Policy Gradient

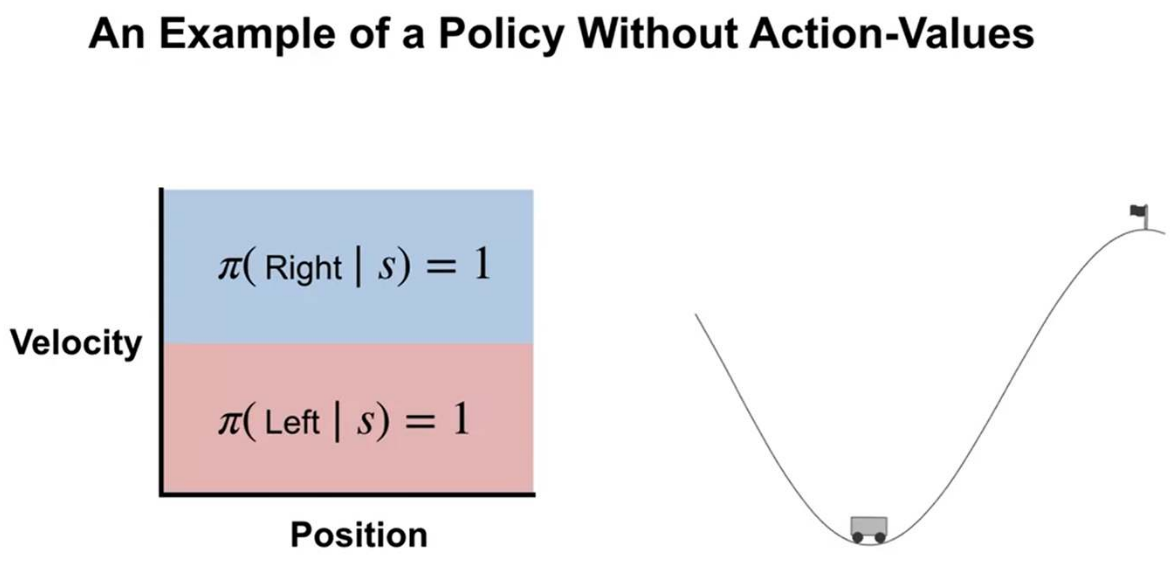
By Rohit Pardasani

References:

https://mitpress.mit.edu/books/reinforcement-learning

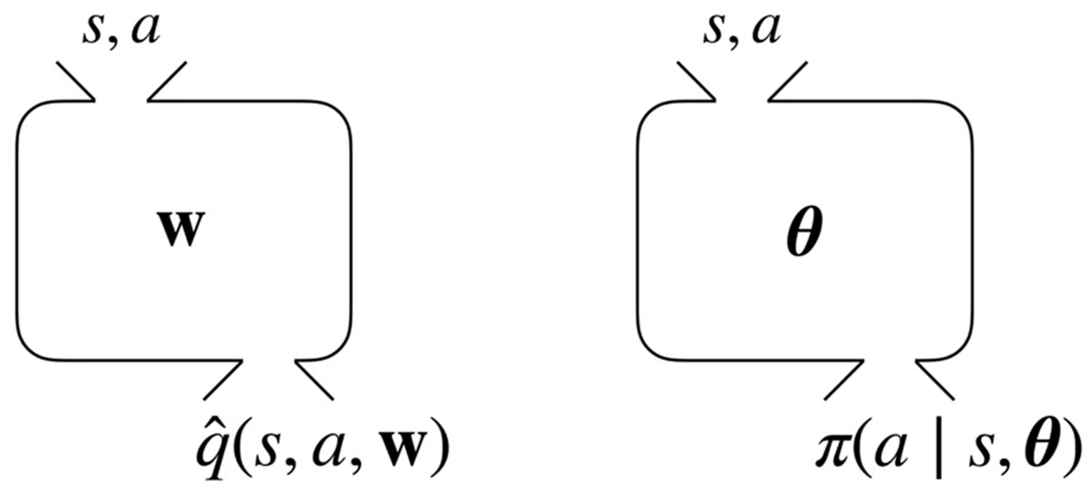
https://www.coursera.org/specializations/reinforcement-learning

Learning Policies Directly



Let's think about what it means to specify a policy directly.
We'll do this in mountain car.
Previously, we used epsilon-greedy to convert approximate action values into a policy.
But we can also consider policy which maps states directly to actions without first computing action values.
For example, in mountain car, we can define such a policy

Parameterizing Policies Directly


s, a
$\mathbf{w}$
$\hat{q}(s, a, \mathbf{w})$
s, a
$\boldsymbol{\theta}$
$\pi(a \mid s, \boldsymbol{\theta})$

We're not actually going to specify policies by hand. Rather, we will learn them.
We can use the language of function approximation to both represent and learn policies directly.
We'll use the **θ** for the policies parameter vector, distinguishing it from the parameters **W** for the approximate value function.
We use the notation **π (a | s, θ)** , to denote the parameterized policy.
For a given input state and action, the parameterized policy function will output the probability of taking that action in that state.
This mapping will be controlled by the parameters **θ**

Constraints on the Policy Parameterization

$\pi(a \mid s, \boldsymbol{\theta}) \geq 0$ **for all** $a \in \mathscr{A}$ **and** $s \in \mathscr{S}$

$\sum_{a \in \mathscr{A}} \pi(a \mid s, \boldsymbol{\theta}) = 1$ **for all** $s \in \mathscr{S}$

The Softmax Policy Parameterization

$$\pi(a \mid s, \boldsymbol{\theta}) \doteq \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_{b \in \mathcal{A}} e^{h(s,b,\boldsymbol{\theta})}}$$

Action Preference

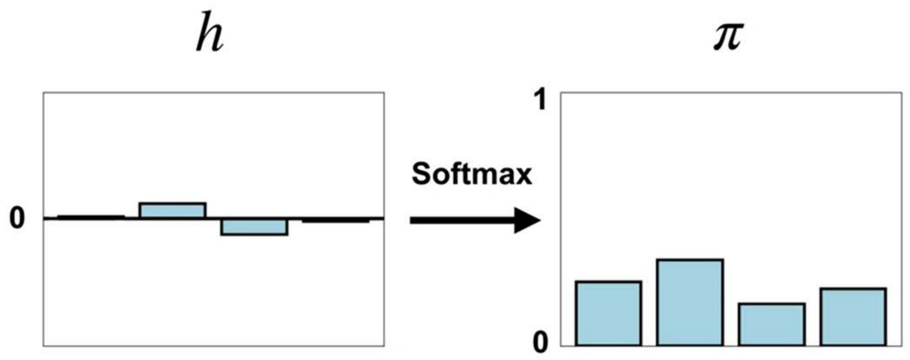
$< \mathbf{x}(s, a), \boldsymbol{\theta} >$

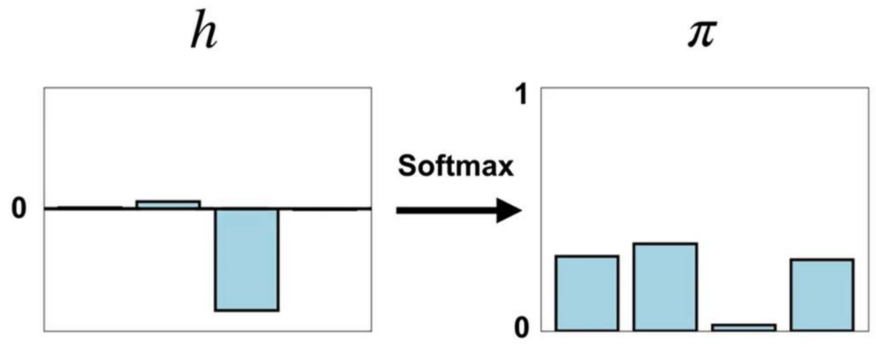
Neural Network

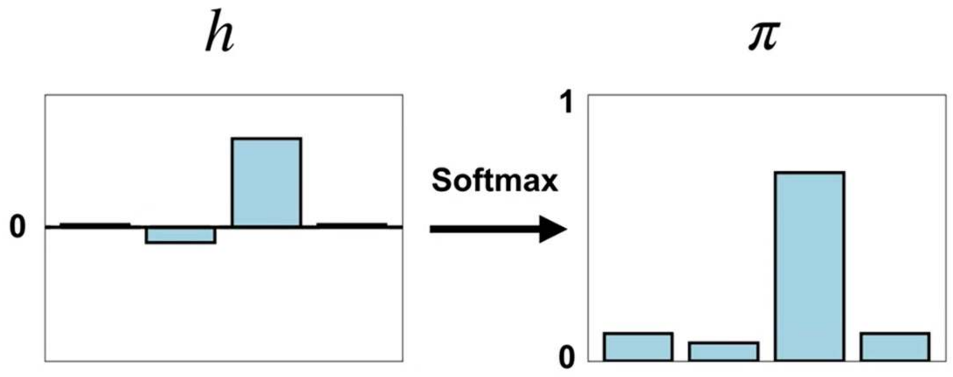
...

Softmax policy is a simple and effective way to parametrize policy while satisfying the policy conditions.
The function h shown here, is called the action preference.
A higher preference for a particular action in a state, means that the action is more likely to be selected.
The action preference is a function of the state and action as well as a parameter vector **θ**.
The action preference, can be parameterized in any way we like since the softmax will enforce the constraints of a probability distribution.
For example, the action preferences could be a linear function of the state action features or something more complex like the output of a neural network.

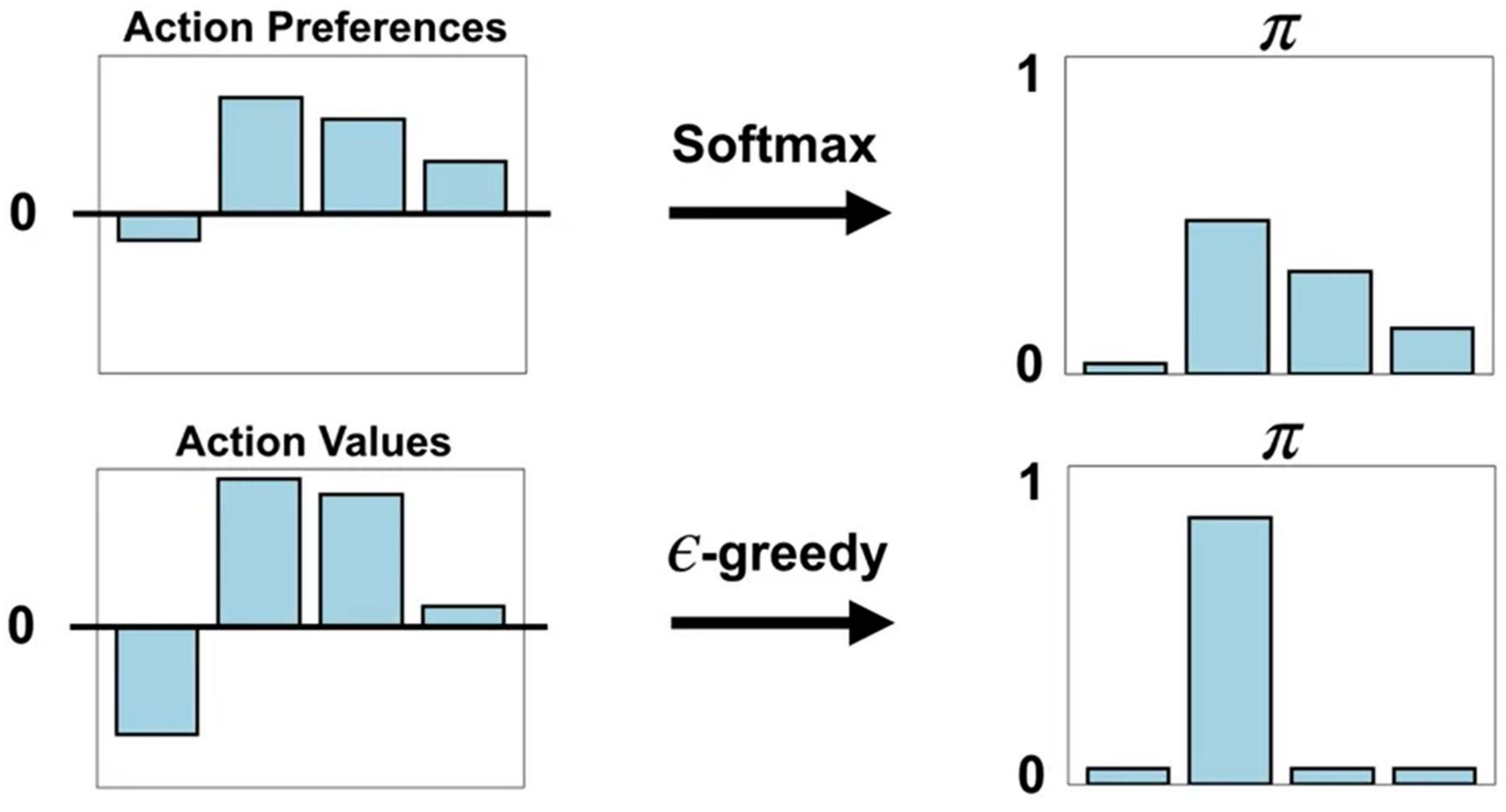
Examples of Softmax Policies







Action preferences are not action values

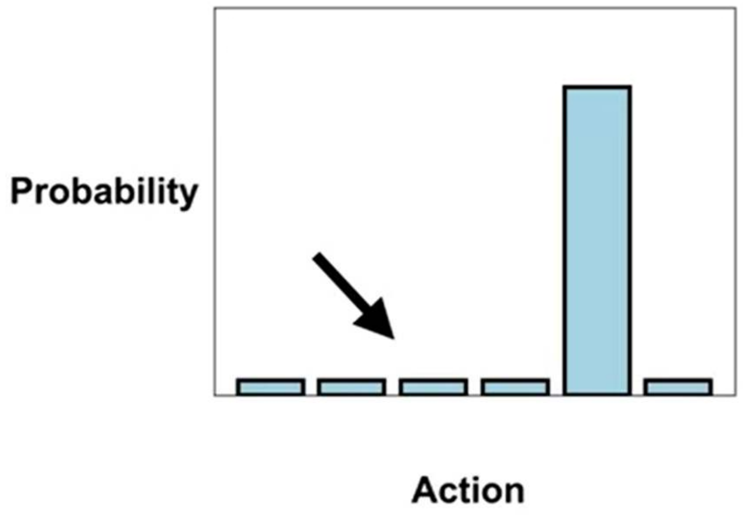


Advantages of Policy Parameterization

The Flexibility of Stochastic Policies

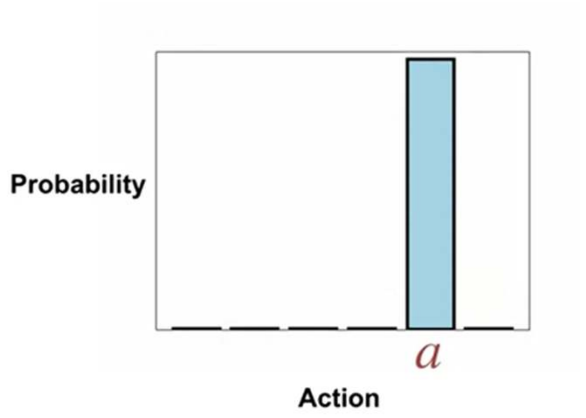
First, the agent can make its policy more greedy over time autonomously.
Why would you want that?
Well, in the beginning, the agent's estimates are not that accurate.
So you would want the agent to explore a lot.
As the estimates become more accurate, the agent should become more and more greedy.

Recall the Epsilon greedy policy we used before.
The Epsilon step chooses a random action to ensure continual exploration. However, the Epsilon probability puts a cap on how good the resulting policy can be.



The Flexibility of Stochastic Policies

$$\pi(a \mid s, \boldsymbol{\theta}) \doteq \frac{e^{h(s,a,\boldsymbol{\theta})} \uparrow}{\sum_{b \in \mathcal{A}} e^{h(s,b,\boldsymbol{\theta})}}$$



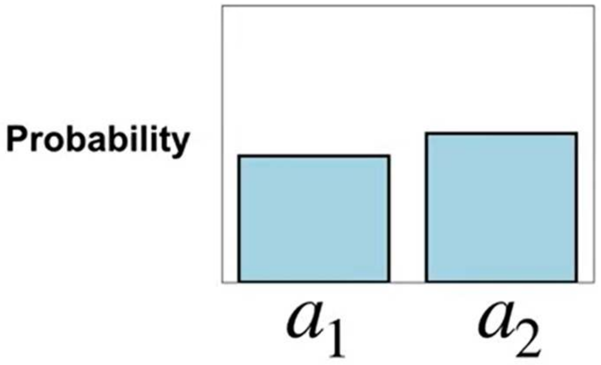
We could, of course, switch to a greedy policy when we think the agent has explored adequately.
But we want our agents to be autonomous.
We don't want them to rely on us to decide when exploration is done.
We can avoid this issue with parameterized policies.
The policy can start off stochastic to guarantee exploration.
Then as learning progresses, the policy can naturally converge towards a deterministic greedy policy.
A softmax policy can adequately approximate a deterministic policy by making one action preference very large.

Stochastic policies might be better than deterministic policies under function approximation

$$\pi_* : \mathcal{S} \to \mathcal{A}$$

$$\pi_*(s_1) = a_1$$

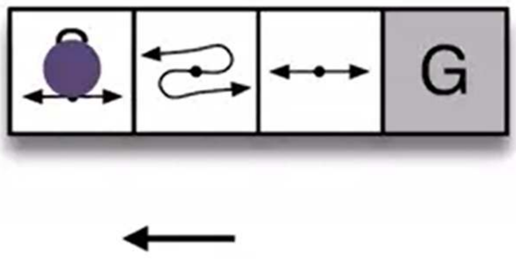
$$\pi_*(s_2) = a_2$$



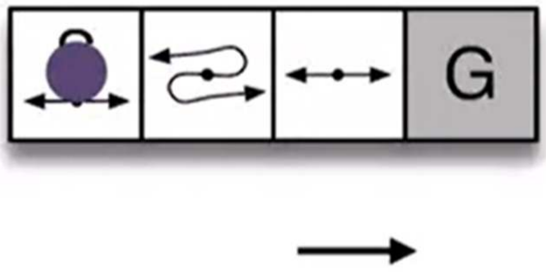
In the tabular setting, we learn there's always a deterministic optimal policy.
In function approximation, we may not be able to represent this deterministic policy.
Instead, if the optimal approximate policy is a stochastic policy, then function approximation might do better.

Stochastic policies can be useful with function approximation

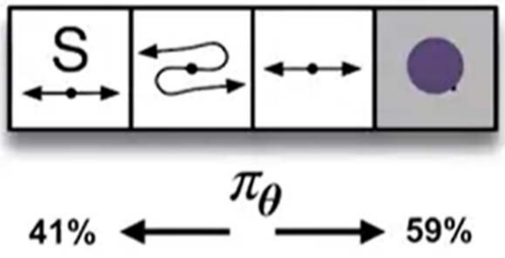
$R = -1$



$v_{\text{always left}}(S) = -\infty$



$v_{\text{always right}}(S) = -\infty$



$v_{\pi_\theta}(S) = -11.6$

Agent needs to reach from S to G.
But in the second state the effect of actions gets switched, thus L causes agent to move right and vice-versa.
Always left and always right are not optimal policies.
Wile a stochastic policy is better.

The Objective for Learning Policies

The Goal of Reinforcement Learning:
Maximizing Reward in the Long Run

$$R_t, R_{t+1}, R_{t+2}, \cdots$$

Formalizing the Goal as an Objective

Episodic: $$G_t = \sum_{t=0}^{T} R_t$$

Continuing: $$G_t = \sum_{t=0}^{\infty} \gamma^t R_t \qquad G_t = \sum_{t=0}^{\infty} R_t - r(\pi)$$

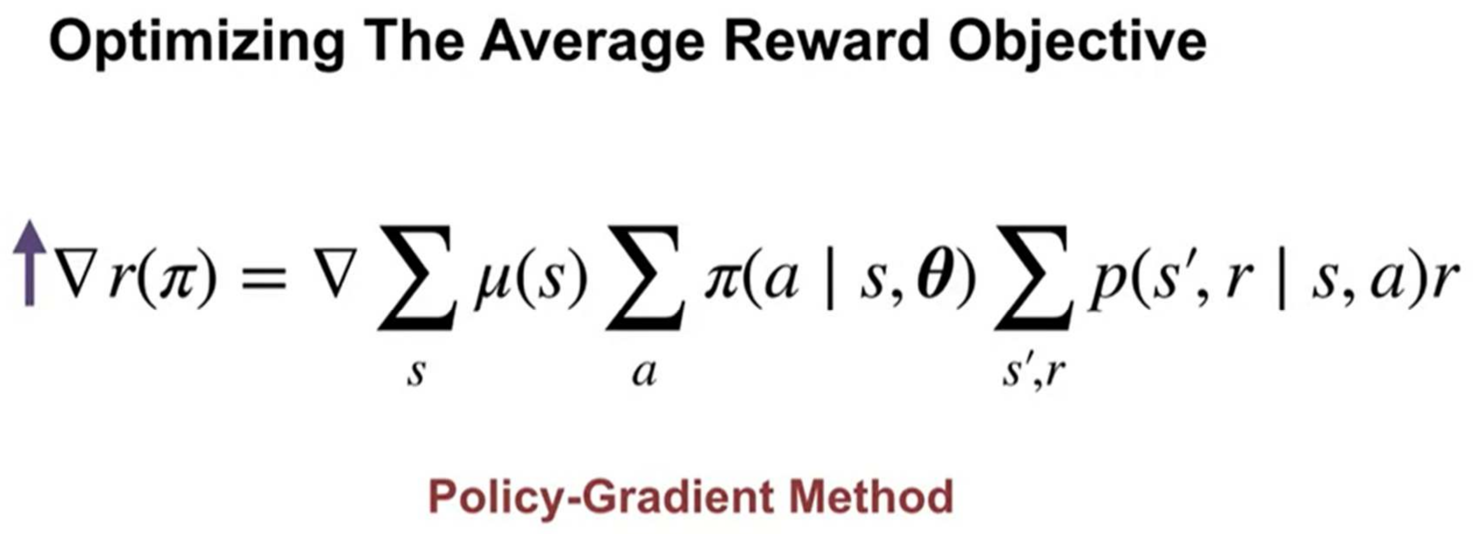
The Average Reward Objective

$$r(\pi) = \sum_{s} \mu(s) \sum_{a} \pi(a \mid s, \boldsymbol{\theta}) \sum_{s',r} p(s', r \mid s, a) r$$

$$r(\pi) = \sum_{s} \mu(s) \sum_{a} \pi(a \mid s, \boldsymbol{\theta}) \underbrace{\boxed{\sum_{s',r} p(s', r \mid s, a) r}}_{\mathbb{E}[R_t \mid S_t = s, A_t = a]}$$

$$r(\pi) = \sum_{s} \mu(s) \underbrace{\boxed{\sum_{a} \pi(a \mid s, \boldsymbol{\theta}) \sum_{s',r} p(s', r \mid s, a) r}}_{\mathbb{E}_\pi[R_t \mid S_t = s]}$$

$$r(\pi) = \underbrace{\boxed{\sum_{s} \mu(s) \sum_{a} \pi(a \mid s, \boldsymbol{\theta}) \sum_{s',r} p(s', r \mid s, a) r}}_{\mathbb{E}_\pi[R_t]}$$



Optimizing The Average Reward Objective

$$\uparrow \nabla r(\pi) = \nabla \sum_{s} \mu(s) \sum_{a} \pi(a \mid s, \boldsymbol{\theta}) \sum_{s',r} p(s', r \mid s, a) r$$

Policy-Gradient Method

Before we were minimizing the mean squared value error, and now we are maximizing an objective.

That means we will want to move in the direction of the gradient rather than the negative gradient.

The Challenge of Policy Gradient Methods

$$\nabla_{\boldsymbol{\theta}} r(\pi) = \nabla_{\boldsymbol{\theta}} \sum_{s} \underset{\nwarrow \text{ Depends on } \theta}{\mu(s)} \sum_{a} \pi(a \mid s, \boldsymbol{\theta}) \sum_{s',r} p(s', r \mid s, a) r$$

$$\nabla_{\mathbf{w}} \overline{VE} = \nabla_{\mathbf{w}} \sum_{s} \underset{\nwarrow \text{ Independent of } \mathbf{w}}{\mu(s)} [v_{\pi}(s) - \hat{v}(s, \mathbf{w})]^2$$

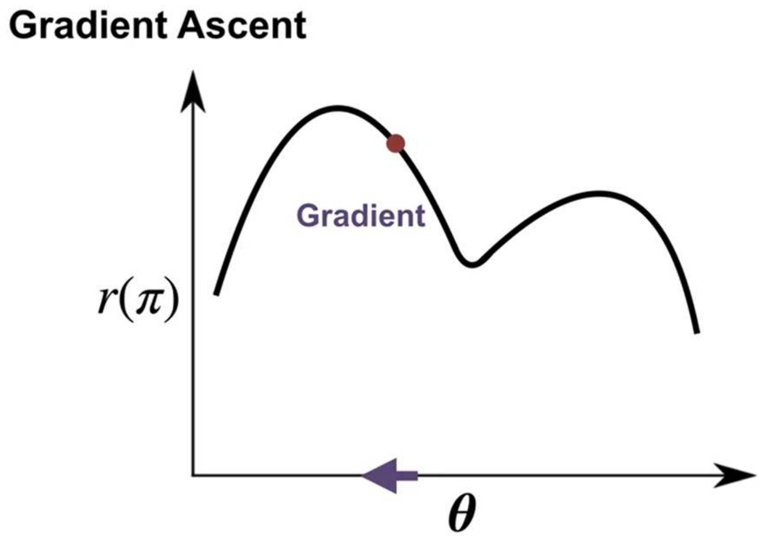
$$\begin{aligned} \nabla_{\mathbf{w}} \overline{VE} &= \nabla_{\mathbf{w}} \sum_{s} \mu(s) [v_{\pi}(s) - \hat{v}(s, \mathbf{w})]^2 \\ &= \sum_{s} \mu(s) \nabla_{\mathbf{w}} [v_{\pi}(s) - \hat{v}(s, \mathbf{w})]^2 \end{aligned}$$

The main difficulty is that modifying our policy π changes the distribution μ.

This contrast value function approximation where we minimized means grid value error under a particular policy. There the distribution μ was fixed. It does not change as the weights and the parameterized value function chains. We were therefore able to estimate gradients for states drawn from μ by simply following the policy.

This is less straightforward, a μ itself depends on the policy we are optimizing. Likely, there's an excellent theoretical answer to this challenge called the **policy gradient theorem**.

The Policy Gradient Theorem



The Gradient of the Objective

$$\boxed{\textbf{Product Rule: } \nabla(f(x)g(x)) = \nabla f(x)g(x) + f(x)\nabla g(x)}$$

$$\nabla r(\pi) = \nabla \sum_s \mu(s) \sum_a \pi(a \mid s, \boldsymbol{\theta}) \sum_{s',r} p(s', r \mid s, a) r$$

$$= \sum_s \nabla \mu(s) \sum_a \pi(a \mid s, \boldsymbol{\theta}) \sum_{s',r} p(s', r \mid s, a) r$$

$$+ \sum_s \mu(s) \nabla \sum_a \pi(a \mid s, \boldsymbol{\theta}) \sum_{s',r} p(s', r \mid s, a) r$$

Difficult to compute the first term because μ changes with π

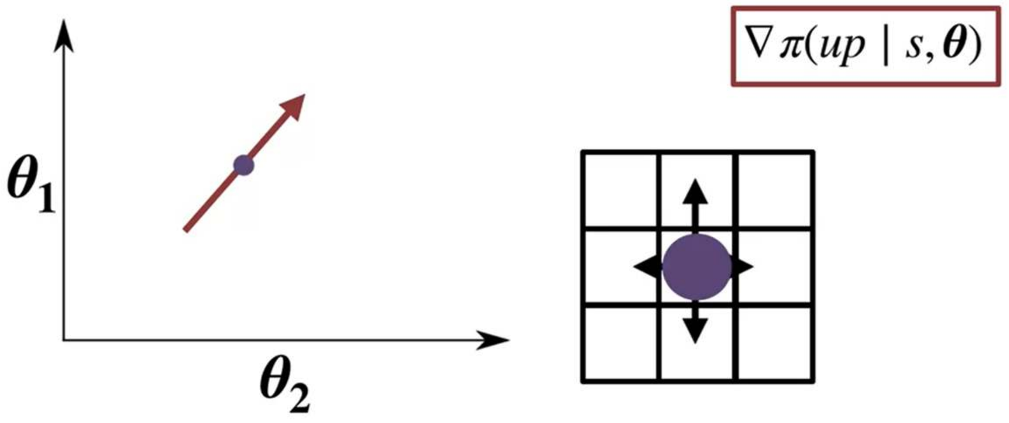
The Policy Gradient Theorem

Gives a easier route

$$\nabla r(\pi) = \sum_s \mu(s) \sum_a \nabla \pi(a \mid s, \boldsymbol{\theta}) q_\pi(s, a)$$

$$\nabla r(\pi) = \sum_s \mu(s) \sum_a \boxed{\nabla \pi(a \mid s, \boldsymbol{\theta}) q_\pi(s, a)}$$

Understanding $\sum_a \nabla \pi(a \mid s, \boldsymbol{\theta}) q_\pi(s, a)$



For simplicity, let's assume the policy is controlled by just two parameters.

The parameters are currently set to the point marked here.

The arrows indicate the agent's action probabilities in a particular state with the current parameter settings.

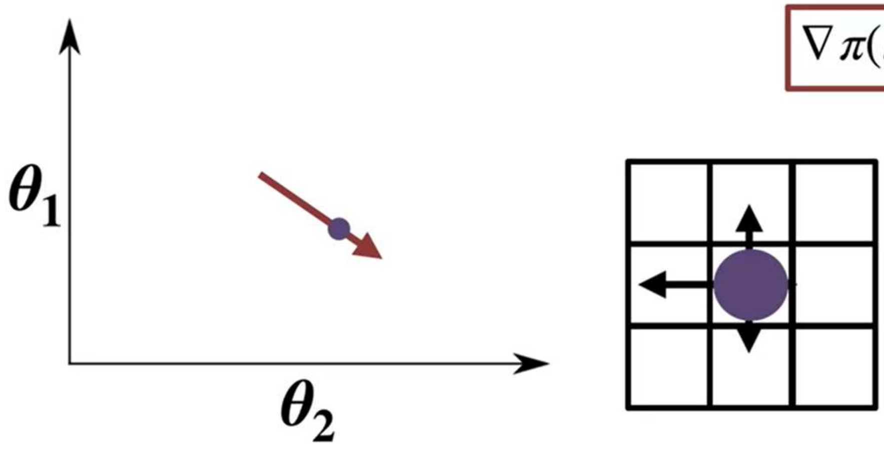
The gradient for the up action might look something like this on the plot.

The gradient tells us how to change the policy parameters to make that action more likely to be selected in the given state.

By moving the parameters in the direction of the gradient, we increase the probability for the up action. This necessarily means decrease in the probability of some of the other actions.

Understanding $\sum_a \nabla \pi(a \mid s, \boldsymbol{\theta}) q_\pi(s, a)$

$\boxed{\nabla \pi(left \mid s, \boldsymbol{\theta})}$

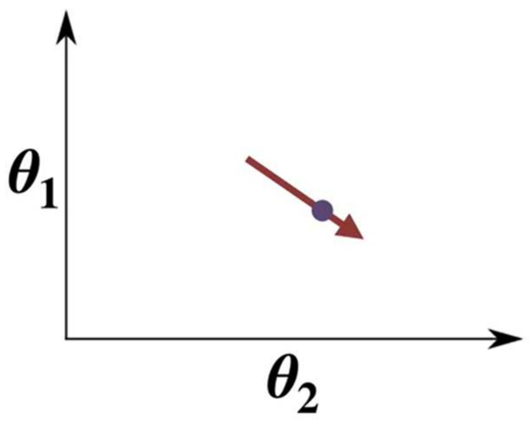


Different actions have different gradients.

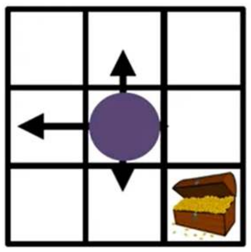
For example, the gradient of the left action probability may look like this.

Moving the parameters in that direction will increase the probability of the left action, and decrease the probability of some of the other actions.

Understanding $\sum_a \nabla \pi(a \mid s, \boldsymbol{\theta}) q_\pi(s, a)$



$$\sum_a \boxed{\nabla \pi(a \mid s, \boldsymbol{\theta}) q_\pi(s, a)}$$



Now, let's bring some reward into the situation and think about what this whole term means.

This is a sum over the gradients of each action probability weighted by the value of the associated action.

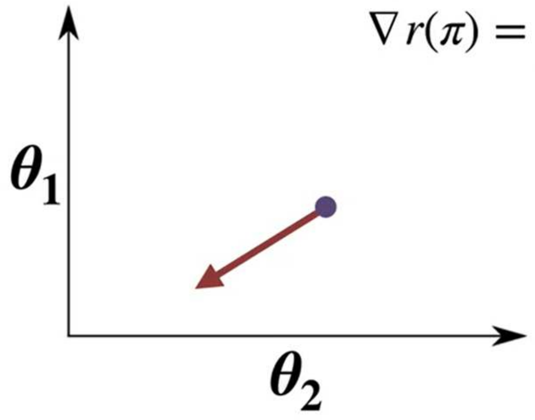
Imagine, we've added a rewarding state in the bottom right.

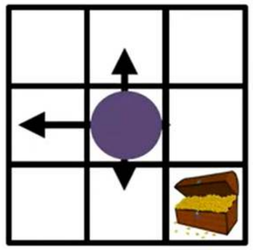
The up and left actions move away from it and should have negative value.

The down and right actions move toward it, and should have positive value.

Understanding $\sum_a \nabla \pi(a \mid s, \boldsymbol{\theta}) q_\pi(s, a)$

$$\nabla r(\pi) = \sum_s \mu(s) \sum_a \boxed{\nabla \pi(a \mid s, \boldsymbol{\theta}) q_\pi(s, a)}$$





$$q_\pi(s, up) = -1.3$$
$$q_\pi(s, left) = -0.9$$
$$q_\pi(s, down) = 0.7$$
$$q_\pi(s, right) = 1.5$$

The precise values will depend on the current policy.

But let's say they look something like this.

The weighted sum gives a direction to move the parameters that decreases the probability of moving up or right since their value is negative, while increasing the probability of moving down or left since their value is positive.

That direction might look like this.
The gradient expression given by the policy gradient theorem takes this expression and sums that over each state.

This gives the direction to move the policy parameters to most rapidly increase the overall average reward.

The Policy Gradient Theorem

$$\nabla r(\pi) = \sum_s \mu(s) \sum_a \nabla \pi(a \mid s, \boldsymbol{\theta}) q_\pi(s, a) \quad \xcancel{\nabla \mu(s)}$$

We now have a simple expression for the gradient.
Importantly, this expression does not contain the gradient of the state distribution mu, which is challenging to estimate.

Estimating the Policy Gradient

Getting Stochastic Samples of the Gradient

$$\nabla r(\pi) = \sum_s \mu(s) \sum_a \nabla \pi(a \mid s, \boldsymbol{\theta}) q_\pi(s, a)$$

$$\sum_a \nabla \pi(a \mid \boxed{S_t}, \boldsymbol{\theta}_t) q_\pi(\boxed{S_t}, a)$$

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \sum_a \nabla \pi(a \mid S_t, \boldsymbol{\theta}_t) q_\pi(S_t, a)$$

$$S_0, A_0, R_1, S_1, A_1, \ldots, \boxed{S_t}, A_t, R_{t+1}, \ldots$$

Computing the sum over states is really impractical. But we can do the same thing we did when deriving our stochastic gradient descent rule for policy evaluation.

We simply make updates from states we observe while following policy π.

This gradient from state St provides an approximation to the gradient of the average reward.

As we discussed before for stochastic gradient descent, we can adjust the weights with this approximation and still guarantee you will reach a stationary point.
This is what the stochastic gradient descent update looks like for the policy parameters.
We could stop here but let's simplify this further.

Unbiasedness of the Stochastic Samples

$$\nabla r(\pi) = \sum_s \mu(s) \sum_a \nabla \pi(a \mid s, \boldsymbol{\theta}) q_\pi(s, a)$$

$$= \mathbb{E}_\mu[\boxed{\sum_a} \nabla \pi(a \mid S, \boldsymbol{\theta}) q_\pi(S, a)]$$

$$S_0, A_0, R_1, S_1, A_1, \ldots, \overset{\sim \mu}{\boxed{S_t}}, A_t, R_{t+1}, \ldots$$

Notice that the sum over states waited by μ can be re-written as an expectation under μ.

Recall that μ is the stationary distribution for pi which reflects state visitation under ∏.

In fact, the state's we observe while following pi are distributed according to μ.

By computing the gradient from a state S_t, we get an unbiased estimate of this expectation.

$$\sum_a \nabla \pi(a \mid S, \boldsymbol{\theta}) q_\pi(S, a)$$

Getting Stochastic Samples with One Action

$$\sum_a \nabla \pi(a \mid S, \boldsymbol{\theta}) q_\pi(S, a)$$

$$= \sum_a \pi(a \mid S, \boldsymbol{\theta}) \boxed{\frac{1}{\pi(a \mid S, \boldsymbol{\theta})} \nabla \pi(a \mid S, \boldsymbol{\theta}) q_\pi(S, a)}$$

$$= \mathbb{E}_\pi[\frac{\nabla \pi(A \mid S, \boldsymbol{\theta})}{\pi(A \mid S, \boldsymbol{\theta})} q_\pi(S, A)]$$

Thinking about our stochastic gradient as an unbiased estimate suggests one other simplification.

Notice that inside the expectation we have a sum over all actions.

We want to make this term even simpler and get rid of the sum over all actions.

If this was an expectation over actions, we could get a stochastic example of this two and so avoid summing over all actions.

Stochastic Gradient Ascent for Policy Parameters

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \frac{\nabla \pi(A_t \mid S_t, \boldsymbol{\theta}_t)}{\pi(A_t \mid S_t, \boldsymbol{\theta}_t)} q_\pi(S_t, A_t)$$

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \nabla \ln \pi(A_t \mid S_t, \boldsymbol{\theta}_t) q_\pi(S_t, A_t)$$

$$\boxed{\nabla \ln\big(f(x)\big) = \frac{\nabla f(x)}{f(x)}} \qquad \nabla \ln \pi(a \mid s, \boldsymbol{\theta}) = \frac{\nabla \pi(a \mid s, \boldsymbol{\theta})}{\pi(a \mid s, \boldsymbol{\theta})}$$

Why use Logarithm? :
One reason is that it is actually simpler to compute the gradient of the logarithm of certain distributions.
The other less important reason is simply that it lets us write this gradient more compactly.

Computing the Update

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \overset{\checkmark}{\nabla} \ln \pi(A_t \mid S_t, \boldsymbol{\theta}_t) \overset{\checkmark}{q}_\pi(S_t, A_t)$$

Actor-Critic Algorithm

Do we have to choose between directly learning the policy parameters and learning a value function?

No. Even within policy gradient methods, value-learning methods like TD still have an important role to play.

In this setup, the parameterized policy plays the role of an actor, while the value function plays the role of a critic, evaluating the actions selected by the actor.

These so-called actor-critic methods, were some of the earliest TD-based methods introduced in reinforcement learning

Approximating the Action Value in the Policy Update

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \nabla \ln \pi(A_t \mid S_t, \boldsymbol{\theta}_t) q_\pi(S_t, A_t)$$

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \nabla \ln \pi(A_t \mid S_t, \boldsymbol{\theta}_t)[R_{t+1} - \bar{R} + \hat{v}(S_{t+1}, \mathbf{w})]$$

But we don't have access to qπ, so we'll have to approximate it.

We can do the usual TD thing, the one-step, bootstrap return.

That is, the differential reward plus the value of the next state.

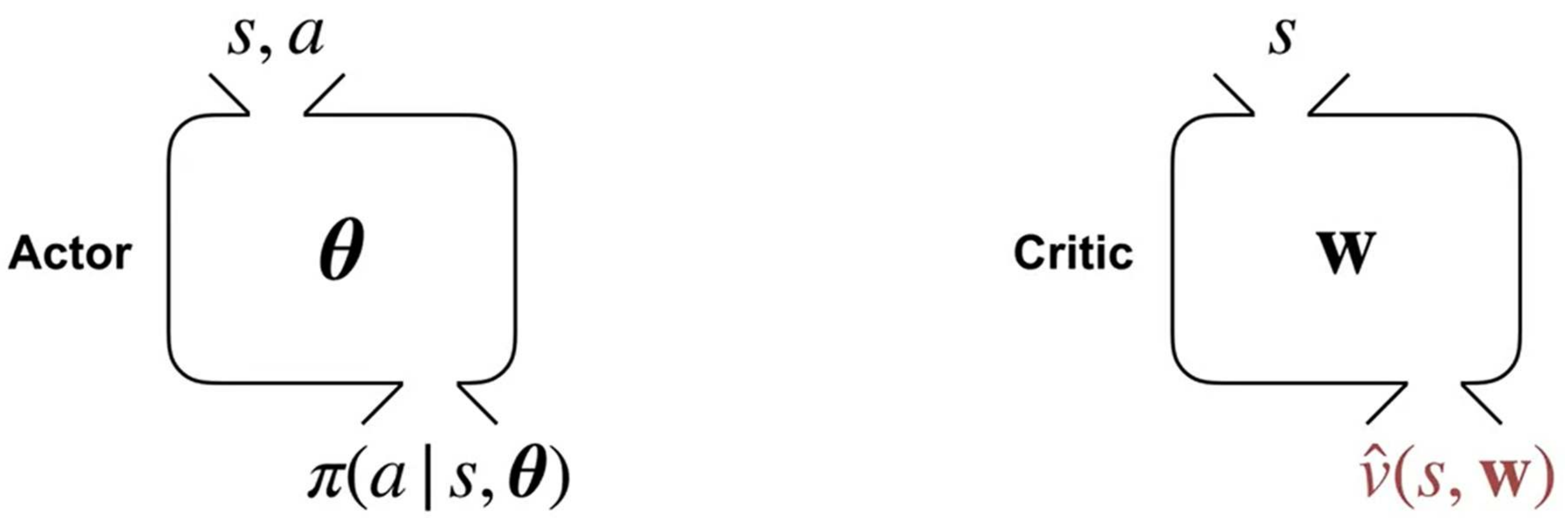
As usual, the parameterize function V hat is learned estimate of the value function.

In this case, V hat is the differential value function.

This is the critic part of the actor-critic algorithm.

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \nabla \ln \pi(A_t \mid S_t, \boldsymbol{\theta}_t)[R_{t+1} - \bar{R} + \hat{v}(S_{t+1}, \mathbf{w})]$$

Average Reward Semi-Gradient TD(0)



The critic provides immediate feedback.
To train the critic, we can use any state value learning algorithm.
We will use the average reward version of semi-gradient TD.
The parameterized policy is the actor.
It uses the policy gradient updates shown here.

Subtracting the Current State's Value Estimate

TD Error δ

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \nabla \ln \pi(A_t \mid S_t, \boldsymbol{\theta}_t)\left[\boxed{R_{t+1} - \bar{R} + \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})}\right]$$

Does not affect the Expected Update

Adding a Baseline

Reduces the update variance

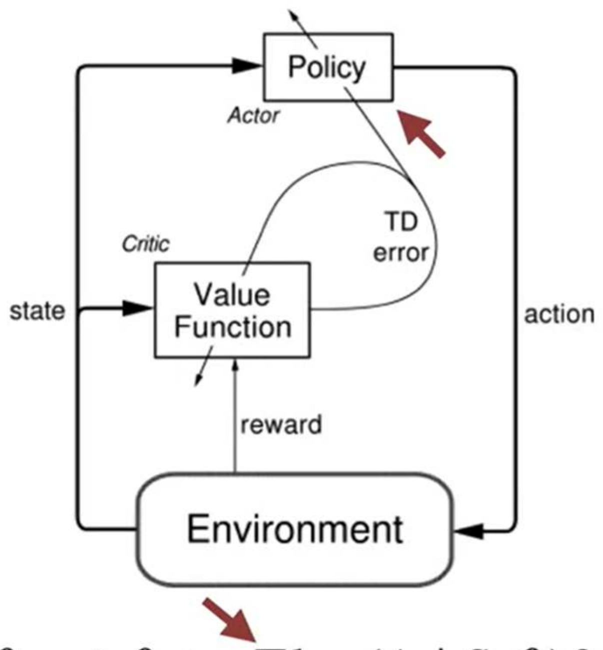
$$\mathbb{E}_\pi\left[\nabla \ln \pi(A_t \mid S_t, \boldsymbol{\theta}_t)[R_{t+1} - \bar{R} + \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})] \mid S_t = s\right]$$

$$= \mathbb{E}_\pi\left[\nabla \ln \pi(A_t \mid S_t, \boldsymbol{\theta}_t)[R_{t+1} - \bar{R} + \hat{v}(S_{t+1}, \mathbf{w})] \mid S_t = s\right]$$

$$-\underbrace{\boxed{\mathbb{E}_\pi\left[\nabla \ln \pi(A_t \mid S_t, \boldsymbol{\theta}_t)\hat{v}(S_t, \mathbf{w}) \mid S_t = s\right]}}_{0}$$

There is one last thing we can do to improve the algorithm.
We can subtract off what is called a baseline.
Instead of using the one-step value estimate alone, we can subtract the value estimate for the state, S_t, to get the update that looks like this.
V hat of S_t is the baseline in this case.
Notice that this expression is equal to the TD error.
The expected value of this update is the same as the previous one.
Subtracting this baseline tends to reduce the variance of the update which results in faster learning.

How the Actor and the Critic Interact



$$\theta_{t+1} \doteq \theta_t + \alpha \nabla \ln \pi(A_t \mid S_t, \theta_t)\delta_t$$

$+$

After we execute an action, we use the TD error to decide how good the action was compared to the average for that state.
If the TD error is positive, then it means the selected action resulted in a higher value than expected.
Taking that action more often should improve our policy.
That is exactly what this update does.
It changes the policy parameters to increase the probability of actions that were better
than expected according to the critic.
Correspondingly, if the critic is disappointed and the TD error is negative, then the probability of the action is decreased.
The actor and the critic learn at the same time, constantly interacting.
The actor is continually changing the policy to exceed the critics expectation, and the critic is constantly updating its value function to evaluate the actor's changing policy.

Actor-Critic (continuing), for estimating $\pi_\theta \approx \pi_*$

```
Input: a differentiable policy parameterization $\pi(a \mid s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s, \mathbf{w})$
Initialize $\bar{R} \in \mathbb{R}$ to $0$
Initialize state-value weights $\mathbf{w} \in \mathbb{R}^d$ and policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g. to $0$)
Algorithm parameters: $\alpha^{\mathbf{w}} > 0, \alpha^{\boldsymbol{\theta}} > 0, \alpha^{\bar{R}} > 0$
Initialize $S \in \mathcal{S}$
Loop forever (for each time step):
    $A \sim \pi(\cdot \mid S, \boldsymbol{\theta})$
    Take action $A$, observe $S'$, $R$
    $\delta \leftarrow R - \bar{R} + \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$
    $\bar{R} \leftarrow \bar{R} + \alpha^{\bar{R}}\delta$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\delta\nabla\hat{v}(S, \mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}}\delta\nabla\ln\pi(A \mid S, \boldsymbol{\theta})$
    $S \leftarrow S'$
```

Actor-Critic with Softmax Policies

Recap: Actor-Critic

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S, \mathbf{w})$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \delta \nabla \ln \pi(A \mid S, \boldsymbol{\theta})$$

Policy Update with a Softmax Policy

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \delta \nabla \ln \pi(A \mid S, \boldsymbol{\theta})$$

$$\pi(a \mid s, \boldsymbol{\theta}) \doteq \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_{b \in \mathcal{A}} e^{h(s,b,\boldsymbol{\theta})}}$$

Features of the Action Preference Function

$$\hat{v}(s, \mathbf{w}) \doteq \mathbf{w}^T \mathbf{x}(s)$$

$$h(s, a, \boldsymbol{\theta}) \doteq \boldsymbol{\theta}^T \mathbf{x}_h(s, a)$$

$$\mathbf{x}_h(s, a) = \begin{bmatrix} x_0(s) \\ x_1(s) \\ x_2(s) \\ x_3(s) \\ x_0(s) \\ x_1(s) \\ x_2(s) \\ x_3(s) \\ x_0(s) \\ x_1(s) \\ x_2(s) \\ x_3(s) \end{bmatrix} \begin{array}{l} \left.\begin{array}{l} \\ \\ \\ \\ \end{array}\right\} a_0 \\ \left.\begin{array}{l} \\ \\ \\ \\ \end{array}\right\} a_1 \\ \left.\begin{array}{l} \\ \\ \\ \\ \end{array}\right\} a_2 \end{array}$$

Actor-Critic with a Softmax Policy

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S, \mathbf{w})$$

$$\nabla \hat{v}(s, \mathbf{w}) = \mathbf{x}(s)$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \delta \nabla \ln \pi(A \mid S, \boldsymbol{\theta})$$

Actor-Critic with a Softmax Policy

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S, \mathbf{w})$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \, \mathbf{x}(S)$$

$$\nabla \hat{v}(s, \mathbf{w}) = \mathbf{x}(s)$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \delta \nabla \ln \pi(A \mid S, \boldsymbol{\theta})$$

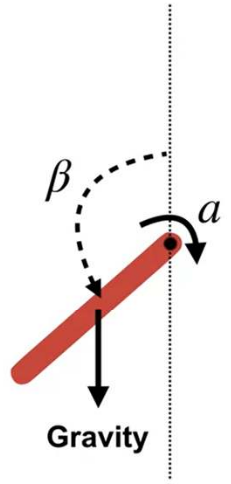
$$\nabla \ln \pi(a \mid s, \boldsymbol{\theta}) = \mathbf{x}_h(s, a) - \sum_b \pi(b \mid s, \boldsymbol{\theta}) \mathbf{x}_h(s, b)$$

Demonstration with Actor-Critic

Pendulum Swing-Up

$$s \doteq \begin{bmatrix} \beta \\ \dot{\beta} \end{bmatrix} \begin{array}{l} \text{Angular Position} \\ \text{Angular Velocity} \end{array}$$

$$a \in \{-1, 0, 1\}$$

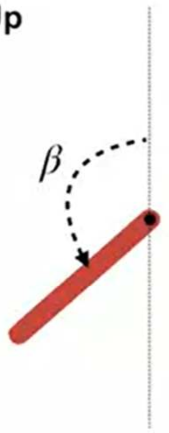


The Reward in Pendulum Swing-Up

$$R \doteq -|\beta|$$

$$-2\pi < \dot{\beta} < 2\pi$$

Continuing Task

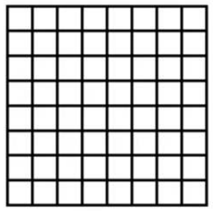


Parameterization and Features

$$\hat{v}(s, \mathbf{w}) \doteq \mathbf{w}^T \mathbf{x}(s) \qquad -\pi < \beta < \pi$$

$$h(s, a, \boldsymbol{\theta}) \doteq \boldsymbol{\theta}^T \mathbf{x}_h(s, a) \qquad -2\pi < \dot{\beta} < 2\pi$$

Tile-Coding Features

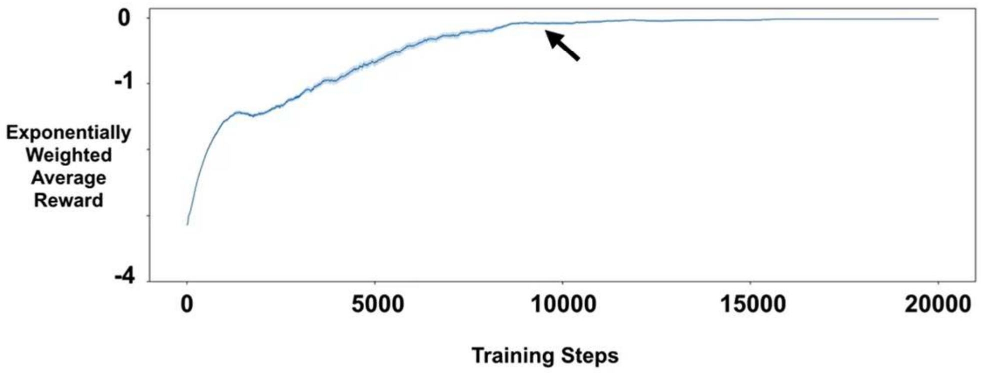


x 32

Step-Sizes

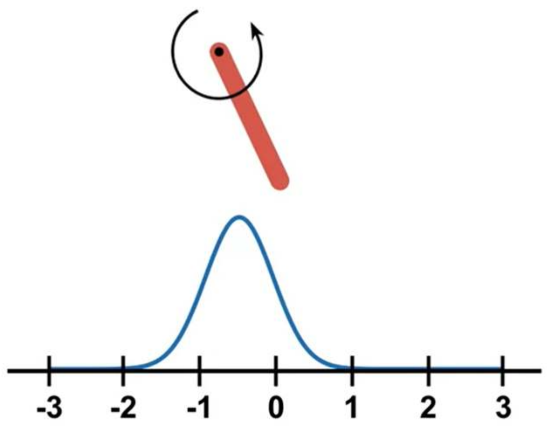
$$\alpha^{\boldsymbol{\theta}} < \alpha^{\mathbf{w}}$$

Results After 100 Runs



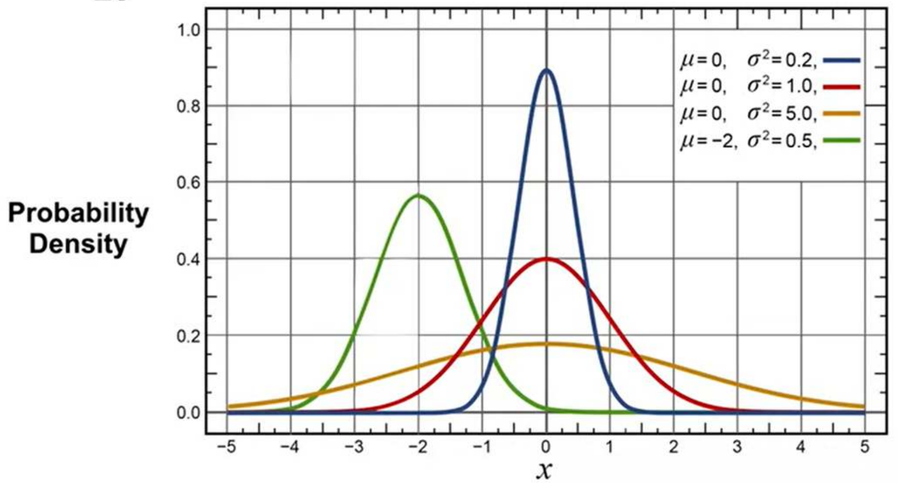
Gaussian Policies for Continuous Actions

Continuous Action Space



Gaussian Distribution

$$p(x) \doteq \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$



Gaussian Policy

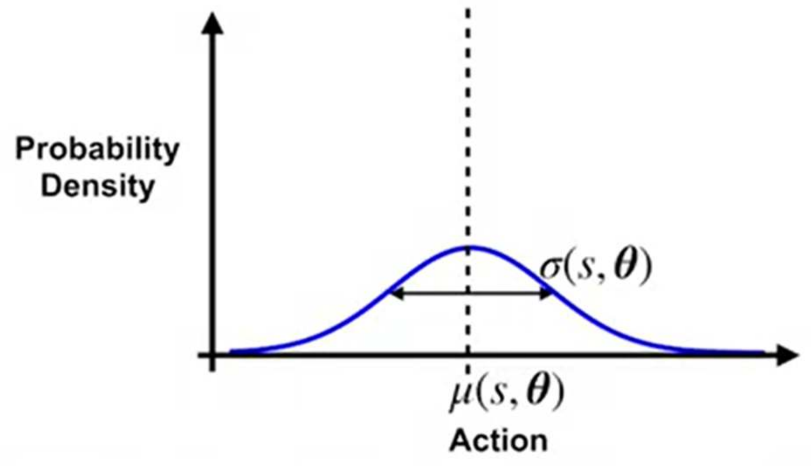
$$\pi(a \mid s, \boldsymbol{\theta}) \doteq \frac{1}{\sigma(s, \boldsymbol{\theta})\sqrt{2\pi}} \exp\left(-\frac{(a - \mu(s, \boldsymbol{\theta}))^2}{2\sigma(s, \boldsymbol{\theta})^2}\right)$$

$$\mu(s, \boldsymbol{\theta}) \doteq \boldsymbol{\theta}_\mu^T \mathbf{x}(s)$$

$$\boldsymbol{\theta} \doteq \begin{bmatrix} \boldsymbol{\theta}_\mu \\ \boldsymbol{\theta}_\sigma \end{bmatrix}$$

$$\sigma(s, \boldsymbol{\theta}) \doteq \exp\left(\boldsymbol{\theta}_\sigma^T \mathbf{x}(s)\right)$$

Sampling Actions From the Gaussian Policy



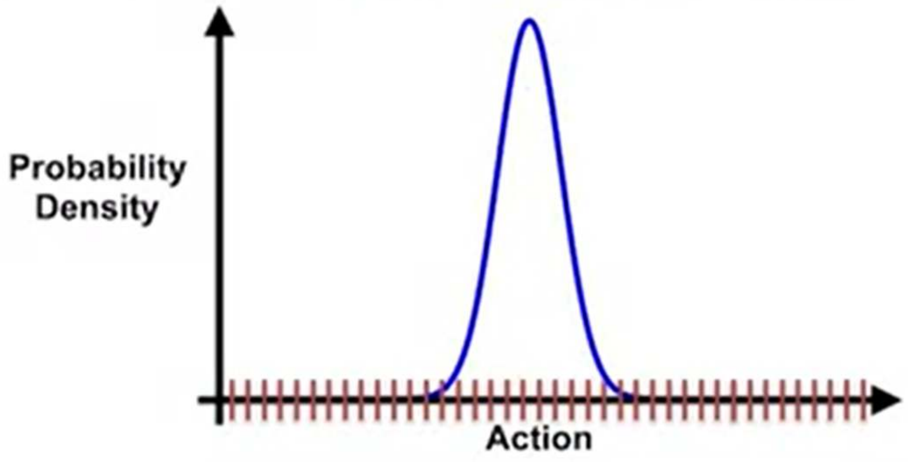
Gradient of the Log of the Gaussian Policy

$$\nabla \ln \pi(a \mid s, \boldsymbol{\theta}_\mu) = \frac{1}{\sigma(s, \boldsymbol{\theta})^2}(a - \mu(s, \boldsymbol{\theta}))\mathbf{x}(s)$$

$$\nabla \ln \pi(a \mid s, \boldsymbol{\theta}_\sigma) = \left(\frac{(a - \mu(s, \boldsymbol{\theta}))^2}{\sigma(s, \boldsymbol{\theta})^2} - 1\right)\mathbf{x}(s)$$

Advantages of Continuous Actions

- It might not be straightforward to choose a proper discrete set of actions
- Continuous actions allow us to generalize over actions



- We can apply actor-critic to problems with continuous actions