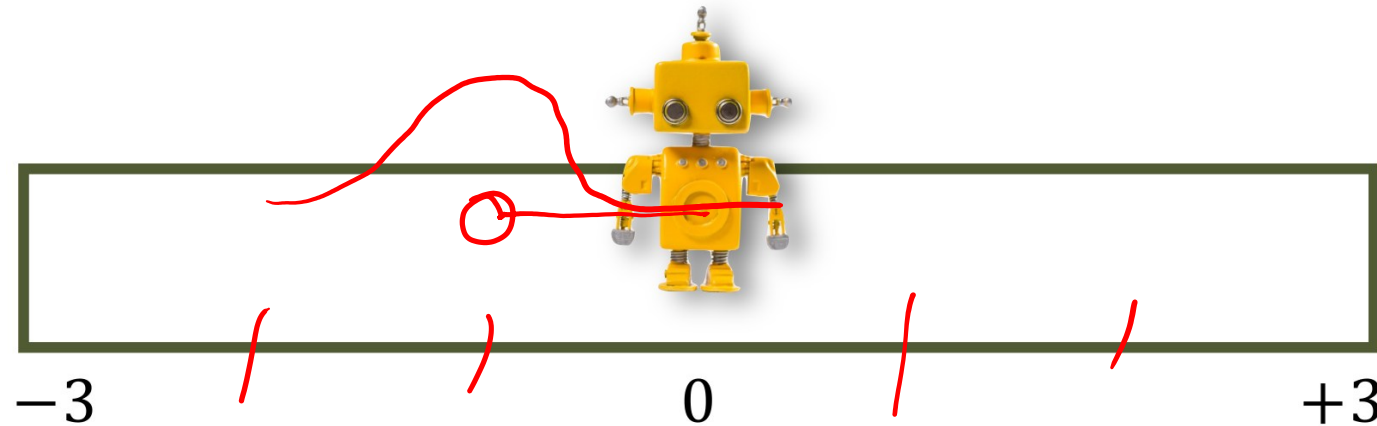


07-380 AI & ML II

Policy Gradient Methods for Deep Reinforcement Learning

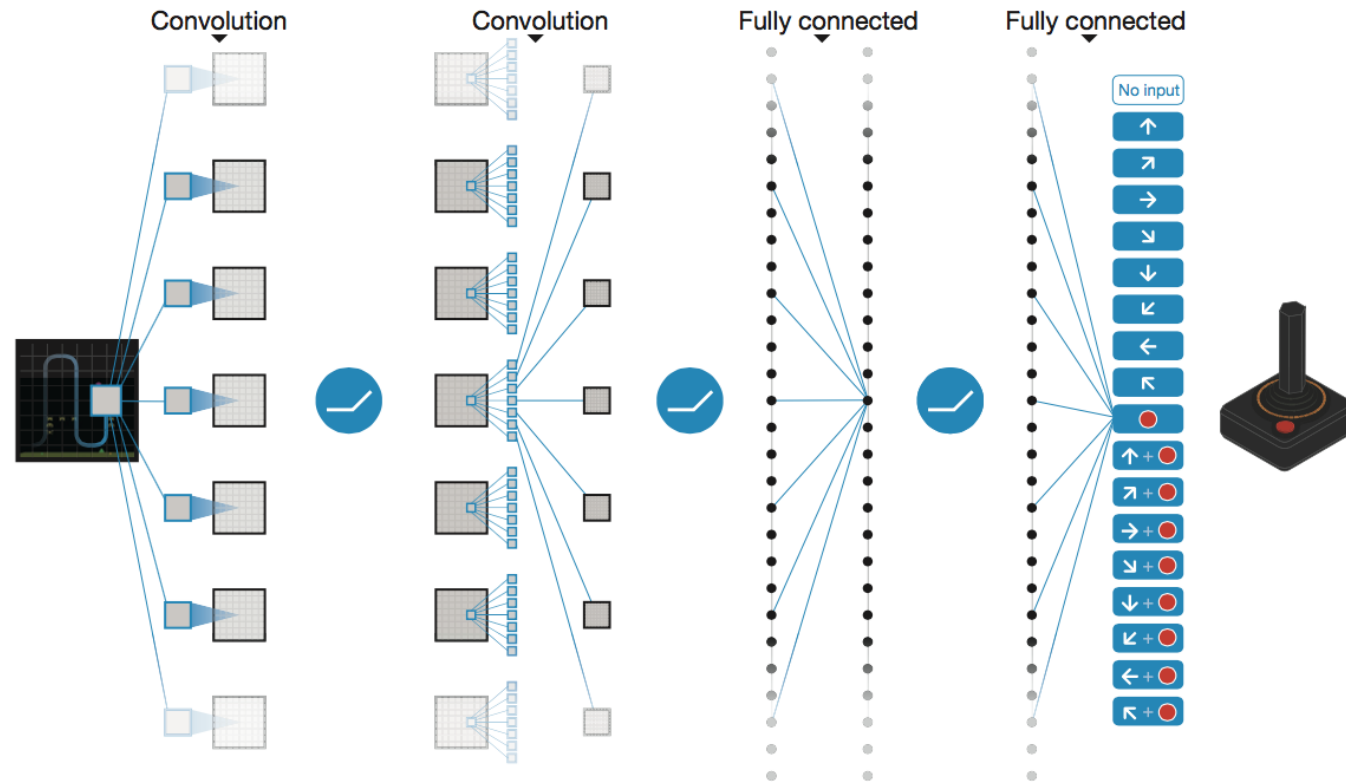
$\pi(s)$
 $\pi(s, a)$
 $L \quad 0.3$
 $R \quad 0.7$



Instructors: Pat Virtue & Mohammad Salameh

Slide credits: Matt Gormley (10-301/10-601) and Geoff Gordon

Deep Reinforcement Learning



Deep Q-Networks (DQN)

Approximate Q-learning: Linear → Network

Deep Q-Networks

Deep Mind, 2015

Used a deep learning network to represent Q:

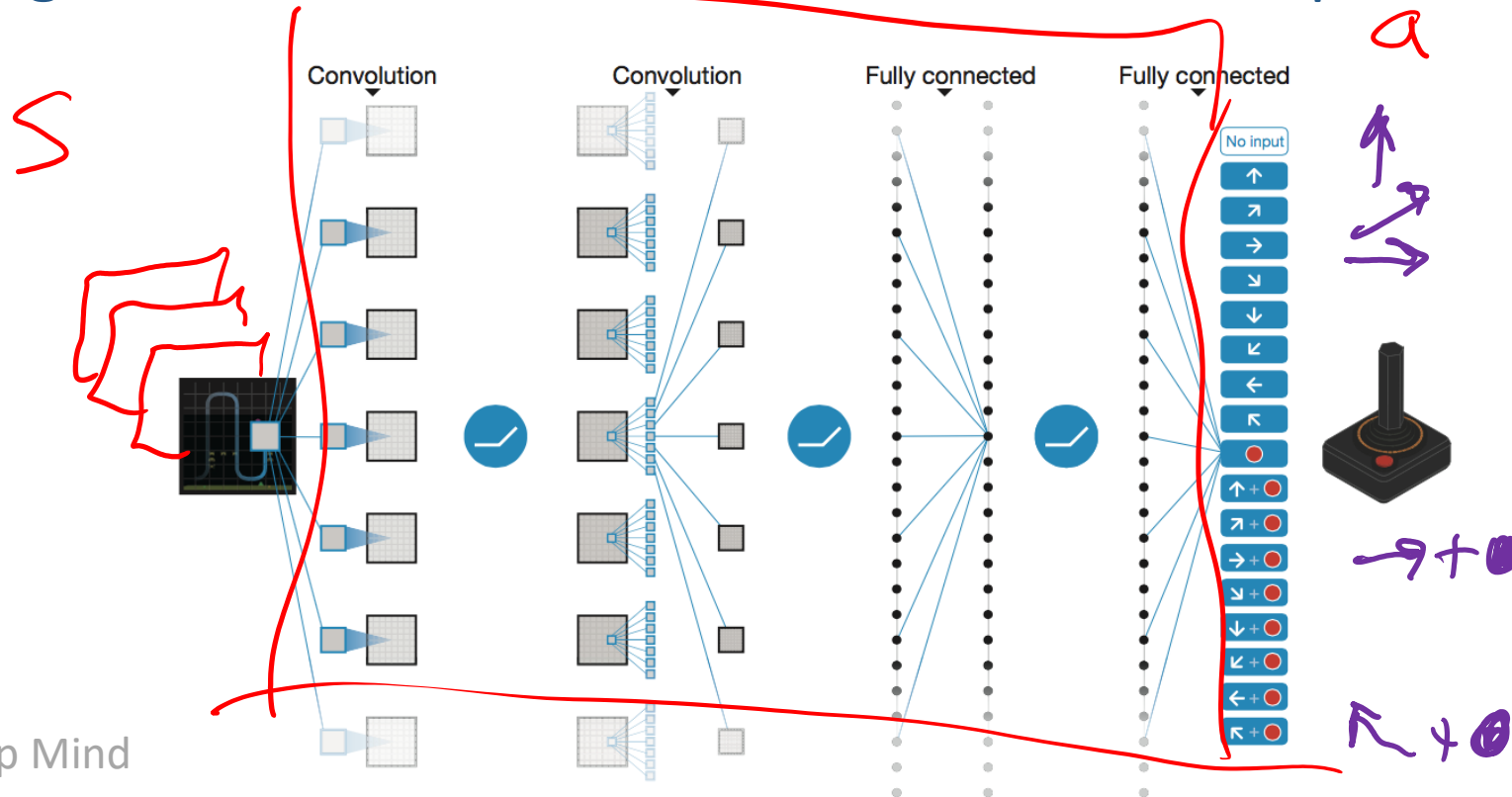
- Input is last 4 images (84x84 pixel values) plus score

49 Atari games, incl. Breakout, Space Invaders, Seaquest, Enduro

$$sample = r + \gamma \max_a Q_{\theta}(s', a')$$

$Q_{\theta}(s, a)$: Neural network

$V_{\theta}(s)$: net
 $\pi_{\theta}(s, a)$: net



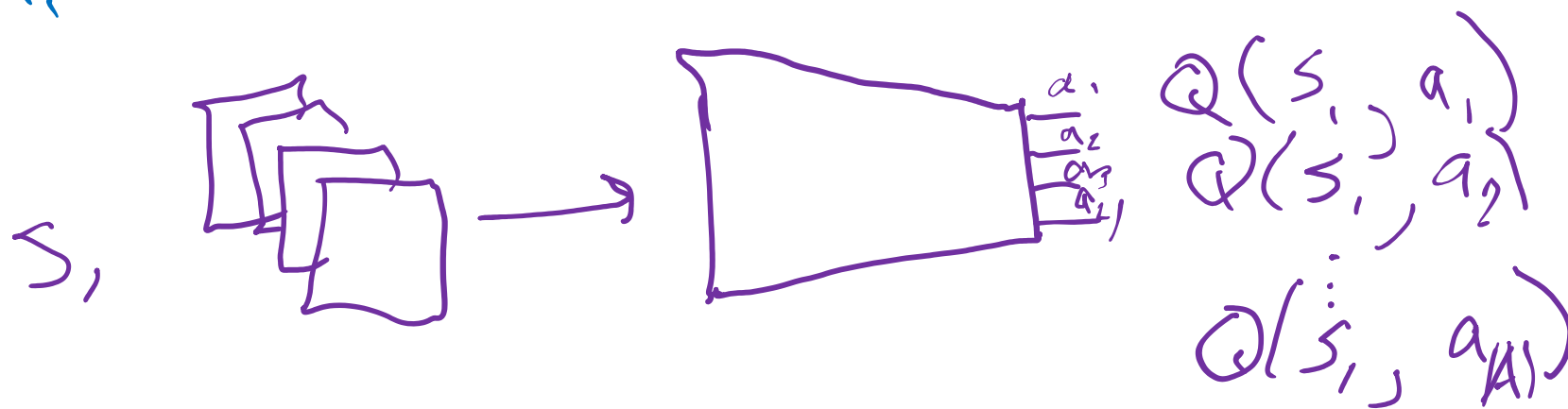
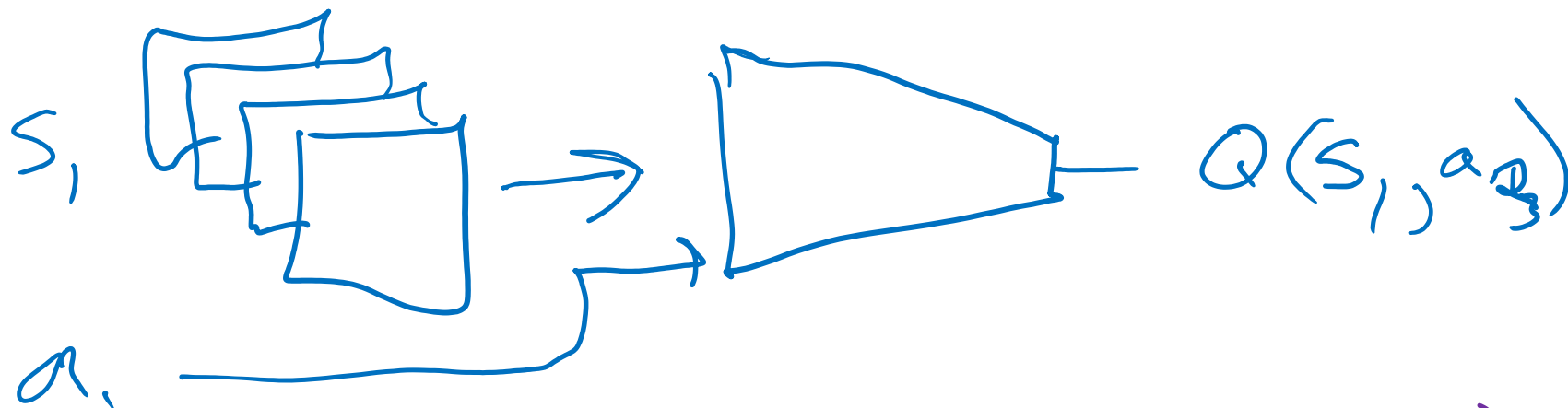
Deep Q-Networks

$$sample = r + \gamma \max_{a'} Q_{\theta}(s', a')$$

Before we approximated Q values with a linear combination of features:

$$Q_w(s, a) = \sum_i w_i f_i(s, a)$$

Now, we'll use a neural net instead. Two options for network input/output:

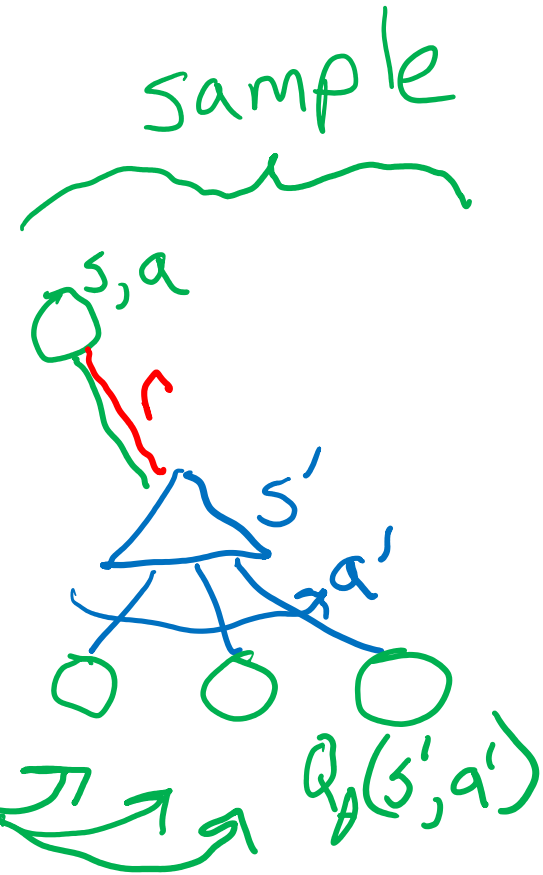
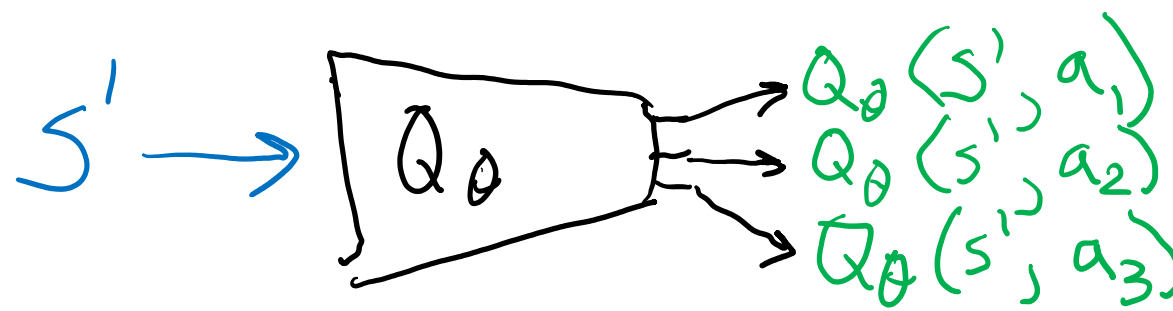
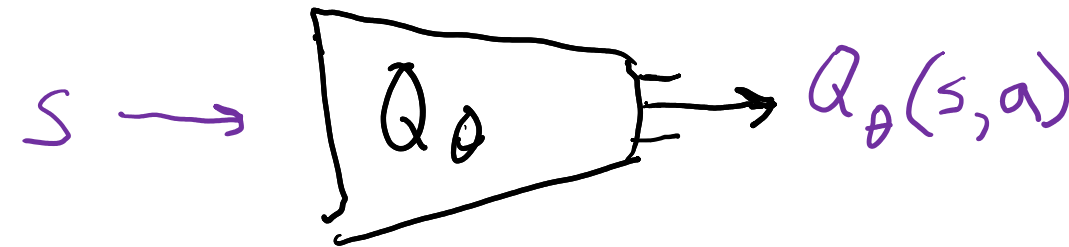


Deep Q-Networks

$$\gamma = \text{sample} = r + \gamma \max_{a'} Q_{\theta}(s', a')$$
$$\hat{\gamma} = Q_{\theta}(s, a)$$

Given state s :

- 1) Choose (any) action a
- 2) Environment gives you reward r and sends you to state s'
- 3) Use network to compute $Q_{\theta}(s, a)$ and **sample**



Deep Q-Networks

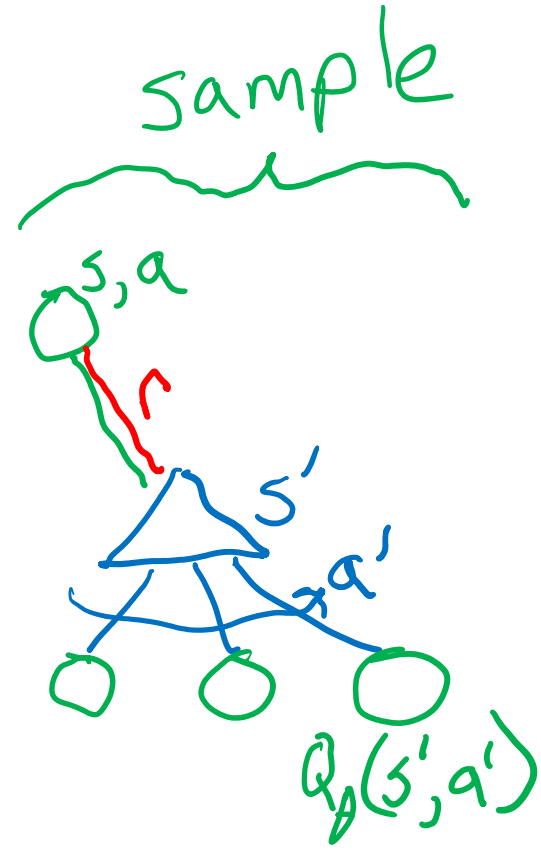
$$\gamma = \text{sample} = r + \gamma \max_{a'} Q_{\theta}(s', a')$$
$$\hat{\gamma} = Q_{\theta}(s, a)$$

Given state s :

- 1) Choose (any) action a
- 2) Environment gives you reward r and sends you to state s'
- 3) Use network to compute $Q_{\theta}(s, a)$ and **sample**
- 4) Update network parameters, θ , with an SGD step

$$\theta \leftarrow \theta + \alpha (\text{sample} - Q_{\theta}(s, a)) \nabla_{\theta} Q(s, a)$$

$$(\gamma - \hat{\gamma})^2$$

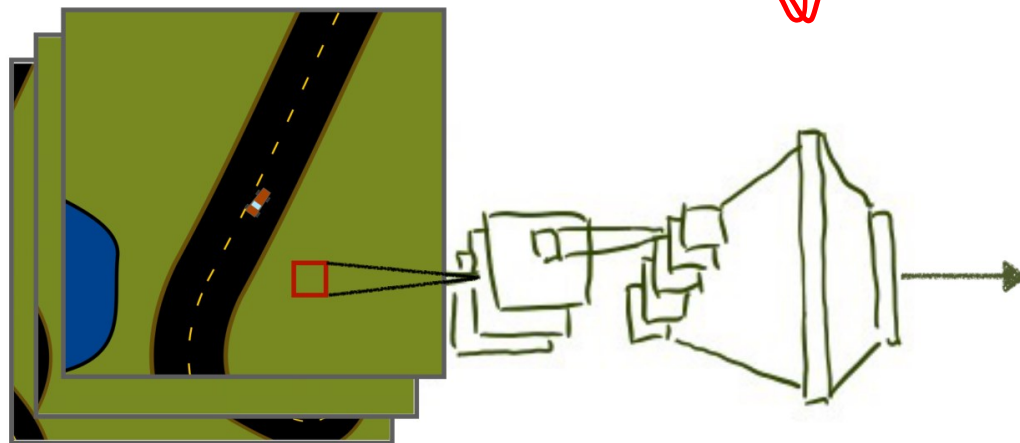


$$V_{\phi}(s)$$

Learning a Value Function with Function Approximation

net

Learning V^π

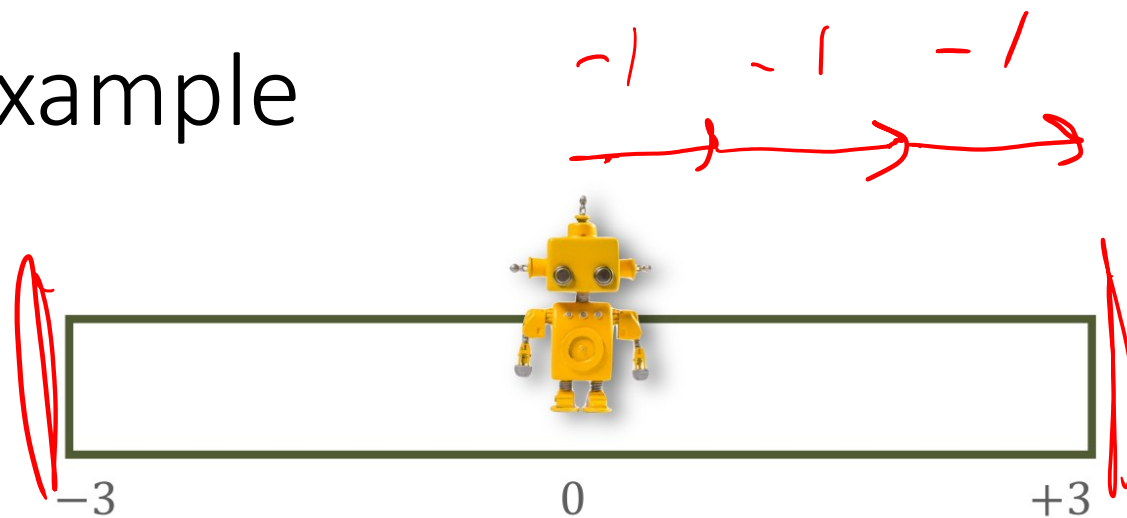


$$V_\phi(s)$$

$$\begin{aligned} \pi(s, a) \\ \hat{a} \sim p(a|s) \end{aligned}$$

- Given a fixed policy π
- Want to train a network $V_\phi^\pi(s)$ w/ parameters ϕ
 - inputs: state info, e.g., stack of images
 - output: value estimate
- Data: follow π , observe one or more trajectories $s_1, a_1, r_1, s_2, a_2, r_2, \dots$
- Each trajectory yields several training examples

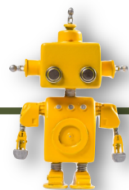
Learning V^π : example



- Environment:
 - state $x \in [-3, 3]$, start at $x = 0$
 - actions L: $x = x - N(1, \sigma^2)$ and R: $x = x + N(1, \sigma^2)$
 - rewards: -1 per action, terminate when $x \notin [-3, 3]$
 - $\gamma = 1, \sigma = \frac{1}{4}$

Some sample trajectories

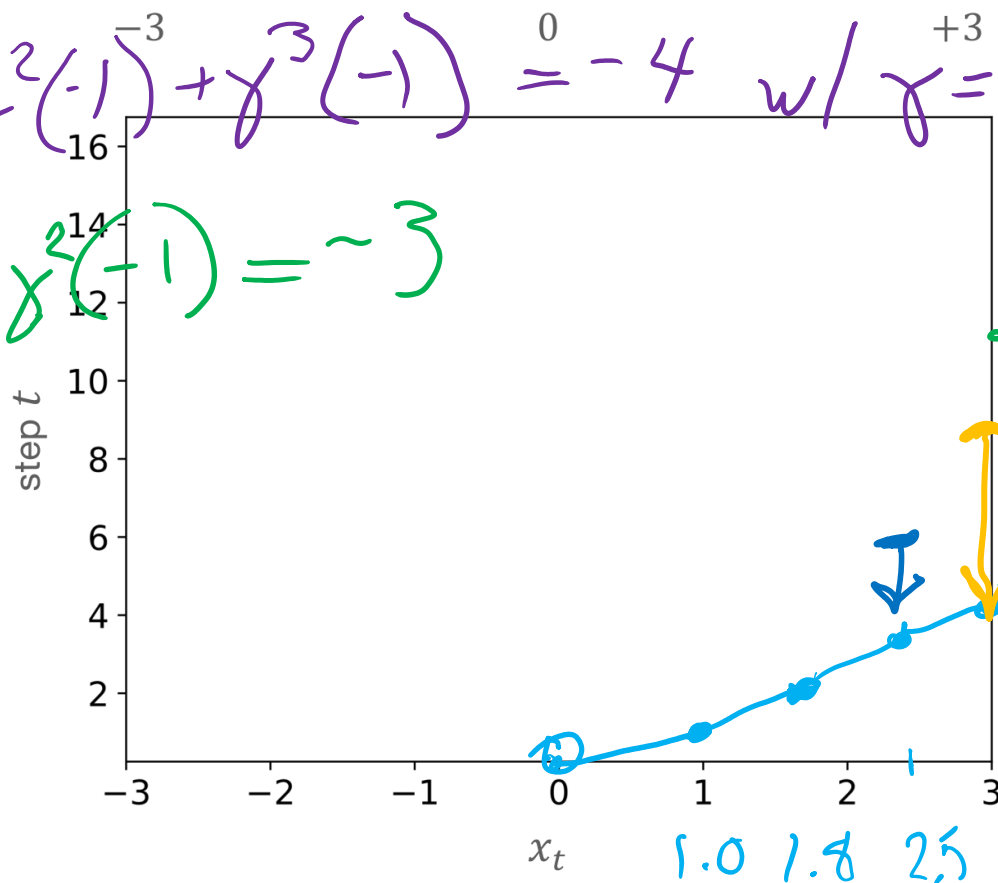
$$R(\pi) = \sum_{t=0}^{T-1} \bar{R}(t)$$



$$\bar{R}(t=0) = -1 + \gamma(-1) + \gamma^2(-1) + \gamma^3(-1) = -4 \text{ w/ } \gamma=1$$

$$\bar{R}(t=1) = -1 + \gamma(-1) + \gamma^2(-1) = -3$$

Each trajectory
yields several
training examples

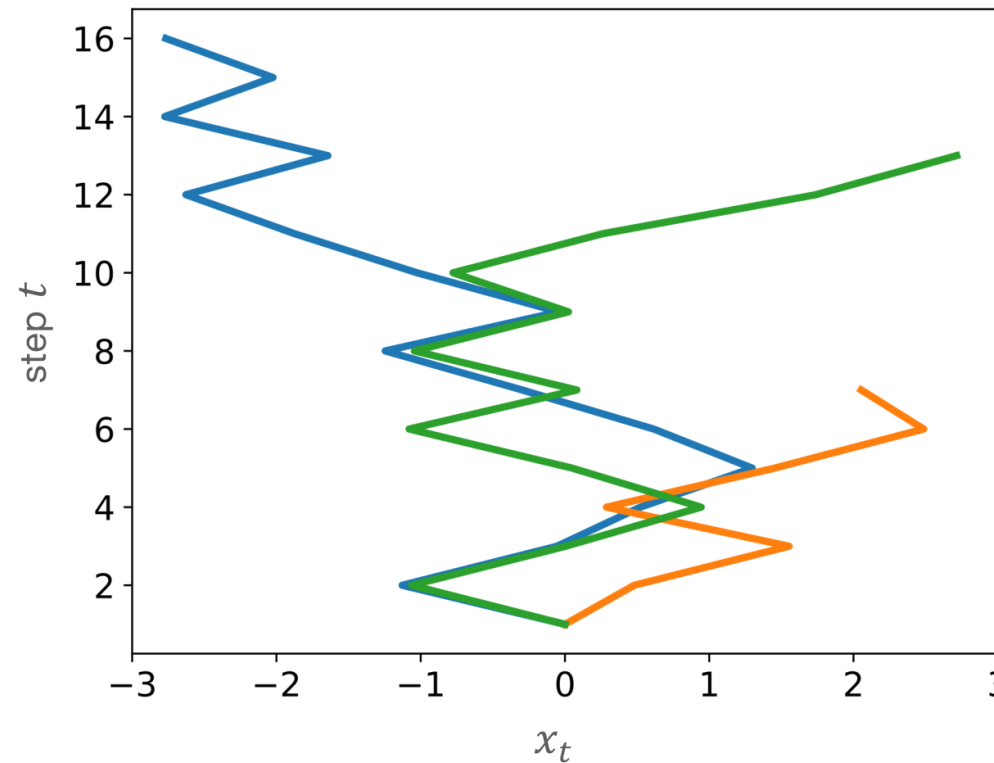
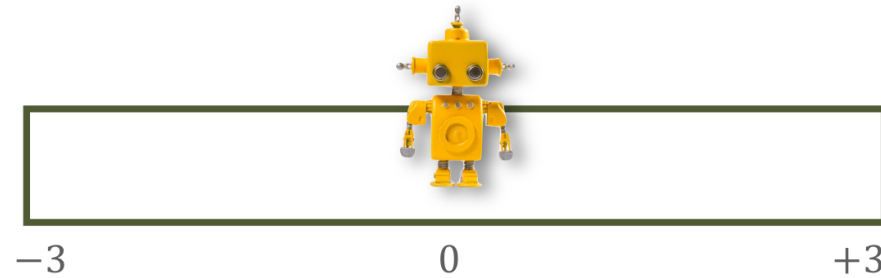


↑ s a r

t=0	0	R	-1
t=1	1.0	R	-1
t=2	1.8	R	-1
t=3	2.5	R	-1

3.0

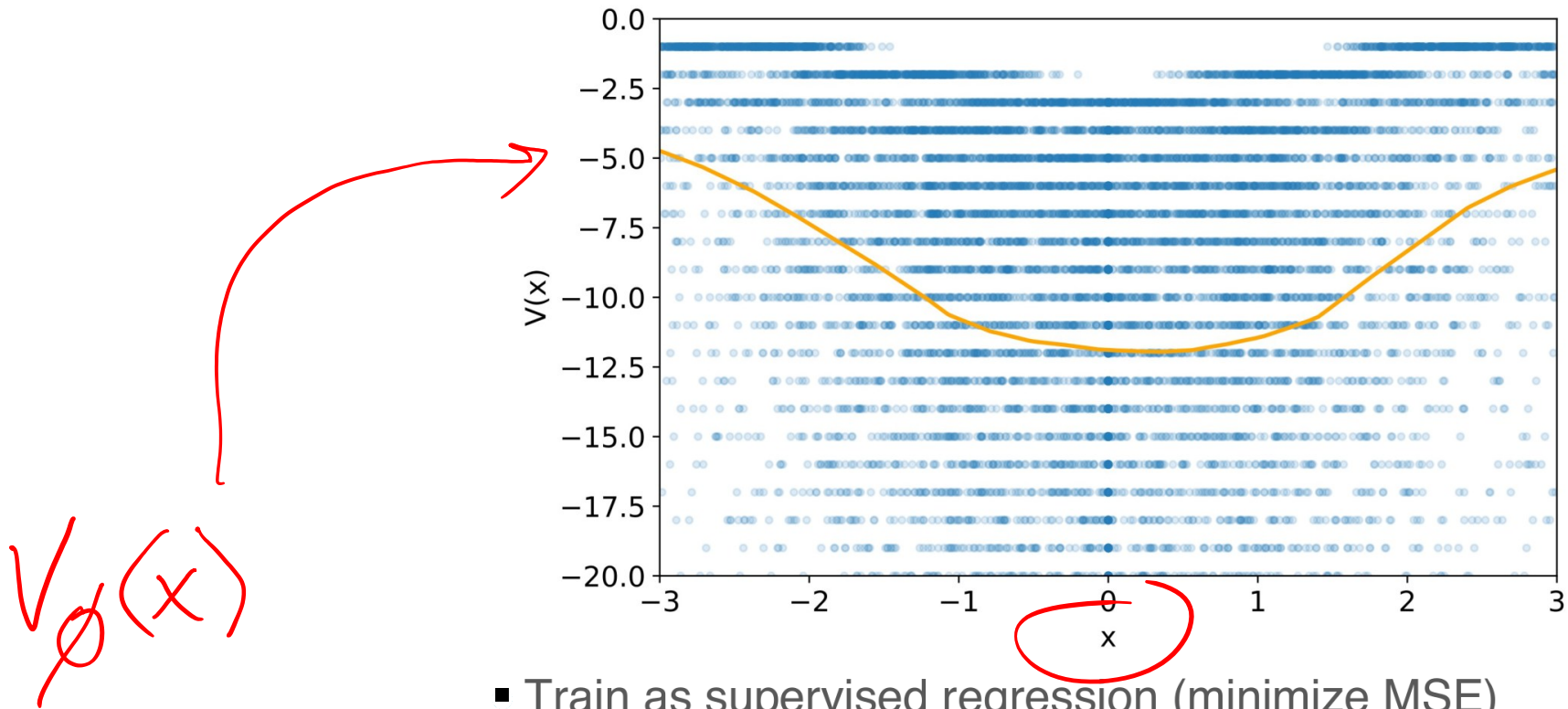
Some sample trajectories



$(x, V(x))$
(0, -13)
(-1, -15)
(+2, -2)
(-2, -1)
...
(-3, -20)

Each trajectory
yields several
training examples

Learned V^π



- Train as supervised regression (minimize MSE)
- Blue dots: training points (1k trajectories, ~12k samples)
- Orange line: fitted V^π (2-layer ReLU net, width 64)
- Note: extremely noisy!

How to Learn V^π given π ?

- **Problem:** Large state space (e.g., high-resolution images): $V(s)$ cannot be represented as a table.
- **Solution:**
 - Use a function approximator (e.g., a neural network) to represent $V(s)$ as $V_\phi(s)$, where ϕ are the parameters.
 - Estimate $V^\pi(s)$ via Monte Carlo: sample trajectories from π and average the returns.
 - Update ϕ via gradient descent to minimize MSE between $V_\phi(s)$ and the Monte Carlo estimates of $V^\pi(s)$.

Example function approximators:

- **Linear function approximation:**
 $V_\phi(s) = \phi^T \mathbf{x}(s)$, where $\mathbf{x}(s)$ is a feature vector representation of state s .
- **One-hidden layer neural network:**
 $V_\phi(s) = \mathbf{w}^T \sigma(\mathbf{W}\mathbf{x}(s) + \mathbf{b})$, where $\mathbf{W}$, $\mathbf{b}$, $\mathbf{w}$ are the parameters of the neural network, $\mathbf{x}(s)$ is a feature vector representation of state s , and σ is a non-linear activation function (e.g., ReLU).

How to Learn V^π given π ?

$$\min J(\phi) = \min \frac{1}{N} \sum (\hat{y}^{(\cdot)} - y)^2$$

Algorithm 1 On-Policy Prediction with Function Approximation

```
1: procedure MONTECARLOPREDICTIONFA( $\pi, \eta, \gamma$ )
2:   Initialize value parameters  $\phi$  randomly
3:   while not converged do
4:     Sample a trajectory  $\tau = [(s_0, a_0, r_0), \dots, (s_T, a_T, r_T)]$  using  $\pi$ 
5:      $\mathbf{g} \leftarrow \sum_{t=0}^T \nabla_{\phi} (V_{\phi}(s_t) - \bar{R}_t)^2$ 
6:     where  $\bar{R}_t = \sum_{t'=t}^T \gamma^{t'-t} r_{t'}$ 
7:      $\phi \leftarrow \phi - \eta \mathbf{g}$ 
8:   return  $V_{\phi}$ 
```

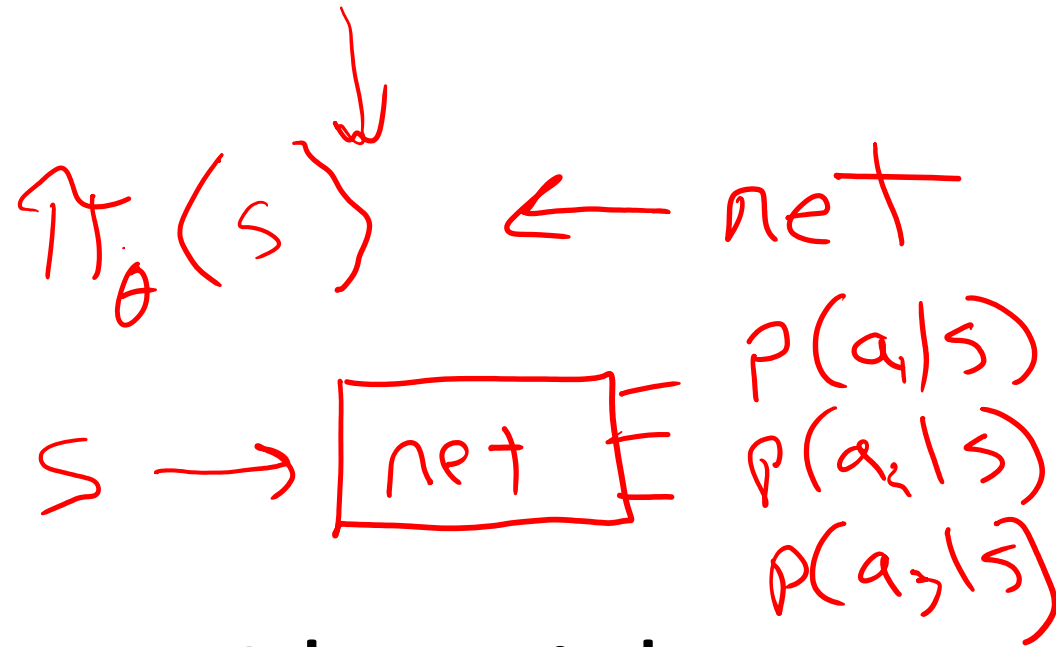
$V_{\phi}(s) - \bar{R}$
↑ ↑
predict sample

$$\leftarrow \sim \nabla J(\phi)$$

$$\nabla_{\phi} (V_{\phi}(s_t) - \bar{R}_t)^2$$

$$2(V_{\phi}(s_t) - \bar{R}_t) \nabla V_{\phi}(s_t)$$

Okay, this idea of using a neural net as a value function approximator is great, but how do we learn a policy now?



Policy Gradient Algorithms

Comparison of Policy Gradient Methods

1) All of the algorithms compute a gradient of the form:

$$\mathbf{g}_\theta = \sum_{t=1}^T \tilde{G}_t \nabla_\theta \log \pi_\theta(a_t | s_t)$$

$$\nabla J(\theta)$$
$$\nabla \pi_\theta(s)$$

where $\tilde{G}_t$ is an estimate of the expected discounted future reward from time step t

2) Some use a V_ϕ , and some do not

Algorithm	$\tilde{G}_t$	Estimate V_ϕ ?
REINFORCE	$\bar{R}_t$	x
REINFORCE with Baseline	$(\bar{R}_t - V_\phi(s_t))$	✓
Actor-Critic	$\hat{Q}_t = r_t + \gamma V_\phi(s_{t+1})$	✓
Advantage Actor-Critic	$\hat{A}_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$	✓

Policy Gradient Methods (generally)

- Assume a stochastic policy $\pi_{\theta}(a \mid s)$ parameterized by θ , that returns the probability of taking action a in state s .
- Assume we have an objective function $J(\theta)$ that we want to maximize (usually maximizing reward under the policy π_{θ}).

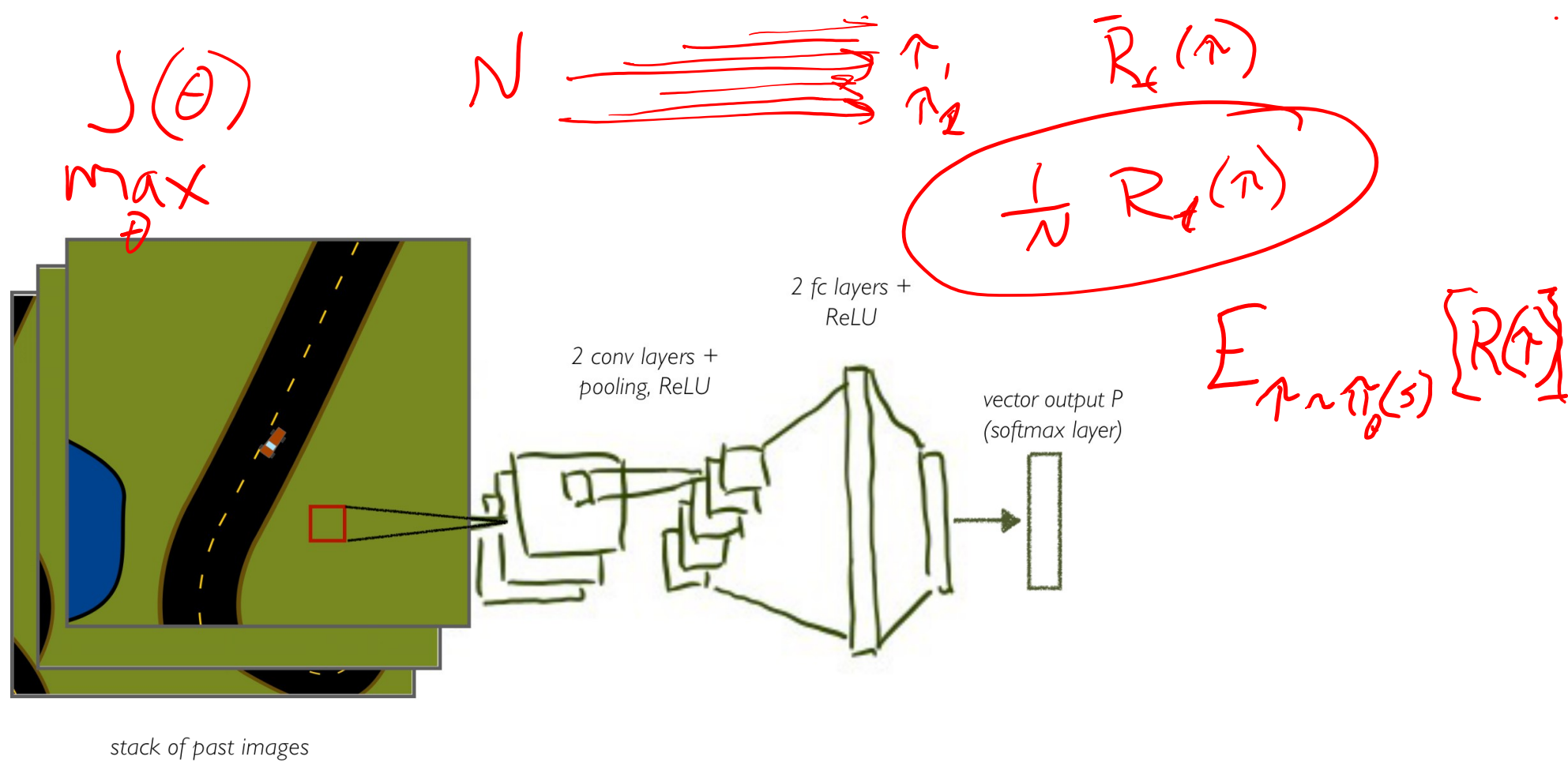
- **Definition:** A **policy gradient method** learns the parameters θ of the policy π_{θ} by performing gradient ascent:

$$\theta \leftarrow \theta + \eta \nabla_{\theta} J(\theta)$$

where η is the learning rate.

- **Definition:** $\nabla_{\theta} J(\theta)$ is called the **policy gradient**.
- (We might also use/learn a value function $V_{\phi}(s)$ parameterized by ϕ , that approximates the expected discounted future reward of being in state s .)

Policy



- Policy is a model too: represents $P(a \mid s, \pi)$
 - note: stochastic! (lets an optimizer make small changes)
- Several common ways to set up:
 - $s \mapsto [P(a_1 \mid s, \pi), P(a_2 \mid s, \pi), \dots, P(a_k \mid s, \pi)]^T$

Policy Gradient Theorem

Standard Objective The goal in RL is to find a policy $\pi_{\theta}(a|s)$ that maximizes the expected cumulative discounted reward:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^T \gamma^t r_t \right] = \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau)]$$

where $\tau = [(s_0, a_0, r_0), \dots, (s_T, a_T, r_T)]$ is a trajectory or episode, $R(\tau)$ is the total reward of the trajectory, $\gamma \in [0, 1]$ is the discount factor, and the expectation is taken over trajectories or episodes obtained by deploying the policy π_{θ} on the environment.

$$\theta \leftarrow \theta + \eta \nabla_{\theta} J(\theta)$$

Theorem 1

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

Theorem 2

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^T \bar{R}_t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

where $\bar{R}_t = \sum_{t'=t}^T \gamma^{t'-t} r_{t'}$ is the discounted return from time step t .

Policy Gradient Theorem

Theorem 1 (Proof) Let $p_{\theta}(\tau) = \prod_{t=0}^T \pi_{\theta}(a_t|s_t)p(s_{t+1}|s_t, a_t)$ be the probability of a trajectory τ under the policy π_{θ} and s_0 is the initial state. Then we can derive the policy gradient as follows:

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau)] \\&= \nabla_{\theta} \sum_{\tau} [R(\tau) p_{\theta}(\tau)] && \text{expand the expectation} \\&= \sum_{\tau} [R(\tau) \nabla_{\theta} p_{\theta}(\tau)] && \text{push in the gradient} \\&= \sum_{\tau} \left[R(\tau) p_{\theta}(\tau) \frac{\nabla_{\theta} p_{\theta}(\tau)}{p_{\theta}(\tau)} \right] && \text{multiply by 1.0} \\&= \sum_{\tau} [R(\tau) p_{\theta}(\tau) \nabla_{\theta} \log p_{\theta}(\tau)] && \nabla \log(f(x)) = (\nabla f(x))/f(x) \\&= \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau) \nabla_{\theta} \log p_{\theta}(\tau)] && \text{expectation of gradient}\end{aligned}$$

$$\begin{aligned}\nabla p &= \frac{p}{p} \nabla p \\&= p \frac{\nabla p}{p} \downarrow \uparrow \\&= p \nabla \log p\end{aligned}$$

$$\begin{aligned}\nabla \log p &= \frac{1}{p} \nabla p\end{aligned}$$

Policy Gradient Theorem



Theorem 2 (Proof) Let $p_{\theta}(\tau) = \prod_{t=0}^T \pi_{\theta}(a_t|s_t)p(s_{t+1}|s_t, a_t)$

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau) \nabla_{\theta} \log p_{\theta}(\tau)] \quad \text{from Theorem 1}$$

$$= \mathbb{E}_{\tau \sim \pi_{\theta}} \left[R(\tau) \nabla_{\theta} \sum_{t=0}^T (\log \pi_{\theta}(a_t|s_t) + \log p(s_{t+1}|s_t, a_t)) \right] \quad \text{expand } p_{\theta}(\tau)$$

$$= \mathbb{E}_{\tau \sim \pi_{\theta}} \left[R(\tau) \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) \right] \quad \text{transition probs. do not depend on } \theta$$

$$\rightarrow = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^T \bar{R}_t \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) \right] \rightarrow \text{Ther 3}$$

where $\bar{R}_t = \sum_{t'=t}^T \gamma^{t'-t} r_{t'}$ is the discounted return from time step t .

(Last line: because the past reward part has expectation zero when multiplied by the gradient of the current action.)

Policy Gradient Theorem

Theorem 3:

We can obtain an unbiased estimate of the policy gradient by sampling a trajectory τ from the policy π_θ and computing:

one MC sample

$$\nabla J(\theta) \approx \mathbf{g} = \sum_{t=0}^T \bar{R}_t \nabla_\theta \log \pi_\theta(a_t | s_t)$$

(This is a **Monte Carlo Estimate** of the gradient!)

where $\bar{R}_t = \sum_{t'=t}^T \gamma^{t'-t} r_{t'}$ is the discounted return from time step t .

That is, the expectation of $\mathbf{g}$ is equal to the policy gradient:

$$\mathbb{E}_{\tau \sim \pi_\theta}[\mathbf{g}] = \nabla J(\theta)$$

$$\theta \leftarrow \theta + \eta \tilde{\mathbf{g}}$$

Policy gradient intuition

- Policy gradient: $\mathbf{g} = \sum_{t=0}^T \bar{R}_t \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t | s_t)$
- Score vectors $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t | s_t)$: parameter direction that would increase (log-)probability of taking action a_t^m in state s_t^m
- Scale by $\bar{R}_t$: upweight score vector when reward is large, flip score vector if reward is negative
- On average: a direction that changes policy by taking actions more often if they were associated with high (total future) rewards
 - step along this direction: change policy *multiplicatively* in favor of high-reward actions
- To minimize costs, step along *negative* gradient: take actions *less* often if associated with high costs

Algorithm 1: REINFORCE

If we plug the estimate from the policy gradient theorem into the policy gradient method, we get one of the oldest RL algorithms: REINFORCE [Williams, 1992]

Repeat:

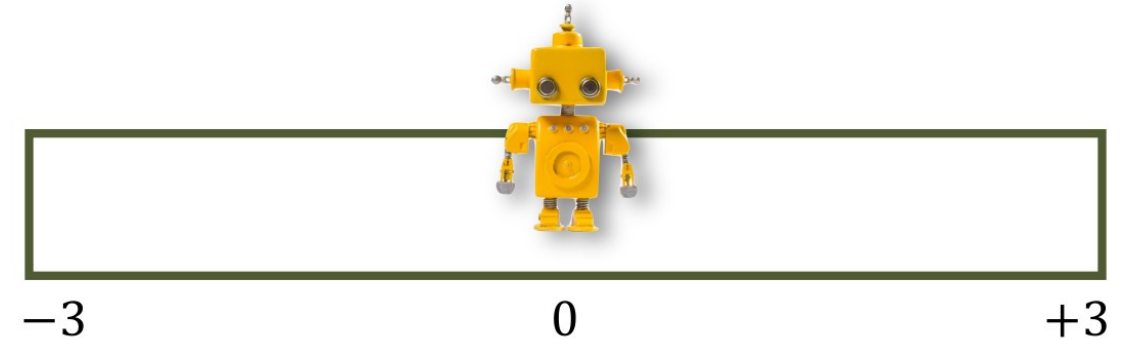
- gather some trajectories under current policy π_θ
- compute gradient estimate g by policy gradient theorem
- update θ by SGD

```
1: procedure REINFORCE( $\eta, \gamma$ )
2:   Initialize policy parameters  $\theta$  randomly
3:   while not converged do
4:     Sample a trajectory  $\tau = [(s_0, a_0, r_0), \dots, (s_T, a_T, r_T)]$  from  $\pi_\theta$ 
5:      $g \leftarrow \sum_{t=1}^T \bar{R}_t \nabla_\theta \log \pi_\theta(a_t | s_t)$ 
6:     where  $\bar{R}_t = \sum_{t'=t}^T \gamma^{t'-t} r_{t'}$ 
7:      $\theta \leftarrow \theta + \eta g$ 
8:   return  $\pi_\theta$ 
```

REINFORCE Example

Environment:

- state $x \in [-3, 3]$, start at $x = 0$
- actions
 - $L : x = x - \mathcal{N}(1, \sigma^2)$
 - $R : x = x + \mathcal{N}(1, \sigma^2)$
- rewards: -1 per action, terminate when $x \notin [-3, 3]$
- $\gamma = 1, \sigma = \frac{1}{4}$



Policy:

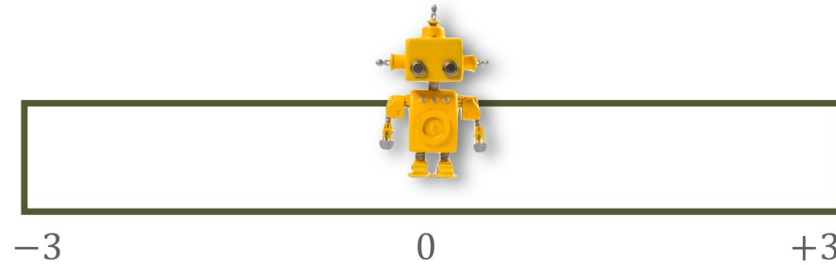
"network"
log regression

$$\pi_{w,b}(R \mid x) = \frac{1}{1 + e^{-(wx+b)}}$$

$$\pi_{w,b}(L \mid x) = \frac{1}{1 + e^{wx+b}}$$

$$\theta = (w, b)^\top$$

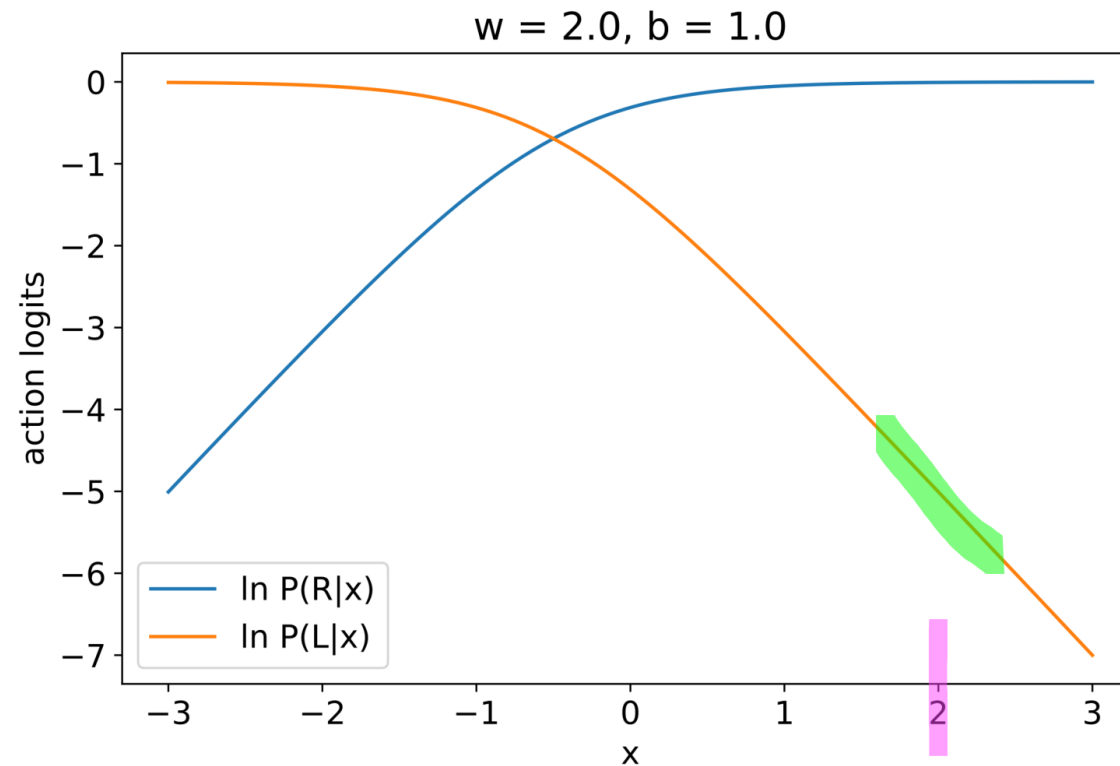
REINFORCE Example



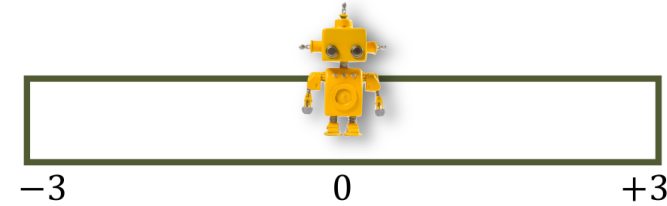
Action logits

$$\nabla_{\theta} \ln P(L | x = 2) = \begin{bmatrix} -1 \\ -2 \end{bmatrix}$$

$$\theta = \begin{bmatrix} w \\ b \end{bmatrix}$$



REINFORCE Example



Policy Gradient:

$$\begin{aligned}\nabla_{\theta} \ln \pi_{w,b}(R \mid x) &= -\nabla_{\theta} \ln(1 + e^{-(wx+b)}) \\ &= -\frac{1}{1 + e^{-(wx+b)}} e^{-(wx+b)} (-\nabla_{\theta}(-wx - b)) \\ &= \frac{1}{1 + e^{wx+b}} \begin{pmatrix} x \\ 1 \end{pmatrix} \\ &= \pi_{w,b}(L \mid x) \begin{pmatrix} x \\ 1 \end{pmatrix}\end{aligned}$$

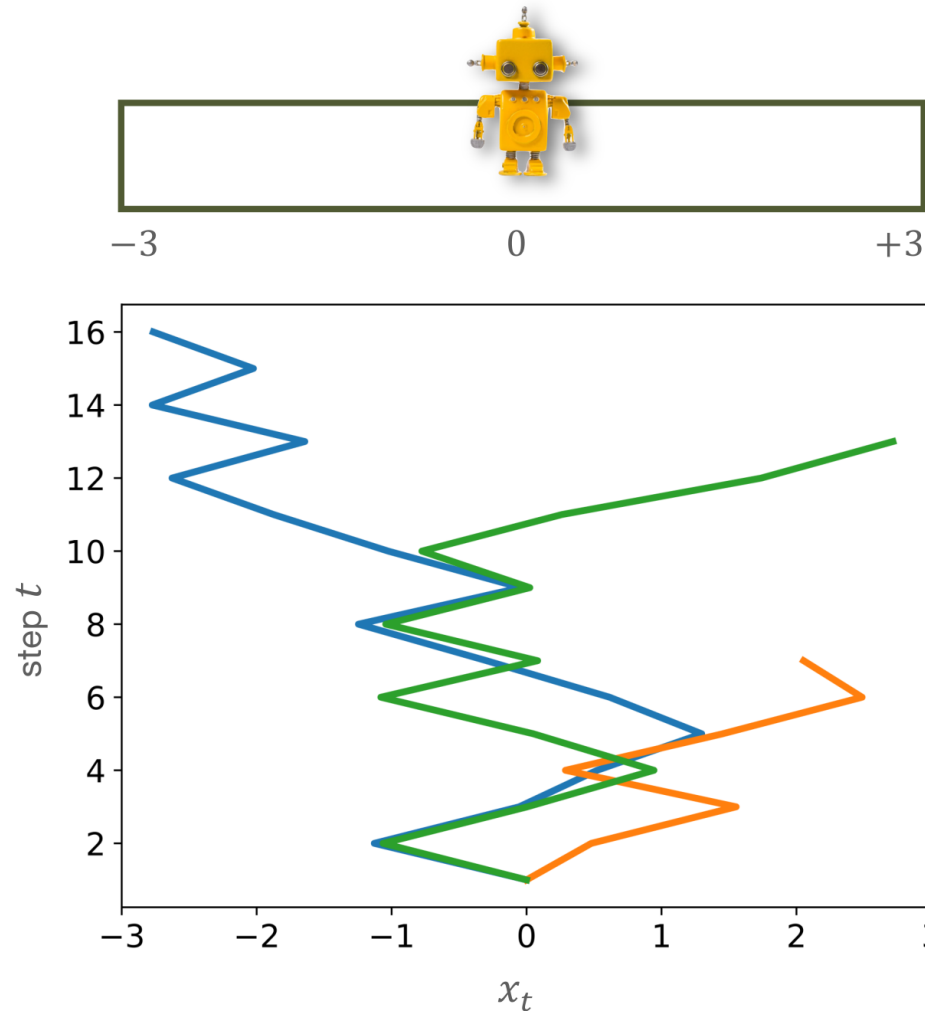
Policy:

$$\begin{aligned}\pi_{w,b}(R \mid x) &= \frac{1}{1 + e^{-(wx+b)}} \\ \pi_{w,b}(L \mid x) &= \frac{1}{1 + e^{wx+b}} \\ \theta &= (w, b)^{\top}\end{aligned}$$

$$\begin{aligned}\nabla_{\theta} \ln \pi_{w,b}(L \mid x) &= -\nabla_{\theta} \ln(1 + e^{wx+b}) \\ &= -\frac{1}{1 + e^{wx+b}} e^{wx+b} (\nabla_{\theta}(wx + b)) \\ &= -\frac{1}{1 + e^{-(wx+b)}} \begin{pmatrix} x \\ 1 \end{pmatrix} \\ &= \pi_{w,b}(R \mid x) \begin{pmatrix} -x \\ -1 \end{pmatrix}.\end{aligned}$$

REINFORCE Example

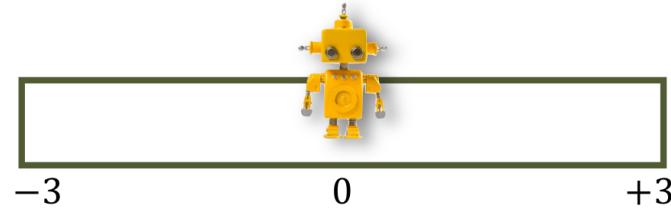
Calculate gradient estimate



(x, a)	Q
(0, R)	-7
(.3, R)	-6
(1.6, L)	-5
(.2, R)	-4
(1.3, R)	-3
(2.4, L)	-2
(2.1, R)	-1

- Start at $w = b = 0$ (so π is uniform random at all x)

REINFORCE Example



Gradient update for a trajectory:

We have one sampled trajectory of (x, a, r) tuples:

$$\tau = [(0, R, -1), (.3, R, -1), (1.6, L, -1), (.2, R, -1), (1.3, R, -1)]$$

the returns-to-go $\bar{R}_t$ are:

$$-5, -4, -3, -2, -1.$$

If $w = b = 0$, then $\pi_{w,b}(R | x) = \pi_{w,b}(L | x) = 0.5$, so

$$\begin{aligned} \mathbf{g} &= \sum_{t=0}^T \bar{R}_t \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t | x_t), \\ &= -5(0.5) \begin{pmatrix} 0 \\ 1 \end{pmatrix} - 4(0.5) \begin{pmatrix} 0.3 \\ 1 \end{pmatrix} - 3(0.5) \begin{pmatrix} -1.6 \\ -1 \end{pmatrix} - 2(0.5) \begin{pmatrix} 0.2 \\ 1 \end{pmatrix} - 1(0.5) \begin{pmatrix} 1.3 \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} 0.95 \\ -4.5 \end{pmatrix}. \end{aligned}$$

We update the policy parameters $\boldsymbol{\theta}$ using the policy gradient $\mathbf{g}$:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \mathbf{g} = \boldsymbol{\theta} + \alpha \begin{pmatrix} 0.95 \\ -4.5 \end{pmatrix}.$$

$$\nabla_{\boldsymbol{\theta}} \log \pi_{w,b}(R | x) = \pi_{w,b}(L | x) \begin{pmatrix} x \\ 1 \end{pmatrix},$$

$$\nabla_{\boldsymbol{\theta}} \log \pi_{w,b}(L | x) = \pi_{w,b}(R | x) \begin{pmatrix} -x \\ -1 \end{pmatrix}.$$

Failure modes of REINFORCE

- Exploration
 - We get a nonzero gradient only if cost/reward is nonzero
 - If feedback is sparse and we start with a random policy, might wander forever without learning anything
 - Imagine: learning to cross a tightrope
- Cancellation (low SNR)
 - Total return Q can scale with horizon, and can vary a lot due to randomness in policy, environment
 - Overall gradient can be much smaller (terms w/ opposite signs)
- Getting stuck
 - When policy gets close to border of simplex, score vectors for unlikely actions get large, probability of seeing them gets small
 - In the limit, $\infty \cdot 0$ (have to sample a really long time to average!)

Failure modes multiply: might have to explore a long time to find feedback, then do it over to average out variance from cancellation, then over again to compensate for being stuck

Upshot: if we try to scale, can only use tiny learning rate

Algorithm 1: REINFORCE

If we plug the estimate from the policy gradient theorem into the policy gradient method, we get one of the oldest RL algorithms: REINFORCE [Williams, 1992]

Repeat:

- gather some trajectories under current policy π_{θ}
- compute gradient estimate g by policy gradient theorem
- update θ by SGD

```
1: procedure REINFORCE( $\eta, \gamma$ )
2:   Initialize policy parameters  $\theta$  randomly
3:   while not converged do
4:     Sample a trajectory  $\tau = [(s_0, a_0, r_0), \dots, (s_T, a_T, r_T)]$  from  $\pi_{\theta}$ 
5:      $g \leftarrow \sum_{t=1}^T \bar{R}_t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$ 
6:     where  $\bar{R}_t = \sum_{t'=t}^T \gamma^{t'-t} r_{t'}$ 
7:      $\theta \leftarrow \theta + \eta g$ 
8:   return  $\pi_{\theta}$ 
```

Algorithm 2: REINFORCE with Baseline

- The policy gradient estimator $\mathbf{g}$ is unbiased, but it can have high variance.
- To reduce variance, we introduce a baseline function $b(s)$ by replacing R_t with $(R_t - b(s_t))$
- $b(s)$ can be any function of the state s , and $\mathbf{g}$ will still be an unbiased estimator of $\nabla J(\boldsymbol{\theta})$
- The value function $V^\pi(s)$ is a common choice for $b(s)$

```
1: procedure REINFORCEWITHBASELINE( $\eta_\theta, \eta_\phi, \gamma$ )
2:   Initialize policy parameters  $\boldsymbol{\theta}$  and baseline parameters  $\phi$  randomly
3:   while not converged do
4:     Sample a trajectory  $\tau = [(s_0, a_0, r_0), \dots, (s_T, a_T, r_T)]$  from  $\pi_\theta$ 
5:      $\mathbf{g}_\theta \leftarrow \sum_{t=1}^T (\bar{R}_t - V_\phi(s_t)) \nabla_{\boldsymbol{\theta}} \log \pi_\theta(a_t | s_t)$ 
6:     where  $\bar{R}_t = \sum_{t'=t}^T \gamma^{t'-t} r_{t'}$ 
7:      $\mathbf{g}_\phi \leftarrow \sum_{t=1}^T \nabla_\phi (V_\phi(s_t) - \bar{R}_t)^2$ 
8:      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta_\theta \mathbf{g}_\theta$ 
9:      $\phi \leftarrow \phi - \eta_\phi \mathbf{g}_\phi$ 
10:  return  $\pi_\theta, V_\phi$ 
```

Actor-critic

$$g = \sum_{t=1}^T Q_t u_t \quad \text{REINFORCE}$$

- Reduce variance: replace $\overline{R_t}$ by conditional expectation
 - $\mathbb{E}(\overline{R_t} \mid s_t, a_t) = Q^\pi(s_t, a_t)$
 - $g_{\text{exactQ}} = \sum_{t=1}^T Q^\pi(s_t, a_t) u_t$ — usually not implementable
- Or use learned approximation
 - $g_{\text{AC}} = \sum_{t=1}^T Q_\phi^\pi(s_t, a_t) u_t$
 - unsafe: introduces bias to gradient estimate due to Q_ϕ^π
 - but still can be highly successful
- Bias means we may make policy worse instead of better
 - if bias is enough to alter gradient more than 90°
 - happens if we already have a good policy, **or** if we have a very small gradient signal (e.g., sparse costs/rewards)

Algorithm 3: Actor-Critic

In practice we treat

$$Q_{\phi}(s_t, a_t) = r_t + \gamma V_{\phi}(s_{t+1})$$

Then we learn $V_{\phi}(s)$

Here are we learning $V_{\phi}(s)$ in just the same way we learned it earlier

```
1: procedure ACTORCRITIC( $\eta_{\theta}, \eta_{\phi}, \gamma$ )
2:   Initialize policy parameters  $\theta$  and value parameters  $\phi$  randomly
3:   while not converged do
4:     Collect a trajectory  $\tau = [(s_0, a_0, r_0), \dots, (s_T, a_T, r_T)]$  from  $\pi_{\theta}$ 
5:      $\mathbf{g}_{\theta} \leftarrow \sum_{t=1}^T \hat{Q}_t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$ 
6:     where  $\hat{Q}_t = r_t + \gamma V_{\phi}(s_{t+1})$ 
7:      $\mathbf{g}_{\phi} \leftarrow \sum_{t=1}^T \nabla_{\phi} (V_{\phi}(s_t) - \hat{Q}_t)^2$ 
8:      $\theta \leftarrow \theta + \eta_{\theta} \mathbf{g}_{\theta}$ 
9:      $\phi \leftarrow \phi - \eta_{\phi} \mathbf{g}_{\phi}$ 
10:  return  $\pi_{\theta}, V_{\phi}$ 
```

- the policy is the **actor**
- the value function is the **critic**

Algorithm 4: Advantage Actor-Critic (A2C)

- In AAC we take the **baseline** and the **critic** together
- even less variance than actor-critic, but bias remains
- $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$ is called the *advantage* of a in s
- large $A^\pi(s, a)$ means it's advantageous to take a in s (vs. following π)

```
1: procedure A2C( $\eta_\theta, \eta_\phi, \gamma$ )
2:   Initialize policy parameters  $\theta$  and value parameters  $\phi$  randomly
3:   while not converged do
4:     Collect a trajectory  $\tau = [(s_0, a_0, r_0), \dots, (s_T, a_T, r_T)]$  from  $\pi_\theta$ 
5:      $\mathbf{g}_\theta \leftarrow \sum_{t=1}^T \hat{A}_t \nabla_\theta \log \pi_\theta(a_t | s_t)$ 
6:     where  $\hat{A}_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$ 
7:      $\mathbf{g}_\phi \leftarrow \sum_{t=1}^T \nabla_\phi (r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t))^2$ 
8:      $\theta \leftarrow \theta + \eta_\theta \mathbf{g}_\theta$ 
9:      $\phi \leftarrow \phi - \eta_\phi \mathbf{g}_\phi$ 
10:  return  $\pi_\theta, V_\phi$ 
```

Algorithm 5: A2C with n-Step Returns

- The advantage actor-critic algorithm uses a **one-step return** to estimate the advantage function, which can lead to high bias in the policy gradient estimate.
- To reduce bias, we can use an **n-step return** to estimate the advantage function
- Closely connected to temporal difference learning, specifically TD(lambda)

```
1: procedure A2C-NSTEP( $\eta_\theta, \eta_\phi, \gamma, n$ )
2:   Initialize policy parameters  $\theta$  and value parameters  $\phi$  randomly
3:   while not converged do
4:     Collect a trajectory  $\tau = [(s_0, a_0, r_0), \dots, (s_T, a_T, r_T)]$  from  $\pi_\theta$ 
5:      $\mathbf{g}_\theta \leftarrow \sum_{t=1}^T \hat{A}_t^{(n)} \nabla_\theta \log \pi_\theta(a_t | s_t)$ 
6:     where  $\hat{A}_t^{(n)} = \hat{R}_t^{(n)} - V_\phi(s_t)$ 
7:     where  $\hat{R}_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n V_\phi(s_{t+n})$ 
8:      $\mathbf{g}_\phi \leftarrow \sum_{t=1}^T \nabla_\phi \left( \hat{R}_t^{(n)} - V_\phi(s_t) \right)^2$ 
9:      $\theta \leftarrow \theta + \eta_\theta \mathbf{g}_\theta$ 
10:     $\phi \leftarrow \phi - \eta_\phi \mathbf{g}_\phi$ 
11:   return  $\pi_\theta, V_\phi$ 
```

Comparison of Policy Gradient Methods

1) All of the algorithms compute a gradient of the form:

$$\mathbf{g}_{\theta} = \sum_{t=1}^T \tilde{G}_t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

where $\tilde{G}_t$ is an estimate of the expected discounted future reward from time step t

2) Some use a V_{ϕ} , and some do not

Algorithm	$\tilde{G}_t$	Estimate V_{ϕ} ?
REINFORCE	$\bar{R}_t$	x
REINFORCE with Baseline	$(\bar{R}_t - V_{\phi}(s_t))$	✓
Actor-Critic	$\hat{Q}_t = r_t + \gamma V_{\phi}(s_{t+1})$	✓
Advantage Actor-Critic	$\hat{A}_t = r_t + \gamma V_{\phi}(s_{t+1}) - V_{\phi}(s_t)$	✓