ECE 1508: Reinforcement Learning

Chapter 5: RL via Policy Gradient

Ali Bereyhi

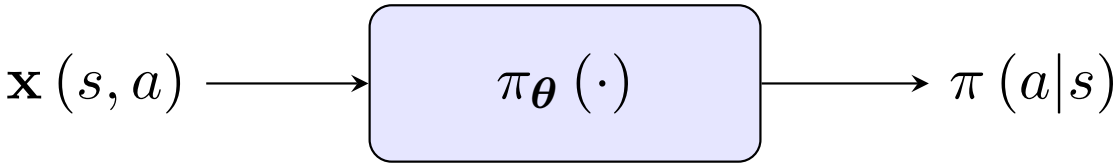
ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

University of Toronto

Summer 2024

Policy Network



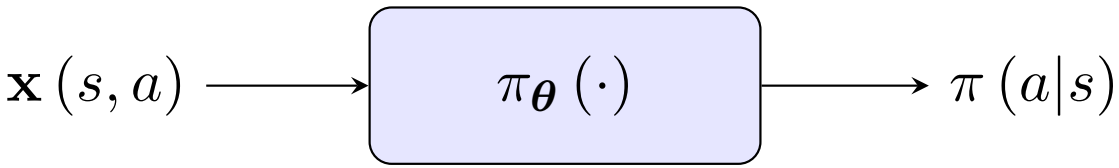
Policy networks are used in two sets of deep RL approaches

- *Policy gradient approaches*
  - ↳ *We do not use a value network and directly approximate optimal policy*
- *Actor-critic approaches*
  - ↳ *We keep the value network to examine the approximated optimal policy*
  - ↳ *This is the most practically-robust approach we can use*

Policy Network

Policy Network

Policy network is an approximation model that maps state-action features to a conditional probability distribution



+ *How can we realize such a network? It is not any network! It should return probabilities!*
– Yes! That's right! Let's see a few examples

Recall: *Feature*

Feature Representation of State-Actions

Feature representation maps each state-action pair into a vector of features that correspond to that state and action, i.e.,

$$\mathbf{x}(\cdot) : \mathbb{S} \times \mathbb{A} \mapsto \mathbb{R}^J$$

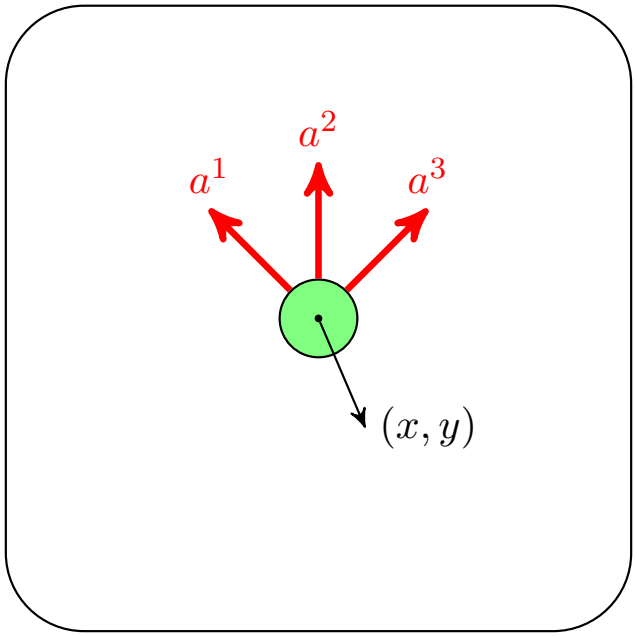
for some integer J that is the feature dimension

Attention

Note that are now in the most general case: states and actions can be either discrete or continuous

Example: *Moving Particle*

We are controlling a moving particle that could move in the 2D space

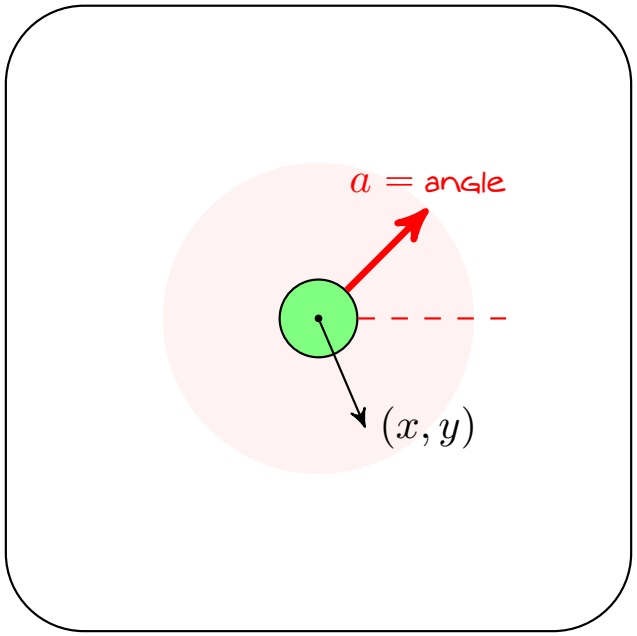


We can set the feature vector

$$\mathbf{x}(s, a) = \begin{bmatrix} x \\ y \\ a \end{bmatrix}$$

Example: *Moving Particle*

We have the same moving particle that could move in any direction



We can set the feature vector

$$\mathbf{x}(s, a) = \begin{bmatrix} x \\ y \\ a \end{bmatrix}$$

New Notation

+ *Shall we see now an example of a policy network?*
– Sure! Just last point to mention before

New Notation

As we think of a generic action and state space, we use a simple notation

$$\int_a f(a) = \begin{cases} \sum_{a \in \mathbb{A}} f(a) & \text{discrete } a \\ \int_{\mathbb{A}} f(a)\,\mathrm{d}a & \text{continuous } a \end{cases}$$

Example: *Softmax*

The most basic example is to assume a linear mapping

$$\pi_{\boldsymbol{\theta}}(a|s) = \boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}(s,a)$$

+ *But how can we guarantee that it returns a probability?! Shall we assume*

$$\int_a \pi_{\boldsymbol{\theta}}(a|s) = \int_a \boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}(s,a) = 1$$

– *Well! We can do that, but there is a better way to convert a linear function into a probability distribution*

Example: *Softmax*

Softmax

Softmax is a vector-activated neuron that maps input $\mathbf{x}(s, a)$ *into*

$$Soft^{\boldsymbol{\theta}}_{\max}(\mathbf{x}(s, a)) = \frac{\exp\left\{\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}(s, a)\right\}}{\int_a \exp\left\{\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}(s, a)\right\}}$$

We can now simply set

$$\pi_{\boldsymbol{\theta}}(a|s) = Soft^{\boldsymbol{\theta}}_{\max}(\mathbf{x}(s, a))$$

As we are going to have

$$\int_a \pi_{\boldsymbol{\theta}}(a|s) = \int_a Soft^{\boldsymbol{\theta}}_{\max}(\mathbf{x}(s, a)) = 1$$

Example: *Gaussian*

Another approach is to use a Gaussian policy that is controllable with some parameters: say at state s we only look at the state representation $\mathbf{x}(s)$

$$\pi_{\boldsymbol{\theta}}(a|s) \equiv \mathcal{N}\left(\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}(s), \sigma^2\right)$$
$$= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{\left(a - \boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}(s)\right)^2}{2\sigma^2}\right\}$$

We may train this network by

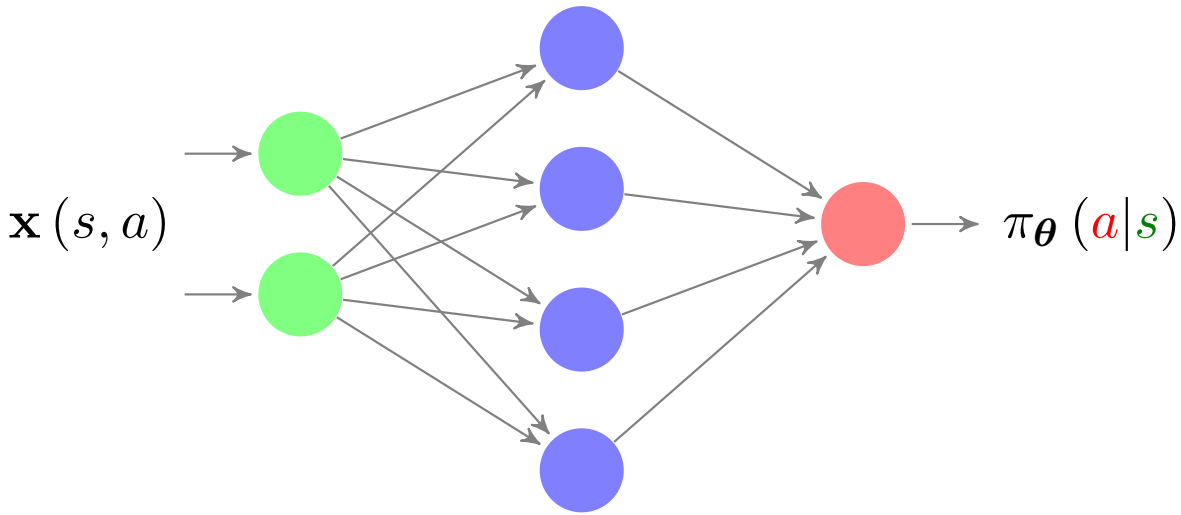
- *either only learning $\boldsymbol{\theta}$*
- *or learning both $\boldsymbol{\theta}$ and σ^2*

Example: *DPN*

In practice, we are more interested to train

$$\textit{Deep Policy Network} \equiv \textit{DPN}$$

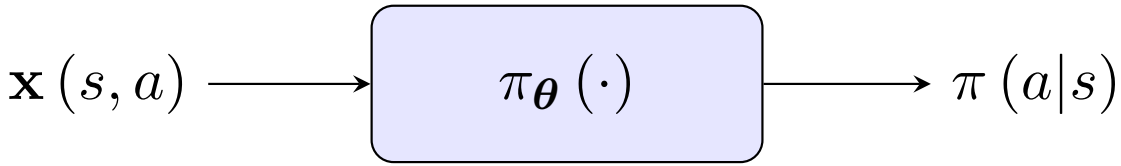
as it can learn a richer class of policies



And we very well know how to make it return a probability distribution!

Training Policy Network

Let's now train the policy network: *assume a general network as*



+ *How can we train it? What should be the loss?*
– *Well! We know what we want?*

We want ro have a policy that maximizes value at all states, i.e.,

$$\boldsymbol{\theta}^{\star} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, v_{\pi_{\boldsymbol{\theta}}}(s)$$

for all states $s \in \mathbb{S}$

Since we are more happy with minimization we can alternatively say ☺

$$\boldsymbol{\theta}^{\star} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} -v_{\pi_{\boldsymbol{\theta}}}(s)$$

Training Policy Network

+ *But that is weird! We have so many states! For which one we should do it?!*
– That's right! We should find a way around it

This naive training reduces to a multi-objective optimization

$$\boldsymbol{\theta}^{\star} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} -v_{\pi_{\boldsymbol{\theta}}}(s)$$

with the number of objectives being as much as the number of states!

Say we have N states: we need to have simultaneously

$$\boldsymbol{\theta}^{\star} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} -v_{\pi_{\boldsymbol{\theta}}}\left(s^1\right) \qquad \ldots \qquad \boldsymbol{\theta}^{\star} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} -v_{\pi_{\boldsymbol{\theta}}}\left(s^N\right)$$

which is not necessarily possible!

Finding Loss

A classical remedy to such multi-objective optimization is to scalarize

$$\boldsymbol{\theta}^{\star} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} -p\left(s^1\right) v_{\pi_{\boldsymbol{\theta}}}\left(s^1\right) - \ldots - p\left(s^N\right) v_{\pi_{\boldsymbol{\theta}}}\left(s^N\right)$$

Or better to say: to minimize the average return over all states, i.e.,

$$\begin{aligned} \mathcal{J}\left(\pi_{\boldsymbol{\theta}}\right) &= \int_{s} v_{\pi_{\boldsymbol{\theta}}}(s)\, p(s) \\ &= \mathbb{E}_{S \sim p}\left\{v_{\pi_{\boldsymbol{\theta}}}(S)\right\} \end{aligned}$$

+ *OK! But what is $p(s)$?! Do we have it? Or shall we assume it?*
– *Neither and both ☺ Let's try a simple setting first*

Finding Loss

Let's consider a simple case: *we have an episodic environment whose a sample trajectory looks like*

$$\tau : S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

We denote the whole trajectory by τ to keep our notation simple

---

Assume we have no discount; then, we could say that a sample return is

$$G_0 = \sum_{t=0}^{T-1} R_{t+1}$$

and that the value for sample state S_0

$$v_{\pi_{\boldsymbol{\theta}}}(S_0) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\{G_0 | S_0\}$$

Finding Loss

Say we fix our starting state to $S_0 = s_0$: *we get a sample trajectory as*

$$\tau(s_0) : s_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

The value of the starting state s_0 is then given by

$$\begin{aligned} v_{\pi_{\boldsymbol{\theta}}}(s_0) &= \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\{G_0|s_0\} \\ &= \int_{r_1,\ldots,r_T} \int_{s_1,\ldots,s_T} \int_{a_0,\ldots,a_{T-1}} \left(\sum_{t=0}^{T-1} r_{t+1}\right) \pi_{\boldsymbol{\theta}}(a_0|s_0)\, p(s_1, r_1|s_0, a_0) \\ &\qquad \ldots (a_{T-1}|s_{T-1})\, p(s_T, r_T|s_{T-1}, a_{T-1}) \\ &= \int_{\tau(s_0)} \left(\sum_{t=0}^{T-1} r_{t+1}\right) \prod_{t=0}^{T-1} \pi_{\boldsymbol{\theta}}(a_t|s_t)\, p(s_{t+1}, r_{t+1}|s_t, a_t) \end{aligned}$$

Finding Loss

Say we fix our starting state to $S_0 = s_0$: *we get a sample trajectory as*

$$\tau(s_0) : s_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

Let's define the return of this trajectory as

$$g(\tau(s_0)) = \sum_{t=0}^{T-1} r_{t+1}$$

This an outcome of random variable

$$G(\tau(s_0)) = \sum_{t=0}^{T-1} R_{t+1}$$

We can now write

$$v_{\pi_{\boldsymbol{\theta}}}(s_0) = \int_{\tau(s_0)} g(\tau(s_0)) \prod_{t=0}^{T-1} \pi_{\boldsymbol{\theta}}(a_t|s_t)\, p(s_{t+1}, r_{t+1}|s_t, a_t)$$

Finding Loss

Say we fix our starting state to $S_0 = s_0$: *we get a sample trajectory as*

$$\tau(s_0) : s_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

Note that we could look at this term as an expectation

$$\begin{aligned} v_{\pi_{\boldsymbol{\theta}}}(s_0) &= \int_{\tau(s_0)} g(\tau(s_0)) \prod_{t=0}^{T-1} \pi_{\boldsymbol{\theta}}(a_t|s_t)\, p(s_{t+1}, r_{t+1}|s_t, a_t) \\ &= \mathbb{E}_{\tau(s_0)\sim\prod_{t=0}^{T-1} \pi_{\boldsymbol{\theta}}(a_t|s_t)p(s_{t+1},r_{t+1}|s_t,a_t)} \{G(\tau(s_0))\} \end{aligned}$$

Initial Conclusion

Distribution of $\tau(s_0)$ for a given s_0 which includes all next states is specified by policy and environment

Finding Loss

Now, let's assume a randomly chosen starting state S_0: *then, we have*

$$\tau : S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

We choose it with some distribution $p(s_0)$; thus, we have

$$\begin{aligned}
\mathcal{J}(\pi_{\boldsymbol{\theta}}) &= \mathbb{E}_{S_0 \sim p}\left\{v_{\pi_{\boldsymbol{\theta}}}(S_0)\right\} \\
&= \int_{s_0} \int_{\tau(s_0)} \mathrm{g}(\tau(s_0))\, p(s_0) \prod_{t=0}^{T-1} \pi_{\boldsymbol{\theta}}(a_t|s_t)\, p(s_{t+1}, r_{t+1}|s_t, a_t) \\
&= \int_{\tau} \mathrm{g}(\tau)\, p(s_0) \prod_{t=0}^{T-1} \pi_{\boldsymbol{\theta}}(a_t|s_t)\, p(s_{t+1}, r_{t+1}|s_t, a_t)
\end{aligned}$$

average over all possible trajectories

Finding Loss: *Estimating Form*

Now, let's define the overall distribution of trajectory τ as

$$p_{\pi_{\boldsymbol{\theta}}}(\tau) = p(s_0) \prod_{t=0}^{T-1} \pi_{\boldsymbol{\theta}}(a_t|s_t)\, p(s_{t+1}, r_{t+1}|s_t, a_t)$$

Then we could compute the average return of the environment as

$$\begin{aligned} \mathcal{J}(\pi_{\boldsymbol{\theta}}) &= \int_{\tau} g(\tau)\, p_{\pi_{\boldsymbol{\theta}}}(\tau) \\ &= \mathbb{E}_{\tau \sim p_{\pi_{\boldsymbol{\theta}}}} \{G(\tau)\} \end{aligned}$$

return of trajectory

distribution of trajectory

Final Conclusion

Part of distribution of τ is assumed and remaining by policy and environment

Training Policy Network: *Gradient Descent*

We have the loss ready: *let's start training the policy network*

+ *What do you mean by training?*
– *Simply, we want to find the network parameters that minimize the loss*

$$\boldsymbol{\theta}^{\star} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} -\mathcal{J}\left(\pi_{\boldsymbol{\theta}}\right)$$

We can use gradient descent: we consider learning rate α and update as

$$\begin{aligned}\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \alpha\nabla\left\{-\mathcal{J}\left(\pi_{\boldsymbol{\theta}}\right)\right\}\\ &\leftarrow \boldsymbol{\theta} + \alpha\nabla\mathcal{J}\left(\pi_{\boldsymbol{\theta}}\right)\end{aligned}$$

So, we need to compute $\nabla\mathcal{J}\left(\pi_{\boldsymbol{\theta}}\right)$ with respect to $\boldsymbol{\theta}$

Training Policy Network: *Gradient Descent*

We are using gradient descent (ascent)

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla \mathcal{J}\left(\pi_{\boldsymbol{\theta}}\right)$$

and we need the gradient: so, we write

$$\nabla \mathcal{J}\left(\pi_{\boldsymbol{\theta}}\right) = \nabla \int_{\tau} \mathrm{g}\left(\tau\right) p_{\pi_{\boldsymbol{\theta}}}\left(\tau\right) = \int_{\tau} \mathrm{g}\left(\tau\right) \nabla p_{\pi_{\boldsymbol{\theta}}}\left(\tau\right)$$

$$= \int_{\tau} \mathrm{g}\left(\tau\right) \nabla \left\{ \prod_{t=0}^{T-1} \pi_{\boldsymbol{\theta}}\left(a_t | s_t\right) p\left(s_{t+1}, r_{t+1} | s_t, a_t\right) p\left(s_0\right) \right\}$$

- \+ *It looks challenging!*
- – *Let's take a deeper look*

Training Policy Network: *Gradient Descent*

There is a trick that might help us in this respect

the so-called log-derivative trick

Log-Derivative Trick

For any positive function $f(\cdot) : \mathbb{R}^J \mapsto \mathbb{R}_+$ *we have by definition*

$$\nabla f(\boldsymbol{\theta}) = f(\boldsymbol{\theta}) \nabla \log f(\boldsymbol{\theta})$$

Let's apply the log-derivative trick to our problem

Training Policy Network: *Gradient Descent*

Applying the log-derivative trick to our problem, we have

$$\begin{aligned}
\nabla \mathcal{J}\left(\pi_{\boldsymbol{\theta}}\right) &= \int_{\tau} \mathrm{g}\left(\tau\right) \nabla p_{\pi_{\boldsymbol{\theta}}}\left(\tau\right) \\
&= \int_{\tau} \mathrm{g}\left(\tau\right) p_{\pi_{\boldsymbol{\theta}}}\left(\tau\right) \nabla \log p_{\boldsymbol{\theta}}\left(\tau\right) \\
&= \int_{\tau} \left[\mathrm{g}\left(\tau\right) \nabla \log p_{\pi_{\boldsymbol{\theta}}}\left(\tau\right)\right] p_{\pi_{\boldsymbol{\theta}}}\left(\tau\right) = \mathbb{E}_{\tau \sim p_{\pi_{\boldsymbol{\theta}}}} \left\{G\left(\tau\right) \nabla \log p_{\pi_{\boldsymbol{\theta}}}\left(\tau\right)\right\}
\end{aligned}$$

+ *Why should that be helpful?!*
- *Let's see how* $\log p_{\pi_{\boldsymbol{\theta}}}\left(\tau\right)$ *looks*

Training Policy Network: *Gradient Descent*

Consider one instant trajectory: *we have a particular outcome*

$$\tau : s_0, a_0 \xrightarrow{r_1} s_1, a_1 \xrightarrow{r_2} \cdots \xrightarrow{r_{T-1}} s_{T-1}, a_{T-1} \xrightarrow{r_T} s_T$$

Using the definition of $p_{\boldsymbol{\theta}}(\tau)$*, we can write*

$$\log p_{\pi_{\boldsymbol{\theta}}}(\tau) = \log \left\{ p(s_0) \prod_{t=0}^{T-1} \pi_{\boldsymbol{\theta}}(a_t|s_t)\, p(s_{t+1}, r_{t+1}|s_t, a_t) \right\}$$

$$= \underbrace{\log p(s_0)}_{\text{does not depend in } \boldsymbol{\theta}} + \sum_{t=0}^{T-1} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) + \underbrace{\sum_{t=0}^{T-1} \log p(s_{t+1}, r_{t+1}|s_t, a_t)}_{\text{does not depend in } \boldsymbol{\theta}}$$

Training Policy Network: *Gradient Descent*

Consider one instant trajectory: *we have a particular outcome*

$$\tau : s_0, a_0 \xrightarrow{r_1} s_1, a_1 \xrightarrow{r_2} \cdots \xrightarrow{r_{T-1}} s_{T-1}, a_{T-1} \xrightarrow{r_T} s_T$$

The gradient of $\log p_{\boldsymbol{\theta}}(\tau)$ *is hence given by*

$$\nabla \log p_{\pi_{\boldsymbol{\theta}}}(\tau) = \nabla \sum_{t=0}^{T-1} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) = \sum_{t=0}^{T-1} \nabla \log \pi_{\boldsymbol{\theta}}(a_t|s_t)$$

If we have a random sample trajectory

$$\tau : S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

we can similarly write

$$\nabla \log p_{\pi_{\boldsymbol{\theta}}}(\tau) = \sum_{t=0}^{T-1} \nabla \log \pi_{\boldsymbol{\theta}}(A_t|S_t)$$

Training Policy Network: *Gradient Descent*

Back to our main problem: *we have a random trajectory*

$$\tau : S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

and want to find the gradient of loss; so, we can write

$$\begin{aligned}
\nabla \mathcal{J}(\pi_{\boldsymbol{\theta}}) &= \mathbb{E}_{\tau \sim p_{\pi_{\boldsymbol{\theta}}}} \{ G(\tau) \nabla \log p_{\pi_{\boldsymbol{\theta}}}(\tau) \} \\
&= \mathbb{E}_{\tau \sim p_{\pi_{\boldsymbol{\theta}}}} \left\{ G(\tau) \sum_{t=0}^{T-1} \nabla \log \pi_{\boldsymbol{\theta}}(A_t | S_t) \right\} \\
&= \mathbb{E}_{\tau \sim p_{\pi_{\boldsymbol{\theta}}}} \left\{ \left(\sum_{t=0}^{T-1} R_{t+1} \right) \sum_{t=0}^{T-1} \nabla \log \pi_{\boldsymbol{\theta}}(A_t | S_t) \right\}
\end{aligned}$$

We can estimate it via Monte-Carlo!

Training Policy Network: *SGD*

Say we set the weights of policy network to $\boldsymbol{\theta}$: *we sample* K *trajectories*

$$\tau : S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

from the environment using policy $\pi_{\boldsymbol{\theta}}$*, and then estimate the gradient as*

$$\hat{\nabla}\mathcal{J}\left(\pi_{\boldsymbol{\theta}}\right) = \frac{1}{K}\sum_{k=1}^{K} G\left(\tau_k\right) \sum_{t=0}^{T-1} \nabla \log \pi_{\boldsymbol{\theta}}\left(A_t\left[k\right] | S_t\left[k\right]\right)$$

We can then use gradient descent to update $\boldsymbol{\theta}$ *as*

$$\begin{aligned}
\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \alpha \hat{\nabla}\mathcal{J}\left(\pi_{\boldsymbol{\theta}}\right) \\
&\leftarrow \boldsymbol{\theta} + \frac{\alpha}{K}\sum_{k=1}^{K} G\left(\tau_k\right) \sum_{t=0}^{T-1} \nabla \log \pi_{\boldsymbol{\theta}}\left(A_t\left[k\right] | S_t\left[k\right]\right)
\end{aligned}$$

Training Policy Network: *SGD*

+ *Isn't that again too slow?! We should wait for a single update!*
- Sure! We can go for SGD

Using SGD, we could take a single sample gradient

$$\hat{\nabla}\mathcal{J}(\pi_{\boldsymbol{\theta}}) = G(\tau)\sum_{t=0}^{T-1}\nabla\log\pi_{\boldsymbol{\theta}}(A_t|S_t)$$

and then update the policy network as

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha G(\tau)\sum_{t=0}^{T-1}\nabla\log\pi_{\boldsymbol{\theta}}(A_t|S_t)$$

First Policy Gradient Algorithm

`PG_v1():`

1: *Initiate with* $\boldsymbol{\theta}$ *and learning rate* α
2: ***for*** *episode* $= 1 : K$ ***do***
3: *Sample a trajectory with policy* $\pi_{\boldsymbol{\theta}}$

$$\tau : S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

4: *Compute return* $G(\tau)$
5: ***for*** $t = 0 : T - 1$ ***do***
6: *Update policy network* $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha G(\tau) \nabla \log \pi_{\boldsymbol{\theta}}(A_t | S_t)$
7: ***end for***
8: ***end for***

+ *Is it a kind of known algorithm?*
– With a bit of modification *it reduces to REINFORCE algorithm proposed by Ronald J. Williams in 1992*

REINFORCE: *First Official Algorithm*

`REINFORCE():`

1: *Initiate with* $\boldsymbol{\theta}$ *and learning rate* α
2: ***for*** *episode* $= 1 : K$ ***do***
3: *Sample a trajectory with policy* $\pi_{\boldsymbol{\theta}}$

$$\tau : S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

4: ***for*** $t = 0 : T - 1$ ***do***
5: *Update policy network* $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha G_t \nabla \log \pi_{\boldsymbol{\theta}} (A_t | S_t)$
6: ***end for***
7: ***end for***

+ *But we are now computing a different gradient? Why should it work?!*
– This is because of the *Policy Gradient Theorem* *which says that we should update proportional to* $\nabla \log \pi_{\boldsymbol{\theta}} (A_t | S_t)$

SGD: *General Setting*

Let's have a more generic analysis: *assume we start at a random state S_0 that is chosen according to*

$$S_0 \sim p(s_0)$$

We start acting via the policy $\pi_{\boldsymbol{\theta}}$ and transit to a new state

$$S_0, A_0 \xrightarrow{R_1} S_1$$

We could then say that the average value of the policy is

$$\mathcal{J}(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{S_0 \sim p}\{v_{\pi_{\boldsymbol{\theta}}}(S_0)\}$$

We need the gradient of this value against $\boldsymbol{\theta}$ to train the policy network

SGD: *General Setting*

We can open up the loss expression

$$\mathcal{J}\left(\pi_{\boldsymbol{\theta}}\right) = \int_{s_0} v_{\pi_{\boldsymbol{\theta}}}\left(s_0\right) p\left(s_0\right)$$

and write the gradient as

$$\begin{aligned}\nabla \mathcal{J}\left(\pi_{\boldsymbol{\theta}}\right) &= \nabla \int_{s} v_{\pi_{\boldsymbol{\theta}}}\left(s_0\right) p\left(s_0\right) \\ &= \int_{s} \nabla v_{\pi_{\boldsymbol{\theta}}}\left(s_0\right) p\left(s_0\right)\end{aligned}$$

Let's compute $\nabla v_{\pi_{\boldsymbol{\theta}}}\left(s_0\right)$

SGD: *General Setting*

We can use the marginalization rule to expand $v_{\pi_{\boldsymbol{\theta}}}(s_0)$

$$v_{\pi_{\boldsymbol{\theta}}}(s_0) = \int_{a_0} q_{\pi_{\boldsymbol{\theta}}}(s_0, a_0)\, \pi_{\boldsymbol{\theta}}(a_0|s_0)$$

So the gradient $\nabla v_{\pi_{\boldsymbol{\theta}}}(s_0)$ *is computed using chain rule as*

$$\begin{aligned}\nabla v_{\pi_{\boldsymbol{\theta}}}(s_0) &= \nabla \int_{a_0} q_{\pi_{\boldsymbol{\theta}}}(s_0, a_0)\, \pi_{\boldsymbol{\theta}}(a_0|s_0) \\ &= \int_{a_0} \nabla q_{\pi_{\boldsymbol{\theta}}}(s_0, a_0)\, \pi_{\boldsymbol{\theta}}(a_0|s_0) + \int_{a_0} q_{\pi_{\boldsymbol{\theta}}}(s_0, a_0)\, \nabla \pi_{\boldsymbol{\theta}}(a_0|s_0)\end{aligned}$$

Let's compute $\nabla q_{\pi_{\boldsymbol{\theta}}}(s_0, a_0)$ *next*

SGD: *General Setting*

We can use Bellman equation to expand $q_{\pi_{\boldsymbol{\theta}}}(s_0, a_0)$ *as*

$$q_{\pi_{\boldsymbol{\theta}}}(s_0, a_0) = \mathcal{R}(s_0, a_0) + \gamma \int_{s_1} v_{\pi_{\boldsymbol{\theta}}}(s_1)\, p(s_1|s_0, a_0)$$

So the gradient reads

$$\begin{aligned}
\nabla q_{\pi_{\boldsymbol{\theta}}}(s_0, a_0) &= \nabla \left\{ \mathcal{R}(s_0, a_0) + \gamma \int_{s_1} v_{\pi_{\boldsymbol{\theta}}}(s_1)\, p(s_1|s_0, a_0) \right\} \\
&= \underbrace{\nabla \mathcal{R}(s_0, a_0)}_{0} + \gamma \int_{s_1} \nabla v_{\pi_{\boldsymbol{\theta}}}(s_1)\, p(s_1|s_0, a_0) \\
&= \gamma \int_{s_1} \nabla v_{\pi_{\boldsymbol{\theta}}}(s_1)\, p(s_1|s_0, a_0)
\end{aligned}$$

SGD: *General Setting*

Now, let's put back all gradients gradually towards beginning of computation

$$\begin{aligned}\nabla \mathcal{J}(\pi_{\boldsymbol{\theta}}) &= \int_{s_0} \nabla v_{\pi_{\boldsymbol{\theta}}}(s_0)\, p(s_0) \\ &= \int_{s_1} \nabla v_{\pi_{\boldsymbol{\theta}}}(s_1)\, p_{\pi_{\boldsymbol{\theta}}}(s_1) + \int_{s_0}\int_{a_0} q_{\pi_{\boldsymbol{\theta}}}(s_0, a_0)\, \nabla \pi_{\boldsymbol{\theta}}(a_0|s_0)\, p(s_0)\end{aligned}$$

where we define the marginal distribution of s_1 as

$$p_{\pi_{\boldsymbol{\theta}}}(s_1) = \int_{s_0}\int_{a_0} p(s_1|s_0, a_0)\, \pi_{\boldsymbol{\theta}}(a_0|s_0)\, p(s_0)$$

SGD: *General Setting*

Since $p_{\pi_{\boldsymbol{\theta}}}(s_1)$ and $p(s_0)$ are distributions, we have

$$\int_{s_1} p_{\pi_{\boldsymbol{\theta}}}(s_1) = \int_{s_0} p(s_0) = 1$$

So, we could modify our final expression as

$$\begin{aligned}
\nabla \mathcal{J}(\pi_{\boldsymbol{\theta}}) = & \int_{s_1} \nabla v_{\pi_{\boldsymbol{\theta}}}(s_1)\, p_{\pi_{\boldsymbol{\theta}}}(s_1) \int_{s_0} p(s_0) \\
& + \int_{s_0}\int_{a_0} q_{\pi_{\boldsymbol{\theta}}}(s_0, a_0)\, \nabla \pi_{\boldsymbol{\theta}}(a_0|s_0)\, p(s_0) \int_{s_1} p_{\pi_{\boldsymbol{\theta}}}(s_1) \\
= & \int_{s_1}\int_{s_0} p_{\pi_{\boldsymbol{\theta}}}(s_1)\, p(s_0) \left[\nabla v_{\pi_{\boldsymbol{\theta}}}(s_1) + \int_{a_0} q_{\pi_{\boldsymbol{\theta}}}(s_0, a_0)\, \nabla \pi_{\boldsymbol{\theta}}(a_0|s_0) \right]
\end{aligned}$$

SGD: *General Setting*

If we keep on progressing in the trajectory as $t \to \infty$*, we will see*

$$\nabla \mathcal{J}(\pi_{\boldsymbol{\theta}}) = \int_s d_{\pi_{\boldsymbol{\theta}}}(s) \int_a q_{\pi_{\boldsymbol{\theta}}}(s,a) \nabla \pi_{\boldsymbol{\theta}}(a|s)$$

for some distribution $d_{\pi_{\boldsymbol{\theta}}}(s)$ *that is the average marginal distribution of states under policy* $\pi_{\boldsymbol{\theta}}$*, i.e.,*

$$\int_s d_{\pi_{\boldsymbol{\theta}}}(s) = 1$$

Finally, using the log-derivative trick we have

$$\nabla \mathcal{J}(\pi_{\boldsymbol{\theta}}) = \int_s d_{\pi_{\boldsymbol{\theta}}}(s) \int_a q_{\pi_{\boldsymbol{\theta}}}(s,a) \pi_{\boldsymbol{\theta}}(a|s) \nabla \log \pi_{\boldsymbol{\theta}}(a|s)$$

Policy Gradient Theorem

This can be equivalently written as

$$\nabla \mathcal{J}(\pi_{\boldsymbol{\theta}}) = \int_s \int_a d_{\pi_{\boldsymbol{\theta}}}(s)\, \pi_{\boldsymbol{\theta}}(a|s) q_{\pi_{\boldsymbol{\theta}}}(s,a)\, \nabla \log \pi_{\boldsymbol{\theta}}(a|s)$$
$$= \mathbb{E}_{S\sim d_{\pi_{\boldsymbol{\theta}}}, A|S\sim\pi_{\boldsymbol{\theta}}} \{q_{\pi_{\boldsymbol{\theta}}}(S,A)\, \nabla \log \pi_{\boldsymbol{\theta}}(A|S)\}$$

which concludes the policy gradient theorem proved by Sutton et al. in 1992

Policy Gradient Theorem

For a policy network with non-zero probabilities, the gradient of the average trajectory return is always given by

$$\nabla \mathcal{J}(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{S\sim d_{\pi_{\boldsymbol{\theta}}}, A|S\sim\pi_{\boldsymbol{\theta}}} \{q_{\pi_{\boldsymbol{\theta}}}(S,A)\, \nabla \log \pi_{\boldsymbol{\theta}}(A|S)\}$$

Policy Gradient Theorem: *Implication*

Policy Gradient Theorem

For a policy network with non-zero probabilities, the gradient of the average trajectory return is always given by

$$\nabla \mathcal{J}(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{S \sim d_{\pi_{\boldsymbol{\theta}}}, A|S \sim \pi_{\boldsymbol{\theta}}} \{ q_{\pi_{\boldsymbol{\theta}}}(S, A) \nabla \log \pi_{\boldsymbol{\theta}}(A|S) \}$$

+ *OK! That sounds nice! But what is special about it?!*
– It says *to train a policy network, you only need gradient of log likelihood*
+ *Then what?!*
– *Well! We could have much more complicated terms! We will talk about it more in the next sections*

Policy Gradient Theorem: *Point of Departure*

Policy Gradient Theorem

For a policy network with non-zero probabilities, the gradient of the average trajectory return is always given by

$$\nabla \mathcal{J}(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{S \sim d_{\pi_{\boldsymbol{\theta}}}, A|S \sim \pi_{\boldsymbol{\theta}}} \{ q_{\pi_{\boldsymbol{\theta}}}(S, A) \nabla \log \pi_{\boldsymbol{\theta}}(A|S) \}$$

Policy gradient theorem is the base of

$$\textit{Policy Gradient Methods} \equiv \textit{PGM}$$

It gives a feasible approach for training a policy network; however, depending on how we use it we can end up with various PGMs

PGMs in Nutshell

Policy Gradient Theorem

For a policy network with non-zero probabilities, the gradient of the average trajectory return is always given by

$$\nabla \mathcal{J}\left(\pi_{\boldsymbol{\theta}}\right) = \mathbb{E}_{S \sim d_{\pi_{\boldsymbol{\theta}}}, A|S \sim \pi_{\boldsymbol{\theta}}} \left\{ q_{\pi_{\boldsymbol{\theta}}}\left(S, A\right) \nabla \log \pi_{\boldsymbol{\theta}}\left(A|S\right) \right\}$$

PGMs can roughly divided into three classes

1. *Vanilla PGM estimates* $q_{\pi_{\boldsymbol{\theta}}}\left(S, A\right)$ *and* $\nabla \log \pi_{\boldsymbol{\theta}}\left(A|S\right)$ *via sampling*
2. *Baseline PGM that reduces estimation variance by temporal unbiasing trick*
3. *Trust region PGM enables reuse of older samples to improve efficiency*
   - ↳ *This is what we learn in the next section of this chapter*

We are going through them in the same order!

Vanilla PGM: *Basic SGD*

Vanilla PGM is pretty straightforward: *sample environment with a trajectory and train policy network via SGD using result of policy gradient theorem*

- *Use SGD to update $\boldsymbol{\theta}$ in each time, i.e., update as $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla \mathcal{J}(\pi_{\boldsymbol{\theta}})$*
- *Compute gradient using policy gradient theorem*

$$\nabla \mathcal{J}(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{S \sim d_{\pi_{\boldsymbol{\theta}}}, A|S \sim \pi_{\boldsymbol{\theta}}} \{ q_{\pi_{\boldsymbol{\theta}}}(S, A) \nabla \log \pi_{\boldsymbol{\theta}}(A|S) \}$$

- *Estimate the gradient via individual samples, i.e.,*

$$\hat{\nabla} \mathcal{J}(\pi_{\boldsymbol{\theta}}) = Q_t \nabla \log \pi_{\boldsymbol{\theta}}(A_t|S_t)$$

where Q_t is an estimator of action-value at pair (S_t, A_t) in sample

$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

Vanilla PGM: *Generic Form*

`VanillaPGM()`:

1: *Initiate with* $\boldsymbol{\theta}$ *and learning rate* α
2: *Use a* ***Q-estimator*** `QEst()`
3: ***while*** *interacting* ***do***
4: *Sample the environment with policy* $\pi_{\boldsymbol{\theta}}$

$$S_0, A_0 \xrightarrow{R_1} S_{t+1}, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

5: ***for*** $t = 0 : T-1$ ***do***
6: Set $Q_t =$ `QEst`(S_t, A_t)
7: Update policy network $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha Q_t \nabla \log \pi_{\boldsymbol{\theta}} (A_t | S_t)$
8: ***end for***
9: ***end while***

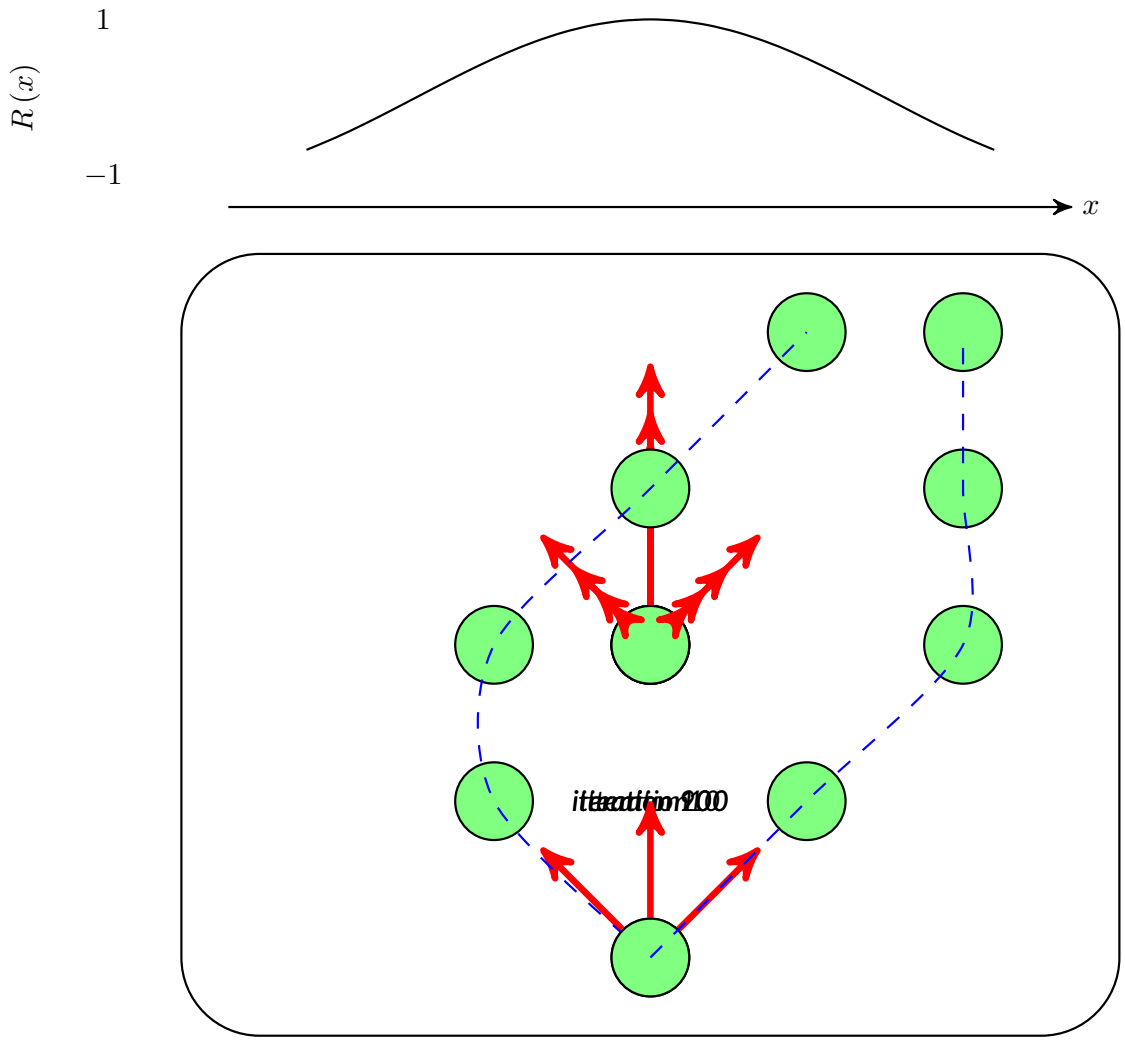
Revisiting REINFORCE

It is easy to see that `REINFORCE()` is a vanilla PGM: *here, we set estimator of action-value to be*

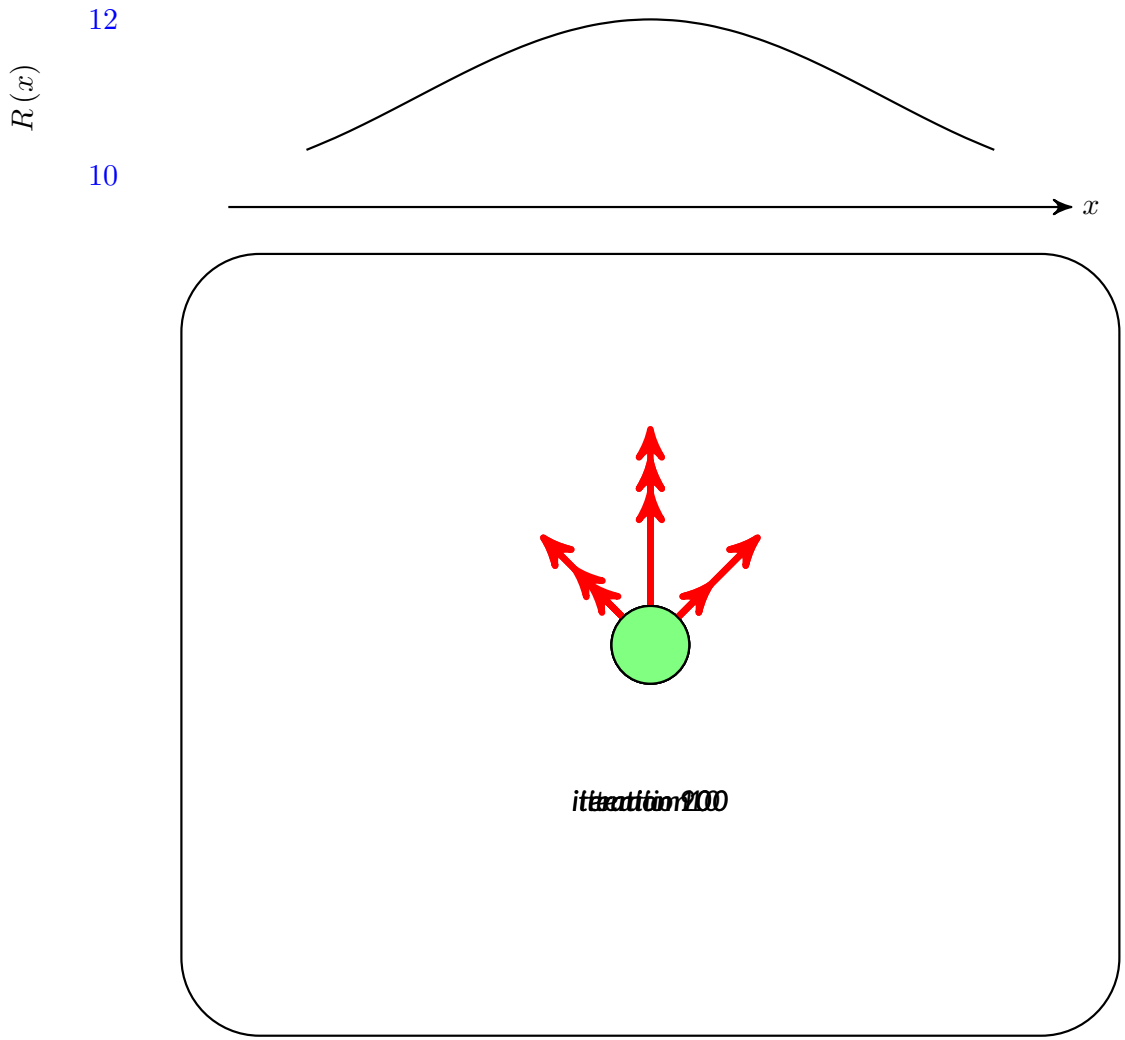
$$Q_t = G_t = \sum_{i=t}^{T-1} \gamma^i R_{i+1}$$

+ *But in our initial derivation, we saw derived G_0 instead of G_t!*
– *Well! If we replace in the policy gradient theorem, we could see that it would be still an estimator if we replace G_t with G_0*
+ *So, we have many estimators! How can we choose among them?!*
– *This is what we do in baseline PGM*
+ *What about using TD to estimate action-values?*
– *We could do it! But there will be a bit of complications. We will see it soon!*

Example: *Controlling Moving Particle – Case I*



Example: *Controlling Moving Particle – Case II*



Vanilla PGM: *Bias Issue*

This is a crucial observation: *with a simple shift in value, vanilla PGM slows significantly in convergence!*

`VanillaPGM()`:

1: *Initiate with* $\boldsymbol{\theta}$ *and learning rate* α
2: *Use a* ***Q-estimator*** `QEst()`
3: ***while*** *interacting* ***do***
4: *Sample the environment with policy* $\pi_{\boldsymbol{\theta}}$

$$S_0, A_0 \xrightarrow{R_1} S_{t+1}, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

5: ***for*** $t = 0 : T-1$ ***do***
6: Set $Q_t =$ `QEst`(S_t, A_t)
7: Update as $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \boxed{Q_t} \nabla \log \pi_{\boldsymbol{\theta}} (A_t | S_t)$ ← here is the trouble
8: ***end for***
9: ***end while***

Vanilla PGM: *Bias Issue*

This is a crucial observation: *with a simple shift in value, vanilla PGM slows significantly in convergence!*

Let's look at this update rule

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha Q_t \nabla \log \pi_{\boldsymbol{\theta}} \left(A_t | S_t\right)$$

With larger values, Q_t becomes larger, hence

- *if $\nabla \log \pi_{\boldsymbol{\theta}} \left(A_t | S_t\right)$ becomes a small positive*
  - ↳ *$\boldsymbol{\theta}$ increases largely*
- *if $\nabla \log \pi_{\boldsymbol{\theta}} \left(A_t | S_t\right)$ becomes a small negative*
  - ↳ *$\boldsymbol{\theta}$ drops largely*

We need to have Q_t concentrated around zero

Vanilla PGM: *Bias Issue*

We need to have Q_t concentrated around zero

+ *But, wait a moment! We derived this expression from policy gradient theorem! If we change Q_t to something else, we are deviating from it!*
– Well! This is not necessarily true!

Let's try an experiment: in the gradient term given by policy gradient algorithm, we replace the action-value term with a shifted one, i.e., replace $q_{\pi_{\boldsymbol{\theta}}}(S, A)$ with

$$q'_{\pi_{\boldsymbol{\theta}}}(S, A) = q_{\pi_{\boldsymbol{\theta}}}(S, A) - u(S)$$

The term $u(S)$ can change with state, but it is fixed in terms of actions

Unbiasing Policy Gradient

Replacing $q'_{\pi_{\boldsymbol{\theta}}}(S, A)$ into the gradient expression, we have

$$\begin{aligned}
\mathcal{E}(\pi_{\boldsymbol{\theta}}) &= \mathbb{E}_{S\sim d_{\pi_{\boldsymbol{\theta}}}, A|S\sim\pi_{\boldsymbol{\theta}}}\left\{q'_{\pi_{\boldsymbol{\theta}}}(S, A)\nabla\log\pi_{\boldsymbol{\theta}}(A|S)\right\} \\
&= \mathbb{E}_{S\sim d_{\pi_{\boldsymbol{\theta}}}, A|S\sim\pi_{\boldsymbol{\theta}}}\left\{(q_{\pi_{\boldsymbol{\theta}}}(S, A) - u(S))\nabla\log\pi_{\boldsymbol{\theta}}(A|S)\right\} \\
&= \mathbb{E}_{S\sim d_{\pi_{\boldsymbol{\theta}}}}\left\{\mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left\{(q_{\pi_{\boldsymbol{\theta}}}(S, A) - u(S))\nabla\log\pi_{\boldsymbol{\theta}}(A|S)\,|S\right\}\right\} \\
&= \underbrace{\mathbb{E}_{S\sim d_{\pi_{\boldsymbol{\theta}}}}\left\{\mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left\{q_{\pi_{\boldsymbol{\theta}}}(S, A)\nabla\log\pi_{\boldsymbol{\theta}}(A|S)\,|S\right\}\right\}}_{\nabla\mathcal{J}(\pi_{\boldsymbol{\theta}})} \\
&\quad - \underbrace{\mathbb{E}_{S\sim d_{\pi_{\boldsymbol{\theta}}}}\left\{u(S)\,\mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left\{\nabla\log\pi_{\boldsymbol{\theta}}(A|S)\,|S\right\}\right\}}_{?}
\end{aligned}$$

So, we have

$$\mathcal{E}(\pi_{\boldsymbol{\theta}}) = \nabla\mathcal{J}(\pi_{\boldsymbol{\theta}}) - \mathbb{E}_{S\sim d_{\pi_{\boldsymbol{\theta}}}}\left\{u(S)\,\mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left\{\nabla\log\pi_{\boldsymbol{\theta}}(A|S)\,|S\right\}\right\}$$

To compute the second term, we can use a simple trick

Gradient Averaging Trick

Assume $X \sim p_{\boldsymbol{\theta}}(x)$: *$X$ is distributed by a distribution that is parameterized by some $\boldsymbol{\theta}$. We can then write*

$$\mathbb{E}_{p_{\boldsymbol{\theta}}}\left\{\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(X)\right\} = \mathbb{E}_{p_{\boldsymbol{\theta}}}\left\{\frac{\nabla p_{\boldsymbol{\theta}}(X)}{p_{\boldsymbol{\theta}}(X)}\right\} = \int_x \frac{\nabla p_{\boldsymbol{\theta}}(x)}{p_{\boldsymbol{\theta}}(x)} p_{\boldsymbol{\theta}}(x)$$

$$= \int_x \nabla p_{\boldsymbol{\theta}}(x) = \nabla \underbrace{\int_x p_{\boldsymbol{\theta}}(x)}_{1} = \nabla 1 = 0$$

Lemma: *Gradient Averaging*

For any parameterized distribution $p_{\boldsymbol{\theta}}(x)$, we have

$$\mathbb{E}_{p_{\boldsymbol{\theta}}}\left\{\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(X)\right\} = 0$$

Unbiasing Policy Gradient

This concludes that

$$\mathcal{E}(\pi_{\boldsymbol{\theta}}) = \nabla \mathcal{J}(\pi_{\boldsymbol{\theta}}) - \mathbb{E}_{S \sim d_{\pi_{\boldsymbol{\theta}}}} \left\{ u(S) \underbrace{\mathbb{E}_{\pi_{\boldsymbol{\theta}}} \{\nabla \log \pi_{\boldsymbol{\theta}}(A|S) | S\}}_{0} \right\}$$

$$= \nabla \mathcal{J}(\pi_{\boldsymbol{\theta}})$$

In other words: we can add any term that is fixed in terms of actions to our value estimator without any harm to policy gradient theorem

- *This fixed term is often called baseline*
- *It can improve convergence behavior*
- *It's something to be engineered in general*
  - ↳ *But no worries! We will see an obvious choice shortly* ☺

Policy Gradient Theorem with *Baseline*

Policy Gradient with Baseline

For a policy network with non-zero probabilities, the gradient of the average trajectory return is always given by

$$\nabla \mathcal{J}(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{S \sim d_{\pi_{\boldsymbol{\theta}}}, A|S \sim \pi_{\boldsymbol{\theta}}} \left\{ \left(q_{\pi_{\boldsymbol{\theta}}}(S, A) - u(S) \right) \nabla \log \pi_{\boldsymbol{\theta}}(A|S) \right\}$$

Baseline

policy evaluation

for any baseline $u(\cdot)$

Baseline PGM: *General Form*

`BaselinePGM()`:

Initiate with $\boldsymbol{\theta}$ *and learning rate* α
Use a ***Q-estimator*** `QEst()`
while *interacting* ***do***
    Sample the environment with policy $\pi_{\boldsymbol{\theta}}$

$$S_0, A_0 \xrightarrow{R_1} S_{t+1}, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

    for $t = 0 : T-1$ ***do***
        Set $Q_t =$ `QEst`(S_t, A_t)
        Compute *baseline estimator* $B_t = u(S_t)$
        Update policy network $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha (Q_t - B_t) \nabla \log \pi_{\boldsymbol{\theta}} (A_t | S_t)$
    end for
end while

Value Function: *Obvious Choice of Baseline*

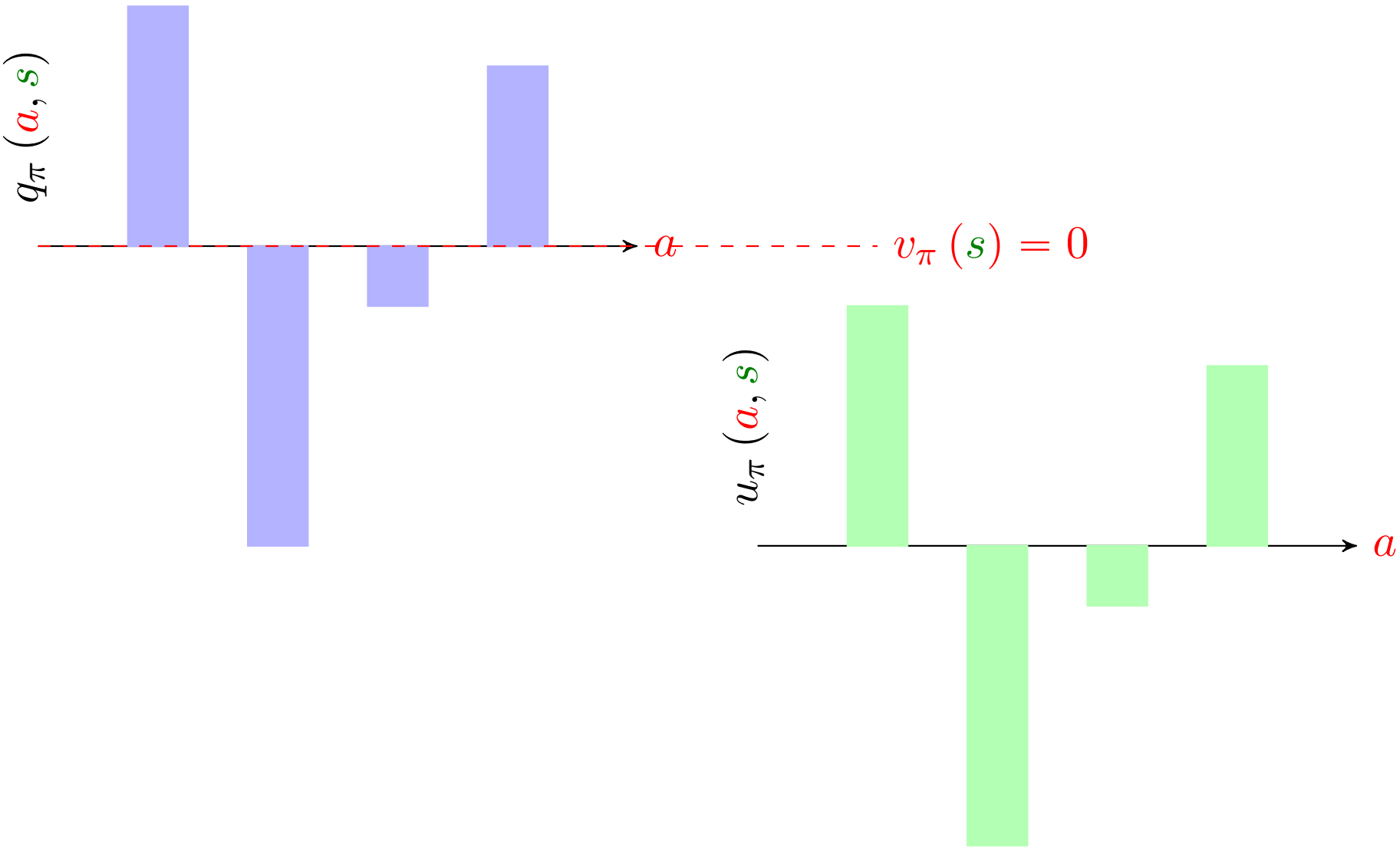
+ *What is the obvious choice of baseline?!*
– *Value function $v_{\pi_{\boldsymbol{\theta}}}(s)$!*
+ *How is it obvious?!*
– *In this case, shifted action-value represents the co-called advantage*

Advantage

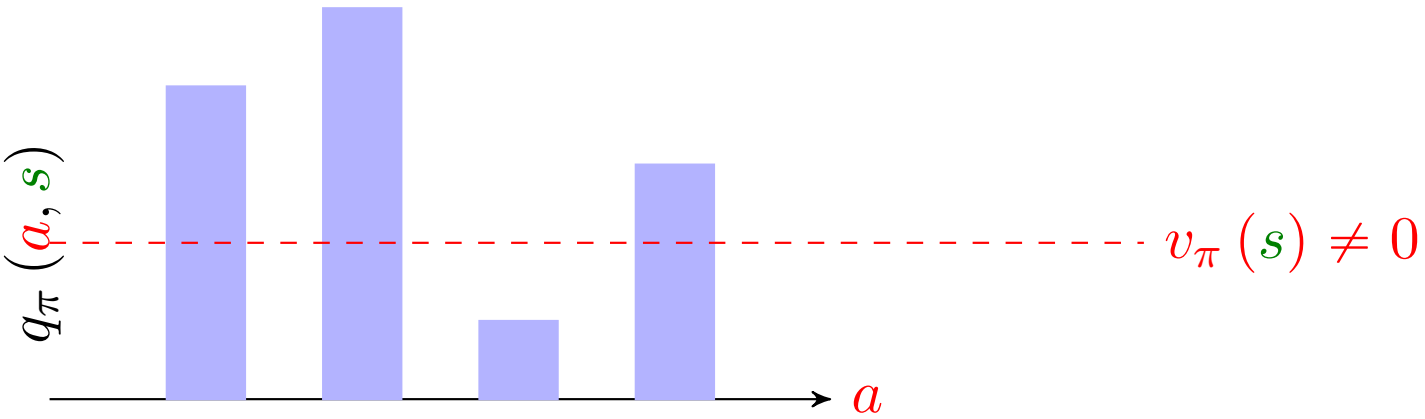
Given policy π, the advantage of action a at state s is defined as

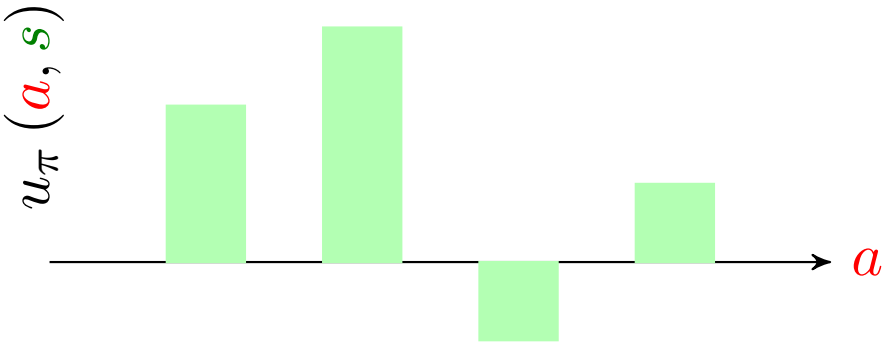
$$u_\pi(a, s) = q_\pi(a, s) - v_\pi(s)$$

Advantage: *Visualization*



Advantage: *Visualization*





Advantage at any *state* concentrates around *zero*

Baseline PGM: *Advantage Optimization*

Policy Gradient with Advantage

For a policy network with non-zero probabilities, the gradient of the average trajectory return is also given by

$$\begin{aligned}\nabla \mathcal{J}\left(\pi_{\boldsymbol{\theta}}\right) &= \mathbb{E}_{S\sim d_{\pi_{\boldsymbol{\theta}}}, A|S\sim\pi_{\boldsymbol{\theta}}}\left\{u_{\pi_{\boldsymbol{\theta}}}\left(S,A\right)\nabla \log \pi_{\boldsymbol{\theta}}\left(A|S\right)\right\} \\ &= \mathbb{E}_{S\sim d_{\pi_{\boldsymbol{\theta}}}, A|S\sim\pi_{\boldsymbol{\theta}}}\left\{\left(q_{\pi_{\boldsymbol{\theta}}}\left(S,A\right) - v_{\pi_{\boldsymbol{\theta}}}\left(S\right)\right)\nabla \log \pi_{\boldsymbol{\theta}}\left(A|S\right)\right\}\end{aligned}$$

+ *But how can we find an estimator for advantage?*
– *If we can estimate action-values, we can obviously use*

$$v_{\pi_{\boldsymbol{\theta}}}\left(s\right) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left\{q_{\pi_{\boldsymbol{\theta}}}\left(s,A\right)\right\} = \int_{a} q_{\pi_{\boldsymbol{\theta}}}\left(s,a\right)\pi_{\boldsymbol{\theta}}\left(a|s\right)$$

Baseline PGM: *Advantage Optimization*

`AdvantagePGM()`:

1: *Initiate with* $\boldsymbol{\theta}$ *and learning rate* α
2: *Use a* ***Q-estimator*** `QEst()`
3: ***while*** *interacting* ***do***
4: *Sample the environment with policy* $\pi_{\boldsymbol{\theta}}$

$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

5: ***for*** $t = 0 : T-1$ ***do***
6: Set $Q_t =$ `QEst`(S_t, A_t)
7: Compute *value* $V_t = \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \{Q_t | S_t\}$ *and sample advantage* $U_t = Q_t - V_t$
8: Update policy network $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha U_t \nabla \log \pi_{\boldsymbol{\theta}} (A_t | S_t)$
9: ***end for***
10: ***end while***

PGM with Temporal Difference Estimate

+ *We have only considered Monte Carlo approach to estimate values! Why don't we use TD?!*
– Well! If we only work with a policy network, it could be challenging

---

Say we have a particular sample trajectory that looks at time t as

$$S_t, A_t \xrightarrow{R_{t+1}} S_{t+1}$$

If we use TD-0 to estimate $q_{\pi_{\boldsymbol{\theta}}}(S_t, A_t)$, we would write

$$\hat{q}_{\pi_{\boldsymbol{\theta}}}(S_t, A_t) = R_{t+1} + \gamma \hat{v}_{\pi_{\boldsymbol{\theta}}}(S_{t+1})$$

where do we get this estimate?!

Estimating via TD in PGM: *Main Challenge*

In value-based RL, we gradually find an estimate for values

- *In tabular RL, we make a Q-table and update it*
- *In deep RL, we train a value network, e.g., a DQN*

In pure PGM, we have neither of them!

+ *Then what can we do? We cannot always use Monte Carlo! What if the environment is not episodic?*
– *Well there are three solutions with only one of them working!*

Estimating via TD in PGM: *Possible Solutions*

1. *We may stick to Monte Carlo approach*
   - ↳ *It has high variance and is hard to use in non-episodic environments*
2. *We may try to evaluate the policy in each iteration*
   - ↳ *This is in practice computationally infeasible*
3. *We may train a value network in addition to the policy network*
   - ↳ *This describe the class of actor-critic methods*
     - ↳ *An actor who plays with the policy network and update it via PGM*
     - ↳ *A critic who evaluates the policy with a value network and update it with DQL*
   - ↳ *We will get to these methods in the next chapter*

For now, let's make an idealistic assumption: we assume that we can really evaluate a policy, i.e., given $\pi_{\boldsymbol{\theta}}$ for any $\boldsymbol{\theta}$

$$\text{we can compute } v_{\pi_{\boldsymbol{\theta}}}(s) \text{ and } q_{\pi_{\boldsymbol{\theta}}}(s, A)$$

We will later get rid of this idealistic assumption by the help of value networks

Estimating via TD in PGM: *Idealistic Case*

With this assumption, we can rewrite our algorithms in an online form, e.g.,

`AdvantagePGM()`:

1: *Initiate with* $\boldsymbol{\theta}$ *and learning rate* α
2: ***while*** *interacting* ***do***
3: *Sample the environment with policy* $\pi_{\boldsymbol{\theta}}$

$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

4: ***for*** $t = 0 : T-1$ ***do***
5: *Compute* $q_{\pi_{\boldsymbol{\theta}}}(S_t, A_t)$ *and* $v_{\pi_{\boldsymbol{\theta}}}(S_t) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\{q_{\pi_{\boldsymbol{\theta}}}(S_t, A_t)|S_t\}$
6: *Compute sample advantage* $U_t = q_{\pi_{\boldsymbol{\theta}}}(S_t, A_t) - v_{\pi_{\boldsymbol{\theta}}}(S_t)$
7: ***end for***
8: *Update policy network*

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \sum_{t=0}^{T-1} U_t \nabla \log \pi_{\boldsymbol{\theta}}(A_t|S_t)$$

9: ***end while***

TD Error as Advantage Estimator

$$S_t, A_t \xrightarrow{R_{t+1}} S_{t+1}$$

Let's look at the sample advantage: we have

$$U_t = q_{\pi_{\boldsymbol{\theta}}}(S_t, A_t) - v_{\pi_{\boldsymbol{\theta}}}(S_t)$$

Using Bellman's equation, we can write

$$U_t = \mathbb{E}\{R_{t+1}\} + \gamma \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\{v_{\pi_{\boldsymbol{\theta}}}(S_{t+1}) | S_t, A_t\} - v_{\pi_{\boldsymbol{\theta}}}(S_t)$$

which we can be estimated by

$$\hat{U}_t = R_{t+1} + \gamma v_{\pi_{\boldsymbol{\theta}}}(S_{t+1}) - v_{\pi_{\boldsymbol{\theta}}}(S_t)$$

This is the TD-0 error: TD error is an estimator of advantage!

Advantage PGM: *Online via TD Estimate*

`AdvantagePGM()`:

1: *Initiate with* $\boldsymbol{\theta}$ *and learning rate* α
2: ***while*** *interacting* ***do***
3: *Sample the environment with policy* $\pi_{\boldsymbol{\theta}}$

$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

4: ***for*** $t = 0 : T-1$ ***do***
5: *Compute sample advantage* $U_t = R_{t+1} + \gamma v_{\pi_{\boldsymbol{\theta}}}(S_{t+1}) - v_{\pi_{\boldsymbol{\theta}}}(S_t)$
6: ***end for***
7: *Update policy network*

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \sum_{t=0}^{T-1} U_t \nabla \log \pi_{\boldsymbol{\theta}}(A_t | S_t)$$

8: ***end while***

\+ *And, we do not need action-values!*

– Right! *Value function is enough*

Advantage PGM: *Online via TD Estimate*

Obviously, we can find a more robust estimator via TD-n

`AdvantagePGM(`n`):`

1: *Initiate with* $\boldsymbol{\theta}$ *and learning rate* α
2: ***while** interacting **do***
3: *Sample the environment with policy* $\pi_{\boldsymbol{\theta}}$

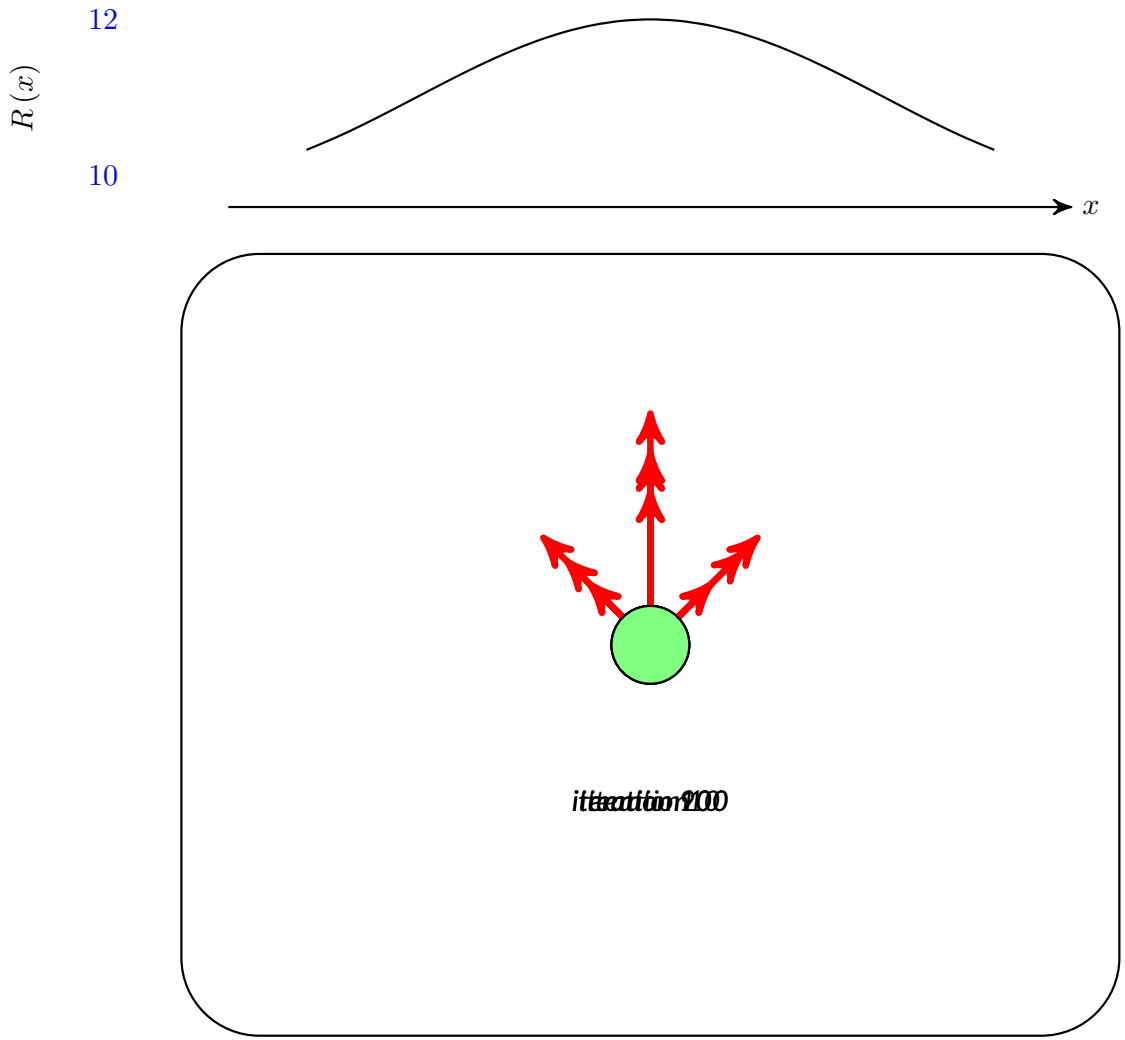
$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

4: ***for*** $t = 0 : T - n - 1$ ***do***
5: *Compute* $U_t = \sum_{i=0}^{n} \gamma^i R_{t+i+1} + \gamma^{n+1} v_{\pi_{\boldsymbol{\theta}}}(S_{t+n+1}) - v_{\pi_{\boldsymbol{\theta}}}(S_t)$
6: ***end for***

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \sum_{t=0}^{T-n-1} U_t \nabla \log \pi_{\boldsymbol{\theta}}(A_t | S_t)$$

7: ***end while***

Example: *Controlling Moving Particle – Case II*



Crucial Challenge in PGM: *Sample Inefficiency*

After implementing `AdvantagePGM()`, we see: *though using baseline, the variance reduces, it still needs long time to converge*

the main reason is that PGM is sample inefficient

In all above algorithms

- *We sample* $S_0, A_0 \xrightarrow{R_1} \cdots \xrightarrow{R_T} S_T$ *and use it for update*
- *We never get back to this sample*

This is generally a big challenge in PGMs!

\+ *Can't we do what we did in DQL?!*

– *You mean experience reply?!*

\+ *Right! Just keep previous samples in a buffer and reuse them again*

– *Well! The issue is that those samples were collected by other policies, i.e.,* $\pi_{\boldsymbol{\theta}}$ *for old* $\boldsymbol{\theta}$*. Through time, we have gone far away from them!*

Solution: *Trust Region PGM*

+ *So was it with DQL! How did we get rid of that?!*
– *We were playing off-policy, so we did not need to have sample with the target policy*
+ *So, isn't there any way to improve the sample efficiency?*
– *There is one and we do know it!*

The solution to this challenge is to use the idea of importance sampling: recall that if $X \sim p(x)$ *we could write*

$$\mathbb{E}_q\{X\} = \int_x xq(x) = \int_x x\frac{q(x)}{p(x)}p(x) = \mathbb{E}_p\left\{X\frac{q(X)}{p(X)}\right\}$$

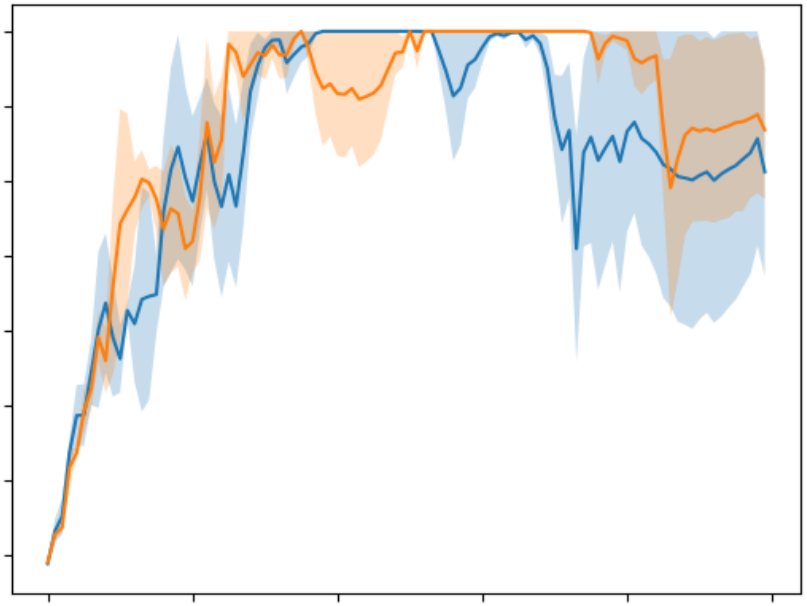
This leads to PGMs with trust region that we will learn next!

Observing Basic PGM

Let's break the problem down: *even by using baseline, we still observe instability while we use PGM*

If we plot the average reward we collect through time

- *We might see it getting improved up to some point*
- *It then could drop drastically at some other point*



Main Reason: *Estimate Variance*

The main reason *for this behavior is high variance of the gradient estimator: recall that we estimate the gradient of average value by*

$$\hat{\nabla}\mathcal{J}\left(\pi_{\boldsymbol{\theta}}\right) = \sum_{t=0}^{T-1} U_t \nabla \log \pi_{\boldsymbol{\theta}}\left(A_t | S_t\right)$$

This is true that

$$\mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left\{\hat{\nabla}\mathcal{J}\left(\pi_{\boldsymbol{\theta}}\right)\right\} = \nabla\mathcal{J}\left(\pi_{\boldsymbol{\theta}}\right)$$

However, its variance, i.e.,

$$\mathbb{V}\mathrm{ar}\left\{\hat{\nabla}\mathcal{J}\left(\pi_{\boldsymbol{\theta}}\right)\right\} = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left\{\left(\hat{\nabla}\mathcal{J}\left(\pi_{\boldsymbol{\theta}}\right) - \mathcal{J}\left(\pi_{\boldsymbol{\theta}}\right)\right)^2\right\}$$

could be very large: one given sample take us far away from the true direction!

Reducing Variance: *Using Mini-Batches*

+ *But, isn't that always the case in SGD?! We assume that those errors cancel each other out! Right?!*
– That's right! *But apparently, it's not working here!*

To find out an explanation to this behavior, let's try to reduce the variance by using larger mini-batches

- *Collect B sample trajectories by policy $\pi_{\boldsymbol{\theta}}$*
- *Compute the gradient estimator for each trajectory*

$$\hat{\nabla}_b \mathcal{J}\left(\pi_{\boldsymbol{\theta}}\right) = \sum_{t=0}^{T-1} U_t \nabla \log \pi_{\boldsymbol{\theta}}\left(A_t | S_t\right)$$

- *Average the estimators to get a better estimator*

$$\hat{\nabla} \mathcal{J}\left(\pi_{\boldsymbol{\theta}}\right) = \frac{1}{B} \sum_{b=1}^{B} \hat{\nabla}_b \mathcal{J}\left(\pi_{\boldsymbol{\theta}}\right)$$

Advantage PGM: *Mini-Batch Version*

`AdvantagePGM()`:

Initiate with $\boldsymbol{\theta}$ *and learning rate* α

while *interacting* ***do***

  for *mini-batch* $b = 1 : B$ ***do***

    Sample the environment with policy $\pi_{\boldsymbol{\theta}}$

$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

    for $t = 0 : T-1$ ***do***

      Compute sample advantage $U_t = R_{t+1} + \gamma v_{\pi_{\boldsymbol{\theta}}}(S_{t+1}) - v_{\pi_{\boldsymbol{\theta}}}(S_t)$

    end for

    Compute sample gradient $\hat{\nabla}_b = \sum_{t=0}^{T-1} U_t \nabla \log \pi_{\boldsymbol{\theta}}(A_t|S_t)$

  end for

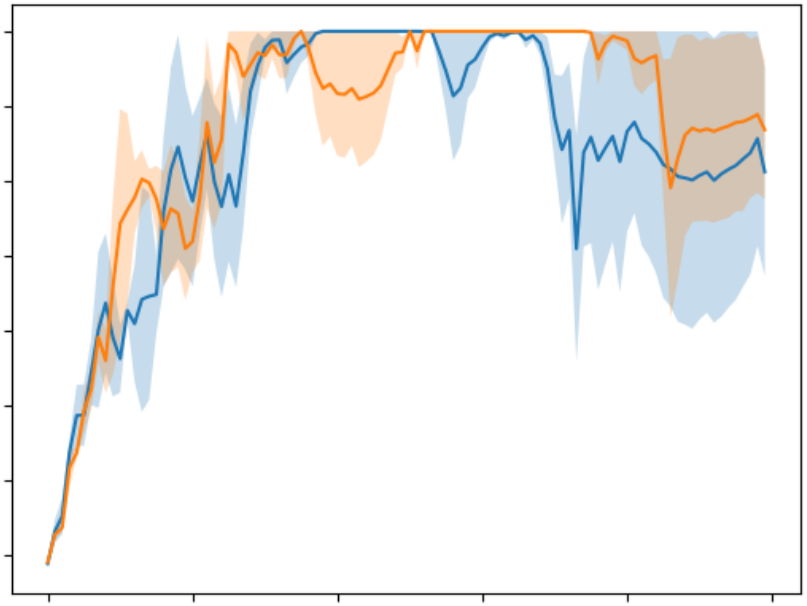
  Update policy network

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \frac{\alpha}{B} \sum_{b=1}^{B} \hat{\nabla}_b$$

end while

Observing Mini-Batch

After trying mini-batch PGM: *we see that the variance of the curve slightly improves; however, we still see that problem*



+ *What does this say then?*
– *It says that the problem is simply coming from high variance*

Alternative Look at *Advantage Optimization*

In the latest version of PGM: *we saw that the gradient of the average value can be computed as*

$$\nabla \mathcal{J}(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \{ u_{\pi_{\boldsymbol{\theta}}}(S, A) \nabla \log \pi_{\boldsymbol{\theta}}(A|S) \}$$

Let's assume that we can compute it exactly: then, we will muse gradient descent to find optimal $\boldsymbol{\theta}$, i.e.,

`AdvantageGD()`:

1: *Start with some initial* $\boldsymbol{\theta}_0$
2: ***for*** $k = 1 : K$ ***do***
3: *Compute the exact gradient*

$$\nabla \mathcal{J}(\pi_{\boldsymbol{\theta}_k}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}_k}} \left\{ u_{\pi_{\boldsymbol{\theta}_k}}(S, A) \nabla \log \pi_{\boldsymbol{\theta}}(A|S) \,|_{\boldsymbol{\theta}=\boldsymbol{\theta}_k} \right\}$$

4: *Update the parameters as* $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha \nabla \mathcal{J}(\pi_{\boldsymbol{\theta}_k})$
5: ***end for***

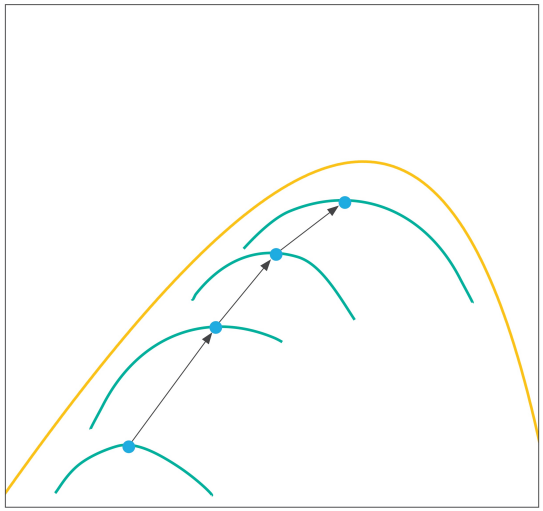
Alternative Look: *Surrogate Function*

Let us now define the following surrogate function at point $\boldsymbol{\theta}_k$

$$\mathcal{L}_k\left(\pi_{\boldsymbol{\theta}}\right) = \mathbb{E}_{\pi_{\boldsymbol{\theta}_k}}\left\{u_{\pi_{\boldsymbol{\theta}_k}}\left(S, A\right)\frac{\pi_{\boldsymbol{\theta}}\left(A|S\right)}{\pi_{\boldsymbol{\theta}_k}\left(A|S\right)}\right\}$$

We should pay attention that

1. *We have a sequence of surrogate functions*
   - ↳ *Each function is defined locally at point* $\boldsymbol{\theta}_k$
2. $\pi_{\boldsymbol{\theta}}\left(A|S\right)$ *is the only term that belongs to* $\boldsymbol{\theta}$
   - ↳ *Everything else depends on* $\boldsymbol{\theta}_k$ *which is a constant*



Alternative Look: *Surrogate Function*

$$\mathcal{L}_k\left(\pi_{\boldsymbol{\theta}}\right) = \mathbb{E}_{\pi_{\boldsymbol{\theta}_k}}\left\{u_{\pi_{\boldsymbol{\theta}_k}}\left(S,A\right)\frac{\pi_{\boldsymbol{\theta}}\left(A|S\right)}{\pi_{\boldsymbol{\theta}_k}\left(A|S\right)}\right\}$$

Let's compute the gradient of this surrogate function at $\boldsymbol{\theta} = \boldsymbol{\theta}_k$

$$\begin{aligned}\nabla\mathcal{L}_k\left(\pi_{\boldsymbol{\theta}_k}\right) &= \mathbb{E}_{\pi_{\boldsymbol{\theta}_k}}\left\{u_{\pi_{\boldsymbol{\theta}_k}}\left(S,A\right)\frac{\nabla\pi_{\boldsymbol{\theta}}\left(A|S\right)}{\pi_{\boldsymbol{\theta}_k}\left(A|S\right)}\right\}\\ &= \mathbb{E}_{\pi_{\boldsymbol{\theta}_k}}\left\{u_{\pi_{\boldsymbol{\theta}_k}}\left(S,A\right)\nabla\log\pi_{\boldsymbol{\theta}}\left(A|S\right)|_{\boldsymbol{\theta}=\boldsymbol{\theta}_k}\right\}\\ &= \nabla\mathcal{J}\left(\pi_{\boldsymbol{\theta}_k}\right)\end{aligned}$$

This is exactly the gradient that we update our policy with!

Alternative Look: *GD with Surrogate Function*

So, we could re-write the gradient descent loop as

`AdvantageGD():`

1: *Start with some initial* $\boldsymbol{\theta}_0$
2: ***for*** $k = 1 : K$ ***do***
3: *Compute the exact gradient* $\nabla \mathcal{L}_k\left(\pi_{\boldsymbol{\theta}_k}\right)$
4: *Update the parameters as* $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha \nabla \mathcal{L}_k\left(\pi_{\boldsymbol{\theta}_k}\right)$
5: ***end for***

\+ *Now, what's the point?! It's still same problem!*

– Sure! But, let's see what we are doing now

In each iteration we add gradient scaled with α *to the previous parameters*

– *Why we do that?*

\+ *We want to increase* $\mathcal{L}_k\left(\pi_{\boldsymbol{\theta}}\right)$ *maximally*

– *Exactly! So, why don't we simply replace* $\boldsymbol{\theta}_{k+1}$ *with its maximizer?!*

Alternative Look: *GD with Surrogate Function*

We re-write the gradient descent loop as follows

`AdvantageGD()`:

1: *Start with some initial* $\boldsymbol{\theta}_0$
2: ***for*** $k = 1 : K$ ***do***
3: *Compute the surrogate function* $\mathcal{L}_k\left(\pi_{\boldsymbol{\theta}}\right)$
4: *Update the parameters as* $\boldsymbol{\theta}_{k+1} = \operatorname{argmax}_{\boldsymbol{\theta}} \mathcal{L}_k\left(\pi_{\boldsymbol{\theta}}\right)$
5: ***end for***

This algorithm algorithm does the learning rate tuning by itself

It gets us rid of specifying the learning rate α

+ *Nice job! But, how are we su[supposed] to find the surrogate function?*
– *You could guess! By sampling!*

Understanding Surrogate Function

Let's get back to the definition of the surrogate function:*it is easy to interprete it as importance sampling*

$$\mathcal{L}_k\left(\pi_{\boldsymbol{\theta}}\right) = \mathbb{E}_{\pi_{\boldsymbol{\theta}_k}}\left\{u_{\pi_{\boldsymbol{\theta}_k}}\left(S,A\right)\frac{\pi_{\boldsymbol{\theta}}\left(A|S\right)}{\pi_{\boldsymbol{\theta}_k}\left(A|S\right)}\right\}$$

We sample trajectories by policy $\pi_{\boldsymbol{\theta}_k}$

- *We collect advantage samples* $u_{\pi_{\boldsymbol{\theta}_k}}\left(S,A\right)$
- *We do know the policy samples* $\pi_{\boldsymbol{\theta}_k}\left(A|S\right)$

But we compute the average with respect to $\pi_{\boldsymbol{\theta}}$

This gives us a very good explanation of why PGM is unstable

Review: *Importance Sampling Trade-off*

Consider the basic setting in importance sampling:

we have samples from $X \sim p(x)$ *but we want to estimate* $X \sim q(x)$

If we could sample from $q(x)$

$$\mu = \mathbb{E}_q \{X\}$$

The variance of the estimate is

$$\begin{aligned} \mathbb{V}\mathrm{ar}\{X\} &= \mathbb{E}_q \left\{ (X - \mu)^2 \right\} \\ &= \mathbb{E}_q \left\{ X^2 \right\} - \mu^2 = \sigma^2 \end{aligned}$$

Review: *Importance Sampling Trade-off*

Consider the basic setting in importance sampling:

we have samples from $X \sim p(x)$ *but we want to estimate* $X \sim q(x)$

Now that we sample $p(x)$

$$\bar{\mu} = \mathbb{E}_p \left\{ X \frac{q(X)}{p(X)} \right\} = \mathbb{E}_q \{X\} = \mu$$

The variance of this estimate is

$$\mathbb{V}\mathrm{ar}\{X\} = \mathbb{E}_p \left\{ \left(X \frac{q(X)}{p(X)} \right)^2 \right\} - \mu^2 = \int_x x^2 \frac{q^2(x)}{p^2(x)} p(x) - \mu^2$$

$$= \int_x x^2 \frac{q(x)}{p(x)} q(x) - \mu^2 = \mathbb{E}_q \left\{ X^2 \frac{q(X)}{p(X)} \right\} - \mu^2 \neq \sigma^2$$

Review: *Importance Sampling Trade-off*

Consider the basic setting in importance sampling:

we have samples from $X \sim p(x)$ *but we want to estimate* $X \sim q(x)$

If we could sample from $q(x)$

$$\mu = \mathbb{E}_q \{X\}$$

and the estimate variance is

$$\sigma^2 = \mathbb{E}_q \{X^2\} - \mu^2$$

Now that we sample $p(x)$

$$\mu = \mathbb{E}_q \{X\}$$

and the estimate variance is

$$\bar{\sigma}^2 = \mathbb{E}_q \left\{ X^2 \frac{q(X)}{p(X)} \right\} - \mu^2$$

With importance sampling, variance scales with ratio of the distributions

Back to Surrogate: *Root of High Variance*

Gradient descent based on surrogate functions optimizes

$$\mathcal{L}_k\left(\pi_{\boldsymbol{\theta}}\right) = \mathbb{E}_{\pi_{\boldsymbol{\theta}_k}}\left\{u_{\pi_{\boldsymbol{\theta}_k}}\left(S, A\right)\frac{\pi_{\boldsymbol{\theta}}\left(A|S\right)}{\pi_{\boldsymbol{\theta}_k}\left(A|S\right)}\right\}$$

We can assume that by sampling the environment, we are estimating this surrogate function by importance sampling from samples of policy $\pi_{\boldsymbol{\theta}_k}$:let $\hat{\mathcal{L}}_k\left(\pi_{\boldsymbol{\theta}}\right)$ be our estimate; then we could say

$$\mathbb{V}\mathrm{ar}\left\{\hat{\mathcal{L}}_k\left(\pi_{\boldsymbol{\theta}}\right)\right\} \propto \frac{\pi_{\boldsymbol{\theta}}\left(A|S\right)}{\pi_{\boldsymbol{\theta}_k}\left(A|S\right)}$$

- *This explains why we see unreliable estimates in PGM!*
- \+ *How exactly?!*
- *Let's break it down!*

Back to Surrogate: *Root of High Variance*

$$\mathbb{V}\mathrm{ar}\left\{\hat{\mathcal{L}}_k\left(\pi_{\boldsymbol{\theta}}\right)\right\} \propto \frac{\pi_{\boldsymbol{\theta}}\left(A|S\right)}{\pi_{\boldsymbol{\theta}_k}\left(A|S\right)}$$

Looking at this variance, we could say: we should not naively update $\boldsymbol{\theta}_k$ by maximizing its surrogate function. We should update it such that

1. *surrogate function is maximized, and*
2. *the policy specified by $\pi_{\boldsymbol{\theta}_{k+1}}$ is rather close to $\pi_{\boldsymbol{\theta}_k}$*

\+ *But aren't we doing that?! We just change the policy parameters slightly, i.e., $\left\|\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k\right\|^2$ is typically small*

– *Well! That doesn't say anything about difference between $\pi_{\boldsymbol{\theta}_{k+1}}$ and $\pi_{\boldsymbol{\theta}_k}$*

Observation: *Sensitivity of Policy Network*

Sensitivity of Policy

Even if we change parameters $\boldsymbol{\theta}$ slightly, $\pi_{\boldsymbol{\theta}}$ can change hugely!

Given this observation, we could modify our gradient descent loop as

`AdvantageGD():`

1: *Start with some initial* $\boldsymbol{\theta}_0$
2: ***for*** $k = 1 : K$ ***do***
3: *Compute the surrogate function* $\mathcal{L}_k\left(\pi_{\boldsymbol{\theta}}\right)$
4: *Update the parameters as*

$$\boldsymbol{\theta}_{k+1} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \mathcal{L}_k\left(\pi_{\boldsymbol{\theta}}\right) \quad \textit{subject to} \quad \pi_{\boldsymbol{\theta}} \textit{ and } \pi_{\boldsymbol{\theta}_k} \textit{ are close}$$

5: ***end for***

\+ *How can we quantify "$\pi_{\boldsymbol{\theta}}$ and $\pi_{\boldsymbol{\theta}_k}$ being close"?*

Review: *Kullback-Leibler Divergence*

KL Divergence

Kullback-Leibler divergence between two distributions p and q is defined as

$$D_{\mathrm{KL}}(p\|q) = \mathbb{E}_p\left\{\log\left(\frac{p(X)}{q(X)}\right)\right\} = \int_x \log\left(\frac{p(x)}{q(x)}\right) p(x)$$

We can use this definition to find the divergence between $\pi_{\boldsymbol{\theta}}$ and $\pi_{\boldsymbol{\theta}_k}$

$$\bar{D}_{\mathrm{KL}}(\pi_{\boldsymbol{\theta}}\|\pi_{\boldsymbol{\theta}_K}) = \mathbb{E}_{S\sim d_{\pi_{\boldsymbol{\theta}_k}}}\left\{\mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left\{\log\left(\frac{\pi_{\boldsymbol{\theta}}(A|S)}{\pi_{\boldsymbol{\theta}_k}(A|S)}\right)\right\}\right\}$$

Review: *Properties of KL Divergence*

KL divergence shows interesting properties

- *It is zero when distributions are the same*

$$D_{\mathrm{KL}}\left(p \| p\right) = 0$$

and increases when they get more different

- *It is always non-negative, i.e., for any p and q*

$$D_{\mathrm{KL}}\left(p \| q\right) \geqslant 0$$

↳ *This property is often called Gibbs' inequality*

But remember that KL divergence is asymmetric, i.e.,

$$D_{\mathrm{KL}}\left(p \| q\right) \neq D_{\mathrm{KL}}\left(q \| p\right)$$

Trust Region Policy Gradient Method

Back to our modified gradient descent loop, we could write

`AdvantageGD()`:

1: *Start with some initial* $\boldsymbol{\theta}_0$
2: ***for*** $k = 1 : K$ ***do***
3: *Compute the surrogate function* $\mathcal{L}_k(\pi_{\boldsymbol{\theta}})$
4: *Update the parameters as*

$$\boldsymbol{\theta}_{k+1} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \mathcal{L}_k(\pi_{\boldsymbol{\theta}}) \quad \textit{subject to} \quad \bar{D}_{\mathrm{KL}}(\pi_{\boldsymbol{\theta}} \| \pi_{\boldsymbol{\theta}_k}) \leqslant d_{\max}$$

5: ***end for***

This modified approach is called

Trust Region PGM

since it computes the best policy gradient within a trusted resion

Surrogate Optimization: *Exact Solution*

+ *How can we solve the optimization in the loop then?*
– *As you could guess, we are going to find a way around it!*

The concrete way to solve it is to use regularization: we want to solve

$$\boldsymbol{\theta}_{k+1} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \mathcal{L}_k\left(\pi_{\boldsymbol{\theta}}\right) \quad \textit{subject to} \quad \bar{D}_{\mathrm{KL}}\left(\pi_{\boldsymbol{\theta}} \| \pi_{\boldsymbol{\theta}_k}\right) \leqslant d_{\max}$$

We solve instead

$$\boldsymbol{\theta}_{k+1} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \mathcal{L}_k\left(\pi_{\boldsymbol{\theta}}\right) - \beta\left(\bar{D}_{\mathrm{KL}}\left(\pi_{\boldsymbol{\theta}} \| \pi_{\boldsymbol{\theta}_k}\right) - d_{\max}\right)$$

for some β that potentially minimizes the regularized objective

This is going to be computationally very expensive!

Surrogate Optimization: *Natural Policy Gradient*

We instead use Taylor expansion to approximate both surrogate and constraint

Taylor Expansion

An analytic function $f(x)$ can be expanded around point x_0 as

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2} f''(x_0)(x - x_0)^2 + \cdots$$

Let's start with the surrogate function

$$\mathcal{L}_k(\pi_{\boldsymbol{\theta}}) = \mathcal{L}_k(\pi_{\boldsymbol{\theta}_k}) + \nabla \mathcal{L}_k(\pi_{\boldsymbol{\theta}_k})^{\mathsf{T}}(\boldsymbol{\theta} - \boldsymbol{\theta}_k) + \varepsilon$$

In Assignment 3, we show $\mathcal{L}_k(\pi_{\boldsymbol{\theta}_k}) = 0$: so, setting $\nabla_k = \nabla \mathcal{L}_k(\pi_{\boldsymbol{\theta}_k})$ we get

$$\mathcal{L}_k(\pi_{\boldsymbol{\theta}}) \approx \nabla_k^{\mathsf{T}}(\boldsymbol{\theta} - \boldsymbol{\theta}_k)$$

Surrogate Optimization: *Natural Policy Gradient*

Next, we go for constraint term

$$\bar{D}_{\mathrm{KL}}\left(\pi_{\boldsymbol{\theta}} \| \pi_{\boldsymbol{\theta}_k}\right) = \bar{D}_{\mathrm{KL}}\left(\pi_{\boldsymbol{\theta}_k} \| \pi_{\boldsymbol{\theta}_k}\right) + \nabla \bar{D}_{\mathrm{KL}}\left(\pi_{\boldsymbol{\theta}} \| \pi_{\boldsymbol{\theta}_k}\right)\big|_{\boldsymbol{\theta}_k}^{\mathsf{T}} \left(\boldsymbol{\theta} - \boldsymbol{\theta}_k\right) + \frac{1}{2}\left(\boldsymbol{\theta} - \boldsymbol{\theta}_k\right)^{\mathsf{T}} \nabla^2 \bar{D}_{\mathrm{KL}}\left(\pi_{\boldsymbol{\theta}} \| \pi_{\boldsymbol{\theta}_k}\right)\big|_{\boldsymbol{\theta}_k} \left(\boldsymbol{\theta} - \boldsymbol{\theta}_k\right) + \varepsilon$$

In Assignment 3, we show that

$$\bar{D}_{\mathrm{KL}}\left(\pi_{\boldsymbol{\theta}_k} \| \pi_{\boldsymbol{\theta}_k}\right) = 0$$
$$\nabla \bar{D}_{\mathrm{KL}}\left(\pi_{\boldsymbol{\theta}} \| \pi_{\boldsymbol{\theta}_k}\right)\big|_{\boldsymbol{\theta}_k} = \mathbf{0}$$

So, by defining $\mathbf{H}_k = \nabla^2 \bar{D}_{\mathrm{KL}}\left(\pi_{\boldsymbol{\theta}} \| \pi_{\boldsymbol{\theta}_k}\right)\big|_{\boldsymbol{\theta}_k}$ *we have*

$$\bar{D}_{\mathrm{KL}}\left(\pi_{\boldsymbol{\theta}} \| \pi_{\boldsymbol{\theta}_k}\right) \approx \frac{1}{2}\left(\boldsymbol{\theta} - \boldsymbol{\theta}_k\right)^{\mathsf{T}} \mathbf{H}_k \left(\boldsymbol{\theta} - \boldsymbol{\theta}_k\right)$$

Surrogate Optimization: *Natural Policy Gradient*

Now, let's replace these approximations

$$\mathcal{L}_k(\pi_{\boldsymbol{\theta}}) \approx \nabla_k^\intercal (\boldsymbol{\theta} - \boldsymbol{\theta}_k)$$
$$\bar{D}_{\mathrm{KL}}(\pi_{\boldsymbol{\theta}} \| \pi_{\boldsymbol{\theta}_k}) \approx \frac{1}{2} (\boldsymbol{\theta} - \boldsymbol{\theta}_k)^\intercal \mathbf{H}_k (\boldsymbol{\theta} - \boldsymbol{\theta}_k)$$

into the optimization problem

$$\boldsymbol{\theta}_{k+1} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \nabla_k^\intercal (\boldsymbol{\theta} - \boldsymbol{\theta}_k) \quad \textit{subject to} \quad \frac{1}{2} (\boldsymbol{\theta} - \boldsymbol{\theta}_k)^\intercal \mathbf{H}_k (\boldsymbol{\theta} - \boldsymbol{\theta}_k) \leqslant d_{\max}$$

This is a classic linear programming whose solution is given by

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \sqrt{\frac{2 d_{\max}}{\nabla_k^\intercal \mathbf{H}_k^{-1} \nabla_k}} \mathbf{H}_k^{-1} \nabla_k$$

Surrogate Optimization: *Natural Policy Gradient*

This is like classic update with linear correction and tuned learning rate

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \sqrt{\frac{2d_{\max}}{\nabla_k^\intercal \mathbf{H}_k^{-1} \nabla_k}} \mathbf{H}_k^{-1} \nabla_k$$

This is often referred to as

natural policy gradient

It gives a better direction for update; however,

- *It could still not increase surrogate or deviate constraint*
  - ↳ *This is due to the inaccuracy of approximations*
- *It requires estimate of $\mathbf{H}_k$ which is computationally expensive*
- *It also needs to invert estimate of $\mathbf{H}_k$ which is again expensive*

Natural PGM

`NaturalPGM():`

1: *Start with some initial* $\boldsymbol{\theta}$
2: ***while*** *interacting* ***do***
3: ***for*** *mini-batch* $b = 1 : B$ ***do***
4: *Sample the environment with policy* $\pi_{\boldsymbol{\theta}}$

$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

5: ***end for***
6: *Estimate* $\hat{\nabla}$ *and* $\hat{\mathbf{H}}$ *from samples*
7: *Update the parameters as*

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \sqrt{\frac{2d_{\max}}{\hat{\nabla}^\intercal \hat{\mathbf{H}} \hat{\nabla}}} \hat{\mathbf{H}}^{-1} \hat{\nabla}$$

8: ***end while***

TRPO and PPO Algorithms

There are two sets of solutions to overcome the challenges in natural PGM

- *Trust Region Policy Optimization*
  - ↳ *Regularize learning rate via backtracking line*
    - ↳ *This guarantees the satisfaction of constraint*
  - ↳ *Use sampling to find the estimate* $\hat{\mathbf{H}}$
  - ↳ *Use gradient conjugate to compute the inverse*
- *Proximal Policy Optimization*
  - ↳ *Skip all these steps by computationally-efficient clipping*
    - ↳ *The clipping guarantees the satisfaction of constraint*
  - ↳ *We do not need to find estimate* $\hat{\mathbf{H}}$ *anymore*

Natural Policy Gradient: *Main Challenges*

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \sqrt{\frac{2d_{\max}}{\nabla_k^\intercal \mathbf{H}_k^{-1} \nabla_k}} \mathbf{H}_k^{-1} \nabla_k$$

If we update with this rule: *we could see*

1. *the new point $\boldsymbol{\theta}_{k+1}$ does not fulfill what we expect, i.e.,*
   - ↳ *it might do no improvement*

$$\mathcal{J}\left(\pi_{\boldsymbol{\theta}_{k+1}}\right) \leqslant \mathcal{J}\left(\pi_{\boldsymbol{\theta}_k}\right)$$

   - ↳ *it might violate the constraint*

$$\bar{D}_{\mathrm{KL}}\left(\pi_{\boldsymbol{\theta}_{k+1}} \| \pi_{\boldsymbol{\theta}_k}\right) > d_{\max}$$

\+ *But, didn't we solve the optimization problem?!*

– *Well! We did it approximately not exactly*

Natural Policy Gradient: *Main Challenges*

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \sqrt{\frac{2d_{\max}}{\nabla_k^\intercal \mathbf{H}_k^{-1} \nabla_k}} \mathbf{H}_k^{-1} \nabla_k$$

If we update with this rule: *we need to*

2. *compute Hessian of* $\bar{D}_{\mathrm{KL}}\left(\pi_{\boldsymbol{\theta}} \| \pi_{\boldsymbol{\theta}_k}\right)$
   - ↳ *we need to compute all second order derivatives, i.e., for all choices of* i *and* j

$$\frac{\partial^2}{\partial \theta_i \partial \theta_j} \bar{D}_{\mathrm{KL}}\left(\pi_{\boldsymbol{\theta}} \| \pi_{\boldsymbol{\theta}_k}\right)$$

   - ↳ *say we use ResNet-50 with* 2.6×10^7 *trainable parameters*
     - ↳ *we need to compute about* 6.6×10^{14} *derivatives*
     - ↳ *this in principle changes the complexity in orders of magnitude*

Natural Policy Gradient: *Main Challenges*

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \sqrt{\frac{2d_{\max}}{\nabla_k^\intercal \mathbf{H}_k^{-1} \nabla_k}} \mathbf{H}_k^{-1} \nabla_k$$

Say we computed the Hessian: *we need to*

3. *invert the Hessian of* $\mathbf{H}_k \in \mathbb{R}^{D \times D}$
   - ↳ *the complexity scales as* $\mathcal{O}\left(D^{\xi}\right)$
     - ↳ $\xi = 3$ *for classical Gauss-Jordan algorithm*
     - ↳ *it can reduces to* $\xi \approx 2.4$ *if we use more optimized algorithms like CW*
   - ↳ *at the end, this is computationally very expensive*

TRPO: *Backtracking Line*

The first algorithmic approach proposed by Schulman et. al was

$$\textit{Trust Region Policy Optimization} \equiv \textit{TRPO}$$

It uses two simple ideas to overcome the mentioned issues

- *Backtracking line challenge to get rid of the first issue*
- *Conjugate gradient to overcome the other two*

Let's take a look

TRPO: *Backtracking Line*

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \sqrt{\frac{2d_{\max}}{\nabla_k^\intercal \mathbf{H}_k^{-1} \nabla_k}} \mathbf{H}_k^{-1} \nabla_k$$

Through analysis it turns out that: the direction of natural policy gradient is effective; however, the step size might be overshooting

+ *Why don't we scale it back?*
– *Sure! We can do this efficiently via backtracking line*

TRPO: *Backtracking Line*

`BacktrackLine()`:

1: *Choose some* $\alpha < 1$, *set* $i = 0$ *and start with some* $\delta > d_{\max}$
2: ***while*** $\delta > d_{\max}$ ***do***
3: *Replace* $\boldsymbol{\theta}_{k+1}$ *with*

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k + \alpha^i \sqrt{\frac{2d_{\max}}{\nabla_k^\intercal \mathbf{H}_k^{-1} \nabla_k}} \mathbf{H}_k^{-1} \nabla_k$$

4: *Set* $\delta \leftarrow \bar{D}_{\mathrm{KL}}\left(\pi_{\boldsymbol{\theta}_{k+1}} \| \pi_{\boldsymbol{\theta}_k}\right)$
5: *Update* $i \leftarrow i + 1$
6: ***end while***

+ *But we are only checking the constraint?!*
– It turns out that *this could also guarantee policy improvement*

TRPO: *Conjugate Gradient*

The next trick in TRPO is to write down the update in a form that can be computed via conjugate gradient: let's take a look at the update rule

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k + \alpha^i \sqrt{\frac{2d_{\max}}{\nabla_k^\intercal \mathbf{H}_k^{-1} \nabla_k}} \mathbf{H}_k^{-1} \nabla_k$$

We can define the vector

$$\mathbf{y}_k = \mathbf{H}_k^{-1} \nabla_k$$

It is then easy to say that

$$\nabla_k^\intercal \mathbf{H}_k^{-1} \nabla_k = \nabla_k^\intercal \mathbf{H}_k^{-1} \mathbf{I} \nabla_k = \nabla_k^\intercal \mathbf{H}_k^{-1} \underbrace{\mathbf{H}_k \mathbf{H}_k^{-1}}_{\mathbf{I}} \nabla_k$$

$$= \underbrace{\nabla_k^\intercal \mathbf{H}_k^{-1}}_{\mathbf{y}_k^\intercal} \mathbf{H}_k \underbrace{\mathbf{H}_k^{-1} \nabla_k}_{\mathbf{y}_k} = \mathbf{y}_k^\intercal \mathbf{H}_k \mathbf{y}_k$$

TRPO: *Conjugate Gradient*

If we have $\mathbf{y}_k$*, we could update more easily*

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k + \alpha^i \sqrt{\frac{2d_{\max}}{\mathbf{y}_k^\intercal \mathbf{H}_k \mathbf{y}_k}} \mathbf{y}_k$$

Let's see if there is any efficient way to find $\mathbf{y}_k$ *at least approximately*

$$\mathbf{y}_k = \mathbf{H}_k^{-1} \nabla_k \rightsquigarrow \mathbf{H}_k \mathbf{y}_k = \nabla_k$$

Now, let's define $\mathbf{g}(\boldsymbol{\theta}) = \nabla \mathcal{L}_k(\boldsymbol{\theta})$*: obviously, we have*

$$\nabla_k = \mathbf{g}(\boldsymbol{\theta}_k)$$
$$\mathbf{H}_k = \nabla \mathbf{g}(\boldsymbol{\theta}) \big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_k}$$

TRPO: *Conjugate Gradient*

Let's use these facts to expand our equation

$$\begin{aligned} \mathbf{y}_k = \mathbf{H}_k^{-1}\nabla_k \rightsquigarrow \mathbf{H}_k\mathbf{y}_k &= \nabla_k \\ \nabla\mathbf{g}\left(\boldsymbol{\theta}_k\right)\mathbf{y}_k &= \mathbf{g}\left(\boldsymbol{\theta}_k\right) \\ \nabla\left(\mathbf{g}\left(\boldsymbol{\theta}\right)\mathbf{y}_k\right)|_{\boldsymbol{\theta}=\boldsymbol{\theta}_k} &= \mathbf{g}\left(\boldsymbol{\theta}_k\right) \end{aligned}$$

The above functional equation can be solved for $\mathbf{y}_k$ *via conjugate gradient algorithm*[1]*, even without knowing the complete* $\mathbf{H}_k = \nabla\mathbf{g}\left(\boldsymbol{\theta}_k\right)$*!*

In practice, we do the following

- *Compute the gradient estimator* $\hat{\nabla}_k$
- *Compute a sample Hessian* $\hat{\mathbf{H}}_k$
- *Solve* $\hat{\mathbf{H}}_k\mathbf{y}_k = \hat{\nabla}_k$ *via conjugate gradient*

[1] *You could check this tutorial if you are interested to know more about that*

TRPO: *Comments on Estimating Hessian*

As long as we need only an estimate, we can estimate Hessian by sampling: if we extend our derivative in Assignment 3, we will see

$$
\begin{aligned}
\mathbf{H}_k &= \nabla^2 \bar{D}_{\mathrm{KL}}\left(\pi_{\boldsymbol{\theta}} \| \pi_{\boldsymbol{\theta}_k}\right)|_{\boldsymbol{\theta}_k} \\
&= \int_s \int_a d_{\boldsymbol{\theta}_k}(s) \nabla \pi_{\boldsymbol{\theta}_k}(a|s) \nabla \log \pi_{\boldsymbol{\theta}_k}(a|s)^{\mathsf{T}} \\
&= \int_s \int_a \underbrace{d_{\boldsymbol{\theta}_k}(s) \pi_{\boldsymbol{\theta}_k}(a|s)}_{\text{distribution}} \underbrace{\nabla \log \pi_{\boldsymbol{\theta}_k}(a|s) \nabla \log \pi_{\boldsymbol{\theta}_k}(a|s)^{\mathsf{T}}}_{\text{sample outer product}} \\
&= \mathbb{E}_{\pi_{\boldsymbol{\theta}_k}} \left\{ \nabla \log \pi_{\boldsymbol{\theta}_k}(A|S) \nabla \log \pi_{\boldsymbol{\theta}_k}(A|S)^{\mathsf{T}} \right\}
\end{aligned}
$$

This is the Fisher information matrix and can be estimated by sampling

TRPO

`TRPO()`:

Initiate with $\boldsymbol{\theta}$ *and dampening factor* $\alpha < 1$
while *interacting* ***do***
  for *mini-batch* $b = 1 : B$ ***do***
    Sample $S_0, A_0 \xrightarrow{R_1} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$ *by policy* $\pi_{\boldsymbol{\theta}}$
    Compute sample $U_t = R_{t+1} + \gamma v_{\pi_{\boldsymbol{\theta}}}(S_{t+1}) - v_{\pi_{\boldsymbol{\theta}}}(S_t)$ ***for*** $t = 0 : T-1$
    Compute sample gradient as $\hat{\nabla}_b = \sum_t U_t \nabla \log \pi_{\boldsymbol{\theta}}(A_t | S_t)$
  end for
  Compute estimator as $\hat{\nabla} = \texttt{mean}(\hat{\nabla}_1, \ldots, \hat{\nabla}_B)$ *and a Hessian estimator* $\hat{\mathbf{H}}$
  Solve $\hat{\mathbf{H}}\mathbf{y} = \hat{\nabla}$ *for* $\mathbf{y}$ *via conjugate gradient with multiple iterations*
  Backtrack on a line: find minimum integer i *such that*

$$\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta} + \alpha^i \sqrt{\frac{2 d_{\max}}{\mathbf{y}^\intercal \hat{\mathbf{H}} \mathbf{y}}} \mathbf{y}$$

  satisfies $\bar{D}_{\mathrm{KL}}(\pi_{\boldsymbol{\theta}'} \| \pi_{\boldsymbol{\theta}}) \leqslant d_{\max}$
  Update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}'$
end while

Back to Trust Region PGM

`AdvantageGD()`:

1: *Start with some initial* $\boldsymbol{\theta}_0$
2: ***for*** $k = 1 : K$ ***do***
3: *Compute the surrogate function* $\mathcal{L}_k(\pi_{\boldsymbol{\theta}})$
4: *Update the parameters as*

$$\boldsymbol{\theta}_{k+1} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \mathcal{L}_k(\pi_{\boldsymbol{\theta}}) \quad \textit{subject to} \quad \pi_{\boldsymbol{\theta}} \textit{ and } \pi_{\boldsymbol{\theta}_k} \textit{ are close}$$

5: ***end for***

+ *Was this whole "closeness" metric worth it?*
– *Well! Maybe not!*

Back to Trust Region PGM: *Alternative Formulation*

Let's check back what was our concern: we wanted to maximize

$$\mathcal{L}_k\left(\pi_{\boldsymbol{\theta}}\right) = \mathbb{E}_{\pi_{\boldsymbol{\theta}_k}}\left\{u_{\pi_{\boldsymbol{\theta}_k}}\left(S, A\right)\frac{\pi_{\boldsymbol{\theta}}\left(A|S\right)}{\pi_{\boldsymbol{\theta}_k}\left(A|S\right)}\right\}$$

while making sure that

$$\mathbb{V}\mathrm{ar}\left\{\hat{\mathcal{L}}_k\left(\pi_{\boldsymbol{\theta}}\right)\right\} \propto \frac{\pi_{\boldsymbol{\theta}}\left(A|S\right)}{\pi_{\boldsymbol{\theta}_k}\left(A|S\right)}$$

does not explode!

- \+ *Why don't we check the ratio of policies for "closeness"?*
- – *Sounds like a good idea!*

Trust Region PGM: *Ratio-Limited Policy Optimization*

Let's assume $\mathcal{C}(\cdot)$ is a function that limits its argument into a restricted interval of variation: then we can define

$$\tilde{\mathcal{L}}_k(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}_k}}\left\{ u_{\pi_{\boldsymbol{\theta}_k}}(S,A)\, \mathcal{C}\left(\frac{\pi_{\boldsymbol{\theta}}(A|S)}{\pi_{\boldsymbol{\theta}_k}(A|S)} \right) \right\}$$

If we optimize this surrogate function, we proximally satisfy what we want

`LimitedRatioAdvantageGD():`

1: *Start with some initial* $\boldsymbol{\theta}_0$
2: ***for*** $k = 1 : K$ ***do***
3: *Compute the surrogate function* $\mathcal{L}_k(\pi_{\boldsymbol{\theta}})$
4: *Update the parameters as*

$$\boldsymbol{\theta}_{k+1} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \tilde{\mathcal{L}}_k(\pi_{\boldsymbol{\theta}})$$

5: ***end for***

Proximal Policy Optimization

A common form of this approach is used in

$$\textit{Proximal Policy Optimization} \equiv \textit{PPO}$$

In this algorithm, we set

$$\tilde{\mathcal{L}}_k(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}_k}}\left\{\mathcal{L}_k^{\text{Clip}}(S, A, \boldsymbol{\theta})\right\}$$

where $\mathcal{L}_k^{\text{Clip}}(S, A, \boldsymbol{\theta})$ *is importance sample of advantage with clipped ratio,i.e.,*

$$\mathcal{L}_k^{\text{Clip}}(S, A, \boldsymbol{\theta}) = \min\left\{u_{\pi_{\boldsymbol{\theta}_k}}(S, A)\frac{\pi_{\boldsymbol{\theta}}(A|S)}{\pi_{\boldsymbol{\theta}_k}(A|S)}, \ell_\varepsilon\left(u_{\pi_{\boldsymbol{\theta}_k}}(S, A)\right)\right\}$$

for the clipping function

$$\ell_\varepsilon(x) = \begin{cases}(1+\varepsilon)x & x > 0\\ (1-\varepsilon)x & x \leqslant 0\end{cases}$$

Proximal Policy Optimization

+ *This clipping looks quite complicated! How does it restrict the domain of variation?*
– *It is indeed complicated, but we may understand it by a simple example*

Say we have only one sample trajectory with single state S and action A: we hence estimate the restricted surrogate as

$$\tilde{\mathcal{L}}_k\left(\pi_{\boldsymbol{\theta}}\right) \approx \mathcal{L}_k^{\text{Clip}}\left(S, A, \boldsymbol{\theta}\right)$$

Now, say that this sample pair gives sample advantage $u_{\pi_{\boldsymbol{\theta}_k}}\left(S, A\right)$: this can be

- *a positive advantage*
- *a negative advantage*

Let's see output of our restricted surrogate in each case

Proximal Policy Optimization

+ *What happens when we have a positive sample advantage?*
– *In this case, we have*

$$\mathcal{L}_k^{\text{Clip}}(S, A, \boldsymbol{\theta}) = u_{\pi_{\boldsymbol{\theta}_k}}(S, A) \min\left\{\frac{\pi_{\boldsymbol{\theta}}(A|S)}{\pi_{\boldsymbol{\theta}_k}(A|S)}, 1+\varepsilon\right\}$$

Since the advantage is positive, surrogate is optimized by $\boldsymbol{\theta}$ that increases the ratio: the clipping operator lets us do it only up to some $\boldsymbol{\theta}$ that

$$\frac{\pi_{\boldsymbol{\theta}}(A|S)}{\pi_{\boldsymbol{\theta}_k}(A|S)} \leqslant 1+\varepsilon$$

if the ratio happens to be more, it clips it by $1+\varepsilon$

Proximal Policy Optimization

+ *What happens when we have a negative sample advantage?*
– *In this case, we have*

$$\mathcal{L}_k^{\text{Clip}}(S, A, \boldsymbol{\theta}) = u_{\pi_{\boldsymbol{\theta}_k}}(S, A) \max\left\{ \frac{\pi_{\boldsymbol{\theta}}(A|S)}{\pi_{\boldsymbol{\theta}_k}(A|S)}, 1-\varepsilon \right\}$$

Since the advantage is negative, surrogate is maximized by $\boldsymbol{\theta}$ that reduces the ratio: the clipping operator lets us do it only up to some $\boldsymbol{\theta}$ that

$$\frac{\pi_{\boldsymbol{\theta}}(A|S)}{\pi_{\boldsymbol{\theta}_k}(A|S)} \geqslant 1-\varepsilon$$

if the ratio happens to lie below, it clips it by $1-\varepsilon$

Proximal Policy Optimization

Moral of Story

Clipping will keep the maximizer of the restricted surrogate such that the new policy described by the maximizer of the surrogate has controlled variation as compared to $\pi_{\boldsymbol{\theta}_k}$. This controlled variation is tuned by ε

Doing so we are still keeping our new policy within a trust region; however,

- *We don't need to check KL-divergence*
- *We don't need to estimate Hessian*
- *We don't need to implement conjugate gradient algorithm*
- *We don't need backtracking line*

Or in a nutshell: the life becomes much easier ☺

PPO Algorithm

`PPO()`:

Initiate with $\boldsymbol{\theta}$ *and learning* $\alpha < 1$
while *interacting* ***do***
  for *mini-batch* $b = 1 : B$ ***do***
    Sample $S_0, A_0 \xrightarrow{R_1} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$ *by policy* $\pi_{\boldsymbol{\theta}}$
    Compute sample $U_t = R_{t+1} + \gamma v_{\pi_{\boldsymbol{\theta}}}(S_{t+1}) - v_{\pi_{\boldsymbol{\theta}}}(S_t)$ ***for*** $t = 0 : T-1$
  end for
  Compute the restricted surrogate

$$\tilde{\mathcal{L}}(\pi_{\mathbf{x}}) = \texttt{mean}_b \left[\sum_{t=1}^{T} \min \left\{ U_t \frac{\pi_{\mathbf{x}}(A_t|S_t)}{\pi_{\boldsymbol{\theta}}(A_t|S_t)}, \ell_\varepsilon(U_t) \right\} \right]$$

  for $i = 1 : I$ *potentially* $I = 1$ ***do***
    Update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla \tilde{\mathcal{L}}(\pi_{\mathbf{x}})\big|_{\mathbf{x}=\boldsymbol{\theta}}$
  end for
end while

Sample Reuse with TRPO and PPO

+ *Very nice! You did a great job; however, you did not mention anything about sample efficiency!*
  - ↳ *With TRPO and PPO, we can make sure that our updated policy will be within the vicinity of previous policy*
  - ↳ *But, we still sample a mini-batch, apply SGD and drop it!*
- *Well! As long as we are using TRPO and PPO, we can reuse our previous samples for some time! This can help us with sample efficiency*

In practice, we can use experience buffer here as well

- *We collect multiple sample trajectory and save them into into a buffer*
- *We treat the buffer as a dataset and break it into mini-batches*
- *We go multiple epochs over this dataset*
- ⋆ *We remove old trajectories periodically as our policy is getting far gradually*

Sample Reuse with TRPO and PPO

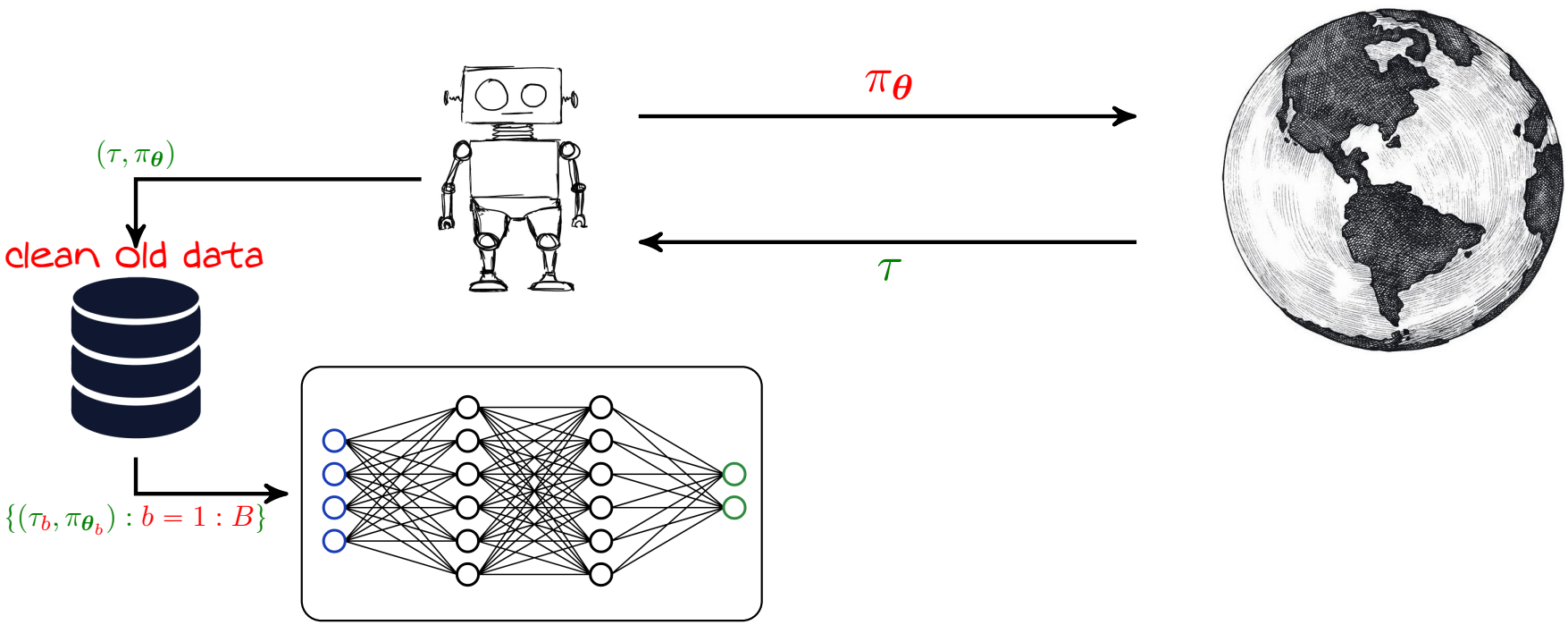
There is a tiny change that we need to consider in this case: when we compute the surrogate function, we should do the importance sampling with the policy that we sampled the trajectory with

For instance, say we sample B trajectories from the buffer

- *It might be that each trajectory has been sampled by one policy $\pi_{\boldsymbol{\theta}_b}$*
- ⚠ *They are all close policies as we clean buffer periodically*
- *If we use PPO, we could compute the surrogate as*

$$\tilde{\mathcal{L}}\left(\pi_{\mathbf{x}}\right) = \texttt{mean}_b\left[\sum_{t=1}^{T} \min\left\{U_t \frac{\pi_{\mathbf{x}}\left(A_t|S_t\right)}{\pi_{\boldsymbol{\theta}_b}\left(A_t|S_t\right)}, \ell_\varepsilon\left(U_t\right)\right\}\right]$$

Sample Reuse with TRPO and PPO: *Visualization*



$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla \tilde{\mathcal{L}}_k \left(\pi_{\boldsymbol{\theta}}\right)$$

multiple epochs

PPO Algorithm: *Sample Efficient Example*

`PPO()`:

1: *Initiate with* $\boldsymbol{\theta}$, *learning* $\alpha < 1$, *and an experience buffer with limited size*
2: ***while*** *interacting* ***do***
3: *Sample* $\tau : S_0, A_0 \xrightarrow{R_1} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$ *by policy* $\pi_{\boldsymbol{\theta}}$
4: ***if*** *experience buffer is full* ***then***
5: *Remove oldest sample*
6: ***end if***
7: *Save sample* $(\tau, \pi_{\boldsymbol{\theta}})$ *into experience buffer as most recent*
8: ***for*** $i = 1 : I$ *potentially for multiple epochs of buffer* ***do***
9: *Sample a mini-batch with* B *trajectories from experience buffer*
10: *Compute the restricted surrogate*

$$\tilde{\mathcal{L}}\left(\pi_{\mathbf{x}}\right) = \texttt{mean}_b \left[\sum_{t=1}^{T} \min \left\{ U_t \frac{\pi_{\mathbf{x}}\left(A_t | S_t\right)}{\pi_{\boldsymbol{\theta}_b}\left(A_t | S_t\right)}, \ell_{\varepsilon}\left(U_t\right) \right\} \right]$$

11: *Update* $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla \tilde{\mathcal{L}}\left(\pi_{\mathbf{x}}\right)|_{\mathbf{x}=\boldsymbol{\theta}}$
12: ***end for***
13: ***end while***

Sample Reuse with TRPO and PPO: *Final Notes*

Though we use experience reply as in DQL, we should note

- *In DQL, we are not very restricted with memory update*
  - ↳ *We could keep all samples and reuse them*
  - ↳ *This was because we could go totally off-policy with DQL*
- *In policy optimization, we are strictly restricted with memory update*
  - ↳ *We could not use very old samples efficiently*
    - ↳ *If we use them, we will have large variance*
  - ↳ *We can only mildly go off-policy*
    - ↳ *We keep a sample and squeeze the its most possible juice*

Important Conclusion

In terms of sample efficiency, we always have

$$\textit{Policy Gradient Methods} \ll \textit{DQL}$$

But they could become more stable than DQL as they directly control the policy

Last Stop: *Actor-Critic Approaches*

We are finished with PGMs

- ✓ *We know how to train efficiently a policy network*
  - ↳ *We can use TRPO and PPO pretty much in any problem*
- ✗ *But we assumed that we have access to the value function*
  - ↳ *This is not really the case in practice!*

We now go for the last chapter, where we learn to

approximate the value function via a value network

This will complete our box of tools and we are ready to solve any RL problem!

Efficient Implementation: *TorchRL*

TorchRL

In larger RL projects, you might find it easier to have access to some pre-implemented modules: TorchRL does that for you

- *It's a library implemented in PyTorch*
- *It contains lots of useful modules, e.g., to implement experience replay*
- *It does not give you implemented algorithms*
  - ↳ *Instead, it gives you modules that you need to implement the algorithm*
- *It's compatible with Gymnasium*

Since we often use PyTorch for DL implementations and Gymnasium to implement environment, TorchRL is a very efficient toolbox

Torch RL: *Sample Modules*

TorchRL

Some sample lines of code

```
from torchrl.collectors import SyncDataCollector
from torchrl.data.replay_buffers import ReplayBuffer
from torchrl.data.replay_buffers.samplers import SamplerWithoutReplacement
from torchrl.data.replay_buffers.storages import LazyTensorStorage
```

Some Resources

- *Take a look at the introductory presentation by Vincent Moens*
- *Go over its documentation at TorchRL page*