

# Reinforcement learning

Episode 6

## Policy gradient methods



Yandex  
Data Factory

LAMBDA 



**British Hedgehog  
Preservation Society**

# Small experiment

The next slide contains a question

Please respond as fast as you can!

# Small experiment



**left or right?**

# Small experiment



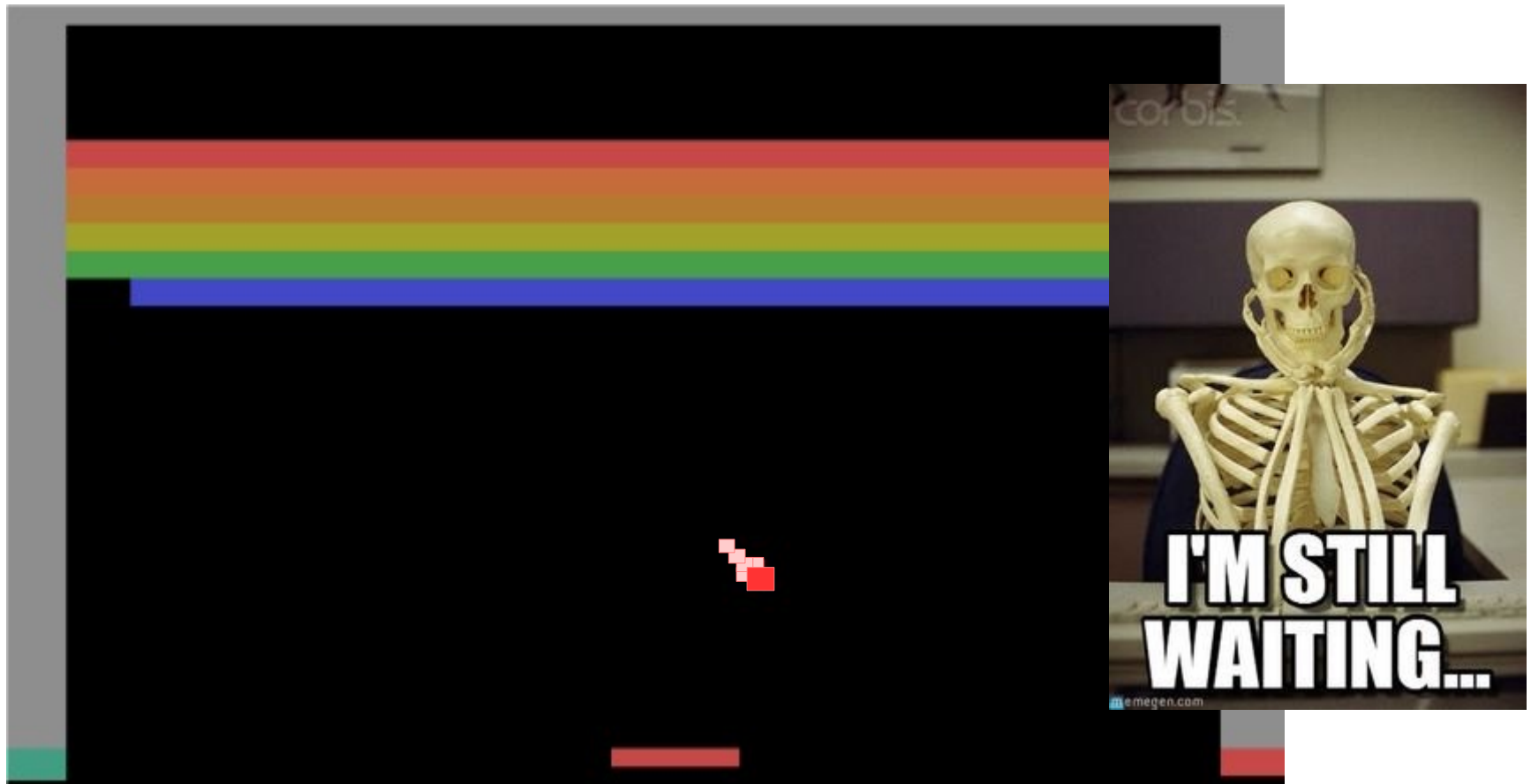
Right! Ready for next one?

# Small experiment



What's  $Q(s, \text{right})$  under  $\gamma=0.99$ ?

# Small experiment



What's  $Q(s, \text{right})$  under  $\gamma=0.99$ ?

# Approximation error

DQN is trained to minimize

$$L \approx E[Q(s_t, a_t) - (r_t + \gamma \cdot \max_{a'} Q(s_{t+1}, a'))]^2$$

## Simple 2-state world

	True	(A)	(B)
$Q(s_0, a_0)$	<i>1</i>	<i>1</i>	<i>2</i>
$Q(s_0, a_1)$	<i>2</i>	<i>2</i>	<i>1</i>
$Q(s_1, a_0)$	<i>3</i>	<i>3</i>	<i>3</i>
$Q(s_1, a_1)$	<i>100</i>	<i>50</i>	<i>100</i>

**Trivia:** Which prediction is better (A/B)?

# Approximation error

DQN is trained to minimize

$$L \approx E[Q(s_t, a_t) - (r_t + \gamma \cdot \max_{a'} Q(s_{t+1}, a'))]^2$$

## Simple 2-state world

	True	(A)	(B)
Q(s0,a0)	1	1	2
Q(s0,a1)	2	2	1
Q(s1,a0)	3	3	3
Q(s1,a1)	100	50	100

better  
policy

less  
MSE



# Approximation error

DQN is trained to minimize

$$L \approx E[Q(s_t, a_t) - (r_t + \gamma \cdot \max_{a'} Q(s_{t+1}, a'))]^2$$

## Simple 2-state world

	True	(A)	(B)
Q(s0,a0)	1	1	2
Q(s0,a1)	2	2	1
Q(s1,a0)	3	3	3
Q(s1,a1)	100	50	100

Q-learning will prefer worse policy (B)!

better  
policy

less  
MSE

# Conclusion

- Often computing q-values is harder than picking optimal actions!
- We could avoid learning value functions by directly learning agent's policy  $\pi_{\theta}(a|s)$

**Trivia:** what algorithm works that way?

(of those we studied)

# Conclusion

- Often computing q-values is harder than picking optimal actions!
- We could avoid learning value functions by directly learning agent's policy  $\pi_{\theta}(a|s)$

**Trivia:** what algorithm works that way?

**e.g. crossentropy method**

# NOT how humans survived

$\text{argmax}[$   
Q(s, pet the tiger)  
Q(s, run from tiger)  
Q(s, provoke tiger)  
Q(s, ignore tiger)  
]



# how humans survived

$$\pi(run|s)=1$$



# Policies

In general, two kinds

- Deterministic policy

$$a = \pi_{\theta}(s)$$

- Stochastic policy

$$a \sim \pi_{\theta}(a|s)$$

**Trivia:** Any case where stochastic is better?

# Policies

In general, two kinds

- Deterministic policy

$$a = \pi_{\theta}(s)$$

- Stochastic policy

$$a \sim \pi_{\theta}(a|s)$$

e.g. rock-paper  
-scissors

**Trivia:** Any case where stochastic is better?

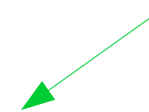
# Policies

In general, two kinds

- Deterministic policy

Genetic algos (week 0)  
Deterministic policy gradient

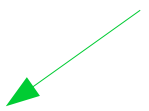
same action each time


$$a = \pi_{\theta}(s)$$

- Stochastic policy

Crossentropy method  
Policy gradient

sampling takes care  
of exploration


$$a \sim \pi_{\theta}(a|s)$$

**Trivia:** how to represent policy in continuous action space?



# Policies

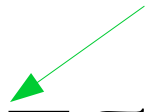
In general, two kinds

- Deterministic policy

Genetic algos (week 0)

Deterministic policy gradient

same action each time

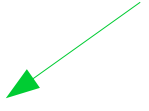

$$a = \pi_{\theta}(s)$$

- Stochastic policy

Crossentropy method

Policy gradient

sampling takes care  
of exploration


$$a \sim \pi_{\theta}(a|s)$$

categorical, normal, mixture of normal, whatever

# Two approaches

- **Value based:**

Learn value function  $Q_{\theta}(s, a)$  or  $V_{\theta}(s)$

Infer policy  $\pi(a|s) = [a = \underset{a}{\operatorname{argmax}} Q_{\theta}(s, a)]$

- **Policy based:**

Explicitly learn policy  $\pi_{\theta}(a|s)$  or  $\pi_{\theta}(s) \rightarrow a$

Implicitly maximize reward over policy

# Recap: crossentropy method

- Initialize NN weights  $\theta_0 \leftarrow \text{random}$
- Loop:
  - Sample N sessions
  - elite = take M best sessions and concatenate

$$\theta_{i+1} = \theta_i + \alpha \nabla \sum_i \log \pi_{\theta_i}(a_i | s_i) \cdot [s_i, a_i \in \text{Elite}]$$

**Trivia:** Can we adapt it to discounted rewards?  
(with  $\gamma$ )

# Recap: crossentropy method

- Initialize NN weights  $\theta_0 \leftarrow \text{random}$
- Loop:
  - Sample N sessions
  - elite = take M best sessions and concatenate

$$\theta_{i+1} = \theta_i + \alpha \nabla \sum_i \log \pi_{\theta_i}(a_i | s_i) \cdot [s_i, a_i \in \text{Elite}]$$

TD version: elite (s,a) that have highest R(s,a)  
(select elites independently from each state)

# Policy gradient main idea

Why so complicated?

We'd rather simply maximize  $R$  over  $\pi$ !

# Objective

Expected reward:

$$J = E_{\substack{s \sim p(s) \\ a \sim \pi_{\theta}(s|a) \\ \dots}} R(s, a, s', a', \dots)$$

Expected discounted reward:

$$J = E_{\substack{s \sim p(s) \\ a \sim \pi_{\theta}(s|a)}} Q(s, a)$$

# Objective

Expected reward:  $R(z)$  setting

$$J = E_{\substack{s \sim p(s) \\ a \sim \pi_{\theta}(s|a) \\ \dots}} R(s, a, s', a', \dots)$$

Expected discounted reward:  $R(s,a) = r + \gamma * R(s',a')$

$$J = E_{\substack{s \sim p(s) \\ a \sim \pi_{\theta}(s|a)}} Q(s, a)$$

“true” Q-function



# Objective

$$J = \mathop{E}_{\substack{s \sim p(s) \\ a \sim \pi_\theta(s|a)}} Q(s, a) = \int_s p(s) \int_a \pi_\theta(a|s) Q(s, a) da ds$$



# Objective

$$J = E_{\substack{s \sim p(s) \\ a \sim \pi_{\theta}(s|a)}} Q(s, a) = \int_s p(s) \int_a \pi_{\theta}(a|s) Q(s, a) da ds$$

Diagram illustrating the components of the objective function  $J$ :

- Agent's policy**: Points to  $\pi_{\theta}(a|s)$ .
- state visitation frequency (may depend on policy)**: Points to  $p(s)$ .
- True action value**: Points to  $Q(s, a)$ .

**Trivia:** how do we compute that?

# Objective

$$J = E_{\substack{s \sim p(s) \\ a \sim \pi_{\theta}(s|a)}} Q(s, a) = \int_s p(s) \int_a \pi_{\theta}(a|s) Q(s, a) da ds$$

$$J \approx \frac{1}{N} \sum_{i=0}^N \sum_{s, a \in z_i} Q(s, a)$$

True action value  
a.k.a.  $E[ R(s, a) ]$

sample N sessions

# Objective

$$J = E_{\substack{s \sim p(s) \\ a \sim \pi_\theta(s|a)}} Q(s, a) = \int_s p(s) \int_a \pi_\theta(a|s) Q(s, a) da ds$$

$$J \approx \frac{1}{N} \sum_{i=0}^N \sum_{s, a \in z_i} Q(s, a)$$

True action value  
a.k.a.  $E[ R(s, a) ]$

sample N sessions

**Can we optimize policy now?**

# Objective

$$J = E_{\substack{s \sim p(s) \\ a \sim \pi_{\theta}(s|a)}} Q(s, a) = \int_s p(s) \int_a \pi_{\theta}(a|s) Q(s, a) da ds$$

parameters “sit” here

$$J \approx \frac{1}{N} \sum_{i=0}^N \sum_{s, a \in z_i} Q(s, a)$$

True action value  
a.k.a.  $E[ R(s, a) ]$

**We don't know how to compute  $dJ/d\theta$**

# Optimization

## Finite differences

- Change policy a little, evaluate

$$\nabla J \approx \frac{J_{\theta+\epsilon} - J_{\theta}}{\epsilon}$$

## Stochastic optimization

- Good old crossentropy method
- Maximize probability of “elite” actions

# Optimization

## Finite differences

- Change policy a little, evaluate

$$\nabla J \approx \frac{J_{\theta+\epsilon} - J_{\theta}}{\epsilon}$$

## Stochastic optimization

- Good old crossentropy method
- Maximize probability of “elite” actions

**Trivia:** any problems with those two?

# Optimization

## Finite differences

- Change policy a little, evaluate

$$\nabla J \approx \frac{J_{\theta+\epsilon} - J_{\theta}}{\epsilon}$$

VERY noisy, especially  
if both J are sampled

## Stochastic optimization

- Good old crossentropy method
- Maximize probability of “elite” actions

“quantile convergence”  
problems with stochastic  
MDPs

# Objective

$$J = E_{\substack{s \sim p(s) \\ a \sim \pi_{\theta}(s|a)}} Q(s, a) = \int_s p(s) \int_a \pi_{\theta}(a|s) Q(s, a) da ds$$

Wish list:

- Analytical gradient
- Easy/stable approximations



# Logderivative trick

Simple math

$$\nabla \log \pi(z) = ? ? ?$$

(try chain rule)

# Logderivative trick

Simple math

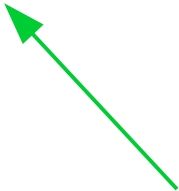
$$\nabla \log \pi(z) = \frac{1}{\pi(z)} \cdot \nabla \pi(z)$$

$$\pi \cdot \nabla \log \pi(z) = \nabla \pi(z)$$

# Policy gradient

Analytical inference

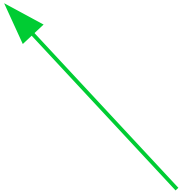
$$\nabla J = \int_s p(s) \int_a \nabla \pi_\theta(a|s) Q(s, a) da ds$$

$$\pi \cdot \nabla \log \pi(z) = \nabla \pi(z)$$


# Policy gradient

## Analytical inference

$$\nabla J = \int_s p(s) \int_a \nabla \pi_\theta(a|s) Q(s, a) da ds$$

$$\pi \cdot \nabla \log \pi(z) = \nabla \pi(z)$$


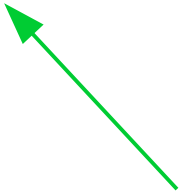
$$\nabla J = \int_s p(s) \int_a \pi_\theta(a|s) \nabla \log \pi_\theta(a|s) Q(s, a) da ds$$

**Trivia:** anything curious about that formula?

# Policy gradient

## Analytical inference

$$\nabla J = \int_s p(s) \int_a \nabla \pi_\theta(a|s) Q(s, a) da ds$$

$$\pi \cdot \nabla \log \pi(z) = \nabla \pi(z)$$


$$\nabla J = \int_s p(s) \int_a \pi_\theta(a|s) \nabla \log \pi_\theta(a|s) Q(s, a) da ds$$

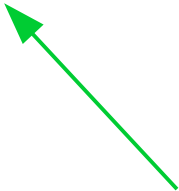
---

that's expectation :)

# Policy gradient

## Analytical inference

$$\nabla J = \int_s p(s) \int_a \nabla \pi_\theta(a|s) Q(s, a) da ds$$

$$\pi \cdot \nabla \log \pi(z) = \nabla \pi(z)$$


$$\nabla J = E_{\substack{s \sim p(s) \\ a \sim \pi_\theta(s|a)}} \nabla \log \pi_\theta(a|s) \cdot Q(s, a)$$

# Policy gradient (REINFORCE)

- Policy gradient

$$\nabla J = E_{\substack{s \sim p(s) \\ a \sim \pi_{\theta}(s|a)}} \nabla \log \pi_{\theta}(a|s) \cdot Q(s, a)$$

- Approximate with sampling

$$\nabla J \approx \frac{1}{N} \sum_{i=0}^N \sum_{s, a \in z_i} \nabla \log \pi_{\theta}(a|s) \cdot Q(s, a)$$

# REINFORCE algorithm

- Initialize NN weights  $\theta_0 \leftarrow \text{random}$
- Loop:
  - Sample N sessions  $\mathbf{z}$  under current  $\pi_\theta(a|s)$
  - Evaluate policy gradient

$$\nabla J \approx \frac{1}{N} \sum_{i=0}^N \sum_{s, a \in \mathbf{z}_i} \nabla \log \pi_\theta(a|s) \cdot Q(s, a)$$

- Ascend  $\theta_{i+1} \leftarrow \theta_i + \alpha \cdot \nabla J$



# REINFORCE algorithm

- Initialize NN weights  $\theta_0 \leftarrow \text{random}$

**Trivia:** is it off- or on-policy?

- Loop:
  - Sample N sessions  $\mathbf{z}$  under current  $\pi_\theta(a|s)$
  - Evaluate policy gradient

$$\nabla J \approx \frac{1}{N} \sum_{i=0}^N \sum_{s, a \in \mathbf{z}_i} \nabla \log \pi_\theta(a|s) \cdot Q(s, a)$$

- Ascend  $\theta_{i+1} \leftarrow \theta_i + \alpha \cdot \nabla J$

# REINFORCE algorithm

- Initialize NN weights  $\theta_0 \leftarrow \text{random}$
- Loop:
  - Sample N sessions  $\mathbf{z}$  under current  $\pi_\theta(a|s)$   
actions under current policy  
= on-policy
  - Evaluate policy gradient

$$\nabla J \approx \frac{1}{N} \sum_{i=0}^N \sum_{s, a \in \mathbf{z}_i} \nabla \log \pi_\theta(a|s) \cdot Q(s, a)$$

- Ascend  $\theta_{i+1} \leftarrow \theta_i + \alpha \cdot \nabla J$

# value-based Vs policy-based

## Value-based

- Q-learning, SARSA, MCTS value-iteration
- Solves harder problem
- Artificial exploration
- Learns from partial experience (temporal difference)
- Evaluates strategy for free :)

## Policy-based

- REINFORCE, CEM
- Solves easier problem
- Innate exploration
- Innate stochasticity
- Support continuous action space
- Learns from full session only



# value-based Vs policy-based

## Value-based

- Q-learning, SARSA, MCTS value-iteration
- Solves harder problem
- Artificial exploration
- Learns from partial experience (temporal difference)
- Evaluates strategy for free :)

## Policy-based

- REINFORCE, CEM

**We'll learn much more soon!**

- Solves easier problem
- Innate exploration
- Innate stochasticity
- Support continuous action space
- ~~Learns from full session only~~



# REINFORCE algorithm

- Initialize NN weights  $\theta_0 \leftarrow \text{random}$
- Loop:
  - Sample N sessions  $\mathbf{z}$  under current  $\pi_\theta(a|s)$
  - Evaluate policy gradient

$$\nabla J \approx \frac{1}{N} \sum_{i=0}^N \sum_{s, a \in \mathbf{z}_i} \nabla \log \pi_\theta(a|s) \cdot Q(s, a)$$

What is better for learning:  
random action in good state  
or  
great action in bad state?

# REINFORCE baseline

- Initialize NN weights  $\theta_0 \leftarrow \text{random}$
- Loop:
  - Sample N sessions  $\mathbf{z}$  under current  $\pi_\theta(a|s)$
  - Evaluate policy gradient

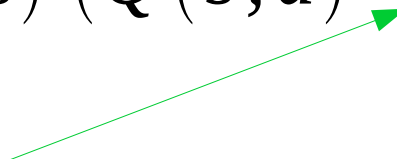
$$\nabla J \approx \frac{1}{N} \sum_{i=0}^N \sum_{s, a \in \mathbf{z}_i} \nabla \log \pi_\theta(a|s) \cdot Q(s, a)$$

$$Q(s, a) = V(s) + A(s, a)$$

Actions influence  $A(s, a)$  only, so  $V(s)$  is irrelevant

# REINFORCE baseline

- Initialize NN weights  $\theta_0 \leftarrow \text{random}$
- Loop:
  - Sample N sessions  $\mathbf{z}$  under current  $\pi_\theta(a|s)$
  - Evaluate policy gradient

$$\nabla J \approx \frac{1}{N} \sum_{i=0}^N \sum_{s, a \in \mathbf{z}_i} \nabla \log \pi_\theta(a|s) \cdot (Q(s, a) - b(s))$$


Anything that doesn't depend on action  
ideally,  $b(s) = V(s)$



# Actor-critic

- Learn both  $V(s)$  and  $\pi_{\theta}(a|s)$
- Hope for best of both worlds :)





# Advantage actor-critic

Idea: learn both  $\pi_{\theta}(a|s)$  and  $V_{\theta}(s)$

Use  $V_{\theta}(s)$  to learn  $\pi_{\theta}(a|s)$  faster!

**Non-trivia:** how can we estimate  $\mathbf{A(s,a)}$   
from  $(s,a,r,s')$  and V-function?

# Advantage actor-critic

Idea: learn both  $\pi_{\theta}(a|s)$  and  $V_{\theta}(s)$

Use  $V_{\theta}(s)$  to learn  $\pi_{\theta}(a|s)$  faster!

$$A(s, a) = Q(s, a) - V(s)$$

$$Q(s, a) = r + \gamma \cdot V(s')$$

$$A(s, a) = r + \gamma \cdot V(s') - V(s)$$

# Advantage actor-critic

Idea: learn both  $\pi_{\theta}(a|s)$  and  $V_{\theta}(s)$

Use  $V_{\theta}(s)$  to learn  $\pi_{\theta}(a|s)$  faster!

$$A(s, a) = Q(s, a) - V(s)$$

$$Q(s, a) = r + \gamma \cdot V(s')$$

$$A(s, a) = r + \gamma \cdot V(s') - V(s)$$

Also: n-step  
version

# Advantage actor-critic

Idea: learn both  $\pi_{\theta}(a|s)$  and  $V_{\theta}(s)$

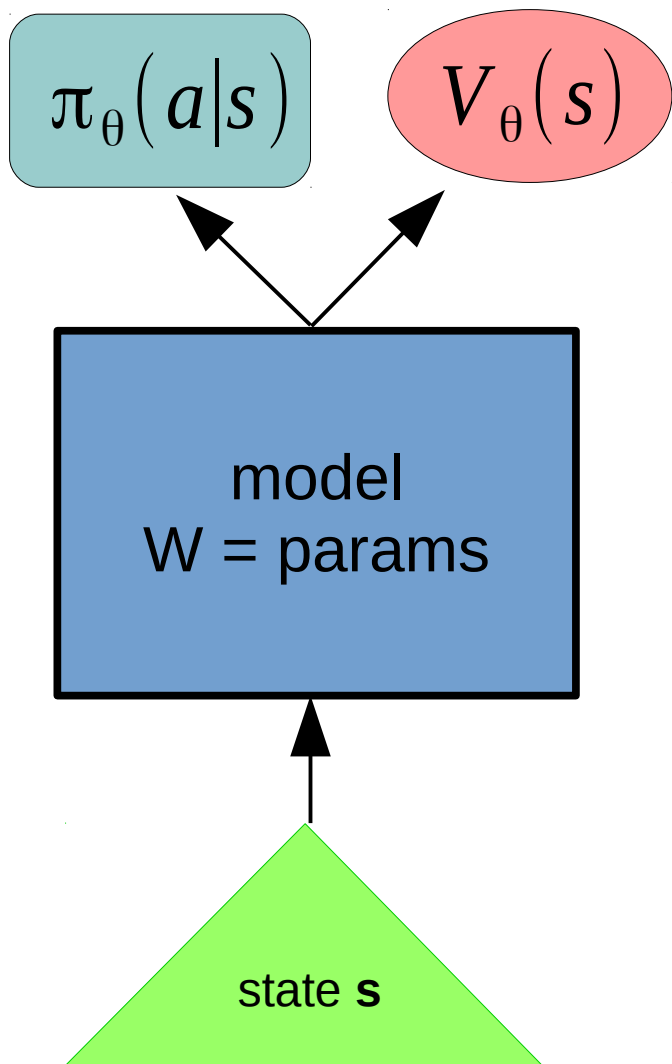
Use  $V_{\theta}(s)$  to learn  $\pi_{\theta}(a|s)$  faster!

$$A(s, a) = r + \gamma \cdot V(s') - V(s)$$

$$\nabla J_{actor} \approx \frac{1}{N} \sum_{i=0}^N \sum_{s, a \in z_i} \nabla \log \pi_{\theta}(a|s) \cdot \underbrace{A(s, a)}_{\text{consider const}}$$

**Trivia:** how do we train  $V$  then?

# Advantage actor-critic



Improve policy:

$$\nabla J_{actor} \approx \frac{1}{N} \sum_{i=0}^N \sum_{s,a \in z_i} \nabla \log \pi_{\theta}(a|s) \cdot A(s, a)$$

Improve value:

$$L_{critic} \approx \frac{1}{N} \sum_{i=0}^N \sum_{s,a \in z_i} (V_{\theta}(s) - [r + \gamma \cdot V(s')])^2$$

# Continuous action spaces

What if there's continuously many actions?

- Robot control: control motor voltage
- Trading: assign money to equity

How does the algorithm change?

# Continuous action spaces

What if there's continuously many actions?

- Robot control: control motor voltage
- Trading: assign money to equity

How does the algorithm change?

it doesn't :)  
Just plug in a different formula for  
 $\pi(a|s)$ , e.g. normal distribution

# Duct tape zone

- $V(s)$  errors less important than in Q-learning
  - actor still learns even if critic is random, just slower
- Regularize with entropy
  - to prevent premature convergence
- Learn on parallel sessions
  - Or super-small experience replay
- Use logsoftmax for numerical stability





Let's code!