RL@Coursera
week 5

Policy gradient methods

Yandex
Data Factory

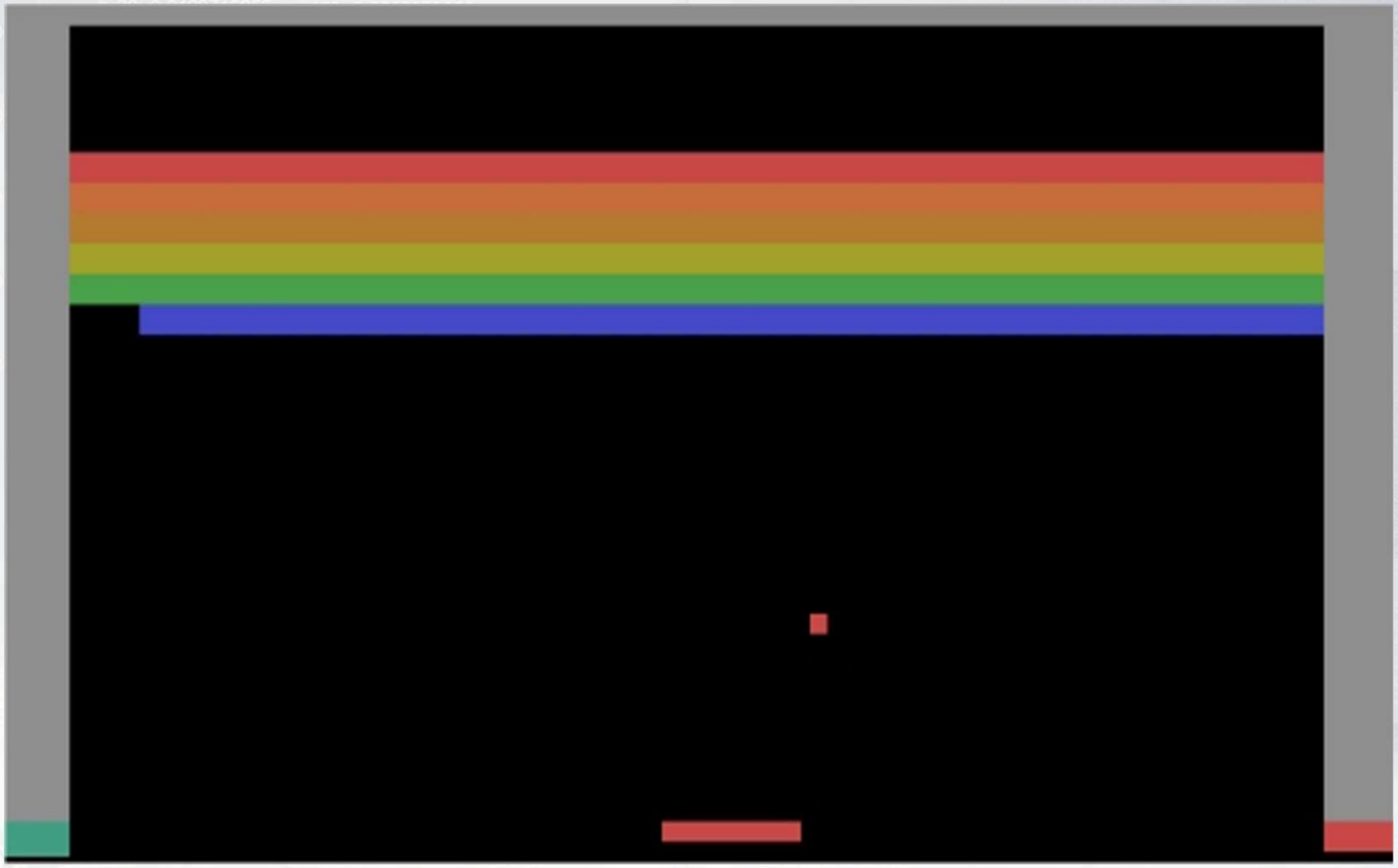
LAMBDA

Small experiment

The next slide contains a question

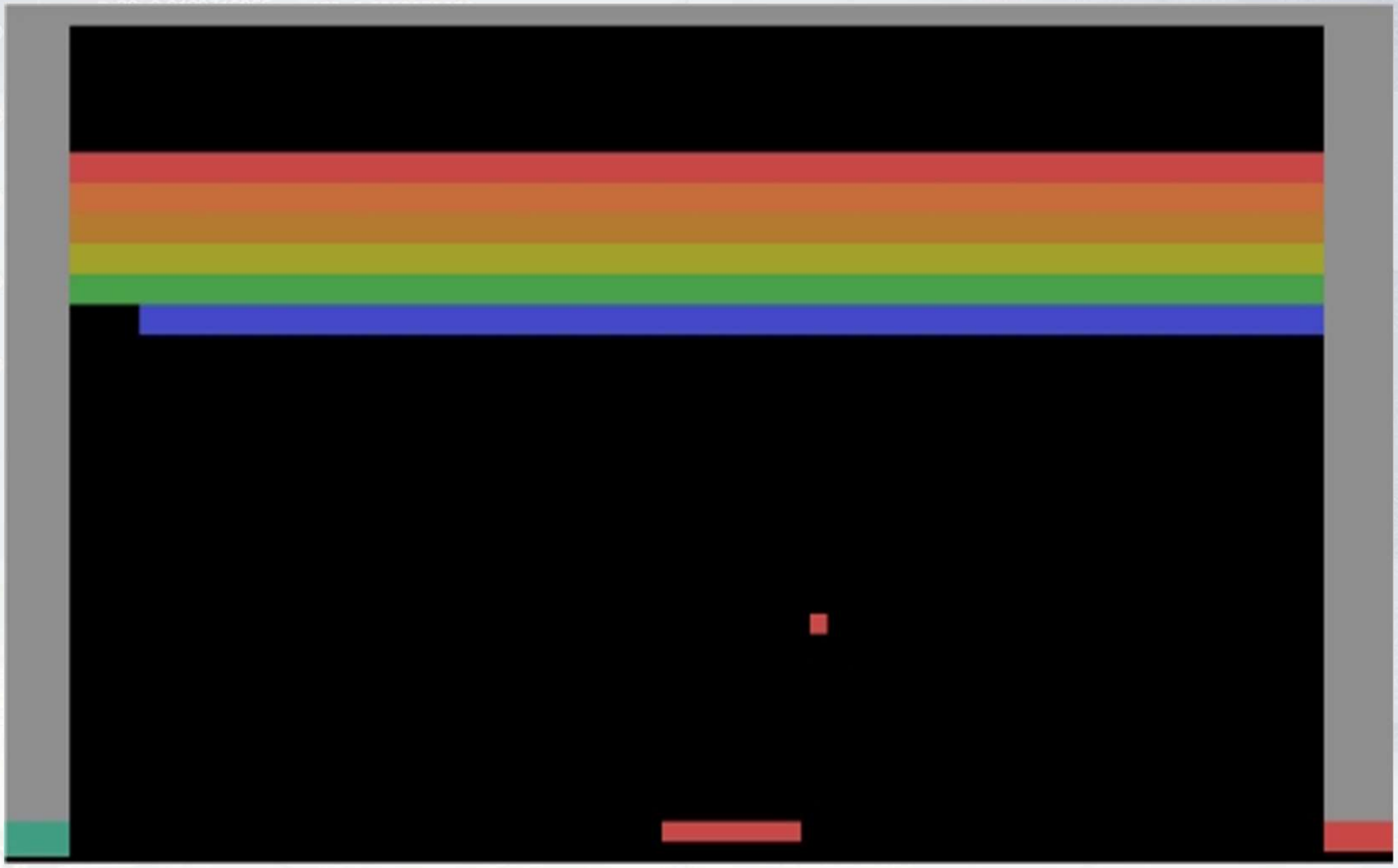
Please respond as fast as you can!

Small experiment



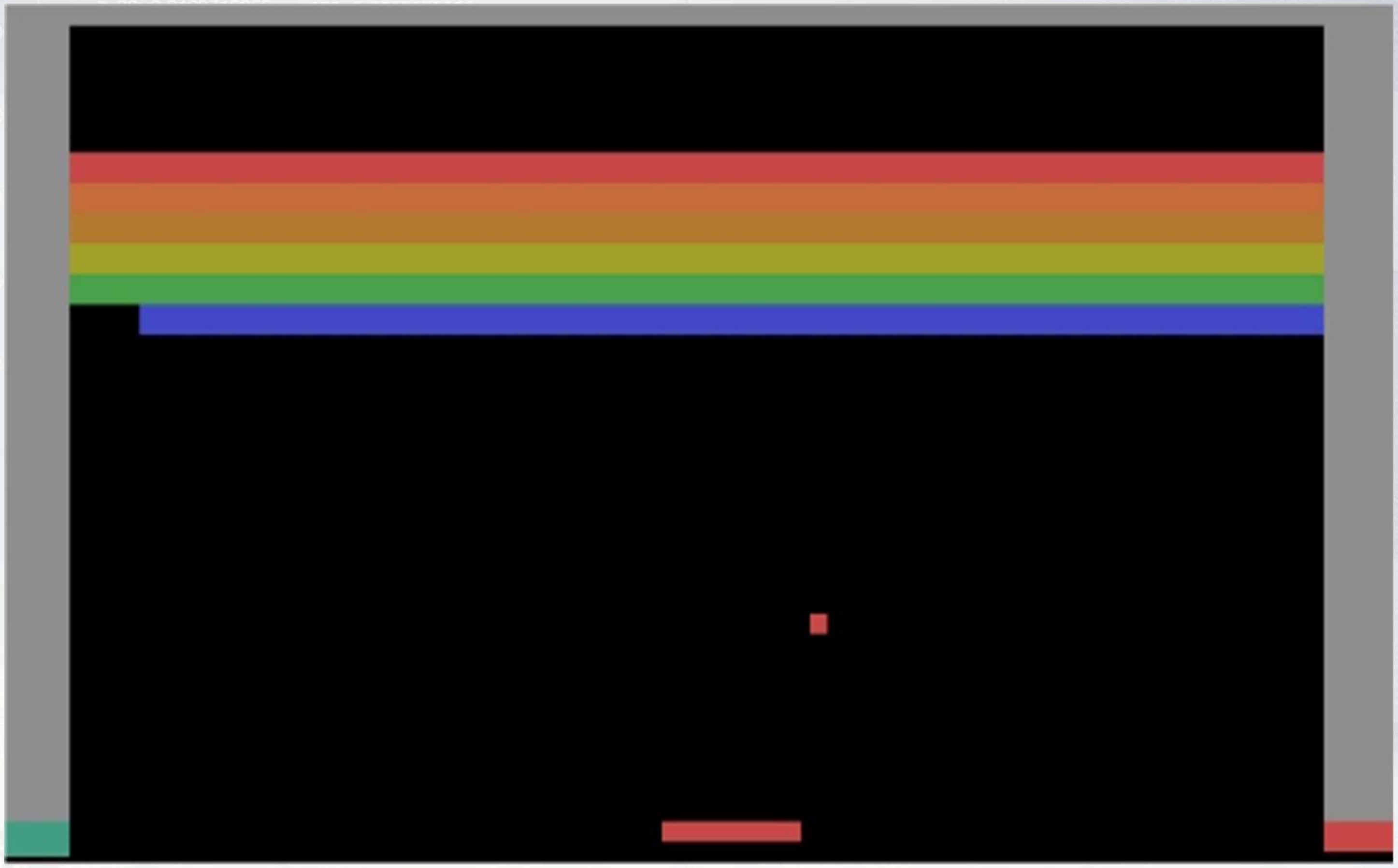
Left or Right?

Small experiment



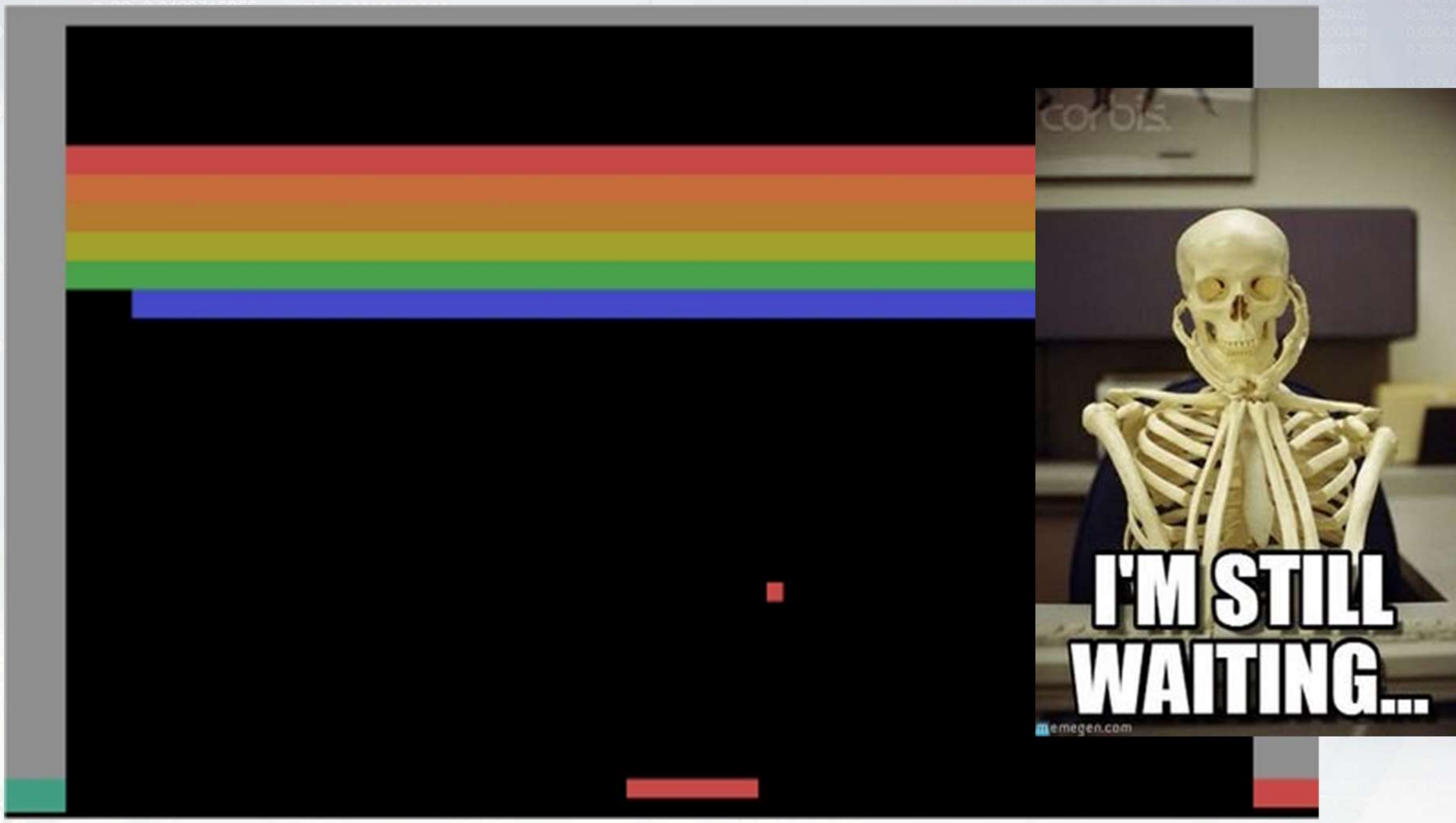
Right! Ready for next one?

Small experiment



What's Q(s,right) under gamma=0.99?

Small experiment



What's Q(s,right) under gamma=0.99?

Approximation error

$$L \approx \; E[Q(s_t,a_t) - \underline{(r_t + \gamma \cdot max_{a'})}]^2$$

Const, not dependent on Q_fn parameters

Assuming 2 nets

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

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

Approximation error

$$L \approx \ E[Q(s_t, a_t) - (r_t + \gamma \cdot max_{a'})]^2$$

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**

while A does not capture Q-values identically, picking $\arg\max_a Q(s,a)$ will still yield a^* everywhere

Approximation error

$$L \approx \; E[Q(s_t, a_t) - (r_t + \gamma \cdot max_{a'})]^2$$

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

not always representative

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

Q: what algorithm works that way?

Conclusion

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

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 $\pi_\theta(s|a)$

Q: what algorithm works that way?

e.g. crossentropy method

- learning prob of actions
- requires lots of sampling
- very efficient in complex env

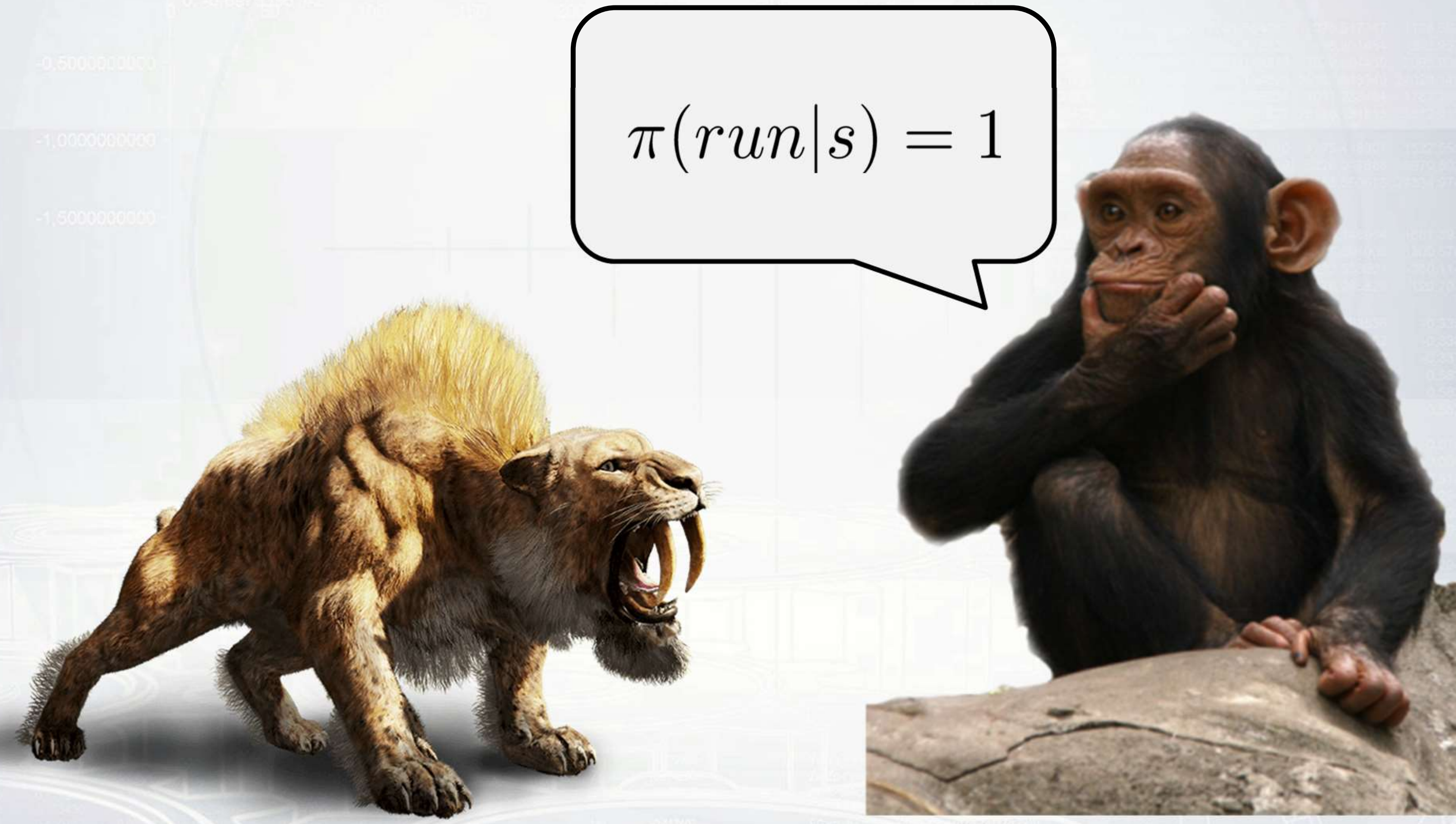
NOT how humans survived

Q-learning

Approach

argmax
[Q(s,pet the tiger)
Q(s,run from tiger)
Q(s,provoke tiger)
Q(s,ignore tiger)]

how humans survived



Policies

In general, two kinds

Deterministic policy

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

Stochastic policy

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

Policies

In general, two kinds

Deterministic policy

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

Stochastic policy

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

* learning prob dist

cross entropy method

Q: Any case where stochastic is better?

Policies

In general, two kinds

Deterministic policy

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

Stochastic policy

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

e.g. rock-paper-scissors

Q: Any case where stochastic is better?

* Having a determinstic policy makes it easier for an adversary to adapt

Policies

In general, two kinds

Deterministic policy

$$a = \pi_\theta(a|s)$$ ← same action each time

Stochastic policy

$$a \backsim \pi_\theta(a|s)$$ ← sampling takes care of exploration

*

With Q-learning, it switch bet random & optimal according to some prob!

Policies

In general, two kinds

Deterministic policy

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

same action each time

Stochastic policy

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

sampling takes care of exploration

Q: how to represent policy in continuous action space?

→ trying some distributions

Policies

In general, two kinds

Deterministic policy

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

same action each time

Stochastic policy

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

sampling takes care of exploration

categorical, normal, mixture of normal, whatever

Two approaches

Having suboptimal values yield suboptimal policy

- **Value based:**

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

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

- **Policy based:**

Explicitly learn policy $\pi_\theta(s,a)$ $\pi_\theta(s) \to a$

Implicitly maximize reward over policy

Recap: crossentropy method

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

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

Policy gradient main idea

Why so complicated?

We'd rather simply maximize G over pi!

Objective

- Expected reward:

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

- Expected discounted reward:

$$J = \mathop{E}_{\substack{s \backsim p(s) \\ a \backsim \pi_\theta(s|a)}} G(s, a)$$

Objective

- Expected reward:

$R(z)$ setting

"Undiscounted Version"

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

- Expected discounted reward:

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

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

Objective

- Consider an 1-step process for simplicity

$$J = \mathop{E}_{\substack{s \backsim p(s) \\ a \backsim \pi_\theta(s|a)}} R(s,a)$$

Objective

- Consider an 1-step process for simplicity

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

Objective

- Consider an 1-step process for simplicity

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

state visitation frequency
(may depend on policy)

naively agent will not affect,
a more complicated agent will
start to affect the state

Reward for
1-step session

what agent directly
influence
e.g NN prediction

Objective

- Consider an 1-step process for simplicity

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

state visitation frequency
(may depend on policy)

Reward for
1-step session

Q: how do we compute that?

Impractical !!

Objective

- Consider an 1-step process for simplicity

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

Reward for agent's action

SAMPLING using MC

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

sample N sessions by following $\pi_\theta(\mathbf{a}|\mathbf{s})$

Objective

- Consider an 1-step process for simplicity

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

Reward for agent's action

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

sample N sessions

Can we optimize policy now?

Objective

- Consider an 1-step process for simplicity

$$J = \mathop{E}_{\substack{s \backsim p(s) \\ a \backsim \pi_\theta(s|a)}} R(s,a) = \int\limits_s p(s) \int\limits_a \pi_\theta(a|s)\ R(s,a)\ da\ ds$$

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/dtheta

not shown when sampling

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

Optimization

problems with such approaches

- Finite differences
  - Change policy a little, evaluate

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

VERY noizy, especially if both Js are sampled

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

stochastic MDPs;
Throws a most sessions away

* We will use cross entropy with some tweaks;
to prevent it from favoring lucky outcomes

Objective

- Consider an 1-step process for simplicity

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

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

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

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

Policy gradient

Analytical inference

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

Doing so, we're approx J via sampling over s & a

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

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

Trivia: Anything curious about that formula?

Policy gradient

Analytical inference

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

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

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

that's expectation

:)

Discounted reward case

To generalize for more than 1-step

Replace R with Q :)

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

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

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

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

that's expectation :)

Policy gradient (REINFORCE)

- Policy gradient

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

- Approximate with sampling [MC version]

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

REINFORCE algorithm

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

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

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

* G obtained from discounted rewards

somehow rep Q-fn

REINFORCE algorithm

- Initialize NN weights $\theta_0 \leftarrow random$

Q: is it on- or off-policy? Q-learning SARSA

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

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

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

REINFORCE algorithm

- Initialize NN weights $\theta_0 \leftarrow random$

actions under current policy

= on-policy

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

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

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

✗ We sample traj under current policy

REINFORCE algorithm

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

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

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

What is better for learning:

random action in good state or

great action in bad state?

REINFORCE baseline

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

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

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

Advantage

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

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

REINFORCE baseline

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

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

baseline, depends on s not a

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

Anything that doesn't depend on action

ideally, $b(s) = V(s)$

* optimal policy will not change doing so

Actor-critic

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

image: youtube: mr tivikov

Advantage actor-critic

- Idea: learn both $\pi_\theta(a|s)$ & $V_\theta(s)$
- Use $V_\theta(s)$ to learn $\pi_\theta(a|s)$ faster!

Non-**trivia:** 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!

Non-**trivia:** 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!

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

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

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

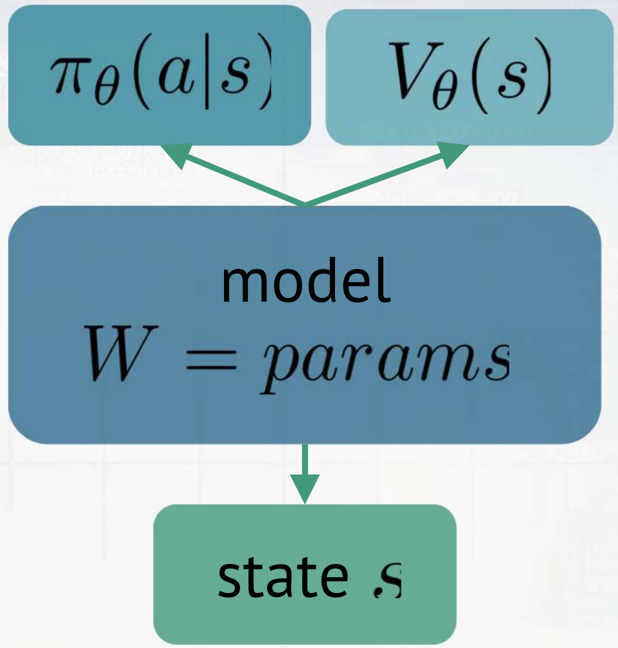
consider const

Trivia: How do we train V then?

Advantage actor-critic

consider const

$\min \mathbb{E}\left[r + \gamma V_\pi(s') - V_\pi(s)\right]^2$



picking action

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

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

est how good particular state

→ improving convergence as well

value-based Vs policy-based

Value-based	Policy-based + Actor-critic
Q-learning, SARSA,value-iteration	REINFORCE, Advantage Actor-Critic, Crossentropy Method

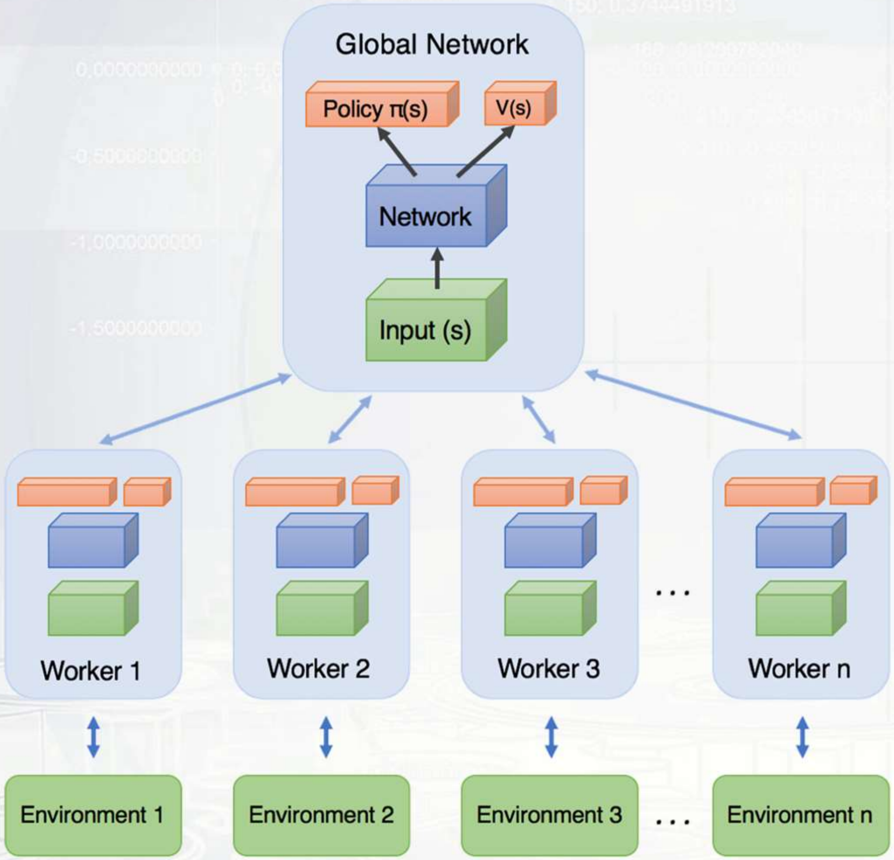
value-based Vs policy-based

Value-based	Policy-based + Actor-critic
Q-learning, SARSA,value-iteration	REINFORCE, Advantage Actor-Critic, Crossentropy Method

Your guess?

Casy Study: A3C

Async Actor Critic

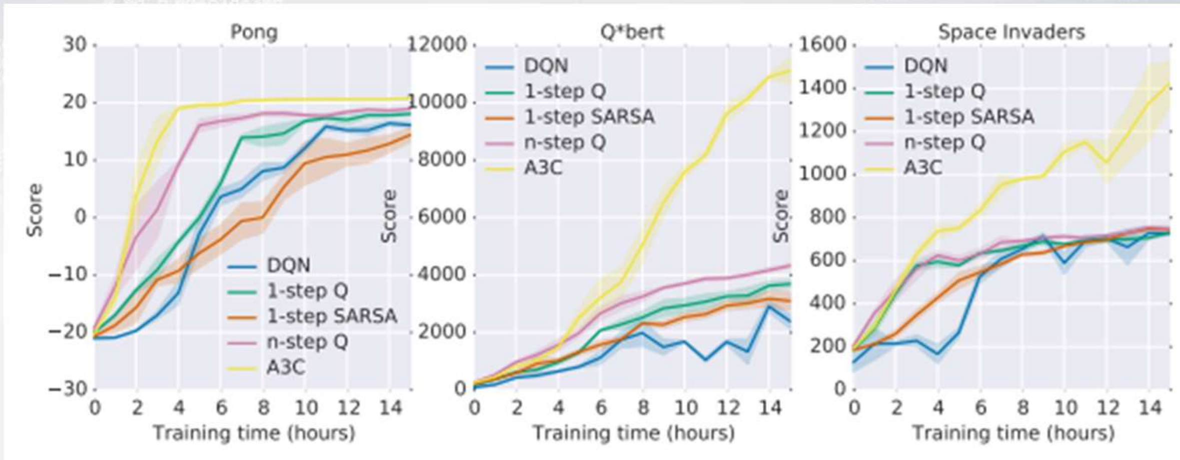


A popular implementation of advantage actor-critic using neural net agent

- No experience replay
- Many parallel sessions
- Asynchronous updates

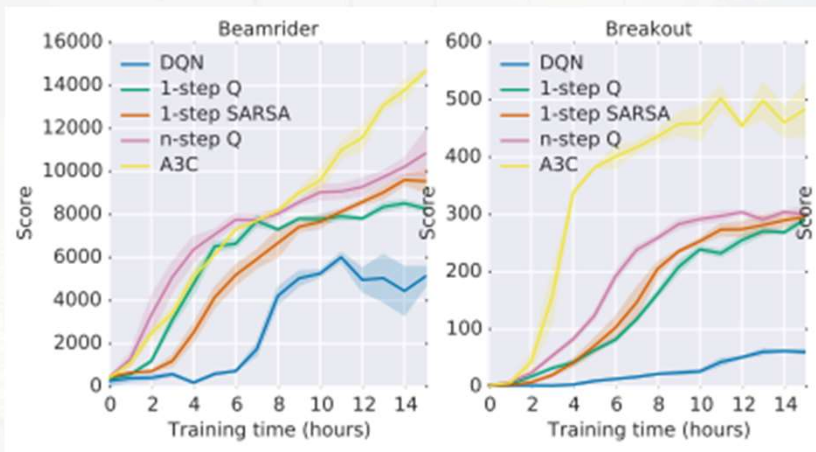
image: http://bit.ly/2z0kBNu

Casy Study: A3C results



A3C + LSTM

at least several seq transitions $[s, a, r, s', a', \ldots, s_K]$ will be required to effectively train RNN inside the agent

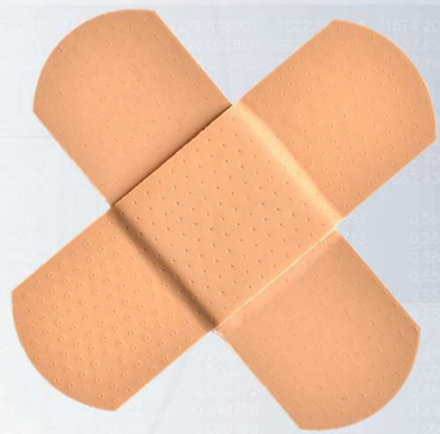


Duct tape zone

LOSS
policy-based ; policy grad
TD, value-based (less important)

- 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

value-based Vs policy-based

learning value

learning policy

Value-based	Policy-based + Actor-critic
Q-learning, SARSA, value-iteration	REINFORCE, Advantage Actor-Critic, Crossentropy Method
Solves harder problem	Solves easier problem
* Explicit exploration	Innate exploration & stochasticity
Evaluates states & actions	* Easier for for continuous actions
Easier to train off-policy ;	* Compatible with supervised learning

increasing sample efficiency

on-policy

Supervised pre-training

Reinforcement learning usually takes longto find optimal policy completely from scratch.

We can use existing knowledge to help it!

- Human experience
- Known heuristic
- Previous system

Supervised pre-training

- Supervised learning:

$$\nabla llh = \underset{x, y_{opt} \backsim D}{E} \nabla \log P_\theta(y_{opt}|x)$$

- Policy gradient:

$$\nabla J = \underset{\substack{s \backsim d(s) \\ a \backsim \pi(a|obs(s))}}{E} \nabla \log \pi_(a|s) \cdot Q(s,a)$$

Supervised pre-training

- Supervised learning:

initializing from some prior

$$\nabla llh = \underset{s, a_{opt} \backsim D}{E} \nabla \log \pi_\theta(a_{opt}|s)$$

- Policy gradient:

improving

$$\nabla J = \underset{\substack{s \backsim d(s) \\ a \backsim \pi(a|obs(s))}}{E} \nabla \log \pi_(a|s) \cdot Q(s,a)$$

Supervised pre-training

- Supervised learning:

$$\nabla llh = \underset{s, a_{opt} \backsim D}{E} \nabla \log \pi_\theta(a_{opt}|s)$$

- Policy gradient:

$$\nabla J = \underset{\substack{s \backsim d(s) \\ a \backsim \pi(a|obs(s))}}{E} \nabla \log \pi_(a|s) \cdot Q(s,a)$$

Q: what's different? (apart from $Q(s,a)$
)

Supervised pre-training

- Supervised learning:

reference

$$\nabla llh = \mathop{E}_{s,a_{opt} \backsim D} \nabla \log \pi_\theta(a_{opt}|s)$$

- Policy gradient:

$$\nabla J = \mathop{E}_{\substack{s \backsim d(s) \\ a \backsim \pi(a|obs(s))}} \nabla \log \pi_(a|s) \cdot Q(s,a)$$

generated

More out there

This domain is huge!

- Trust Region Policy optimization
  - policy gradient on steroids
- Deterministic policy gradients
  - Off-policy & less variance
- Many many more!
  - Research happens as you read this line

narrowing the policy space

Let's code!