



**girafe
ai**

Lecture 06: Policy gradient & REINFORCE

Radoslav Neychev

References

These slides are almost the exact copy of Practical RL course week 6 slides.
Special thanks to YSDA team for making them publicly available.

Original slides link: [week06_policy_based](#)

Small experiment

The next slide contains a question

Please respond as fast as you can!

Small experiment



left or right?

Small experiment



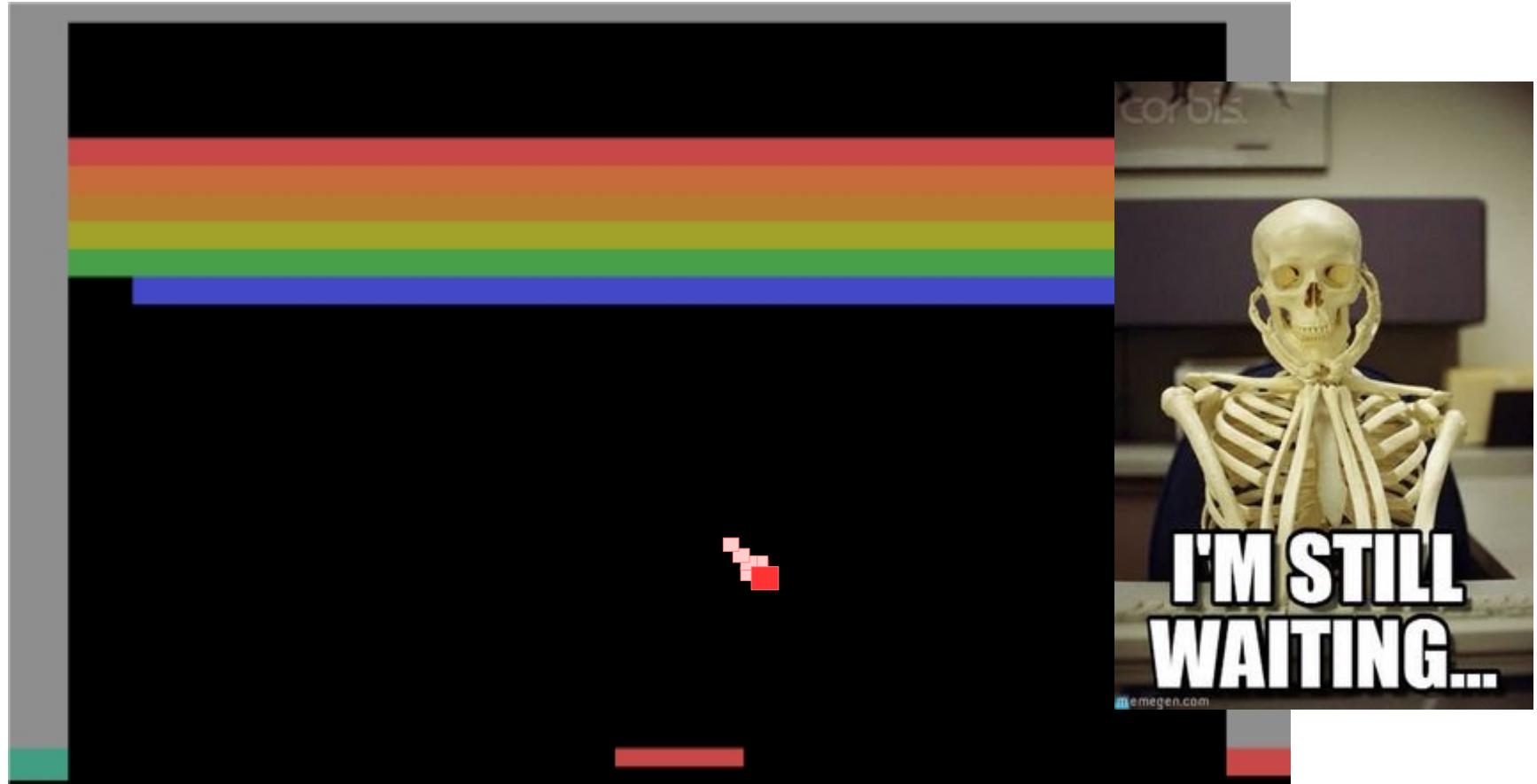
Right! Ready for next one?

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)
$Q(s_0, a_0)$	1	1	2
$Q(s_0, a_1)$	2	2	1
$Q(s_1, a_0)$	3	3	3
$Q(s_1, a_1)$	100	50	100

Q: 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)
$Q(s_0, a_0)$	1	1	2
$Q(s_0, a_1)$	2	2	1
$Q(s_1, a_0)$	3	3	3
$Q(s_1, a_1)$	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)
$Q(s_0, a_0)$	1	1	2
$Q(s_0, a_1)$	2	2	1
$Q(s_1, a_0)$	3	3	3
$Q(s_1, a_1)$	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 $\pi_\theta(a|s)$

Q: what algorithm works that way?
(of those we studied)

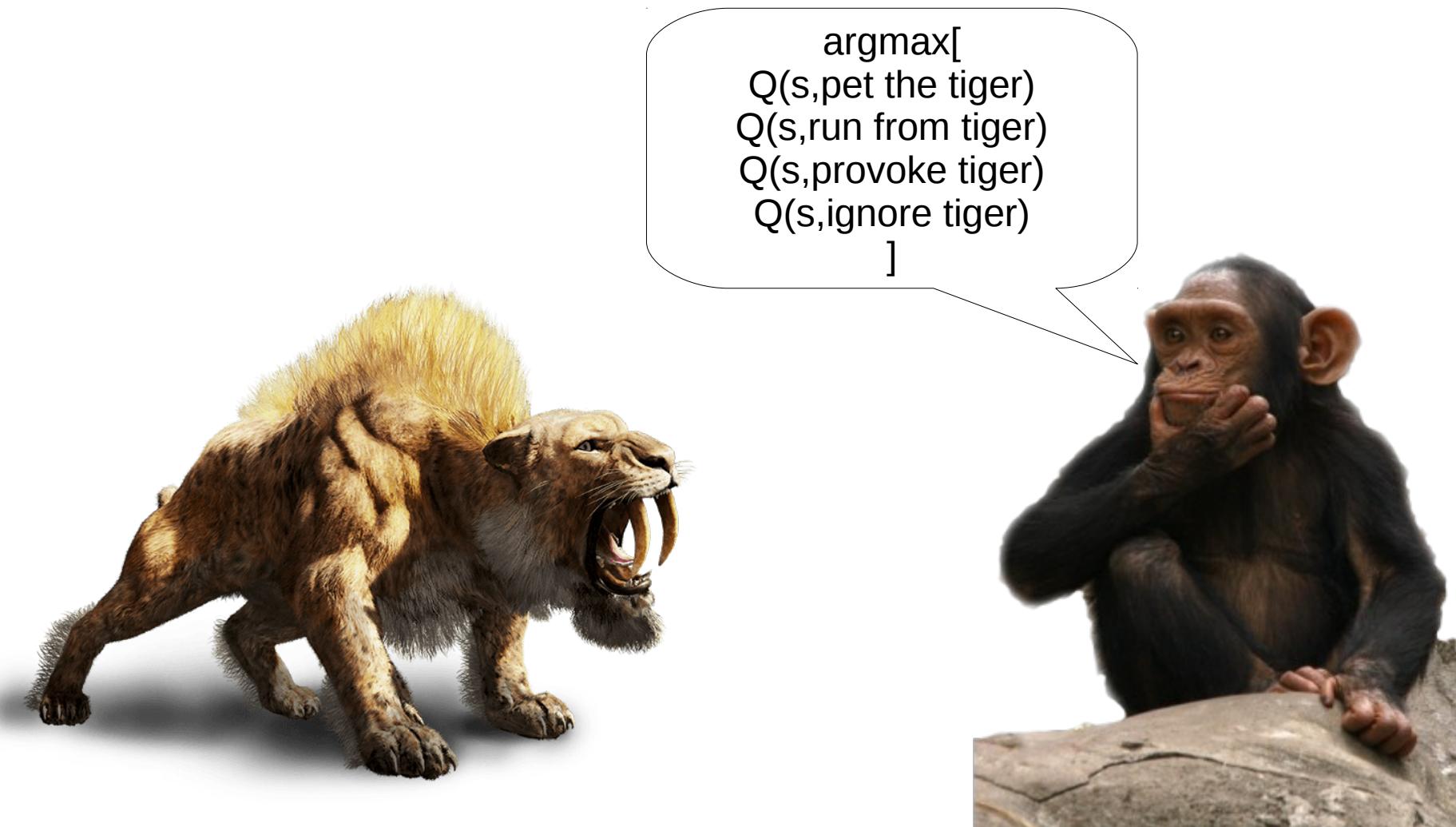
Conclusion

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

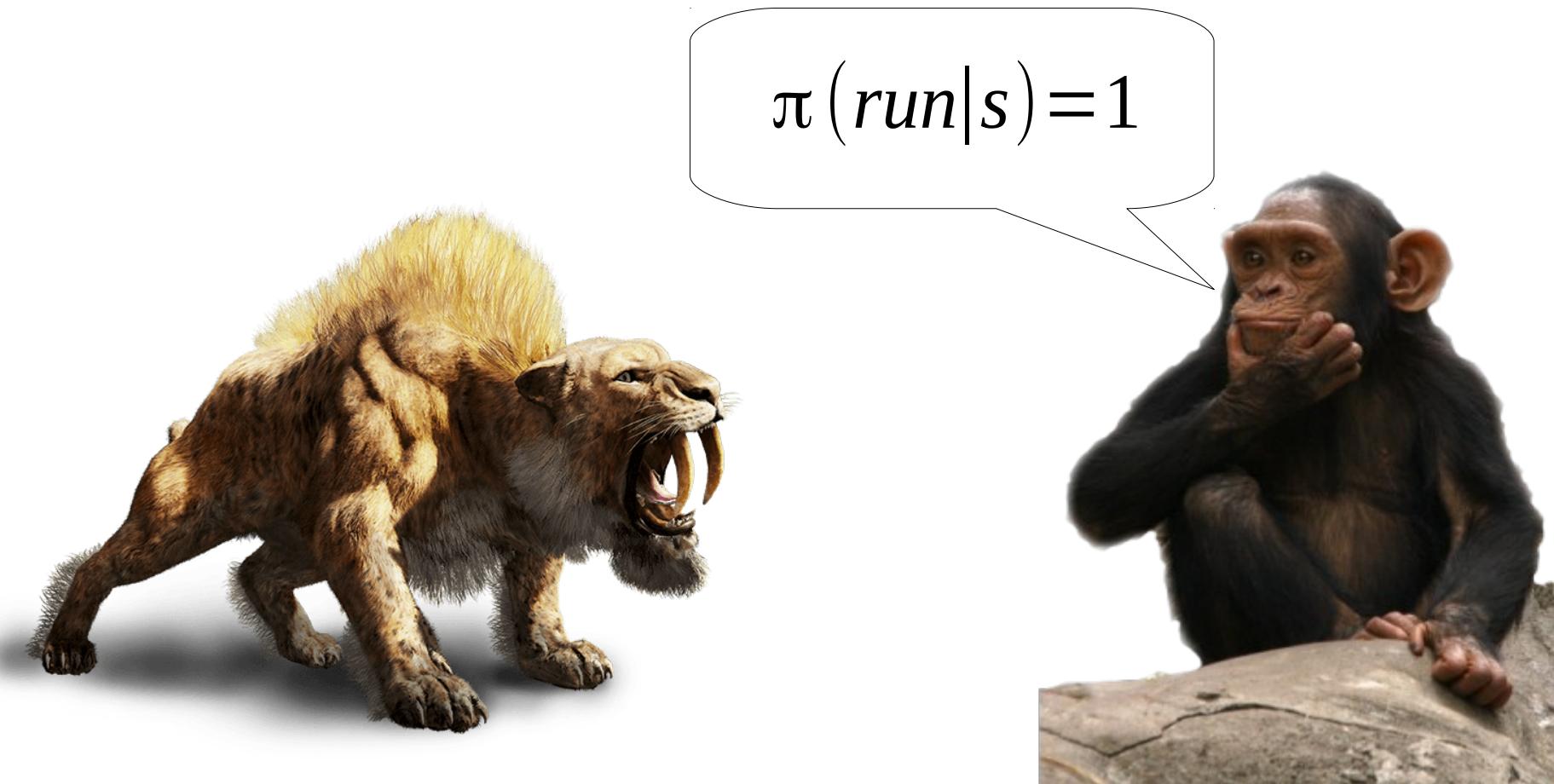
Q: what algorithm works that way?

e.g. crossentropy method

NOT how humans survived



how humans survived



Policies

In general, two kinds

- Deterministic policy

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

- Stochastic policy

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

Q: 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

Q: Any case where stochastic is better?

Policies

In general, two kinds

- Deterministic policy

Genetic algos (week 0)
Deterministic policy gradient

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

same action each time

- Stochastic policy

Crossentropy method
Policy gradient

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

sampling takes care
of exploration

Q: how to represent policy in continuous action space?

Policies

In general, two kinds

- Deterministic policy

Genetic algos (week 0)
Deterministic policy gradient

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

same action each time

- Stochastic policy

Crossentropy method
Policy gradient

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

sampling takes care
of exploration

categorical, normal, mixture of normal, whatever

Two approaches

- **Value based:**

Learn value function

$$Q_\theta(s, a) \text{ or } V_\theta(s)$$

Infer policy

$$a = \underset{a}{\operatorname{argmax}} Q_\theta(s, a)$$

- **Policy based:**

Explicitly learn policy

$$\pi_\theta(a|s) \text{ or } \pi_\theta(s) \rightarrow a$$

Implicitly maximize reward over policy

Recap: crossentropy method

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

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

Recap: crossentropy method

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

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

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

Policy gradient main idea

Why so complicated?

We'd rather maximize reward directly!

Objective

Expected reward:

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

...

Expected discounted reward:

$$J = \underset{\substack{s \sim p(s) \\ a \sim \pi_\theta(s|a)}}{E} G(s, a)$$

Objective

Expected reward: $R(z)$ setting

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

...

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

$$J = \underset{\substack{s \sim p(s) \\ a \sim \pi_\theta(s|a)}}{E} G(s, a)$$

Objective

Consider an 1-step process for simplicity

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

Objective

Consider an 1-step process for simplicity

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

s

a

Reward for 1-step
session

state visitation frequency
(may depend on policy)

Q: how do we compute that?

Objective

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

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

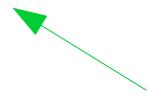


sample N sessions
under current policy

Objective

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

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



sample N sessions

Can we optimize policy now?

Objective

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

parameters “sit” here



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

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

Q: 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

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

Wish list:

- Analytical gradient
- Easy/stable approximations

Logderivative trick

Simple math

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

(try chain rule)

Logderivative trick

Simple math

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

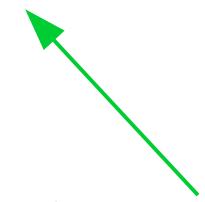
$$\pi \cdot \nabla \log \pi(z) = \nabla \pi(z)$$

Policy gradient

Analytical inference

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

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



Policy gradient

Analytical inference

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

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

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

Q: anything curious about that formula?

Policy gradient

Analytical inference

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

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

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

that's expectation :)

REINFORCE (bandit)

- Initialize NN weights $\theta_0 \leftarrow \text{random}$
- Loop:
 - Sample N sessions \mathbf{z} under current
 - Evaluate policy gradient

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

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

Discounted reward case

- Replace R with Q :)

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

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

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

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

that's expectation :)

REINFORCE (discounted)

- 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

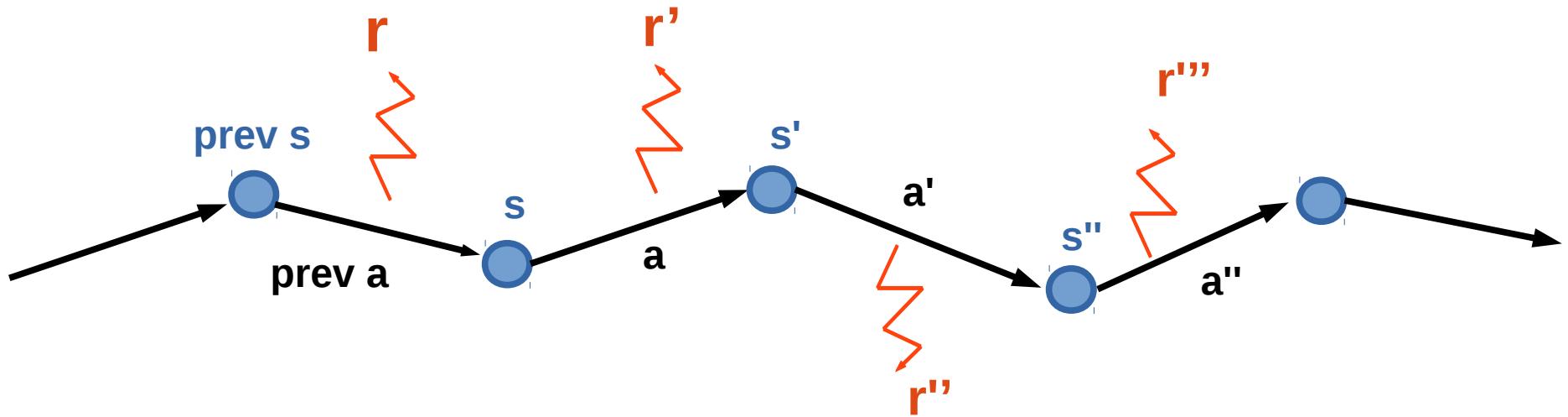
$$\nabla J \approx \frac{1}{N} \sum_{i=0}^N \sum_{s, a \in z_i} \nabla \log \pi_\theta(a|s) \cdot Q(s, a)$$

REINFORCE algorithm

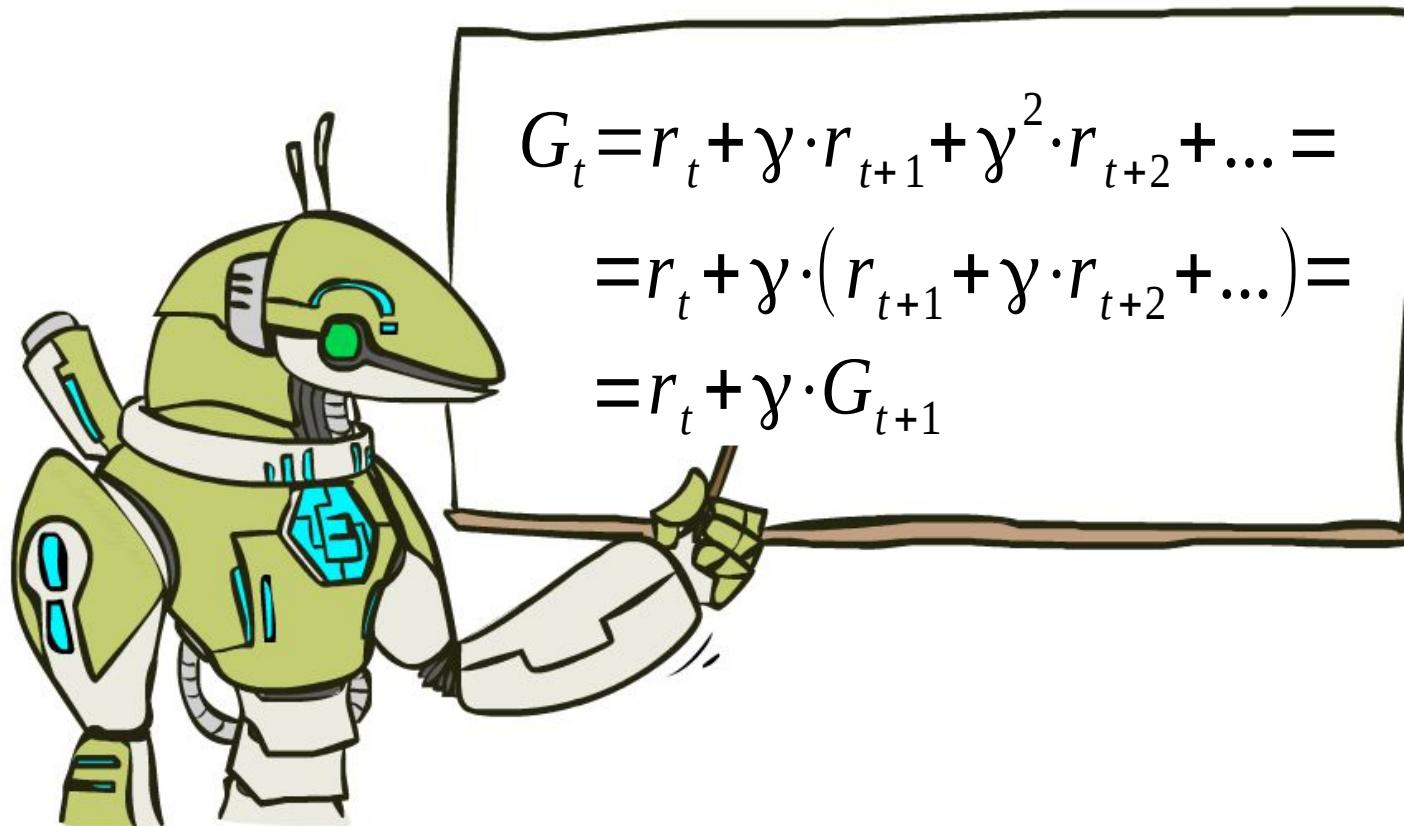
We can estimate Q using G

$$Q_{\pi}(s_t, a_t) = E_s \cdot G(s_t, a_t)$$

$$G_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \dots$$



Recap: discounted rewards



We can use this to compute all G 's
in linear time

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

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

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

REINFORCE algorithm

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

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

- 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 policy $\pi_\theta(a|s)$
 - Evaluate policy gradient

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

- 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~~



- Remember log-derivative trick
- Combining best from both worlds is generally a good idea
- See [this](#) paper for the proof of the policy gradient for discounted rewards
-

REINFORCE baselines

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

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

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

REINFORCE baselines

We can subtract arbitrary baseline $b(s)$

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

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

REINFORCE baselines

We can subtract arbitrary baseline $b(s)$

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

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

Q: Can you simplify the second term?

Note that $b(s)$ does not depend on a

REINFORCE baselines

We can subtract arbitrary baseline $b(s)$

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

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

$$E_{\substack{s \sim p(s) \\ a \sim \pi_\theta(a|s)}} \nabla \log \pi_\theta(a|s) b(s) = b(s) \cdot E_{\substack{s \sim p(s) \\ a \sim \pi_\theta(a|s)}} \nabla \log \pi_\theta(a|s) = 0$$

REINFORCE baselines

We can subtract arbitrary baseline $b(s)$

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

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

Gradient direction doesn't change!

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

REINFORCE baselines

- Gradient direction ∇J stays the same
- Variance may change

Gradient variance: $Var[Q(s, a) - b(s)]$
as a *random variable over (s, a)*

$$Var[Q(s, a)] - 2 \cdot Cov[Q(s, a), b(s)] + Var[b(s)]$$

REINFORCE baselines

- Gradient direction ∇J stays the same
- Variance may change

Gradient variance: $\text{Var}[Q(s, a) - b(s)]$
as a *random variable over (s, a)*

$$\text{Var}[Q(s, a)] - 2 \cdot \text{Cov}[Q(s, a), b(s)] + \text{Var}[b(s)]$$



If $b(s)$ correlates with $Q(s, a)$, variance decreases

REINFORCE baselines

- Gradient direction ∇J stays the same
- Variance may change

Gradient variance: $\text{Var}[Q(s, a) - b(s)]$
as a *random variable over (s, a)*

$$\text{Var}[Q(s, a)] - 2 \cdot \text{Cov}[Q(s, a), b(s)] + \text{Var}[b(s)]$$



Q: can you suggest any such $b(s)$?

REINFORCE baselines

- Gradient direction ∇J stays the same
- Variance may change

Gradient variance: $\text{Var}[Q(s, a) - b(s)]$
as a *random variable over (s, a)*

$$\text{Var}[Q(s, a)] - 2 \cdot \text{Cov}[Q(s, a), b(s)] + \text{Var}[b(s)]$$

Naive baseline: $b = \text{moving average } Q$
over all (s, a) , $\text{Var}[b(s)] = 0,$ $\text{Cov}[Q, b] > 0$

REINFORCE baselines

Better baseline: $b(s) = V(s)$

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

Q: but how do we predict $V(s)$?

Actor-critic

- Learn both $V(s)$ and $\pi_\theta(a|s)$
- Hope for the 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!

Q: how can we estimate $A(s,a)$ 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 $V_\theta(s)$

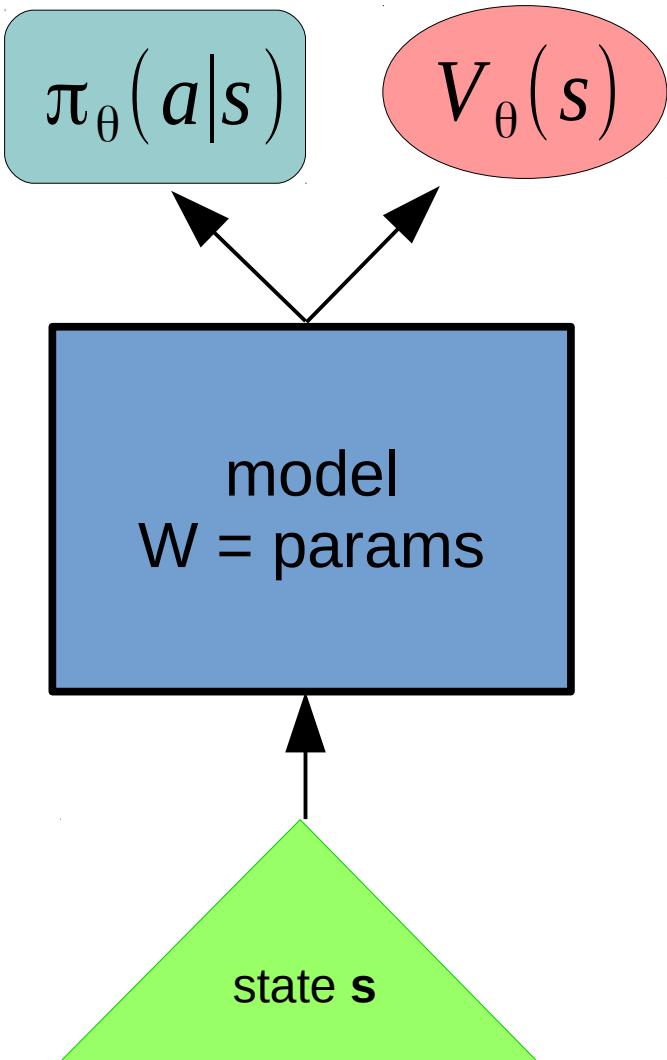
Use $V_\theta(s)$ to learn $\pi_\theta(a|s)$ faster!

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

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

consider
const

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?

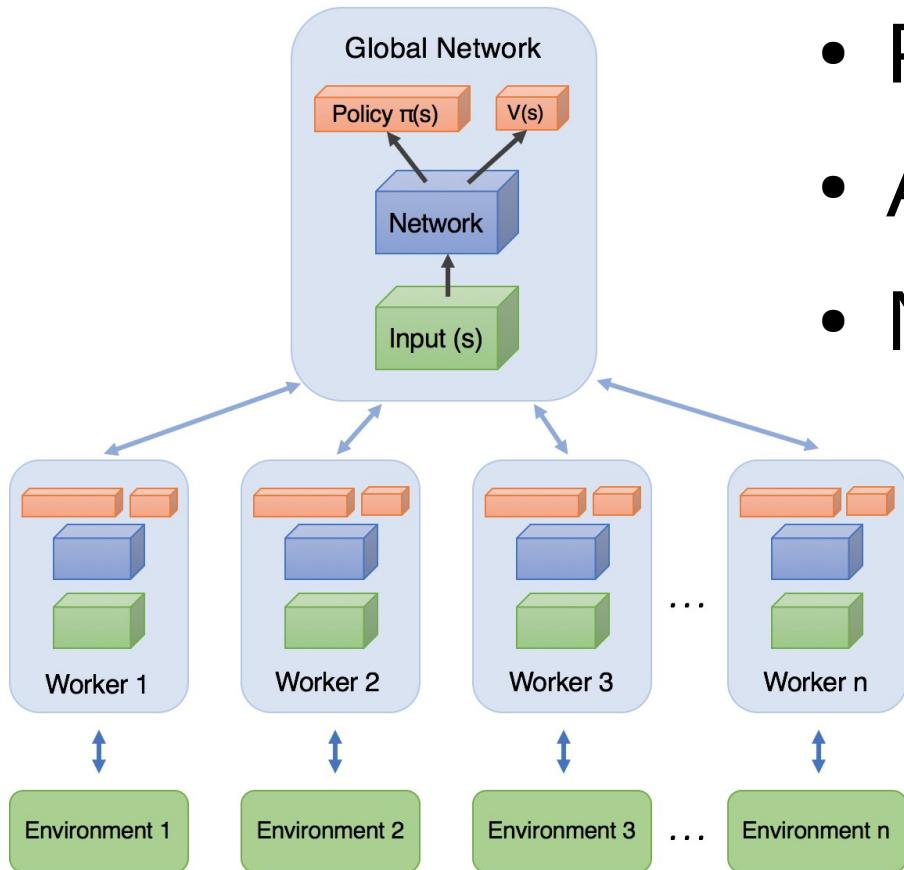
- Robot control: control motor voltage
 - Trading: assign money to equity

How does the algorithm change?

it doesn't :)

Just plug in a different formula for
 $\pi(a|s)$, e.g. normal distribution

Asynchronous advantage actor-critic

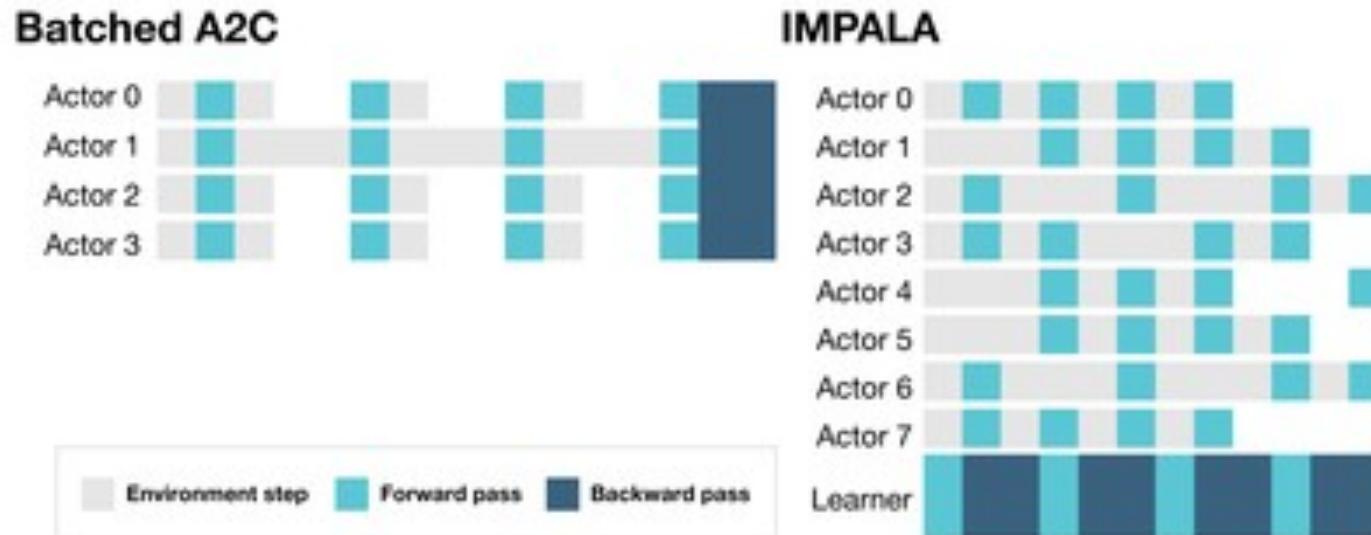


- Parallel game sessions
- Async multi-CPU training
- No experience replay
- LSTM policy
- N-step advantage
- No experience replay

Read more: <https://arxiv.org/abs/1602.01783>

IMPALA

- Massively parallel
- Separate actor / learner processes
 - Small experience replay w/ importance sampling



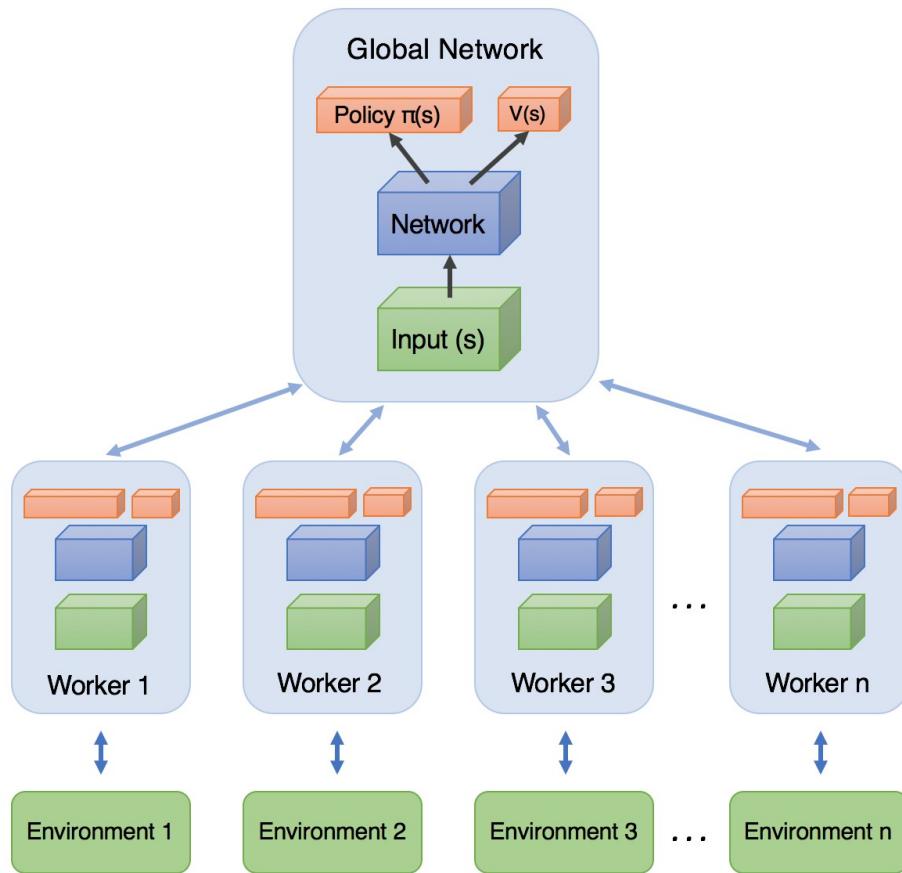
Read more: <https://arxiv.org/abs/1802.01561>

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



Asynchronous advantage actor-critic



- Remember log-derivative trick
- Combining best from both worlds is generally a good idea
- See [this](#) paper for the proof of the policy gradient for discounted rewards
- Time to write some code!