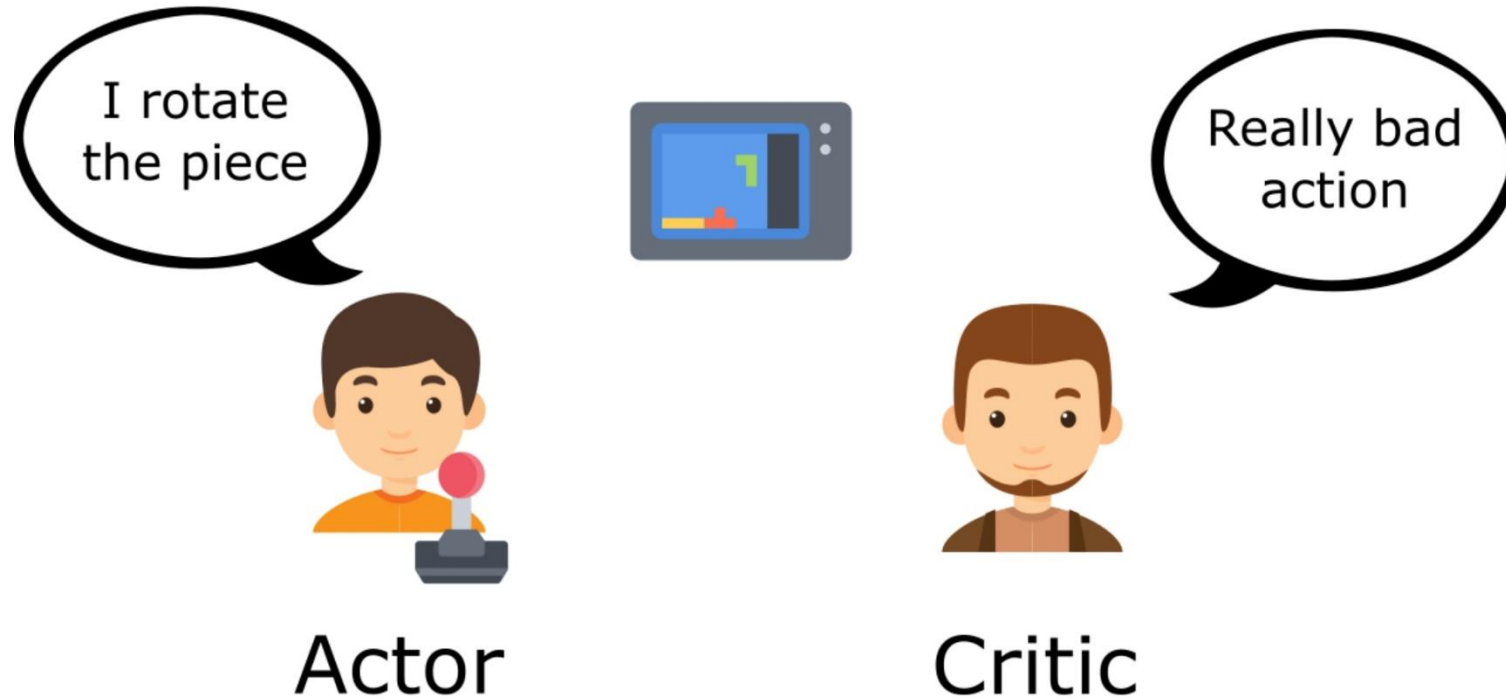


Actor Critic and Proximal Policy Optimization



Instructor: Daniel Brown --- University of Utah

Announcement

- Mid semester feedback. Thank you!
- What y'all like?
 - Exploratory assignments, interesting topics, no exams 😊
 - Experience-based learning
 - Paper reading
- What y'all want to see changed?
 - Zoom options if you are sick.
 - Quizzes: more structure, no paper passing, more frequent, eliminate...
 - End a minute or two early.
 - More reading assignments
 - Record lectures...
 - Less math!
 - More consistency in math.
 - More math!
 - Harder/deeper programming assignments

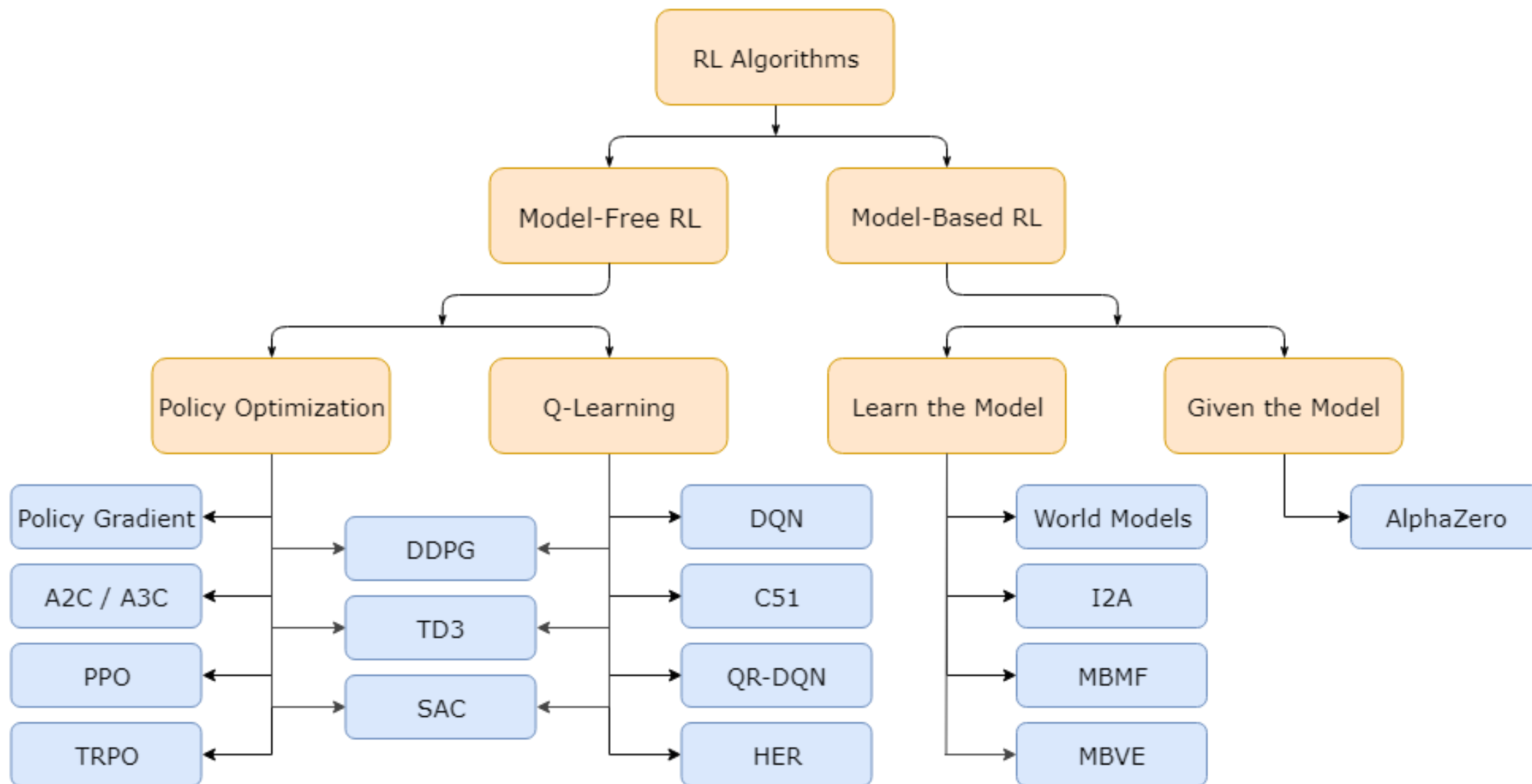
Announcement

- What helps y'all learn?
 - Quizzes, In-class lectures, Recordings, Programming assignments to practice concepts
- What can I do to improve learning?
 - More paper reading assignments, examples of applications.
 - Add subtitles to zoom recordings
 - More interactions and Q&A in lectures
 - Move at a faster pace
 - More structure in homework questions.
 - Harder homework problems
 - Post lecture slides earlier.
 - Discuss pseudo code in lectures
 - More math!
 - Example project ideas.

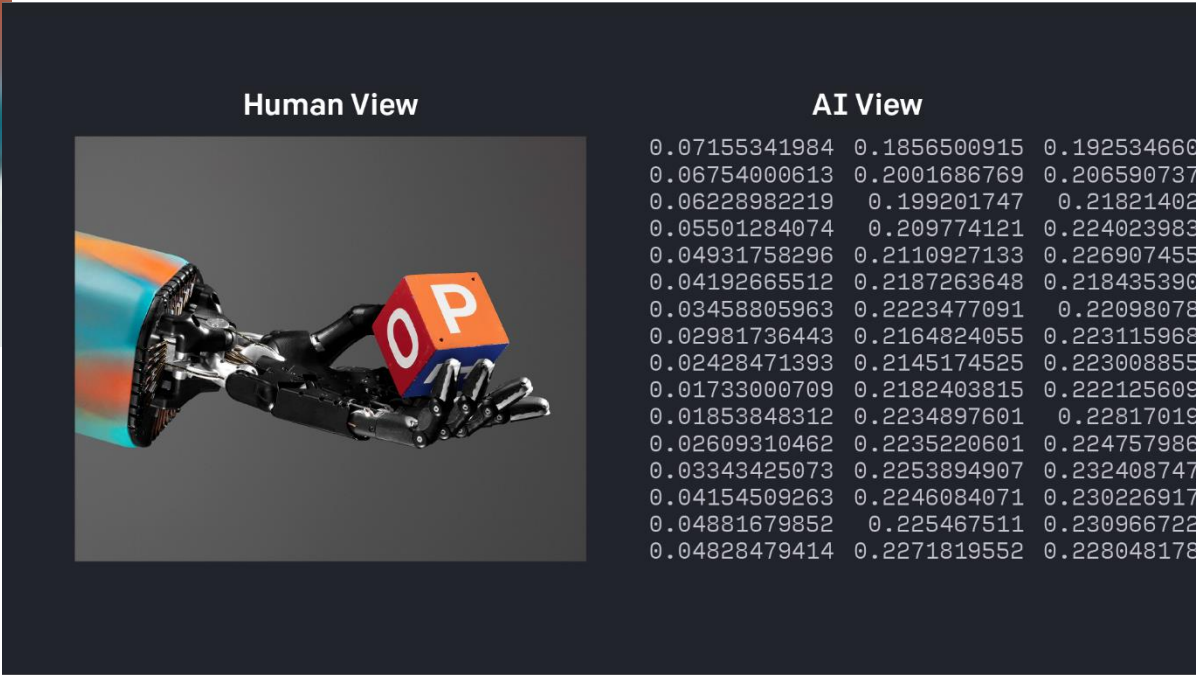
Announcement

- Homework 5

Rough Taxonomy of RL Algorithms



Dexterous Manipulation



OpenAI 5: DOTA 2



Human View



AI View

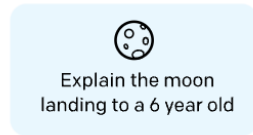
3.006	-1.386	-0.4695	0.883	1	0.84
-0.3154	-0.5425	-0.5	0.866	0	0.82
3.11	-1.36	-0.9336	0.3584	1	0.78
-2.324	2.863	0.9746	0.225	0	0.86
3.037	-1.361	-0.7773	0.6294	1	0.82
-1.387	2.951	0.988	0.1565	0	0.74
3.023	-0.9395	0.05234	-0.9985	0	0.66
2.951	-0.5747	0.01746	1	0	0.72
2.963	-1.303	0.3906	0.9204	0	0.68
2.834	-3.164	0.01746	-1	0	0.68
3.127	-1.368	0.6562	0.755	1	0.55
3.088	-1.366	0.4695	0.883	0	0.55
2.984	-1.398	-0.225	0.9746	1	0.55
3.037	-1.391	0.788	0.6157	0	0.55
3.076	-1.438	0.883	0.4695	0	0.55
-2.412	2.846	0.996	0.08716	1	0.3

RLHF in ChatGPT

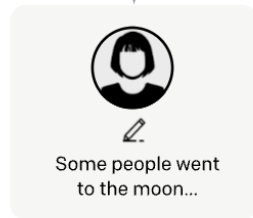
Step 1

Collect demonstration data, and train a supervised policy.

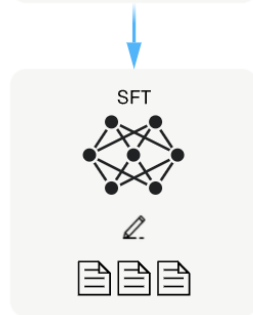
A prompt is sampled from our prompt dataset.



A labeler demonstrates the desired output behavior.



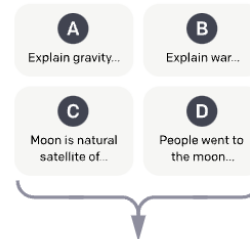
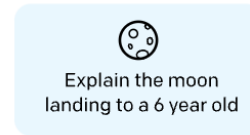
This data is used to fine-tune GPT-3 with supervised learning.



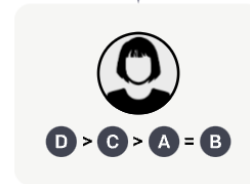
Step 2

Collect comparison data, and train a reward model.

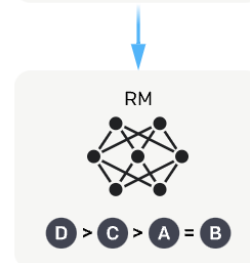
A prompt and several model outputs are sampled.



A labeler ranks the outputs from best to worst.



This data is used to train our reward model.



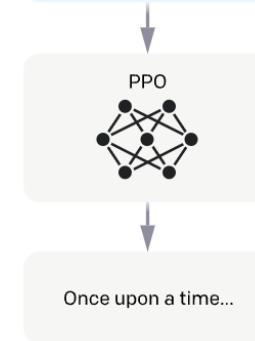
Step 3

Optimize a policy against the reward model using reinforcement learning.

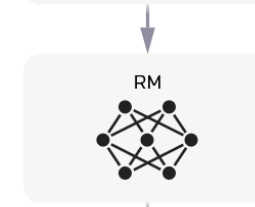
A new prompt is sampled from the dataset.



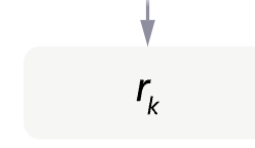
The policy generates an output.



The reward model calculates a reward for the output.



The reward is used to update the policy using PPO.



What is the goal of RL?

- Find a policy that maximizes expected utility (discounted cumulative rewards)

$$\pi^* = \underset{\pi}{\operatorname{argmax}} E_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t R(s, \pi(s), s') \right]$$

The Policy Gradient (REINFORCE)

- We can now perform gradient ascent to improve our policy!

$$\theta_{k+1} \leftarrow \theta_k + \alpha \nabla_{\theta} J(\pi_{\theta}) \Big|_{\theta_k}$$

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

Estimate with a
sample mean over a
set D of policy rollouts
given current
parameters

$$\approx \frac{1}{|D|} \sum_{\tau \in D} \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) R(\tau)$$

Simple Pytorch Pseudocode

```
for episode in range(num_episodes):
    state = env.reset()
    trajectory = []

    while True:
        action, log_prob = select_action(policy_net, state)
        next_state, reward, done, _ = env.step(action)

        trajectory.append((log_prob, reward))
        state = next_state

    if done:
        break
```

```
# Compute returns and policy loss
log_probs, rewards = zip(*trajectory)
returns = compute_returns(rewards, gamma)
policy_loss = -sum(log_prob * G
                    for log_prob, G in zip(log_probs, returns))

# Update policy network
optimizer.zero_grad()
policy_loss.backward()
optimizer.step()
```

Policy Gradient RL Algorithms

- We can directly update the policy to achieve high reward.
- Pros:
 - Directly optimize what we care about: Utility!
 - Naturally handles continuous action spaces!
 - Can learn specific probabilities for taking actions.
 - Often more stable than value-based methods (e.g. DQN).
- Cons:
 - On-Policy -> Sample-inefficient we need to collect a large set of new trajectories every time the policy parameters change.
 - Q-Learning methods are usually more data efficient since they can reuse data from any policy (Off-Policy) and can update per sample.

Many forms of policy gradients

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

What we derived: $\Phi_t = R(\tau),$

Follows a similar
derivation:

$$\Phi_t = \sum_{t'=t}^T R(s_{t'}, a_{t'}, s_{t'+1}),$$

<https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63>

- What is better about the second approach?
 - Focuses on rewards in the future!
 - Less variance -> less noisy gradients.

Many forms of policy gradients

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

$$\Phi_t = \sum_{t'=t}^T R(s_{t'}, a_{t'}, s_{t'+1}),$$

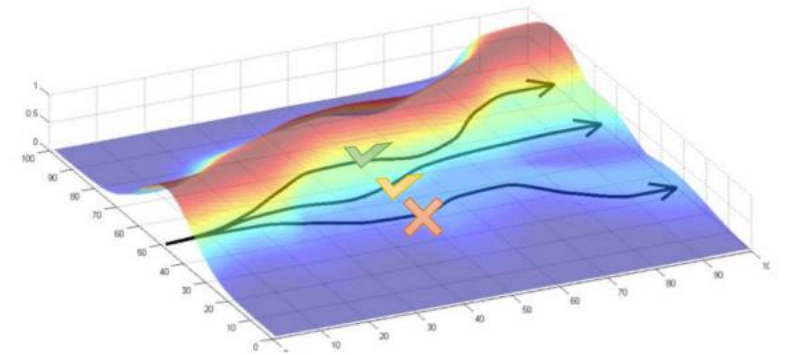
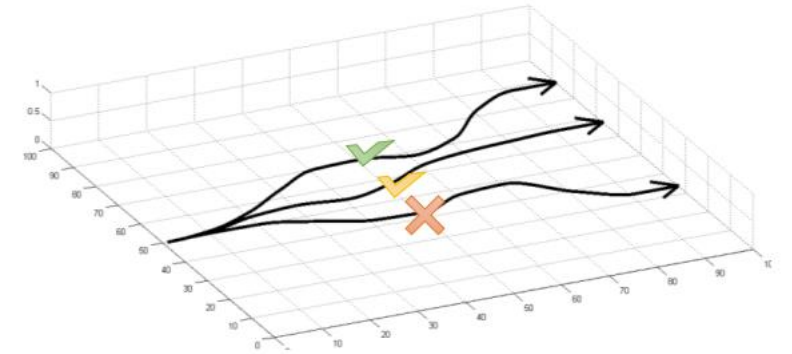
Looks familiar....

$$\Phi_t = Q^{\pi_{\theta}}(s_t, a_t)$$

- Now we have an approach that combines a parameterized policy and a parameterized value function!

Baselines

$$\nabla_{\theta} J(\pi_{\theta}) = E_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) R(\tau) \right]$$
$$\approx \frac{1}{|D|} \sum_{\tau \in D} \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) R(\tau)$$



Baselines

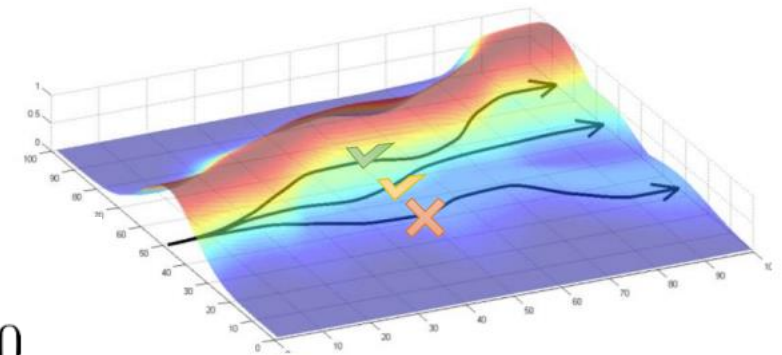
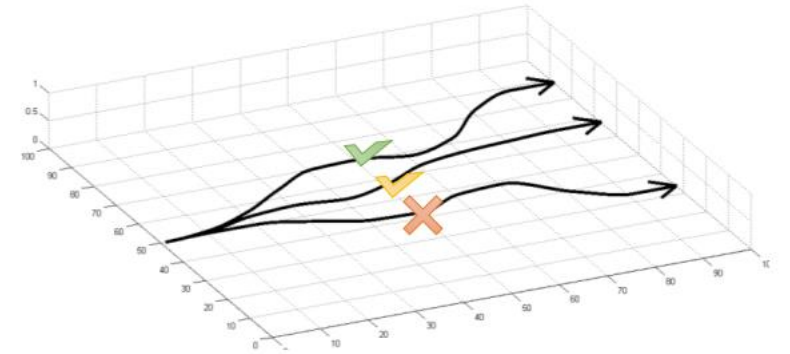
$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \nabla_{\theta} \log p_{\theta}(\tau) [r(\tau) - b]$$

$$b = \frac{1}{N} \sum_{i=1}^N r(\tau)$$

But can we do this?

$$E[\nabla_{\theta} \log p_{\theta}(\tau) b] = \int p_{\theta}(\tau) \nabla_{\theta} \log p_{\theta}(\tau) b d\tau$$

$$= \int \nabla_{\theta} p_{\theta}(\tau) b d\tau = b \nabla_{\theta} \int p_{\theta}(\tau) d\tau = b \nabla_{\theta} 1 = 0$$



Many forms of policy gradients

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

$$\Phi_t = R(\tau), \quad \Phi_t = \sum_{t'=t}^T R(s_{t'}, a_{t'}, s_{t'+1}), \quad \Phi_t = Q^{\pi_{\theta}}(s_t, a_t)$$

$$\Phi_t = \sum_{t'=t}^T R(s_{t'}, a_{t'}, s_{t'+1}) - b(s_t)$$

$$\Phi_t = A^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$$

Advantage Function

I rotate
the piece



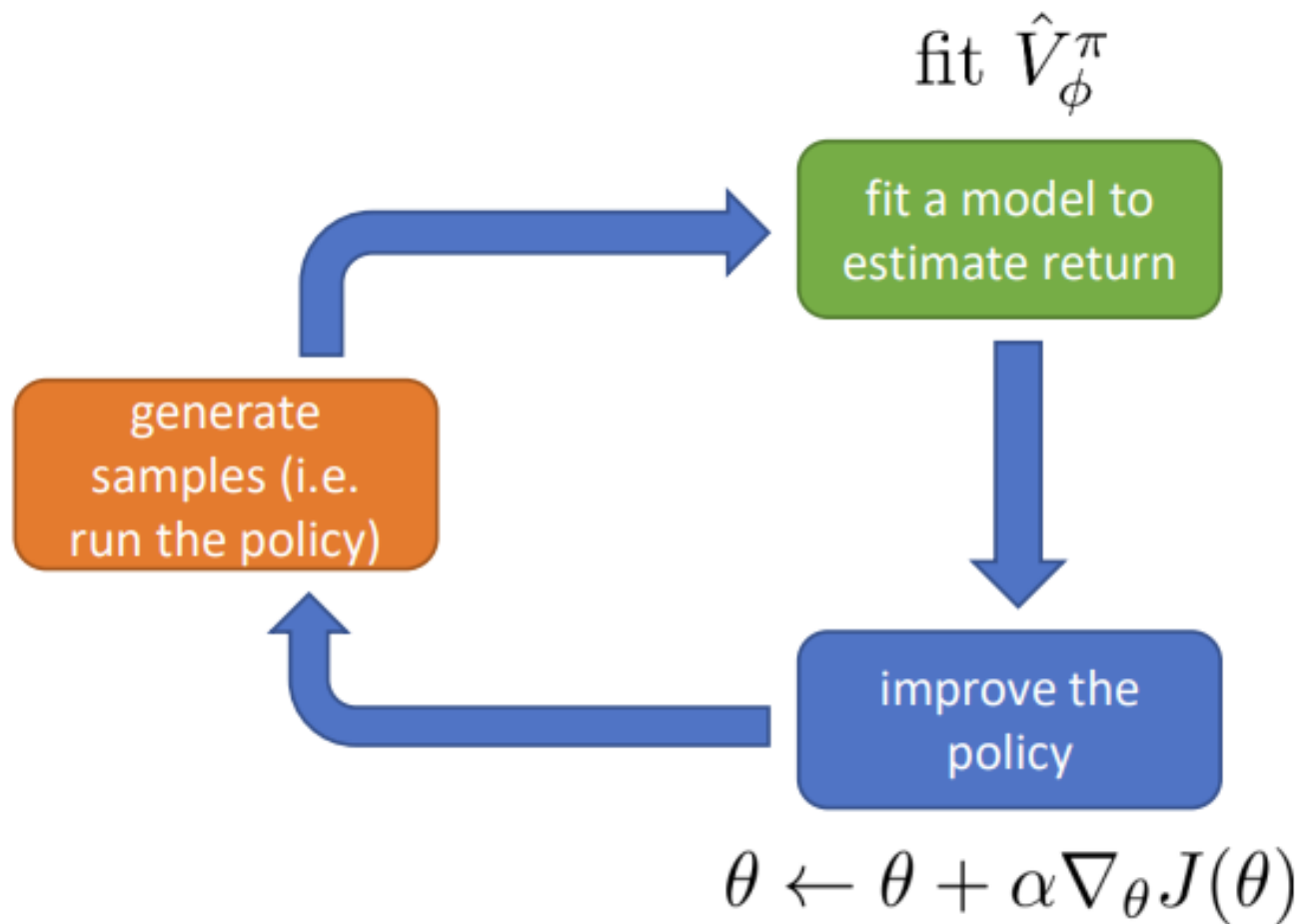
Really bad
action



Actor

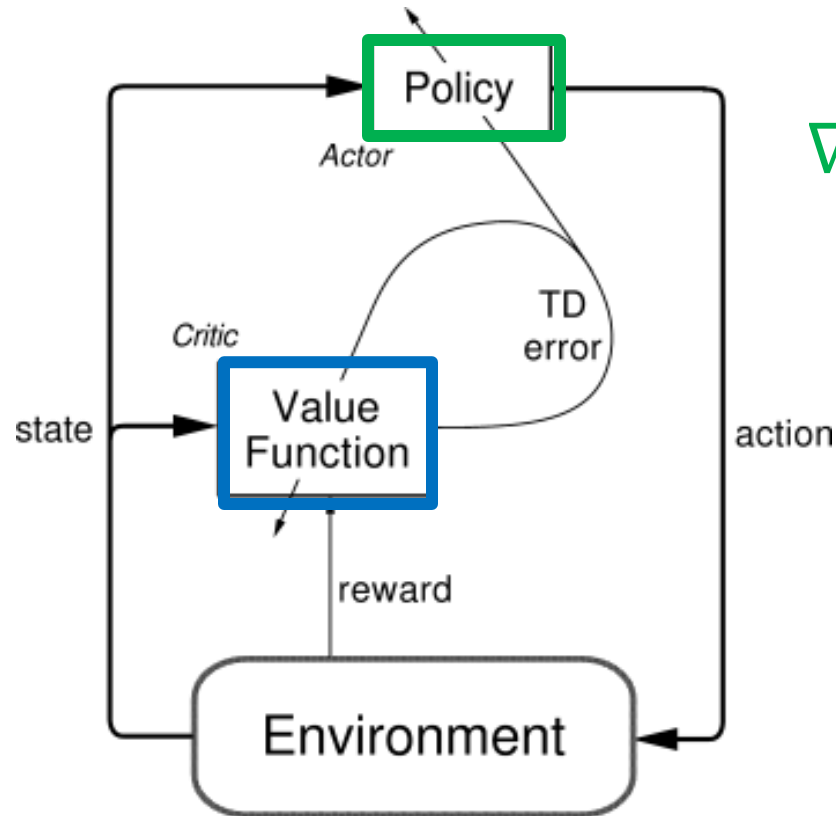


Critic



Actor Critic Algorithms

- Combining value learning with direct policy learning
 - One example is policy gradient using the advantage function



$$\nabla_{\theta} J(\pi_{\theta}) = E_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) Q_w^{\pi_{\theta}}(s_t, a_t) \right]$$

$$\theta_{k+1} \leftarrow \theta_k + \alpha \nabla_{\theta} J(\pi_{\theta}) \Big|_{\theta_k}$$

$$\delta = (r_t + \gamma Q_w^{\pi_{\theta}}(s_{t+1}, a_{t+1}) - Q_w^{\pi_{\theta}}(s_t, a_t))$$

$$w_{k+1} \leftarrow w_k + \alpha \delta_t \nabla_{\theta} Q_w^{\pi_{\theta}}$$

Q Actor Critic Algorithm Pseudo Code

Algorithm 1 Q Actor Critic

Initialize parameters s, θ, w and learning rates α_θ, α_w ; sample $a \sim \pi_\theta(a|s)$.

for $t = 1 \dots T$: **do**

 Sample reward $r_t \sim R(s, a)$ and next state $s' \sim P(s'|s, a)$

 Then sample the next action $a' \sim \pi_\theta(a'|s')$

 Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \log \pi_\theta(a|s)$; Compute the correction (TD error) for action-value at time t:

$$\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$$

 and use it to update the parameters of Q function:

$$w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)$$

 Move to $a \leftarrow a'$ and $s \leftarrow s'$

end for

Adapted from Lilian Weng's post "Policy Gradient algorithms"

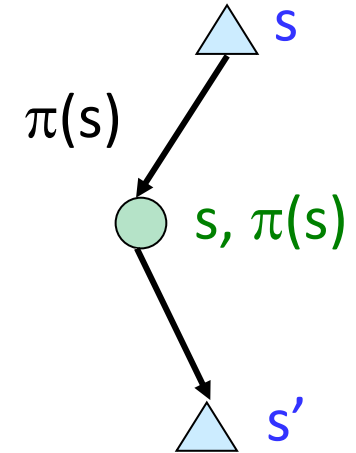
The Advantage Function

$$A(s, a) = \underbrace{Q(s, a)}_{\text{q value for action a in state s}} - \underbrace{V(s)}_{\text{average value of that state}}$$

- Benefits?
- Downsides?

Temporal Difference Learning

- Big idea: learn from every experience!
 - Update $V(s)$ each time we experience a transition (s, a, s', r)
 - Likely outcomes s' will contribute updates more often
- Temporal difference learning of values
 - Policy still fixed, still doing evaluation!
 - Move values toward value of whatever successor occurs: running average



Sample of $V(s)$: $sample = R(s, \pi(s), s') + \gamma V^\pi(s')$

Update to $V(s)$: $V^\pi(s) \leftarrow V^\pi(s) + \alpha(sample - V^\pi(s))$

The Advantage Function

$$A(s, a) = \boxed{Q(s, a)} - V(s)$$

$$\frac{r + \gamma V(s')}{\quad}$$

$$A(s, a) = \underline{r + \gamma V(s') - V(s)}$$

TD Error

Advantage Actor Critic (A2C)

- Combining value learning with direct policy learning



Actor

Policy gradient update

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

$$\begin{aligned} \Phi_t &= A^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) - V^{\pi}(s_t) \\ &\approx r(s_t, a_t) + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t) \end{aligned}$$

Critic

TD-Learning update



$$\delta_t = r(s_t, a_t) + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$$

$$\text{Value} = V^{\pi}(s_t)$$

$$\text{Target} = r(s_t, a_t) + \gamma V^{\pi}(s_{t+1})$$

$$w_{k+1} \leftarrow w_k + \alpha \text{MSE}(\text{value}, \text{target})$$

Asynchronous Advantage Actor Critic (A3C)

Asynchronous Methods for Deep Reinforcement Learning

Volodymyr Mnih¹

Adrià Puigdomènech Badia¹

Mehdi Mirza^{1,2}

Alex Graves¹

Tim Harley¹

Timothy P. Lillicrap¹

David Silver¹

Koray Kavukcuoglu¹

VMNIH@GOOGLE.COM

ADRIAP@GOOGLE.COM

MIRZAMOM@IRO.UMONTREAL.CA

GRAVESA@GOOGLE.COM

THARLEY@GOOGLE.COM

COUNTZERO@GOOGLE.COM

DAVIDSILVER@GOOGLE.COM

KORAYK@GOOGLE.COM

¹ Google DeepMind

² Montreal Institute for Learning Algorithms (MILA), University of Montreal

Asynchronous Advantage Actor Critic (A3C)

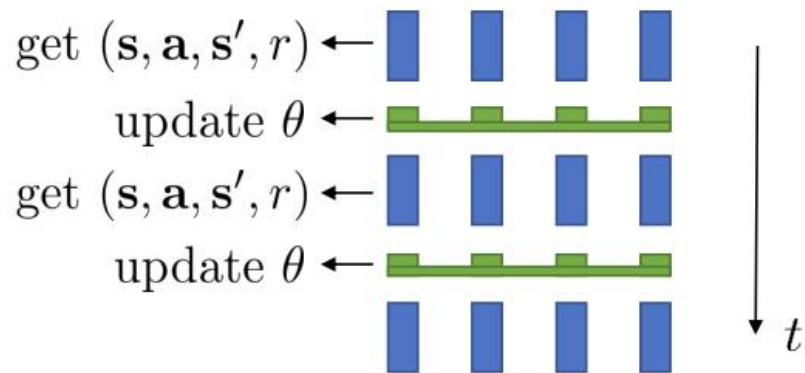
- Adds a few tricks
 1. Multiple parallel workers to collect rollouts in different copies of the same env and update the global policy and value models asynchronously
 2. n-step returns
 3. Entropy regularization
 4. Share neural network weights for actor and critic

Parallel actors

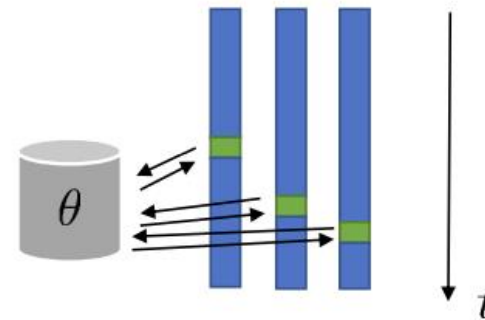
online actor-critic algorithm:

1. take action $\mathbf{a} \sim \pi_{\theta}(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
2. update \hat{V}_{ϕ}^{π} using target $r + \gamma \hat{V}_{\phi}^{\pi}(\mathbf{s}')$ ← works best with a batch (e.g., parallel workers)
3. evaluate $\hat{A}^{\pi}(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \hat{V}_{\phi}^{\pi}(\mathbf{s}') - \hat{V}_{\phi}^{\pi}(\mathbf{s})$
4. $\nabla_{\theta} J(\theta) \approx \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}|\mathbf{s}) \hat{A}^{\pi}(\mathbf{s}, \mathbf{a})$
5. $\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$

synchronized parallel actor-critic



asynchronous parallel actor-critic



N-Step Returns

- At convergence we want $V^\pi(s_t) = E_\pi[r_t + \gamma V^\pi(s_{t+1})]$
- So given experience (s_t, a_t, r_t, s_{t+1}) , TD methods push $V^\pi(s_t)$ towards $r_t + \gamma V^\pi(s_{t+1})$
- But why only look one step ahead? [1-step return]
- In practice we have experience that looks like this

$$(s_0, a_0, r_0, s_1, s_2, a_2, r_2, s_3, \dots, s_t, a_t, r_t, s_{t+1}, \dots)$$

What if we pushed $V^\pi(s_t)$ towards $r_t + \gamma r_{t+1} + \gamma^2 V^\pi(s_{t+2})$?

Or even pushed $V^\pi(s_t)$ towards $r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 V^\pi(s_{t+3})$?

We can generalize this idea to use n-step returns!

N-Step Returns for A3C updates

Given $(s_0, a_0, r_0, s_1, s_2, a_2, r_2, s_3, \dots, s_t, a_t, r_t, s_{t+1}, \dots, r_{T-1}, s_T)$

Compute advantage for each state. If s_T is a terminal state, then define $V_w^\pi(s_T)=0$

$$A(s_t, a_t) = \sum_{i=0}^{T-t-1} \gamma^i r_{t+i} + \gamma^{T-t} V_w^\pi(s_T) - V_w^\pi(s_t)$$

Accumulate gradients for each state and update policy using policy gradient

$$\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) A_w(s_t, a_t)$$

Update Value function based on TD-error using MSE loss

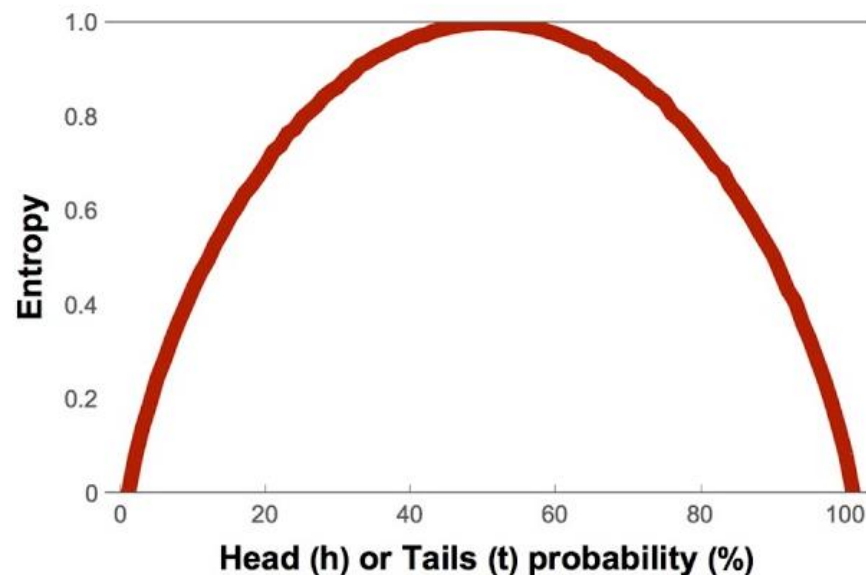
$$\nabla_w \sum_{t=0}^{T-1} \left(\sum_{i=0}^{T-t-1} \gamma^i r_{t+i} + \gamma^{T-t} V_w^\pi(s_T) - V_w^\pi(s_t) \right)^2$$

Shannon Entropy

- Average level of uncertainty associated with a random variable's possible outcomes.

$$H(X) := - \sum_{x \in \mathcal{X}} p(x) \log p(x)$$

$$P(X = heads) = \frac{1}{2} \qquad P(X = tails) = \frac{1}{2}$$

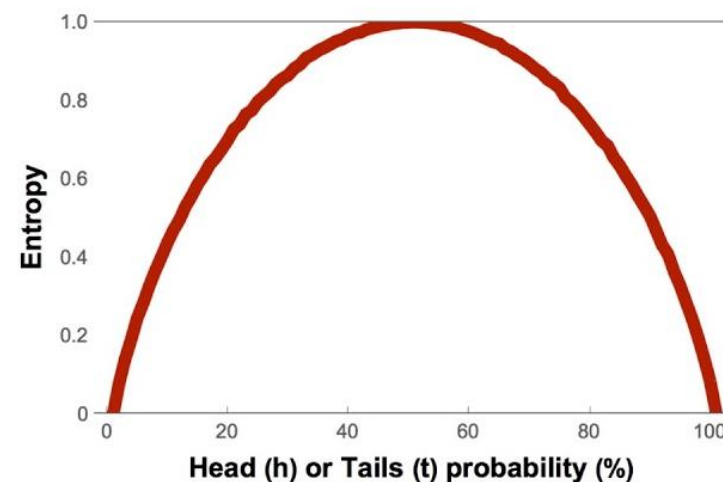


Policy Entropy Bonus

- Improves exploration by discouraging premature convergence to suboptimal deterministic policies.


$$H(\pi) = - \sum_a \pi(a|s) \log \pi(a|s)$$
$$P(X = heads) = \frac{1}{2} \quad P(X = tails) = \frac{1}{2}$$

$$H(\pi) = - \int \pi(a|s) \log \pi(a|s) da$$

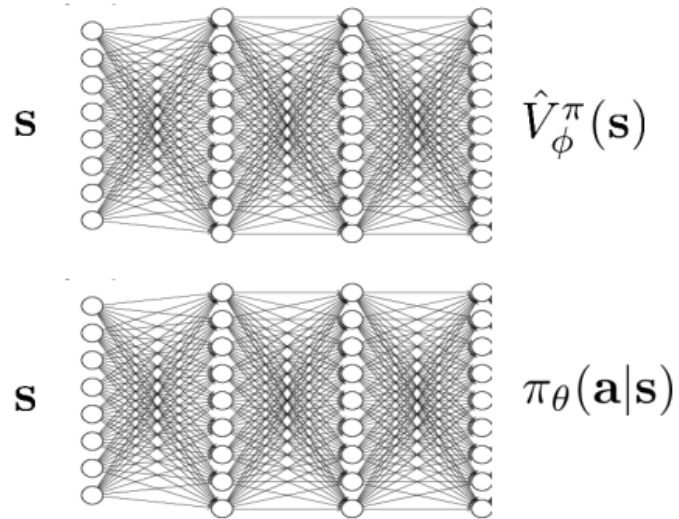


Parameter Sharing

online actor-critic algorithm:

- 
1. take action $\mathbf{a} \sim \pi_{\theta}(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
 2. update \hat{V}_{ϕ}^{π} using target $r + \gamma \hat{V}_{\phi}^{\pi}(\mathbf{s}')$
 3. evaluate $\hat{A}^{\pi}(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \hat{V}_{\phi}^{\pi}(\mathbf{s}') - \hat{V}_{\phi}^{\pi}(\mathbf{s})$
 4. $\nabla_{\theta} J(\theta) \approx \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}|\mathbf{s}) \hat{A}^{\pi}(\mathbf{s}, \mathbf{a})$
 5. $\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$

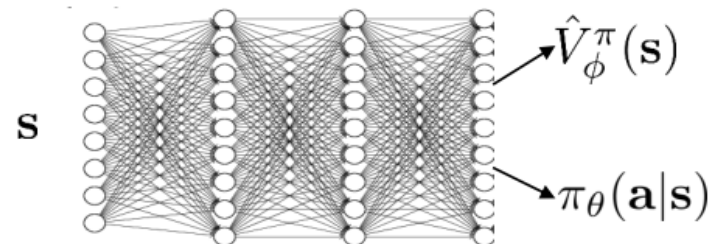
two network design



+ simple & stable

- no shared features between actor & critic

shared network design



Generalized Advantage Estimation (GAE)

Published as a conference paper at ICLR 2016

HIGH-DIMENSIONAL CONTINUOUS CONTROL USING GENERALIZED ADVANTAGE ESTIMATION

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan and Pieter Abbeel

Department of Electrical Engineering and Computer Science

University of California, Berkeley

`{joschu, pcmoritz, levine, jordan, pabbeel}@eecs.berkeley.edu`

- Can we construct all possible n-step returns and average them?

$$\hat{A}_n^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{t+n} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - \hat{V}_\phi^\pi(\mathbf{s}_t) + \gamma^n \hat{V}_\phi^\pi(\mathbf{s}_{t+n})$$

Smaller n results in lower variance, but higher bias

$$\hat{A}_{\text{GAE}}^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{n=1}^{\infty} w_n \hat{A}_n^\pi(\mathbf{s}_t, \mathbf{a}_t)$$

weighted combination of n-step returns

$$w_n \propto \lambda^{n-1} \quad \text{exponential falloff} \quad \text{where } \lambda \in [0,1]$$

$$\hat{A}_{\text{GAE}}^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{\infty} (\gamma\lambda)^{t'-t} \delta_{t'} \quad \delta_{t'} = r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}_{t'+1}) - \hat{V}_\phi^\pi(\mathbf{s}_{t'})$$

← similar effect as discount!

GAE Pseudo Code

```
#predict values based on sequence of states in a trajectory  
vals = predict_values(states)  
# the next two lines implement GAE-Lambda advantage calculation  
deltas = rews[:-1] + gamma * vals[1:] - vals[:-1]  
gae = discount_cumsum(deltas, gamma * lam)
```

Proximal Policy Optimization (PPO)

Proximal Policy Optimization Algorithms


John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov

OpenAI

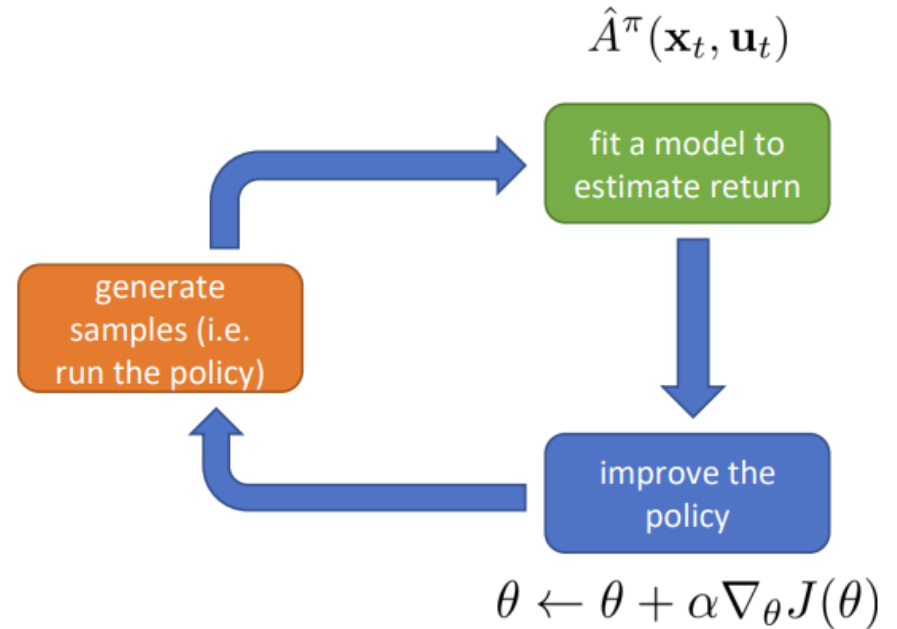
`{joschu, filip, prafulla, alec, oleg}@openai.com`

Why does the policy gradient work?

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \hat{A}_{i,t}^{\pi}$$


- 
1. Estimate $\hat{A}^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$ for current policy π
 2. Use $\hat{A}^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$ to get *improved* policy π'

look familiar?




Why does the policy gradient work?

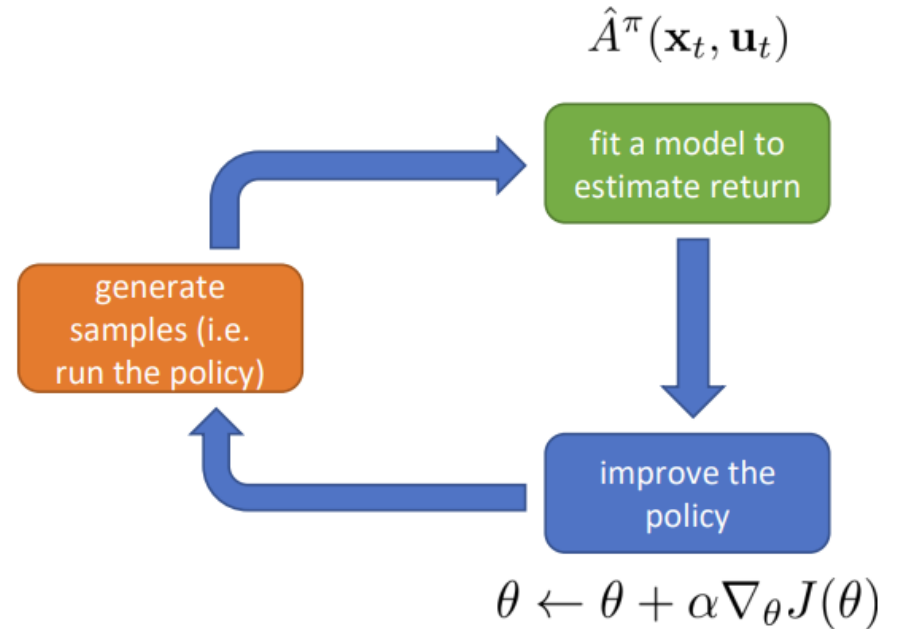
$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \hat{A}_{i,t}^{\pi}$$

- 
1. Estimate $\hat{A}^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$ for current policy π
 2. Use $\hat{A}^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$ to get *improved* policy π'

look familiar?

policy iteration algorithm:

- 
1. evaluate $A^{\pi}(\mathbf{s}, \mathbf{a})$
 2. set $\pi \leftarrow \pi'$



Proximal Policy Optimization (PPO)

- One of the most popular deep RL algorithms
- Used to train ChatGPT and other LLMs

Motivation:

- Many Policy Gradient algorithms have stability problems.
- This can be avoided if we avoid making too big of a policy update.



<https://huggingface.co/blog/deep-rl-ppo>

Proximal Policy Iteration (PPO)

- Measure how much we are changing policy compared with previous policy using a ratio:

$$ratio_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}$$

- Clip policy gradient update based on this ratio:

$$\theta_{k+1} = \arg \max_{\theta} \mathbb{E}_{s, a \sim \pi_{\theta_k}} [L(s, a, \theta_k, \theta)]$$

$$L(s, a, \theta_k, \theta) = \min \left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), \quad \text{clip} \left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1 - \epsilon, 1 + \epsilon \right) A^{\pi_{\theta_k}}(s, a) \right)$$

Proximal Policy Iteration (PPO)

- Simpler way to write clip objective:

$$L(s, a, \theta_k, \theta) = \min \left(\frac{\pi_{\theta}(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), \quad g(\epsilon, A^{\pi_{\theta_k}}(s, a)) \right)$$

where

$$g(\epsilon, A) = \begin{cases} (1 + \epsilon)A & A \geq 0 \\ (1 - \epsilon)A & A < 0 \end{cases}$$

Proximal Policy Iteration (PPO)

- Simpler way to write clip objective:

$$L(s, a, \theta_k, \theta) = \min \left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), \quad g(\epsilon, A^{\pi_{\theta_k}}(s, a)) \right)$$

where

$$g(\epsilon, A) = \begin{cases} (1 + \epsilon)A & A \geq 0 \\ (1 - \epsilon)A & A < 0 \end{cases}$$

What if the advantage is positive?

$$L(s, a, \theta_k, \theta) = \min \left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1 + \epsilon) \right) A^{\pi_{\theta_k}}(s, a)$$

We want to increase $\pi_\theta(a|s)$, but not too much!

Once $\pi_\theta(a|s) > (1 + \epsilon)\pi_{\theta_k}(a|s)$ the min kicks in and limits our policy update.

Proximal Policy Iteration (PPO)

- Simpler way to write clip objective:

$$L(s, a, \theta_k, \theta) = \min \left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), \quad g(\epsilon, A^{\pi_{\theta_k}}(s, a)) \right)$$

where

$$g(\epsilon, A) = \begin{cases} (1 + \epsilon)A & A \geq 0 \\ (1 - \epsilon)A & A < 0 \end{cases}$$

What if the advantage is negative?

$$L(s, a, \theta_k, \theta) = \max \left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1 - \epsilon) \right) A^{\pi_{\theta_k}}(s, a)$$

We want to decrease $\pi_\theta(a|s)$, but not too much!

Once $\pi_\theta(a|s) < (1 - \epsilon)\pi_{\theta_k}(a|s)$ the max kicks in and limits our policy update.

Algorithm 1 PPO-Clip

- 1: Input: initial policy parameters θ_0 , initial value function parameters ϕ_0
- 2: **for** $k = 0, 1, 2, \dots$ **do**
- 3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
- 4: Compute rewards-to-go \hat{R}_t .
- 5: Compute advantage estimates, \hat{A}_t (using any method of advantage estimation) based on the current value function V_{ϕ_k} .
- 6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg \max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \min \left(\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.

- 7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg \min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \left(V_{\phi}(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

- 8: **end for**
-

Lots of other tricks used

- Additional advantage normalization
- Early stopping with KL-divergence
- Etc.

The 37 Implementation Details of Proximal Policy Optimization: <https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/>