
\[ \{R_1, R_2, \ldots, R_t\} = t \text{ random perturbations of } \pi \]
\[ \text{evaluate( } \{R_1, R_2, \ldots, R_t\} \text{ )} \]
\[ \text{for } k = 1 \text{ to } N \text{ do} \]
\[ \text{Avg}_{+\epsilon, k} \leftarrow \text{average score for all } R_i \text{ that have a positive perturbation in dimension } k \]
\[ \text{Avg}_{+0, k} \leftarrow \text{average score for all } R_i \text{ that have a zero perturbation in dimension } k \]
\[ \text{Avg}_{-\epsilon, k} \leftarrow \text{average score for all } R_i \text{ that have a negative perturbation in dimension } k \]
\[ \text{if } \text{Avg}_{+0, k} > \text{Avg}_{+\epsilon, k} \text{ and } \text{Avg}_{+0, k} > \text{Avg}_{-\epsilon, k} \text{ then} \]
\[ A_k \leftarrow 0 \]
\[ \text{else} \]
\[ A_k \leftarrow \text{Avg}_{+\epsilon, k} - \text{Avg}_{-\epsilon, k} \]
\[ \text{end if} \]
\[ \text{end for} \]
\[ A \leftarrow \frac{A}{|A|} \eta \]
\[ \pi \leftarrow \pi + A \]
\[ \text{end while} \]
Trajectory $\tau$ is a state action sequence $s_0, a_0, s_1, a_1, \ldots s_H, a_H$

Trajectory reward: $R(\tau) = \sum_{t=0}^{H} R(s_t, a_t)$

A reasonable policy objective then is $U(\theta) = \mathbb{E}_{\tau \sim P(\tau; \theta)} R(\tau)$

$$\max_{\theta} U(\theta) = \mathbb{E}_{\tau \sim P(\tau; \theta)}[R(\tau)] = \sum_{\tau} P(\tau; \theta) R(\tau)$$

Probability of a trajectory: $P(\tau; \theta) = \prod_{t=0}^{H} P(s_{t+1} \mid s_t, a_t) \cdot \pi_\theta(a_t \mid s_t)$

Our problem is to compute $\nabla_\theta U(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim P(\tau; \theta)}[R(\tau)]$
Computing derivatives of expectations w.r.t. variables that parameterize the distribution, not the quantity inside the expectation

$$\max_{\theta} \mathbb{E}_{x \sim P(x; \theta)} f(x)$$

Assumptions:

- **P** is a probability density function that is continuous and differentiable
- **P** is easy to sample from

$$\max_{\theta} \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[ R(\tau) \right]$$
Derivatives of expectations

\[ \nabla_{\theta} \mathbb{E}_x f(x) = \nabla_{\theta} \mathbb{E}_{x \sim P_\theta(x)} [f(x)] \]
Derivatives of expectations

\[ \nabla_{\theta} \mathbb{E}_x f(x) = \nabla_{\theta} \mathbb{E}_{x \sim P_{\theta}(x)} [f(x)] \]

\[ = \nabla_{\theta} \sum_x P_{\theta}(x) f(x) \]
Derivatives of expectations

\[ \nabla_{\theta} \mathbb{E}_x f(x) = \nabla_{\theta} \mathbb{E}_{x \sim P_{\theta}(x)} [f(x)] \]

\[ = \nabla_{\theta} \sum_x P_{\theta}(x)f(x) \]

\[ = \sum_x \nabla_{\theta} P_{\theta}(x)f(x) \quad \text{Why?} \]
Derivatives of expectations

\[ \nabla_\theta \mathbb{E}_x f(x) = \nabla_\theta \mathbb{E}_{x \sim P_\theta(x)} [f(x)] \]

\[ = \nabla_\theta \sum_x P_\theta(x)f(x) \]

\[ = \sum_x \nabla_\theta P_\theta(x)f(x) \]

What is the problem here?
Derivatives of expectations

\[ \nabla_\theta \mathbb{E}_x f(x) = \nabla_\theta \mathbb{E}_{x \sim P_\theta(x)} [f(x)] \]

\[ = \nabla_\theta \sum_x P_\theta(x)f(x) \]

\[ = \sum_x \nabla_\theta P_\theta(x)f(x) \]

\[ = \sum_x P_\theta(x) \frac{\nabla_\theta P_\theta(x)}{P_\theta(x)} f(x) \]
Derivatives of expectations

\[ \nabla_{\theta} \mathbb{E}_{x} f(x) = \nabla_{\theta} \mathbb{E}_{x \sim P_{\theta}(x)} [f(x)] \]

\[ = \nabla_{\theta} \sum_{x} P_{\theta}(x) f(x) \]

\[ = \sum_{x} \nabla_{\theta} P_{\theta}(x) f(x) \]

\[ = \sum_{x} P_{\theta}(x) \frac{\nabla_{\theta} P_{\theta}(x)}{P_{\theta}(x)} f(x) \]

\[ = \sum_{x} P_{\theta}(x) \nabla_{\theta} \log P_{\theta}(x) f(x) \]
Derivatives of expectations

\[ \nabla_\theta \mathbb{E}_x f(x) = \nabla_\theta \mathbb{E}_{x \sim P_\theta(x)} [f(x)] \]

\[ = \nabla_\theta \sum_x P_\theta(x) f(x) \]

\[ = \sum_x \nabla_\theta P_\theta(x) f(x) \]

\[ = \sum_x P_\theta(x) \frac{\nabla_\theta P_\theta(x)}{P_\theta(x)} f(x) \]

\[ = \sum_x P_\theta(x) \nabla_\theta \log P_\theta(x) f(x) \]

\[ = \mathbb{E}_{x \sim P_\theta(x)} [\nabla_\theta \log P_\theta(x) f(x)] \]

What have we achieved?
Derivatives of expectations

\[
\nabla_\theta \mathbb{E}_x f(x) = \nabla_\theta \mathbb{E}_{x \sim P_\theta(x)} [f(x)] \\
= \nabla_\theta \sum_x P_\theta(x)f(x) \\
= \sum_x \nabla_\theta P_\theta(x)f(x) \\
= \sum_x P_\theta(x) \frac{\nabla_\theta P_\theta(x)}{P_\theta(x)} f(x) \\
= \sum_x P_\theta(x) \nabla_\theta \log P_\theta(x)f(x) \\
= \mathbb{E}_{x \sim P_\theta(x)} [\nabla_\theta \log P_\theta(x)f(x)]
\]

From the law of large numbers, I can obtain an unbiased estimator for the gradient by sampling!
Derivatives of expectations

\[
\nabla_\theta \mathbb{E}_x f(x) = \nabla_\theta \mathbb{E}_{x \sim P_\theta(x)} [f(x)] \\
= \nabla_\theta \sum_x P_\theta(x) f(x) \\
= \sum_x \nabla_\theta P_\theta(x) f(x) \\
= \sum_x P_\theta(x) \frac{\nabla_\theta P_\theta(x)}{P_\theta(x)} f(x) \\
= \sum_x P_\theta(x) \nabla_\theta \log P_\theta(x) f(x) \\
= \mathbb{E}_{x \sim P_\theta(x)} \left[ \nabla_\theta \log P_\theta(x) f(x) \right] \\
\approx \frac{1}{N} \sum_{i=1}^N \nabla_\theta \log P_\theta(x^{(i)}) R(x^{(i)})
\]

From the law of large numbers, I can obtain an unbiased estimator for the gradient by sampling!
Derivatives of expectations

\[ \nabla_\theta \mathbb{E}_x f(x) = \nabla_\theta \mathbb{E}_{x \sim P_\theta(x)} [f(x)] \]

\[ = \nabla_\theta \sum_x P_\theta(x) f(x) \]

\[ = \sum_x \nabla_\theta P_\theta(x) f(x) \]

\[ = \sum_x P_\theta(x) \frac{\nabla_\theta P_\theta(x)}{P_\theta(x)} f(x) \]

\[ = \sum_x P_\theta(x) \nabla_\theta \log P_\theta(x) f(x) \]

\[ = \mathbb{E}_{x \sim P_\theta(x)} \left[ \nabla_\theta \log P_\theta(x) f(x) \right] \]

\[ \approx \frac{1}{N} \sum_{i=1}^N \nabla_\theta \log P_\theta(x^{(i)}) f(x^{(i)}) \]

From the law of large numbers, I can obtain an unbiased estimator for the gradient by sampling!

For Gaussian \( p(x) \)
Derivatives of expectations

\[
\nabla_\theta \mathbb{E}_x f(x) = \nabla_\theta \mathbb{E}_{x \sim P_\theta(x)} [f(x)]
\]

\[
= \sum_x \nabla_\theta P_\theta(x) f(x)
\]

\[
= \sum_x P_\theta(x) \frac{\nabla_\theta P_\theta(x)}{P_\theta(x)} f(x)
\]

\[
= \sum_x P_\theta(x) \nabla_\theta \log P_\theta(x) f(x)
\]

\[
= \mathbb{E}_{x \sim P_\theta(x)} [\nabla_\theta \log P_\theta(x) f(x)]
\]

\[
\approx \frac{1}{N} \sum_{i=1}^N \nabla_\theta \log P_\theta(x^{(i)}) f(x^{(i)})
\]

From the law of large numbers, I can obtain an unbiased estimator for the gradient by sampling!
Derivatives of expectations

\[
\nabla_\theta \mathbb{E}_x f(x) = \nabla_\theta \mathbb{E}_{x \sim P_\theta(x)} [f(x)] \\
= \nabla_\theta \sum_x P_\theta(x)f(x) \\
= \sum_x \nabla_\theta P_\theta(x)f(x) \\
= \sum_x P_\theta(x) \frac{\nabla_\theta P_\theta(x)}{P_\theta(x)} f(x) \\
= \sum_x P_\theta(x) \nabla_\theta \log P_\theta(x)f(x) \\
= \mathbb{E}_{x \sim P_\theta(x)} [\nabla_\theta \log P_\theta(x)f(x)] \\
\approx \frac{1}{N} \sum_{i=1}^N \nabla_\theta \log P_\theta(x^{(i)})f(x^{(i)})
\]

From the law of large numbers, I can obtain an unbiased estimator for the gradient by sampling!

For Gaussian \( p(x) \)

samples \( x \) and \( \nabla_\theta \log p(x) \) for the mean

score function \( f \)
Derivatives of the policy objective

$$\max_\theta \ U(\theta) = \mathbb{E}_{\tau \sim P_\theta(\tau)} [R(\tau)]$$
Derivatives of the policy objective

$$\max_{\theta} \ U(\theta) = \mathbb{E}_{\tau \sim P_\theta(\tau)} [R(\tau)]$$

$$\nabla_\theta U(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim P_\theta(\tau)} [R(\tau)]$$

$$= \nabla_\theta \sum_\tau P_\theta(\tau) R(\tau)$$

$$= \sum_\tau \nabla_\theta P_\theta(\tau) R(\tau)$$
Derivatives of the policy objective

\[
\max_{\theta} U(\theta) = \mathbb{E}_{\tau \sim P_\theta(\tau)} [R(\tau)]
\]

\[
\nabla_\theta U(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim P_\theta(\tau)} [R(\tau)]
\]

\[
= \nabla_\theta \sum_\tau P_\theta(\tau) R(\tau)
\]

\[
= \sum_\tau \nabla_\theta P_\theta(\tau) R(\tau)
\]

\[
= \sum_\tau P_\theta(\tau) \frac{\nabla_\theta P_\theta(\tau)}{P_\theta(\tau)} R(\tau)
\]

\[
= \sum_\tau P_\theta(\tau) \nabla_\theta \log P_\theta(\tau) R(\tau)
\]

\[
= \mathbb{E}_{\tau \sim P_\theta(\tau)} [\nabla_\theta \log P_\theta(\tau) R(\tau)]
\]
Derivatives of the policy objective

$$\max_\theta \ U(\theta) = \mathbb{E}_{\tau \sim \mathcal{P}_\theta(\tau)} \ [R(\tau)]$$

$$\nabla_\theta U(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim \mathcal{P}_\theta(\tau)} \ [R(\tau)]$$

$$= \nabla_\theta \sum_\tau \mathcal{P}_\theta(\tau) R(\tau)$$

$$= \sum_\tau \nabla_\theta P_\theta(\tau) R(\tau)$$

$$= \sum_\tau P_\theta(\tau) \frac{\nabla_\mu P_\theta(\tau)}{P_\theta(\tau)} R(\tau)$$

$$= \sum_\tau P_\theta(\tau) \nabla_\theta \log P_\theta(\tau) R(\tau)$$

$$= \mathbb{E}_{\tau \sim \mathcal{P}_\theta(\tau)} \ [\nabla_\theta \log P_\theta(\tau) R(\tau)]$$

Approximate the gradient with empirical estimate from N sampled trajectories:

$$\nabla_\theta U(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log P_\theta(\tau^{(i)}) R(\tau^{(i)})$$
\[ \nabla_\theta \log P(\tau^{(i)}; \theta) = \nabla_\theta \log \left[ \prod_{t=0}^{T} P(s_{t+1}^{(i)} | s_t^{(i)}, a_t^{(i)}) \cdot \pi_\theta(a_t^{(i)} | s_t^{(i)}) \right] \]

\[ = \nabla_\theta \sum_{t=0}^{T} \log P(s_{t+1}^{(i)} | s_t^{(i)}, a_t^{(i)}) + \log \pi_\theta(a_t^{(i)} | s_t^{(i)}) \]

\[ = \nabla_\theta \sum_{t=0}^{T} \log \pi_\theta(a_t^{(i)} | s_t^{(i)}) \]

\[ = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)}) \]

\[ \nabla_\theta U(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log P_\theta(\tau^{(i)})R(\tau^{(i)}) \]

\[ \nabla_\theta U(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\alpha_t^{(i)} | s_t^{(i)})R(\tau^{(i)}) \]
Intuition

\[ \nabla_\theta U(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\alpha_t^{(i)} | s_t^{(i)}) R(\tau^{(i)}) \]

- Gradient tries to:
  - Increase probability of paths with positive R
  - Decrease probability of paths with negative R

❗ Likelihood ratio changes probabilities of experienced paths, does not try to change the paths (\(<\rightarrow\) Path Derivative)
Likelihood ratio gradient estimator

\[ \max_{\theta} U(\theta) = \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[ R(\tau) \right] \]

\[ \nabla_\theta U(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[ R(\tau) \right] = \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[ \nabla_\theta \log P_\theta(\tau) R(\tau) \right] \]

An unbiased estimator of this gradient:

\[ \hat{g} = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log P_\theta(\tau^{(i)}) R(\tau^{(i)}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\alpha_t^{(i)} \mid s_t^{(i)}) R(\tau^{(i)}) \]

\[ \mathbb{E}[\hat{g}] = \nabla_\theta U(\theta) \]
Pong from Pixels
Policy network
Policy network

e.g.,

height width

[80 x 80]
array of
Policy network

```
h = np.dot(W1, x)  # compute hidden layer neuron activations
h[h<0] = 0   # ReLU nonlinearity: threshold at zero
logp = np.dot(W2, h)  # compute log probability of going up
p = 1.0 / (1.0 + np.exp(-logp))  # sigmoid function (gives probability of going up)
```
E.g. 200 nodes in the hidden network, so:

\[ ((80 \times 80) \times 200 + 200) + [200 \times 1 + 1] = \sim 1.3M \text{ parameters} \]
Network does not see this. Network sees $80 \times 80 = 6,400$ numbers. It gets a reward of $+1$ or $-1$, some of the time.

Q: How do we efficiently find a good setting of the 1.3M parameters?
Random search

Evolutionary methods

Approximation to the gradient via finite differences

Likelihood ratio policy gradients
Suppose we had the training labels...
(we know what to do in any state)

(x1, UP)
(x2, DOWN)
(x3, UP)
...


Suppose we had the training labels…
(we know what to do in any state)

(x1,UP)
(x2,DOWN)
(x3,UP)
...

![Diagram showing a neural network with raw pixels on the left and a hidden layer on the right, with a node indicating the probability of moving UP.]
Suppose we had the training labels…
(we know what to do in any state)

(x1, UP)
(x2, DOWN)
(x3, UP)
...

maximize:

$$\sum_i \log p(y_i | x_i)$$

supervised learning
Except, we don’t have labels...

Should we go UP or DOWN?
Except, we don’t have labels...

“Try a bunch of stuff and see what happens. Do more of the stuff that worked in the future.”

-RL

trial-and-error learning
Let’s just act according to our current policy...

Rollout the policy and collect an episode
Collect many rollouts...

4 rollouts:

[Diagram showing four rollouts with arrows indicating UP and DOWN movements, leading to labels Win, Lose, Lose, Win.]
Not sure whatever we did here, but apparently it was good.
Not sure whatever we did here, but it was bad.
Pretend every action we took here was the correct label.

\[
\text{maximize: } \log p(y_i \mid x_i)
\]

Pretend every action we took here was the wrong label.

\[
\text{maximize: } (-1) \log p(y_i \mid x_i)
\]
Supervised Learning

maximize:

$$\sum_i \log p(y_i | x_i)$$

For images $x_i$ and their labels $y_i$. 
Supervised Learning

maximize:

$$\sum_i \log p(y_i | x_i)$$

For images $x_i$ and their labels $y_i$.
Supervised Learning

maximize:

$$\sum_i \log p(y_i | x_i)$$

For images $x_i$ and their labels $y_i$.

Reinforcement Learning

1) we have no labels so we sample:

$$y_i \sim p(\cdot | x_i)$$
Supervised Learning

maximize:

\[ \sum_i \log p(y_i | x_i) \]

For images \( x_i \) and their labels \( y_i \).

---

Reinforcement Learning

1) we have no labels so we sample:

\[ y_i \sim p(\cdot | x_i) \]

2) once we collect a batch of rollouts:

\[ \sum_i A_i \ast \log p(y_i | x_i) \]
Supervised Learning

maximize:

$$\sum_i \log p(y_i | x_i)$$

For images $x_i$ and their labels $y_i$.

---

Reinforcement Learning

1) we have no labels so we sample:

$$y_i \sim p(\cdot | x_i)$$

2) once we collect a batch of rollouts: maximize:

$$\sum_i A_i * \log p(y_i | x_i)$$

We call this the advantage, it’s a number, like +1.0 or -1.0 based on how this action eventually turned out.

Advantage is the same for all actions taken during a trajectory, and depends on the trajectory return (episode return)
Supervised Learning

maximize:

$$\sum_i \log p(y_i | x_i)$$

For images $$x_i$$ and their labels $$y_i$$.

Reinforcement Learning

1) we have no labels so we sample:

$$y_i \sim p(\cdot | x_i)$$

2) once we collect a batch of rollouts:

maximize:

$$\sum_i A_i \ast \log p(y_i | x_i)$$

+ve advantage will make that action more likely in the future, for that state.

-ve advantage will make that action less likely in the future, for that state.

Advantage is the same for all actions taken during a trajectory, and depends on the trajectory return $$R(\tau)$$.
Each action takes the blame for the full trajectory!
Temporal structure

\[
\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}(\alpha^{(i)}_{t} | s^{(i)}_{t}) R(\tau^{(i)}) \\
= \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}(\alpha^{(i)}_{t} | s^{(i)}_{t}) \left( \sum_{k=0}^{H} R(s^{(i)}_{k}, a^{(i)}_{k}) \right) \\
= \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}(\alpha^{(i)}_{t} | s^{(i)}_{t}) \left( \sum_{k=0}^{t-1} R(s^{(i)}_{k}, a^{(i)}_{k}) + \sum_{k=t}^{H} R(s^{(i)}_{k}, a^{(i)}_{k}) \right)
\]

Each action takes the blame for the full trajectory!

These rewards are not caused by actions that come after t
Temporal structure

\[ \hat{g} = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}(\alpha_t^{(i)} | s_t^{(i)}) R(\tau^{(i)}) \]

\[ = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}(\alpha_t^{(i)} | s_t^{(i)}) \left( \sum_{k=0}^{H} R(s_k^{(i)}, a_k^{(i)}) \right) \]

\[ = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}(\alpha_t^{(i)} | s_t^{(i)}) \left( \sum_{k=0}^{t-1} R(s_k^{(i)}, a_k^{(i)}) + \sum_{k=t}^{H} R(s_k^{(i)}, a_k^{(i)}) \right) \]

Consider instead:

\[ \hat{g} = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}(\alpha_t^{(i)} | s_t^{(i)}) \left( \sum_{k=t}^{H} R(s_k^{(i)}, a_k^{(i)}) \right) \]

We can call this the return from \( t \) onwards \( G_t \)
Let's analyze the update:

$$\Delta \theta_t = \alpha G_t \nabla \theta \log \pi_\theta(s_t, a_t)$$

Let's us rewrite is as follows:

$$\theta_{t+1} = \theta_t + \alpha \gamma^t G_t \frac{\nabla \theta \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)}$$

Update is proportional to:
- the product of a return $G_t$ and
- the gradient of the probability of taking the action actually taken,
- divided by the probability of taking that action.
Let’s analyze the update:

\[ \Delta \theta_t = \alpha G_t \nabla_\theta \log \pi_\theta(s_t, a_t) \]

Let’s us rewrite is as follows:

move most in the directions that favor actions that yield the highest return

\[ \theta_{t+1} = \theta_t + \alpha \gamma^t G_t \frac{\nabla_\theta \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \]

Update is inversely proportional to the action probability -- actions that are selected frequently are at an advantage (the updates will be more often in their direction)
For constant $b$, consider this:

$$
\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta} \log P_{\theta}(\tau^{(i)})(R(\tau^{(i)}) - b)
$$

$$
\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta} \log P_{\theta}(\tau^{(i)}) R(\tau^{(i)}) - \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta} \log P_{\theta}(\tau^{(i)}) b
$$

$$
\mathbb{E} \nabla_{\theta} \log P_{\theta}(\tau) b
$$
For constant $b$, consider this:

$$
\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta} \log P_{\theta}(\tau^{(i)})(R(\tau^{(i)}) - b)
$$

$$
\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta} \log P_{\theta}(\tau^{(i)})R(\tau^{(i)}) - \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta} \log P_{\theta}(\tau^{(i)})b
$$

$$
\mathbb{E} \nabla_{\theta} \log P_{\theta}(\tau)b
$$

$$
= \sum_{\tau} P(\tau; \theta) \nabla_{\theta} \log P_{\theta}(\tau)b
$$
Likelihood ratio gradient estimator

For constant $b$, consider this:

$$
\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta} \log P_{\theta}(\tau^{(i)}) (R(\tau^{(i)}) - b)
$$

$$
\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta} \log P_{\theta}(\tau^{(i)}) R(\tau^{(i)}) - \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta} \log P_{\theta}(\tau^{(i)}) b
$$

$$
\mathbb{E} \nabla_{\theta} \log P_{\theta}(\tau)b
$$

$$
= \sum_{\tau} P(\tau; \theta) \nabla_{\theta} \log P_{\theta}(\tau)b
$$

$$
= \sum_{\tau} P(\tau; \theta) \frac{\nabla_{\theta} P_{\theta}(\tau)}{P(\tau; \theta)} b
$$
For constant $b$, consider this:

$$
\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta} \log P_{\theta}(\tau^{(i)}) (R(\tau^{(i)}) - b)
$$

$$
\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta} \log P_{\theta}(\tau^{(i)}) R(\tau^{(i)}) - \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta} \log P_{\theta}(\tau^{(i)}) b
$$

$$
\mathbb{E} \nabla_{\theta} \log P_{\theta}(\tau)b
$$

$$
= \sum_{\tau} P(\tau; \theta) \nabla_{\theta} \log P_{\theta}(\tau)b
$$

$$
= \sum_{\tau} P(\tau; \theta) \frac{\nabla_{\theta} P_{\theta}(\tau)}{P(\tau; \theta)} b
$$

$$
= \sum_{\tau} \nabla_{\theta} P_{\theta}(\tau)b
$$
Likelihood ratio gradient estimator

For constant $b$, consider this:

$$
\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta} \log P_{\theta}(\tau^{(i)}) (R(\tau^{(i)}) - b)
$$

$$
\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta} \log P_{\theta}(\tau^{(i)}) R(\tau^{(i)}) - \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta} \log P_{\theta}(\tau^{(i)}) b
$$

$$
\mathbb{E} \nabla_{\theta} \log P_{\theta}(\tau) b = \sum_{\tau} P(\tau; \theta) \nabla_{\theta} \log P_{\theta}(\tau) b
$$

$$
= \sum_{\tau} P(\tau; \theta) \frac{\nabla_{\theta} P_{\theta}(\tau) b}{P(\tau; \theta)}
$$

$$
= \sum_{\tau} \nabla_{\theta} P_{\theta}(\tau) b
$$

$$
= b \left( \sum_{\tau} \nabla_{\theta} P_{\theta}(\tau) \right)
$$
Likelihood ratio gradient estimator

For constant \(b\), consider this:

\[
\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta} \log P_\theta(\tau^{(i)}) (R(\tau^{(i)}) - b)
\]

\[
\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta} \log P_\theta(\tau^{(i)}) R(\tau^{(i)}) - \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta} \log P_\theta(\tau^{(i)}) b
\]

\[
\mathbb{E} \nabla_{\theta} \log P_\theta(\tau)b
\]
\[
= \sum_{\tau} P(\tau; \theta) \nabla_{\theta} \log P_\theta(\tau)b
\]
\[
= \sum_{\tau} P(\tau; \theta) \frac{\nabla_{\theta} P_\theta(\tau)}{P(\tau; \theta)} b
\]
\[
= \sum_{\tau} \nabla_{\theta} P_\theta(\tau)b
\]
\[
= b \left( \sum_{\tau} \nabla_{\theta} P_\theta(\tau) \right)
\]
\[
= b \left( \nabla_{\theta} \sum_{\tau} P_\theta(\tau) \right)
\]
Likelihood ratio gradient estimator

For constant $b$, consider this:

$$\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta} \log P_{\theta}(\tau^{(i)}) R(\tau^{(i)}) - b$$

$$\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta} \log P_{\theta}(\tau^{(i)}) R(\tau^{(i)}) - \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta} \log P_{\theta}(\tau^{(i)}) b$$

$$\mathbb{E} \nabla_{\theta} \log P_{\theta}(\tau) b$$

$$= \sum_{\tau} P(\tau; \theta) \nabla_{\theta} \log P_{\theta}(\tau) b$$

$$= \sum_{\tau} P(\tau; \theta) \frac{\nabla_{\theta} P_{\theta}(\tau)}{P(\tau; \theta)} b$$

$$= \sum_{\tau} \nabla_{\theta} P_{\theta}(\tau) b$$

$$= b \left( \sum_{\tau} \nabla_{\theta} P_{\theta}(\tau) \right)$$

$$= b \left( \nabla_{\theta} \sum_{\tau} P_{\theta}(\tau) \right)$$

$$= 0$$

We still have an unbiased estimator of the gradient!
Baseline choices

\[
\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}(\alpha_t^{(i)} | s_t^{(i)}) \left( \sum_{k=t}^{H} R(s_k^{(i)}, a_k^{(i)}) - b \right)
\]

- **Constant baseline:** \( b = \mathbb{E}[R(\tau)] \approx \sum_{i=1}^{N} R(\tau^{(i)}) \)

- **Time-dependent baseline:** \( b_t = \sum_{i=1}^{N} \sum_{k=t}^{H} R(s_k^{(i)}, a_k^{(i)}) \)

- **State-dependent expected return:**
  \[
  b(s_t) = \mathbb{E} \left[ r_t + r_{t+1} + r_{t+2} + \ldots + r_{H-1} \right] = V_\pi(s_t)
  \]
Estimate $V_\pi(S_t)$

$$\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\alpha^{(i)}_t | s^{(i)}_t) \left( \sum_{k=t}^{H} R(s^{(i)}_k, a^{(i)}_k) - V_\pi(s^{(i)}_k) \right)$$

MC estimation

Initialize $\phi$

- Collect trajectories $\tau_1, \ldots \tau_N$
- Regress against empirical return:

$$\phi_{i+1} \leftarrow \arg \min_{\phi} \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{H-1} \left( V_\phi^{\pi}(s^{(i)}_t) - \left( \sum_{k=t}^{H-1} R(s^{(i)}_k, u^{(i)}_k) \right) \right)^2$$
Algorithm 1 “Vanilla” policy gradient algorithm

Initialize policy parameter $\theta$, baseline $b$

for iteration=1, 2, ... do

Collect a set of trajectories by executing the current policy
At each timestep in each trajectory, compute
the return $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and
the advantage estimate $\hat{A}_t = R_t - b(s_t)$.  
Re-fit the baseline, by minimizing $\|b(s_t) - R_t\|^2$, 
summed over all trajectories and timesteps.
Update the policy, using a policy gradient estimate $\hat{g}$, 
which is a sum of terms $\nabla_{\theta} \log \pi(a_t \mid s_t, \theta) \hat{A}_t$

end for

~ [Williams, 1992]
Estimate $V_{\pi}(S_t)$

$$\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\alpha^{(i)}_t | s^{(i)}_t) \left( \sum_{k=t}^{H} R(s^{(i)}_k, a^{(i)}_k) - V_\pi(s^{(i)}_k) \right)$$

TD estimation

Initialize $\phi$

- Collect data $\{s, u, s', r\}$
- Fitted V iteration:
  $$\phi_{i+1} \leftarrow \min_\phi \sum_{(s,u,s',r)} \| r + V_{\phi_{i}}(s') - V_\phi(s) \|^2_2 + \lambda \| \phi - \phi_i \|^2_2$$

Bootstrapping!
Actions inherit the blame of the future return

All the random actions we did have been found bad, while they really didn’t matter.

Can I find a better estimator for the cumulative future reward, instead of the return of a single rollout?

we screwed up somewhere here

this was all hopeless and only contributes noise to the gradient
Better estimates for cumulative future reward

$$\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}(\alpha_{t}^{(i)} | s_{t}^{(i)}) \left( \sum_{k=t}^{H} R(s_{k}^{(i)}, a_{k}^{(i)}) - V_{\pi}(s_{k}^{(i)}) \right)$$

- Estimation of Q from single roll-out
  
  $$Q_{\pi}^{\pi}(s, u) = \mathbb{E}[r_0 + r_1 + r_2 + \cdots | s_0 = s, a_0 = a]$$

- = high variance per sample based / no generalization
  
  - Reduce variance by discounting
  
  - Reduce variance by function approximation (=critic)
Monte-Carlo policy gradient still has high variance

We can use a critic to estimate the action-value function:

$$Q_w(s, a) \approx Q^{\pi_\theta}(s, a)$$

Actor-critic algorithms maintain two sets of parameters

- **Critic Updates** action-value function parameters $w$
- **Actor Updates** policy parameters $\theta$, in direction suggested by critic

Actor-critic algorithms follow an approximate policy gradient
Actor-Critic

$$Q^\pi,\gamma (s, u) = \mathbb{E}[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots | s_0 = s, u_0 = u]$$
Actor-Critic

\[ Q^{\pi, \gamma}(s, u) = \mathbb{E}[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots | s_0 = s, u_0 = u] \]
\[ = \mathbb{E}[r_0 + \gamma V^\pi(s_1) | s_0 = s, u_0 = u] \]
$Q^{\pi, \gamma}(s, u) = \mathbb{E}[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots | s_0 = s, u_0 = u]$

$= \mathbb{E}[r_0 + \gamma V^{\pi}(s_1) | s_0 = s, u_0 = u]$

$= \mathbb{E}[r_0 + \gamma r_1 + \gamma^2 V^{\pi}(s_2) | s_0 = s, u_0 = u]$
Actor-Critic

\[ Q^{\pi, \gamma}(s, u) = \mathbb{E}[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots | s_0 = s, u_0 = u] \]
\[ = \mathbb{E}[r_0 + \gamma V^{\pi}(s_1) | s_0 = s, u_0 = u] \]
\[ = \mathbb{E}[r_0 + \gamma r_1 + \gamma^2 V^{\pi}(s_2) | s_0 = s, u_0 = u] \]
\[ = \mathbb{E}[r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 V^{\pi}(s_3) | s_0 = s, u_0 = u] \]
\[ = \cdots \]

- **Async Advantage Actor Critic (A3C) [Mnih et al, 2016]**
  - \( \hat{Q} \) one of the above choices (e.g. \( k=5 \) step lookahead)
Asynchronous Methods for Deep Reinforcement Learning

Volodymyr Mnih
Adrià Puigdomènech Badia
Mehdi Mirza
Alex Graves
Tim Harley
Timothy P. Lillicrap
David Silver
Koray Kavukcuoglu

1 Google DeepMind
2 Montreal Institute for Learning Algorithms (MILA), University of Montreal

VMNIH@GOOGLE.COM
ADRIAP@GOOGLE.COM
MIRZAMOM@IRO.UMONTREAL.CA
GRAVESA@GOOGLE.COM
THARLEY@GOOGLE.COM
COUNTZERO@GOOGLE.COM
DAVIDSILVER@GOOGLE.COM
KORAYK@GOOGLE.COM
Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors \( \theta \) and \( \theta_v \) and global shared counter \( T = 0 \)
// Assume thread-specific parameter vectors \( \theta' \) and \( \theta'_v \)
Initialize thread step counter \( t \leftarrow 1 \)

repeat
  Reset gradients: \( d\theta \leftarrow 0 \) and \( d\theta_v \leftarrow 0 \).
  Synchronize thread-specific parameters \( \theta' = \theta \) and \( \theta'_v = \theta_v \)
  \( t_{\text{start}} = t \)
  Get state \( s_t \)
  repeat
    Perform \( a_t \) according to policy \( \pi(a_t|s_t; \theta') \)
    Receive reward \( r_t \) and new state \( s_{t+1} \)
    \( t \leftarrow t + 1 \)
    \( T \leftarrow T + 1 \)
  until terminal \( s_t \) or \( t - t_{\text{start}} = t_{\text{max}} \)
  \( R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \end{cases} \) // Bootstrap from last state
for \( i \in \{t - 1, \ldots, t_{\text{start}}\} \) do
  \( R \leftarrow r_i + \gamma R \)
  Accumulate gradients wrt \( \theta' \): \( d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i; \theta')(R - V(s_i; \theta'_v)) \)
  Accumulate gradients wrt \( \theta'_v \): \( d\theta_v \leftarrow d\theta_v + \partial (R - V(s_i; \theta'_v))^2 / \partial \theta'_v \)
end for
Perform asynchronous update of \( \theta \) using \( d\theta \) and of \( \theta_v \) using \( d\theta_v \).
until \( T > T_{\text{max}} \)

What is the approximation used for the advantage?

\[
\begin{align*}
R_3 &= r_3 + \gamma V(s_4, \theta'_v) \\
A_3 &= R_3 - V(s_3; \theta'_v) \\
R_2 &= r_2 + \gamma r_3 + \gamma^2 V(s_4, \theta'_v) \\
A_2 &= R_2 - V(s_2; \theta'_v)
\end{align*}
\]
Distributed RL
Distributed Asynchronous RL

The actor critic trained in such asynchronous way is known as A3C.
Distributed Synchronous RL

5. Gradients of all workers are averaged and the central neural net weights are updated

4. Worker gets gradients from losses

3. Worker calculates value and policy loss

2. Worker interacts with environment

1. Worker reset to global network

All worker may have the same actor/critic weights

The actor critic trained in such synchronous way is known as A2C
Advantages of Asynchronous (multi-threaded) RL