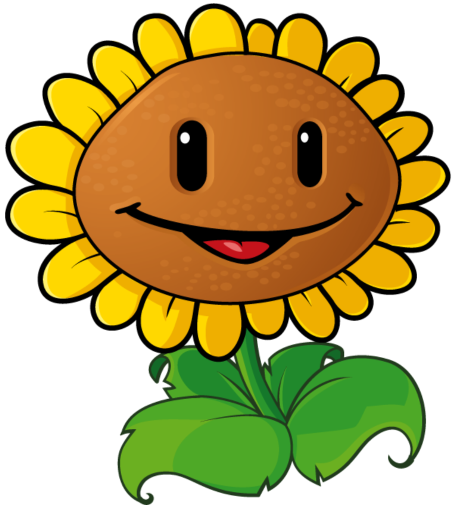
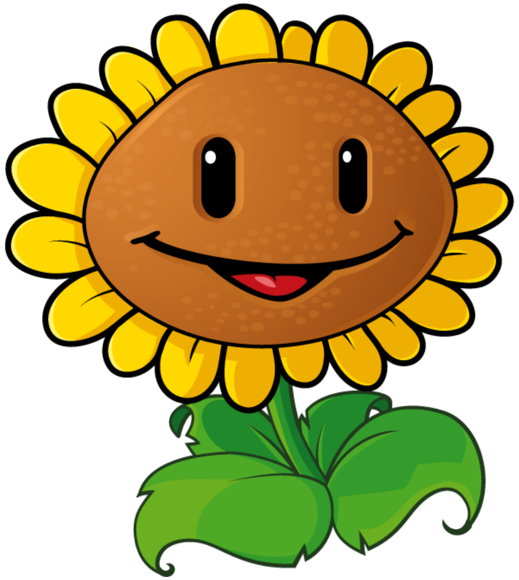
Policy Gradient Methods

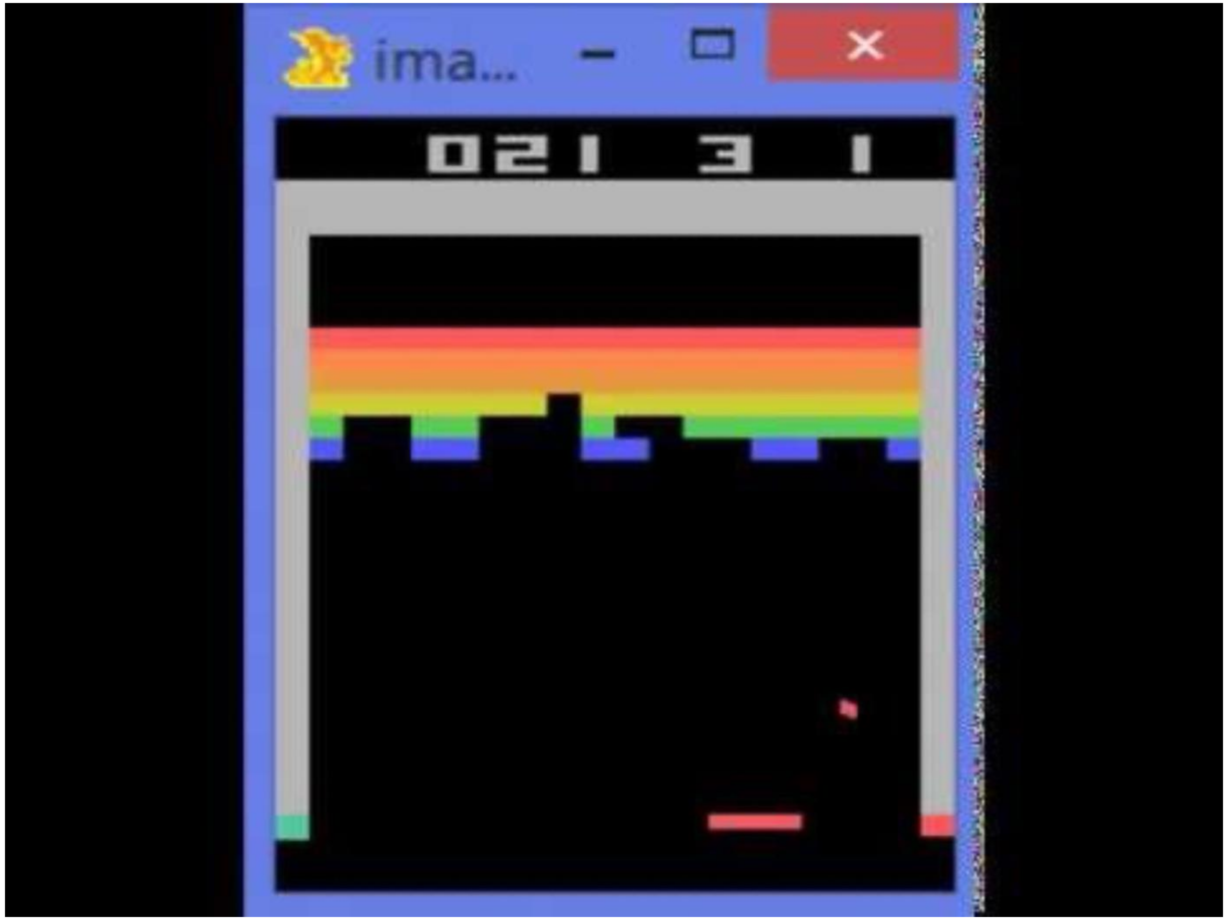
CSCI 410

Eric Ewing





Deep-Q Networks



Video Source: https://www.youtube.com/watch?v=V1eYniJ0Rnk

Q-Values to Policy

What do we do after we learn Q? We need to turn them into a policy.

For a given state, take the action associated with the best Q-value.

$$\pi(s) = argmax_a \, Q(s, a)$$

Policies

Why learn Q-values first and turn them into a policy? Why not just learn a policy?

Policies

Why learn Q-values first and turn them into a policy? Why not just learn a policy?

Policies

Why learn Q-values first and turn them into a policy? Why not just learn a policy?

Need to convert outputs of neural network to be probabilities!

Policies

Why learn Q-values first and turn them into a policy? Why not just learn a policy?

Need to convert outputs of neural network to be probabilities!

Use Softmax activation function:

$$\sigma(z)_i = \frac{z^i}{\sum_j z_j}$$

Output *i*

Vector of intermediate features z

How do we train a policy network?

How do we train a policy network?

Need to find an appropriate loss function.

How do we train a policy network?

Need to find an appropriate loss function.

What’s our objective?

How do we train a policy network?

Need to find an appropriate loss function.

What's our objective?

Find a policy π such that the value of the start state is maximized:

How do we train a policy network?

Need to find an appropriate loss function.

What's our objective?

Find a policy π such that the value of the start state is maximized:

$$\pi = \operatorname{argmax}_{\pi} \left(V(s_0)\right)$$

How do we train a policy network?

Need to find an appropriate loss function.

What's our objective?

Find a policy π such that the value of the start state is maximized:

$$\pi = \text{argmax}_{\pi} \left(V(s_0)\right)$$

How can we maximize $V(s_0)$?

Let $J(\theta)$ be our objective function:

$$J(\theta) = V(s_0)$$

Let $J(\theta)$ be our objective function:

$$J(\theta) = V(s_0)$$

$$J(\theta) = \mathbb{E}[G_0]$$

Let $J(\theta)$ be our objective function:

$$J(\theta) = V(s_0)$$

$$J(\theta) = \mathbb{E}[G_0]$$

$$J(\theta) = \sum_{\tau} \Pr(\tau|\theta)\, G(\tau)$$

Let $J(\theta)$ be our objective function:

$$J(\theta) = V(s_0)$$

$$J(\theta) = \mathbb{E}[G_0]$$

$$J(\theta) = \sum_{\tau} \Pr(\tau|\theta)\, G(\tau)$$

Probability of a trajectory occurring

Returns of a specific trajectory

Let $J(\theta)$ be our objective function:

$$J(\theta) = V(s_0)$$

$$J(\theta) = \mathbb{E}[G_0]$$

$$J(\theta) = \sum_{\tau} \Pr(\tau|\theta)\, G(\tau)$$

Probability of a trajectory occurring

Returns of a specific trajectory

$$\Pr(\tau|\theta) = \Pi_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

Let $J(\theta)$ be our objective function:

$$J(\theta) = V(s_0)$$

$$J(\theta) = \mathbb{E}[G_0]$$

$$J(\theta) = \sum_{\tau} \Pr(\tau|\theta)\, G(\tau)$$

Probability of a trajectory occurring

Returns of a specific trajectory

$$\Pr(\tau|\theta) = \Pi_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

State transition Probability

Probability of taking an action for a given state

Let $J(\theta)$ be our objective function:

$$J(\theta) = V(s_0)$$

$$J(\theta) = \mathbb{E}[G_0]$$

$$J(\theta) = \sum_{\tau} \Pr(\tau|\theta)\, G(\tau)$$

Sum over all possible trajectories

Probability of a trajectory occurring

Returns of a specific trajectory

$$\Pr(\tau|\theta) = \Pi_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

State transition Probability

Probability of taking an action for a given state

Log-Derivative Trick

We can rewrite the derivative of a function using the derivative of the natural log function:

$$\nabla \ln f(x) = \frac{\nabla \mathrm{f}(x)}{f(x)}$$

$$\nabla f(x) = f(x) \nabla \ln \mathrm{f(x)}$$

When applied to $\Pr(\tau|\theta)$:

$$\nabla_\theta \Pr(\tau|\theta) = \Pr(\tau|\theta)\, \nabla_\theta \ln \Pr(\tau|\theta)$$

Log Probability Trick

$$\Pr(\tau|\theta) = \Pi_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

Log Probability Trick

$$\Pr(\tau|\theta) = \Pi_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

$$\nabla_\theta \Pr(\tau|\theta) = \Pr(\tau|\theta)\, \nabla_\theta \ln \Pr(\tau|\theta)$$

This gradient term is what we want to calculate

Log Probability Trick

$$\Pr(\tau|\theta) = \Pi_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

$$\nabla_\theta \Pr(\tau|\theta) = \Pr(\tau|\theta)\,\nabla_\theta \ln \Pr(\tau|\theta)$$

This gradient term is what we want to calculate

$$\nabla_\theta \ln \Pr(\tau|\theta) = \nabla_\theta \sum_{t=0}^{T} \ln[P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)]$$

Log Probability Trick

$$\Pr(\tau|\theta) = \Pi_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

$$\nabla_\theta \Pr(\tau|\theta) = \Pr(\tau|\theta)\, \nabla_\theta \ln \Pr(\tau|\theta)$$

This gradient term is what we want to calculate

$$\nabla_\theta \ln \Pr(\tau|\theta) = \nabla_\theta \sum_{t=0}^{T} \ln[P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)]$$

$$\nabla_\theta \ln \Pr(\tau|\theta) = \nabla_\theta \sum_{t=0}^{T} \ln P(s_{t+1}|s_t, a_t) + \ln \pi_\theta(a_t|s_t)$$

Log Probability Trick

$$\Pr(\tau|\theta) = \Pi_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

$$\nabla_\theta \Pr(\tau|\theta) = \Pr(\tau|\theta)\, \nabla_\theta \ln \Pr(\tau|\theta)$$

This gradient term is what we want to calculate

Log of product -> sum of logs

$$\nabla_\theta \ln \Pr(\tau|\theta) = \nabla_\theta \sum_{t=0}^{T} \ln P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

Log of product -> sum of logs

$$\nabla_\theta \ln \Pr(\tau|\theta) = \nabla_\theta \sum_{t=0}^{T} \ln P(s_{t+1}|s_t, a_t) + \ln \pi_\theta(a_t|s_t)$$

Derivative of sum -> sum of derivative

$$\nabla_\theta \ln \Pr(\tau|\theta) = \sum_{t=0}^{T} \nabla_\theta \ln P(s_{t+1}|s_t, a_t) + \nabla_\theta \ln \pi_\theta(a_t|s_t)$$

Gradient of a trajectory

$$\nabla_\theta \ln \Pr(\tau|\theta) = \sum_{t=0}^{T} \nabla_\theta \ln P(s_{t+1}|s_t, a_t) + \nabla_\theta \ln \pi_\theta(a_t|s_t)$$

State transition function
does not depend on θ!

$$\nabla_\theta \ln \Pr(\tau|\theta) = \sum_{t=0}^{T} \nabla_\theta \ln \pi_\theta(a_t|s_t)$$

Policy Gradient Derivation

Putting it all back together:

$$J(\theta) = \sum_{\tau} \Pr(\tau|\theta)\, G(\tau)$$

Our Objective

Policy Gradient Derivation

Putting it all back together:

$$J(\theta) = \sum_{\tau} \Pr(\tau|\theta)\, G(\tau)$$

Our Objective

$$\nabla_{\theta} J(\theta) = \sum_{\tau} \nabla_{\theta} \Pr(\tau|\theta)\, G(\tau)$$

Take the gradient

Policy Gradient Derivation

Putting it all back together:

$$J(\theta) = \sum_{\tau} \Pr(\tau|\theta)\, G(\tau)$$

Our Objective

$$\nabla_\theta J(\theta) = \sum_{\tau} \nabla_\theta \Pr(\tau|\theta)\, G(\tau)$$

Take the gradient

$$\nabla_\theta J(\theta) = \sum_{\tau} \Pr(\tau|\theta) G(\tau) \nabla_\theta \ln \Pr(\tau|\theta)$$

Log-Derivative Trick

Policy Gradient Derivation

Putting it all back together:

$$J(\theta) = \sum_{\tau} \Pr(\tau|\theta)\, G(\tau)$$ Our Objective

$$\nabla_\theta J(\theta) = \sum_{\tau} \nabla_\theta \Pr(\tau|\theta)\, G(\tau)$$ Take the gradient

$$\nabla_\theta J(\theta) = \sum_{\tau} \Pr(\tau|\theta) G(\tau) \nabla_\theta \ln \Pr(\tau|\theta)$$ Log-Derivative Trick

$$\nabla_\theta J(\theta) = \sum_{\tau} \left[\Pr(\tau|\theta) G(\tau) \sum_{t=0}^{T} \nabla_\theta \ln \pi_\theta(a_t|s_t)\,\right]$$ Gradient of a Trajectory

Policy Gradient Derivation

Putting it all back together:

$$J(\theta) = \sum_{\tau} \Pr(\tau|\theta)\, G(\tau)$$ Our Objective

$$\nabla_\theta J(\theta) = \sum_{\tau} \nabla_\theta \Pr(\tau|\theta)\, G(\tau)$$ Take the gradient

$$\nabla_\theta J(\theta) = \sum_{\tau} \Pr(\tau|\theta) G(\tau) \nabla_\theta \ln \Pr(\tau|\theta)$$ Log-Derivative Trick

$$\nabla_\theta J(\theta) = \sum_{\tau} [\Pr(\tau|\theta) G(\tau) \sum_{t=0}^{T} \nabla_\theta \ln \pi_\theta(a_t|s_t)\ ]$$ Gradient of a Trajectory

$$\nabla_\theta J(\theta) = \mathbb{E}[G_0 \sum_{t=0}^{T} \nabla_\theta \ln \pi_\theta(a_t|s_t)]$$ Convert back to Expectation

Policy Gradient

Bigger step if better returns

Direction to move in to increase probability of trajectory

$$\nabla_\theta J(\theta) = \mathbb{E}[G_0 \sum_{t=0}^{T} \nabla_\theta \ln \pi_\theta(a_t|s_t)]$$

We will never be able to sum over all possible trajectories...

How do we get around this?

Policy Gradient

Bigger step if better returns

Direction to move in to increase probability of trajectory

$$\nabla_\theta J(\theta) = \mathbb{E}[G_0 \sum_{t=0}^{T} \nabla_\theta \ln \pi_\theta(a_t|s_t)]$$

We will never be able to sum over all possible trajectories…

How do we get around this?

Sampling!

1. Collect n trajectories following policy π_θ
2. $\Pr(\tau|\theta) = 1/n$ for each trajectory
3. Calculate the total return for each trajectory $G(\tau)$

Reward-To-Go Policy Gradient

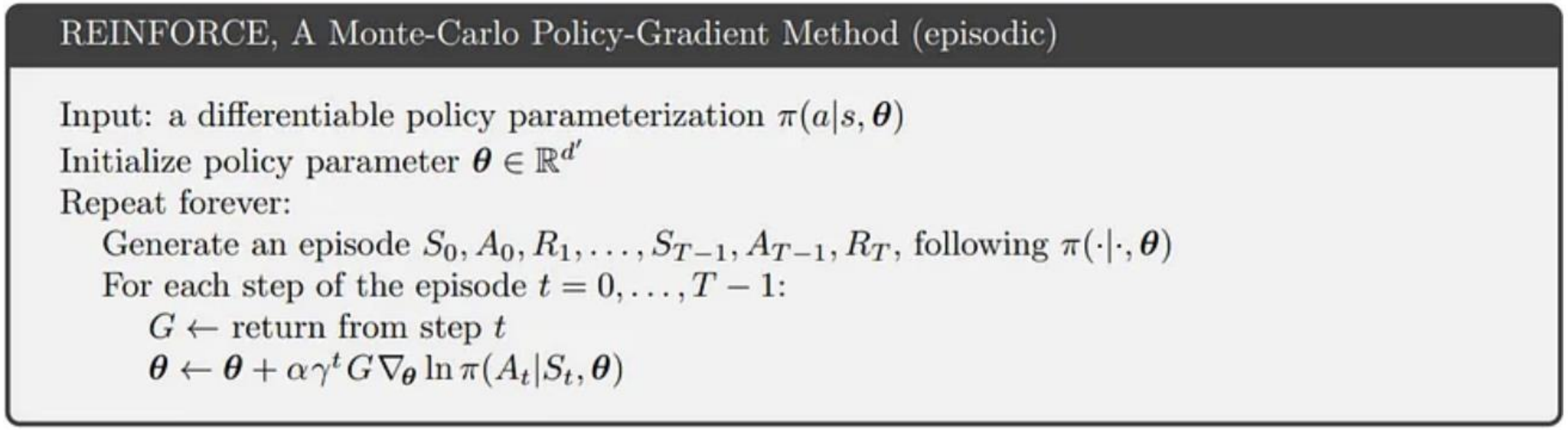
You can also do the policy gradient derivation such that the gradient does not depend on G_0, but on G_t

$$\nabla_\theta J(\theta) = \mathbb{E}[\sum_{t=0}^{T} G_t \nabla_\theta \ln \pi_\theta(a_t|s_t)]$$

Or

$$\nabla_\theta J(\theta) = \mathbb{E}[\sum_{t=0}^{T} Q(s_t, a_t) \nabla_\theta \ln \pi_\theta(a_t|s_t)]$$

REINFORCE (Policy Gradient Learning)

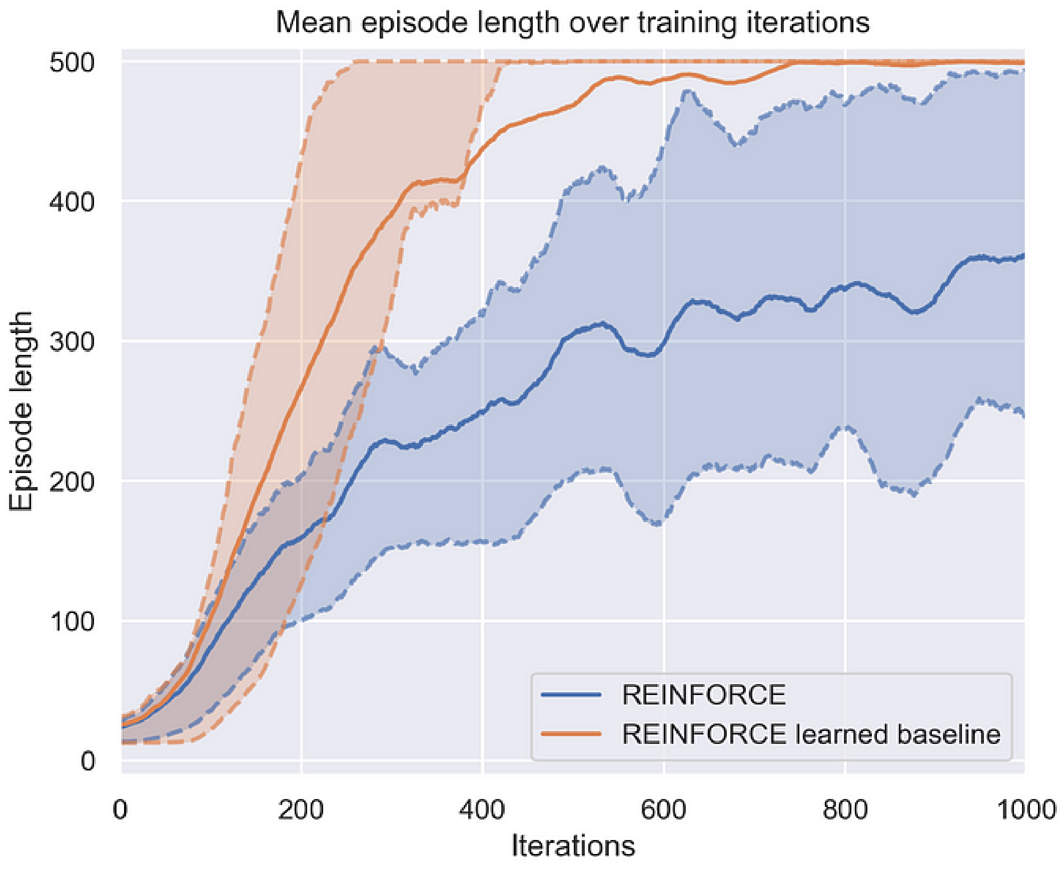

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

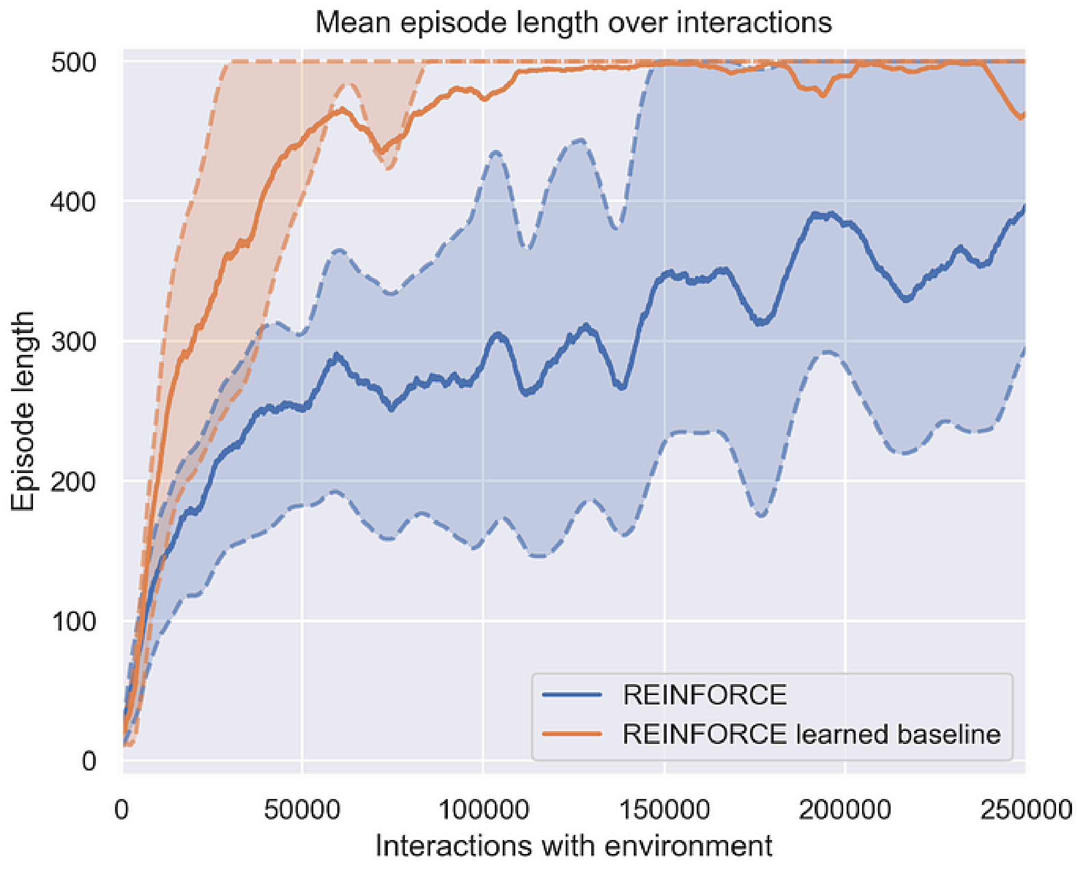
Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$
Repeat forever:
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  For each step of the episode $t = 0, \ldots, T-1$:
    $G \leftarrow$ return from step t
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Source: Sutton and Barto, Reinforcement Learning: An Introduction

REINFORCE Variance

If we could calculate $\nabla_\theta J(\theta)$ exactly (not just for single trajectory/sample), then Policy Gradient would be a great algorithm! (with some minor flaws)





Results on Cartpole

Image Source: https://medium.com/@fork.tree.ai/understanding-baseline-techniques-for-reinforce-53a1e2279b57

Actor-Critic Methods

Actor-Critic Methods

REINFORCE uses the return for a trajectory G_t:

Actor-Critic Methods

REINFORCE uses the return for a trajectory G_t:

$$\nabla_\theta J(\theta) = \mathbb{E}[\sum_{t=0}^{T} G_t \nabla_\theta \ln \pi_\theta(a_t|s_t)]$$

Actor-Critic Methods

REINFORCE uses the return for a trajectory G_t:

$$\nabla_\theta J(\theta) = \mathbb{E}[\sum_{t=0}^{T} G_t \nabla_\theta \ln \pi_\theta(a_t|s_t)]$$

Variance of Returns is always a problem…

Actor-Critic Methods

REINFORCE uses the return for a trajectory G_t:

$$\nabla_\theta J(\theta) = \mathbb{E}[\sum_{t=0}^{T} G_t \nabla_\theta \ln \pi_\theta(a_t|s_t)]$$

Variance of Returns is always a problem...

Actor-Critic Methods learn an approximation of G_t

Actor-Critic Methods

REINFORCE uses the return for a trajectory G_t:

$$\nabla_\theta J(\theta) = \mathbb{E}[\sum_{t=0}^{T} G_t \nabla_\theta \ln \pi_\theta(a_t|s_t)]$$

Variance of Returns is always a problem…

Actor-Critic Methods learn an approximation of G_t

$$Q^\pi(s_t, a_t) = \mathbb{E}[G_t]$$

Actor-Critic Methods

REINFORCE uses the return for a trajectory G_t:

$$\nabla_\theta J(\theta) = \mathbb{E}[\sum_{t=0}^{T} G_t \nabla_\theta \ln \pi_\theta(a_t|s_t)]$$

Variance of Returns is always a problem...

Actor-Critic Methods learn an approximation of G_t

$$Q^\pi(s_t, a_t) = \mathbb{E}[G_t]$$

Actor-Critic Methods

REINFORCE uses the return for a trajectory G_t:

$$\nabla_\theta J(\theta) = \mathbb{E}[\sum_{t=0}^{T} G_t \nabla_\theta \ln \pi_\theta(a_t|s_t)]$$

Variance of Returns is always a problem…

Actor-Critic Methods learn an approximation of G_t

$$Q^\pi(s_t, a_t) = \mathbb{E}[G_t]$$

$$\nabla_\theta J(\theta) = \mathbb{E}[\sum_{t=0}^{T} Q^{\pi_\theta}(s_t, a_t) \nabla_\theta \ln \pi_\theta(a_t|s_t)]$$

Actor-Critic Algorithm: Learn Q^{π} and $\pi(a|s)$

Actor-Critic Algorithm: Learn Q^{π} and $\pi(a|s)$

Actor (policy): Takes actions

Actor-Critic Algorithm: Learn Q^{π} and $\pi(a|s)$

Actor (policy): Takes actions

Critic: Scores the action

Actor-Critic Algorithm: Learn Q^{π} and $\pi(a|s)$

Actor (policy): Takes actions

Critic: Scores the action

That’s not even how that piece moves, Q(s,a)=-5

Actor-Critic Algorithm: Learn Q^{π} and $\pi(a|s)$

Policy network has parameters θ
Q network has parameters *w*

Initialize $\pi_{\theta}, Q_w, \alpha_{\theta}, \alpha_w$

Repeat forever:

Take action a, get new state s' and reward *r*

Sample next action $a' \sim \pi_{\theta}(a|s)$

update $\theta \leftarrow \theta + \alpha_{\theta} Q_w(s,a) \nabla_{\theta} \ln \pi_{\theta}(a|s)$

Calculate TD Error: $\delta = r + \gamma Q_w(s',a') - Q_w(s,a)$

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

$a \leftarrow a', s \leftarrow s'$

Actor-Critic Algorithm: Learn Q^{π} and $\pi(a|s)$

Policy network has parameters θ
Q network has parameters w

Initialize $\pi_\theta, Q_w, \alpha_\theta, \alpha_w$

Repeat forever:

Take action a, get new state s' and reward *r*

Sample next action $a' \sim \pi_\theta(a|s)$

update $\theta \leftarrow \theta + \alpha_\theta Q_w(s,a) \nabla_\theta \ln \pi_\theta(a|s)$

Calculate TD Error: $\delta = r + \gamma Q_w(s', a') - Q_w(s,a)$

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

$a \leftarrow a', s \leftarrow s'$

Like Q-learning and REINFORCE at the same time

Variations on a Theme...

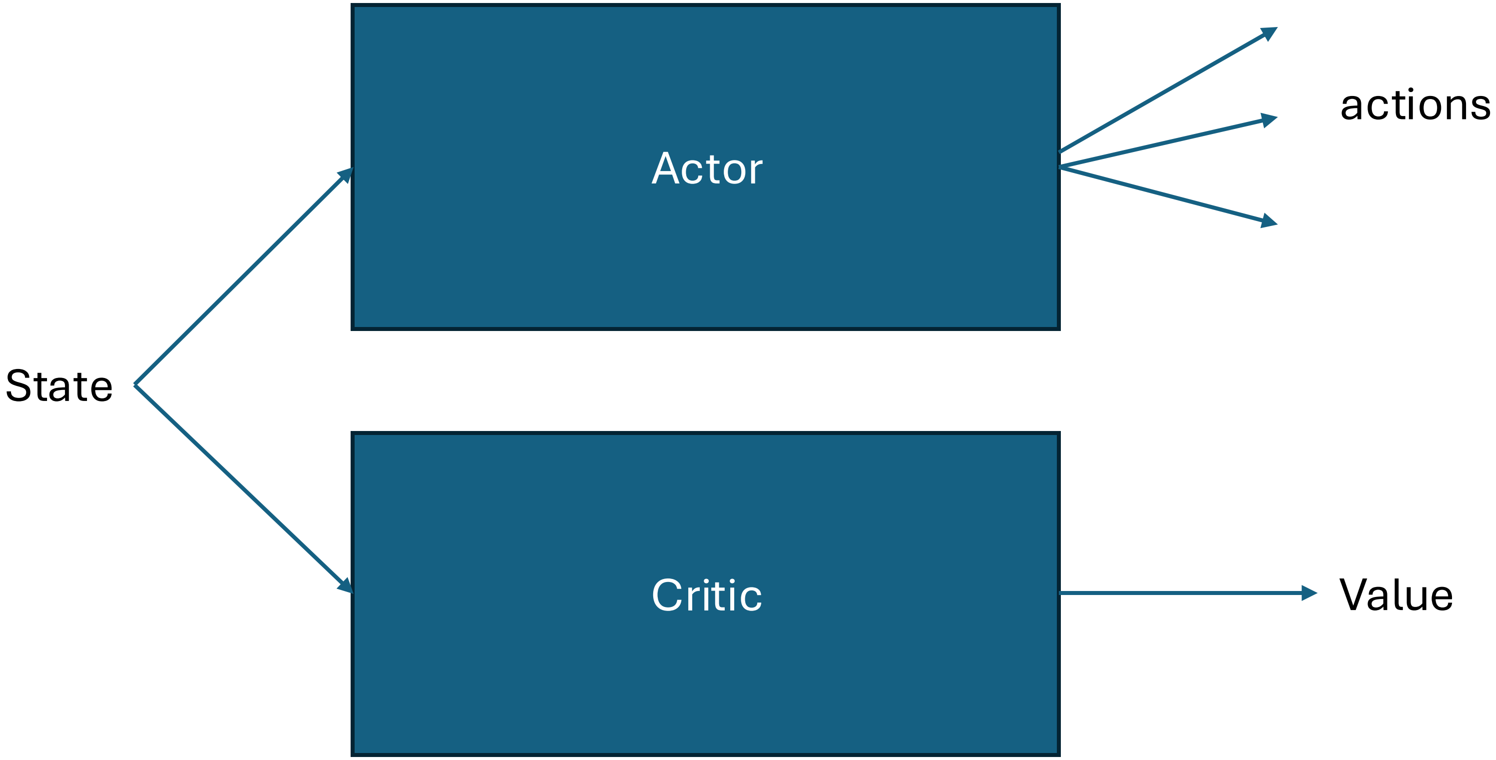
How to estimate $J(\theta)$

(Wikipedia uses R_t instead of G_t)

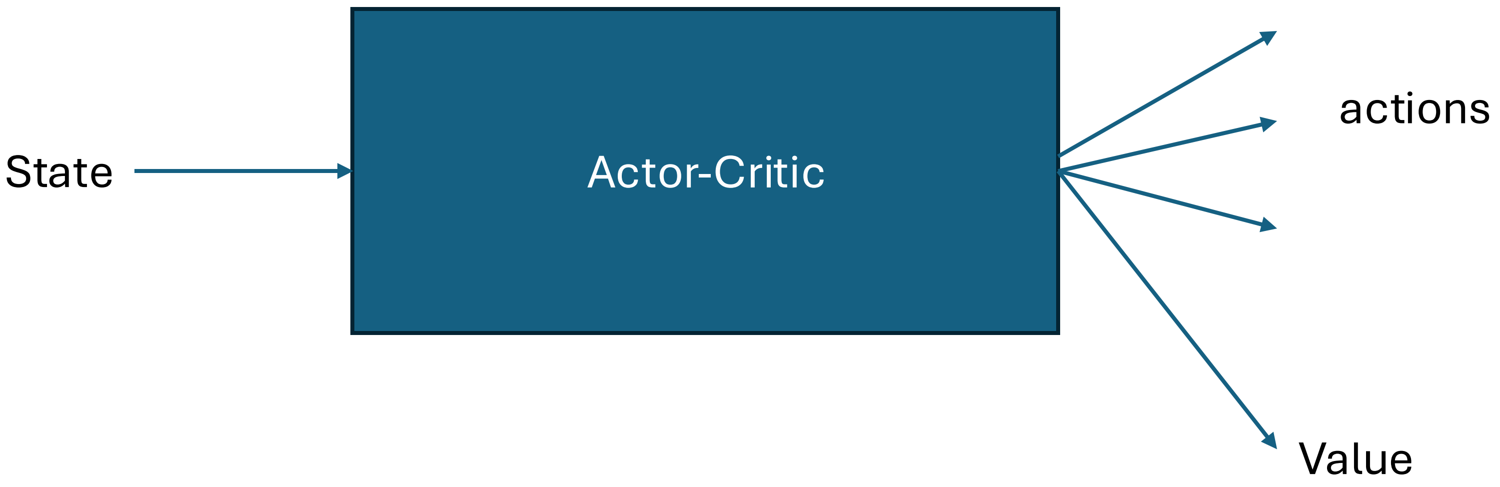
- $\sum_{0 \le i \le T} (\gamma^i R_i)$.
- $\gamma^j \sum_{j \le i \le T} (\gamma^{i-j} R_i)$: the **REINFORCE** algorithm.
- $\gamma^j \sum_{j \le i \le T} (\gamma^{i-j} R_i) - b(S_j)$: the **REINFORCE with baseline** algorithm. Here b is an arbitrary function.
- $\gamma^j \left(R_j + \gamma V^{\pi_\theta}(S_{j+1}) - V^{\pi_\theta}(S_j)\right)$: TD(1) learning.
- $\gamma^j Q^{\pi_\theta}(S_j, A_j)$.
- $\gamma^j A^{\pi_\theta}(S_j, A_j)$: **Advantage Actor-Critic (A2C)**.[3]
- $\gamma^j \left(R_j + \gamma R_{j+1} + \gamma^2 V^{\pi_\theta}(S_{j+2}) - V^{\pi_\theta}(S_j)\right)$: TD(2) learning.
- $\gamma^j \left(\sum_{k=0}^{n-1} \gamma^k R_{j+k} + \gamma^n V^{\pi_\theta}(S_{j+n}) - V^{\pi_\theta}(S_j)\right)$: TD(n) learning.
- $\gamma^j \sum_{n=1}^{\infty} \frac{\lambda^{n-1}}{1-\lambda} \cdot \left(\sum_{k=0}^{n-1} \gamma^k R_{j+k} + \gamma^n V^{\pi_\theta}(S_{j+n}) - V^{\pi_\theta}(S_j)\right)$: TD(λ) learning, also known as **GAE (generalized advantage estimate)**.[4] This is obtained by an exponentially decaying sum of the TD(n) learning terms.

Source: https://en.wikipedia.org/wiki/Actor-critic_algorithm

Actor-Critic Networks



Actor-Critic Networks



Just make sure you use the correct activation function for the different outputs

Deep Q-Learning Revisited

Compute TD-Error: $\delta = r + \gamma \max_{a'} Q(s', a') - Q(s, a)$

Loss Function: $L = \delta^2$

Update model with Gradient Descent

Deep Q-Learning Revisited

Compute TD-Error: $\delta = r + \gamma \max_{a'} Q(s', a') - Q(s, a)$

Loss Function: $L = \delta^2$

Update model with Gradient Descent

Q-Learning uses:

$$\delta = r + \gamma \max_{a'} Q(s', a') - Q(s, a)$$

Actor-Critic Uses:

$$\delta = r + \gamma Q(s', a') \ - Q\ (s, a)$$

Deep Q-Learning Revisited

Compute TD-Error: $\delta = r + \gamma \max_{a'} Q(s', a') - Q(s, a)$

Loss Function: $L = \delta^2$

Update model with Gradient Descent

Q-Learning uses:

$$\delta = r + \gamma \max_{a'} Q(s', a') - Q(s, a)$$

Actor-Critic Uses:

$$\delta = r + \gamma Q(s', a') - Q(s, a)$$

Q-Learning is learning *Optimal Q-values*

Actor-Critic is learning the Q-values for following a specific policy Q^{π}

On-Policy Vs. Off-Policy Learning

RL algorithms collect experiences and learn from these experiences

On-Policy Algorithms have to collect experiences with the policy they are learning

Off-Policy Algorithms can use **any** policy to collect experiences

DQNs Are Off-Policy

In Q-Learning, we typically collect experiences using an ϵ-greedy policy in training

With probability ϵ, take random action.

Else, take action $\operatorname{argmax}_a Q(s, a)$

At test time, we should take the best actions, not the ϵ-greedy actions

$$\operatorname{argmax}_a Q(s, a)$$

DQNs Are Off-Policy

In Q-Learning, we typically collect experiences using an ϵ-greedy policy in training

With probability ϵ, take random action.

Else, take action $\text{argmax}_a\, Q(s, a)$

At test time, we should take the best actions, not the ϵ-greedy actions

$\text{argmax}_a\, Q(s, a)$

These are different policies! DQNs can be trained with any data collection policy at training time

Actor-Critic Algorithm: Learn Q^{π} and $\pi(a|s)$

Initialize π_θ, $Q_w, \alpha_\theta, \alpha_w$

Repeat forever:

Take action a, get new state s' and reward r

Sample next action $a' \sim \pi_\theta(a|s)$

On Policy: Have to take actions according to π_θ

update $\theta \leftarrow \theta + \alpha_\theta Q_w(s,a) \nabla_\theta \ln \pi_\theta(a|s)$

Calculate TD Error: $\delta = r + \gamma Q_w(s', a') - Q_w(s,a)$

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

$a \leftarrow a', s \leftarrow s'$

On-Policy vs Off-Policy Learning

On-Policy vs Off-Policy Learning

Advantage of On-Policy Learning:

On-Policy vs Off-Policy Learning

Advantage of On-Policy Learning:

- More sample efficient, tend to converge faster

On-Policy vs Off-Policy Learning

Advantage of On-Policy Learning:

- More sample efficient, tend to converge faster

Disadvantages of On-Policy Learning:

On-Policy vs Off-Policy Learning

Advantage of On-Policy Learning:

- More sample efficient, tend to converge faster

Disadvantages of On-Policy Learning:

- Can get stuck in local minima
  - Is there a simple policy that performs ok? Would small changes to that policy cause returns to go down temporarily?
  - How do balance exploration in our policy?

On-Policy vs Off-Policy Learning

Advantage of On-Policy Learning:

- More sample efficient, tend to converge faster

Disadvantages of On-Policy Learning:

- Can get stuck in local minima
  - Is there a simple policy that performs ok? Would small changes to that policy cause returns to go down temporarily?
  - How do balance exploration in our policy?

Advantages of Off-Policy Learning:

On-Policy vs Off-Policy Learning

Advantage of On-Policy Learning:

- More sample efficient, tend to converge faster

Disadvantages of On-Policy Learning:

- Can get stuck in local minima
  - Is there a simple policy that performs ok? Would small changes to that policy cause returns to go down temporarily?
  - How do balance exploration in our policy?

Advantages of Off-Policy Learning:

- Can learn from any policy

On-Policy vs Off-Policy Learning

Advantage of On-Policy Learning:

- More sample efficient, tend to converge faster

Disadvantages of On-Policy Learning:

- Can get stuck in local minima
  - Is there a simple policy that performs ok? Would small changes to that policy cause returns to go down temporarily?
  - How do balance exploration in our policy?

Advantages of Off-Policy Learning:

- Can learn from any policy
- More likely to learn an optimal policy

On-Policy vs Off-Policy Learning

Advantage of On-Policy Learning:

- More sample efficient, tend to converge faster

Disadvantages of On-Policy Learning:

- Can get stuck in local minima
  - Is there a simple policy that performs ok? Would small changes to that policy cause returns to go down temporarily?
  - How do balance exploration in our policy?

Advantages of Off-Policy Learning:

- Can learn from any policy
- More likely to learn an optimal policy

Disadvantages of Off-Policy Learning:

On-Policy vs Off-Policy Learning

Advantage of On-Policy Learning:

- More sample efficient, tend to converge faster

Disadvantages of On-Policy Learning:

- Can get stuck in local minima
    - Is there a simple policy that performs ok? Would small changes to that policy cause returns to go down temporarily?
    - How do balance exploration in our policy?

Advantages of Off-Policy Learning:

- Can learn from any policy
- More likely to learn an optimal policy

Disadvantages of Off-Policy Learning:

- Slower…

Off-Policy Learning

Most of the time in RL, collecting the data is computationally expensive.

So far, we've looked at an example, learned from it, and discarded it.

In all our other problems, we always learned from data multiple times (i.e., epochs)

Off-Policy Learning

Most of the time in RL, collecting the data is computationally expensive.

So far, we've looked at an example, learned from it, and discarded it.

In all our other problems, we always learned from data multiple times (i.e., epochs)

Off-Policy Learning

Most of the time in RL, collecting the data is computationally expensive.

Maybe we shouldn't throw away useful data immediately…

So far, we've looked at an example, learned from it, and discarded it.

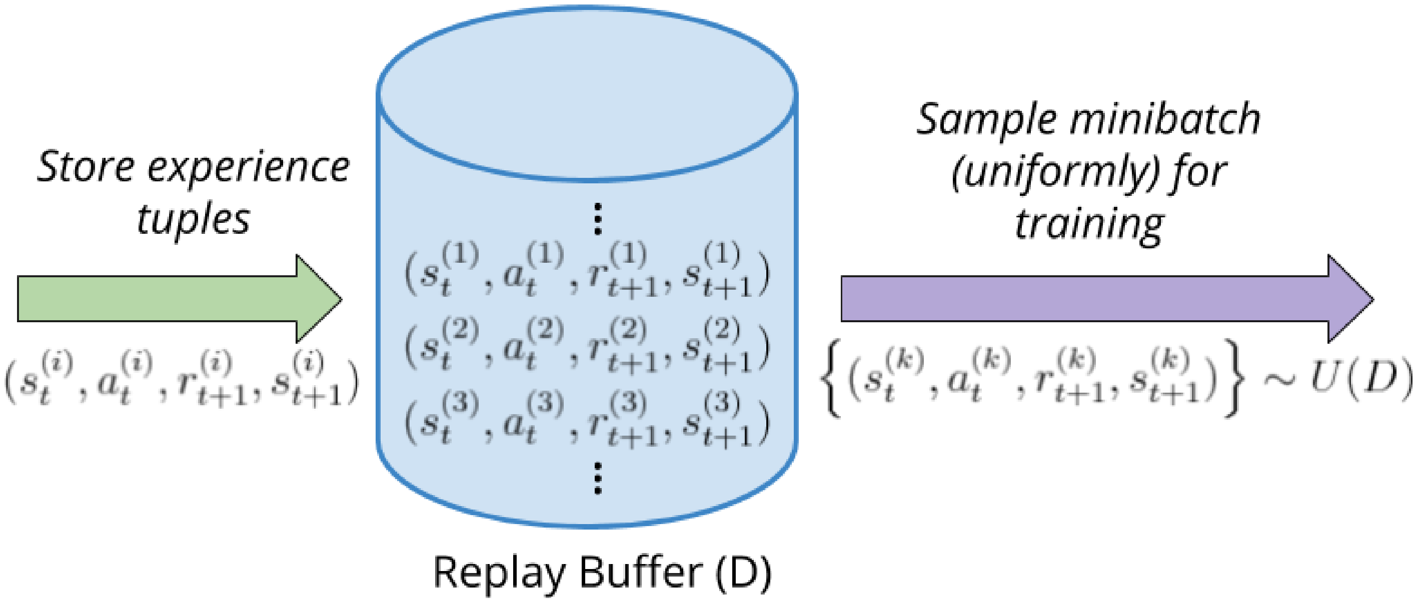
In all our other problems, we always learned from data multiple times (i.e., epochs)

Experience Replay and Replay Buffers

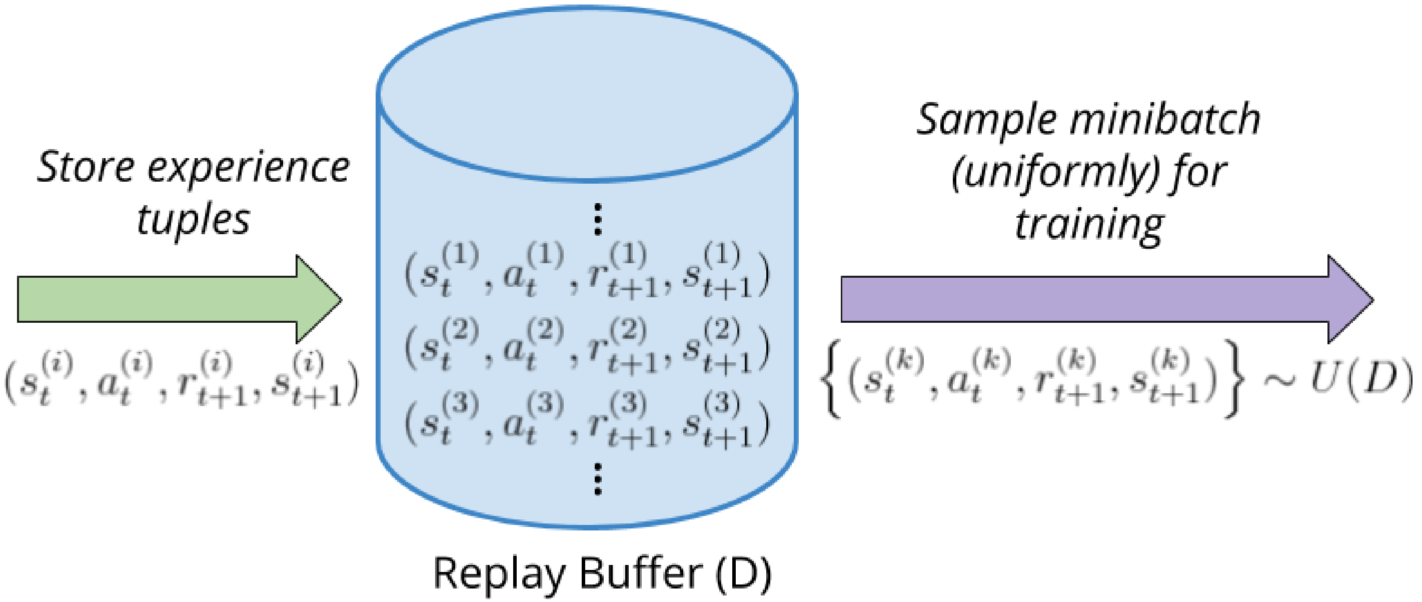
Keep a memory of experiences (state, action, reward, next_state)

As you collect new experiences, remove oldest experiences from buffer

To train model, sample batch of data from buffer

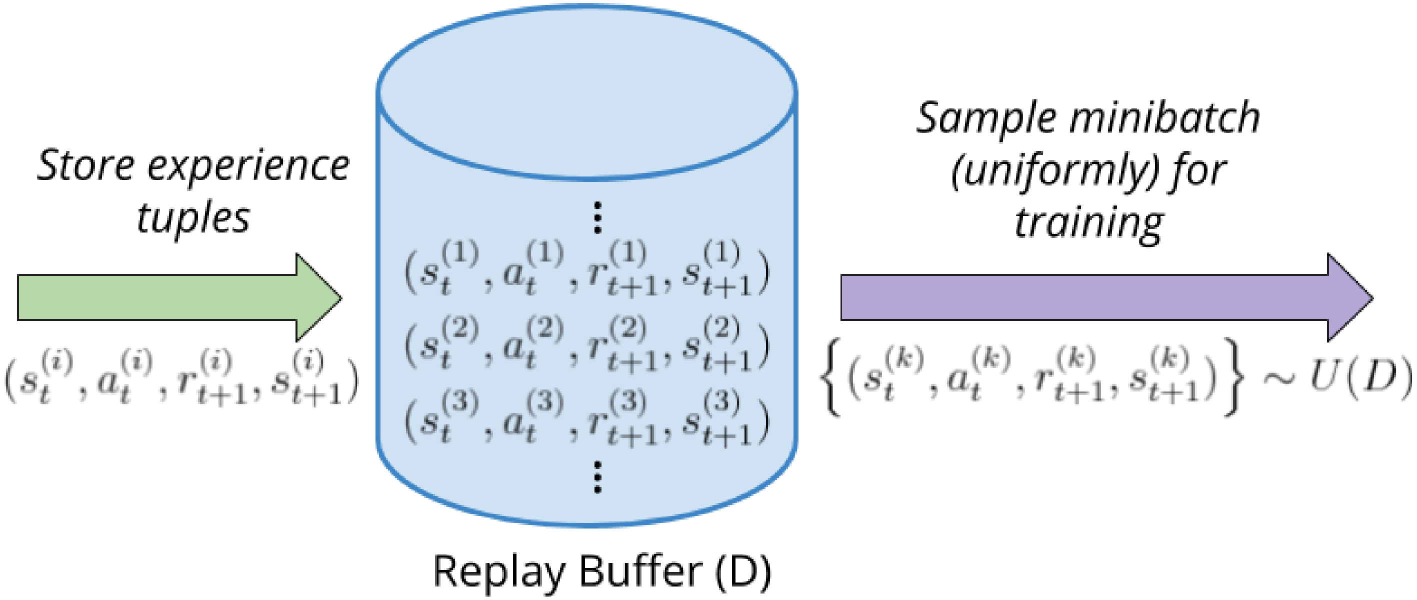


On-Policy Learning



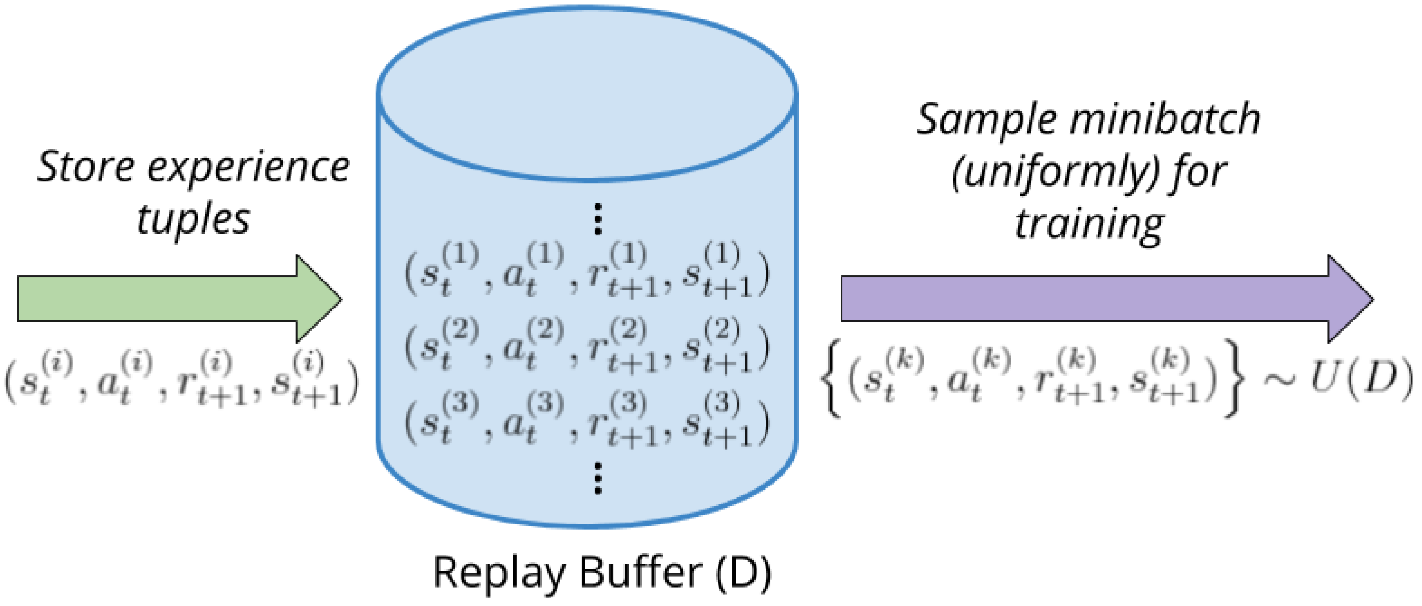
On-Policy Learning

Can we use Replay Buffers with On-Policy learning algorithms (e.g., REINFORCE, Actor-Critic, etc.)?



On-Policy Learning

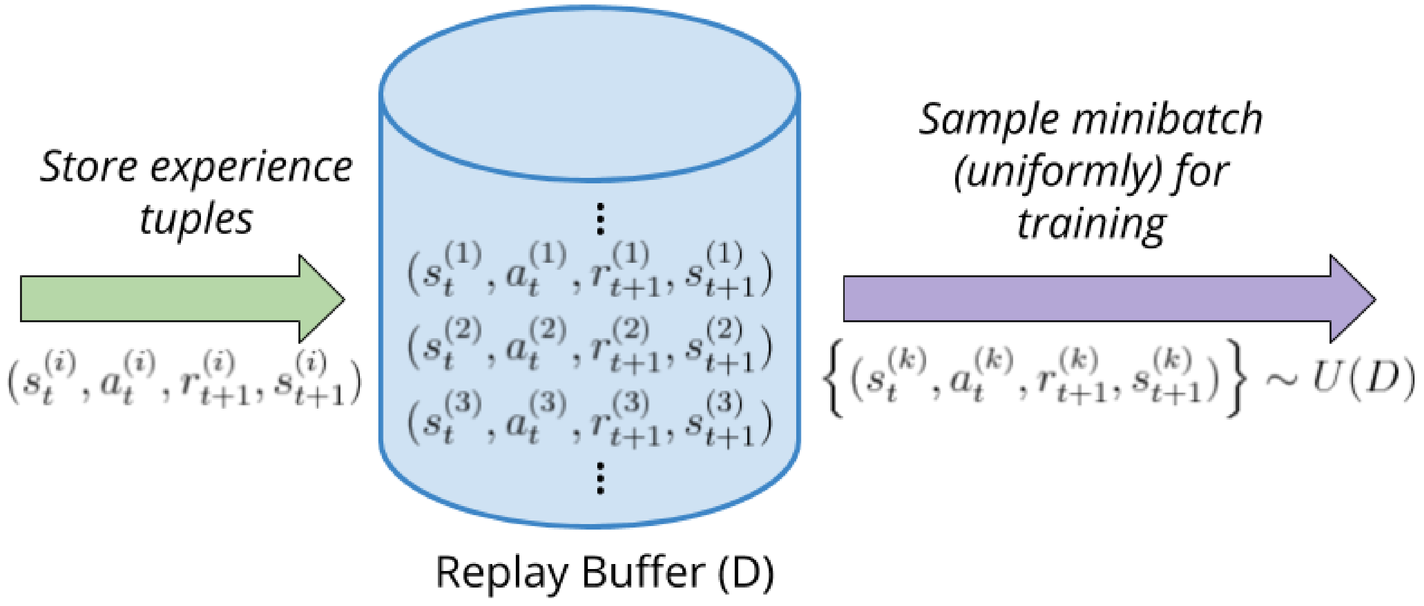
Can we use Replay Buffers with On-Policy learning algorithms (e.g., REINFORCE, Actor-Critic, etc.)?



No! Data in the buffer was collected with an older policy and we can only learn on experiences collected using the current policy...

On-Policy Learning

Can we use Replay Buffers with On-Policy learning algorithms (e.g., REINFORCE, Actor-Critic, etc.)?



No! Data in the buffer was collected with an older policy and we can only learn on experiences collected using the current policy...

But what if we actually could...

Off-Policy Policy Gradient:

Data collected under policy $\beta(a|s)$ (i.e., older version of policy)

We can re-weight our gradient according to the old policy:

$$\rho = \frac{\pi(a|s)}{\beta(a|s)}$$

$$\nabla_\theta J(\theta) = \sum_{(s,a) \in batch} \rho \cdot Q^\pi(s,a) \nabla_\theta \ln \pi(s,a)$$

Actor-Critic with Importance Sampling

Off-Policy Actor Critic: https://arxiv.org/pdf/1205.4839

But what if we actually could...

Off-Policy Policy Gradient:

Data collected under policy $\beta(a|s)$ (i.e., older version of policy)

We can re-weight our gradient according to the old policy:

$$\rho = \frac{\pi(a|s)}{\beta(a|s)}$$

$$\nabla_\theta J(\theta) = \sum_{(s,a)\in batch} \rho \cdot Q^\pi(s,a) \nabla_\theta \ln \pi(s,a)$$

Actor-Critic with Importance Sampling

Store action probabilities $\beta(a|s)$ in replay buffer

Off-Policy Actor Critic: https://arxiv.org/pdf/1205.4839

Trust Region Policy Optimization

Insight: the reason that variance is bad is that it can cause large updates to π_θ

Add a constraint to how large of an update can be applied:

KL-Divergence between old and new policy must be below some hyperparameter Δ

$$D_{KL}(\pi_\theta^{new}(\cdot\,|s)\left|\right|\pi_\theta^{old}(\cdot\,|s)\Big) \leq \Delta$$

$$\rho = \frac{\pi^{new}(a|s)}{\pi^{old}(a|s)}$$

$$J^{TRPO}(\theta) = \mathbb{E}\left[\rho \cdot (r + \gamma V^{\pi^{old}}(s') - V^{\pi^{\mathrm{old}}}(s)\Big)\right]$$

Paper: https://arxiv.org/pdf/1502.05477

Proximal Policy Optimization

TRPO is complicated…

What if instead of constraining the update with KL-Divergence, we clipped the update if it's too big…

$$\rho_{clipped} = clip[\frac{\pi^{new}(a|s)}{\pi^{old}(a|s)}, 1-\epsilon, 1+\epsilon]$$

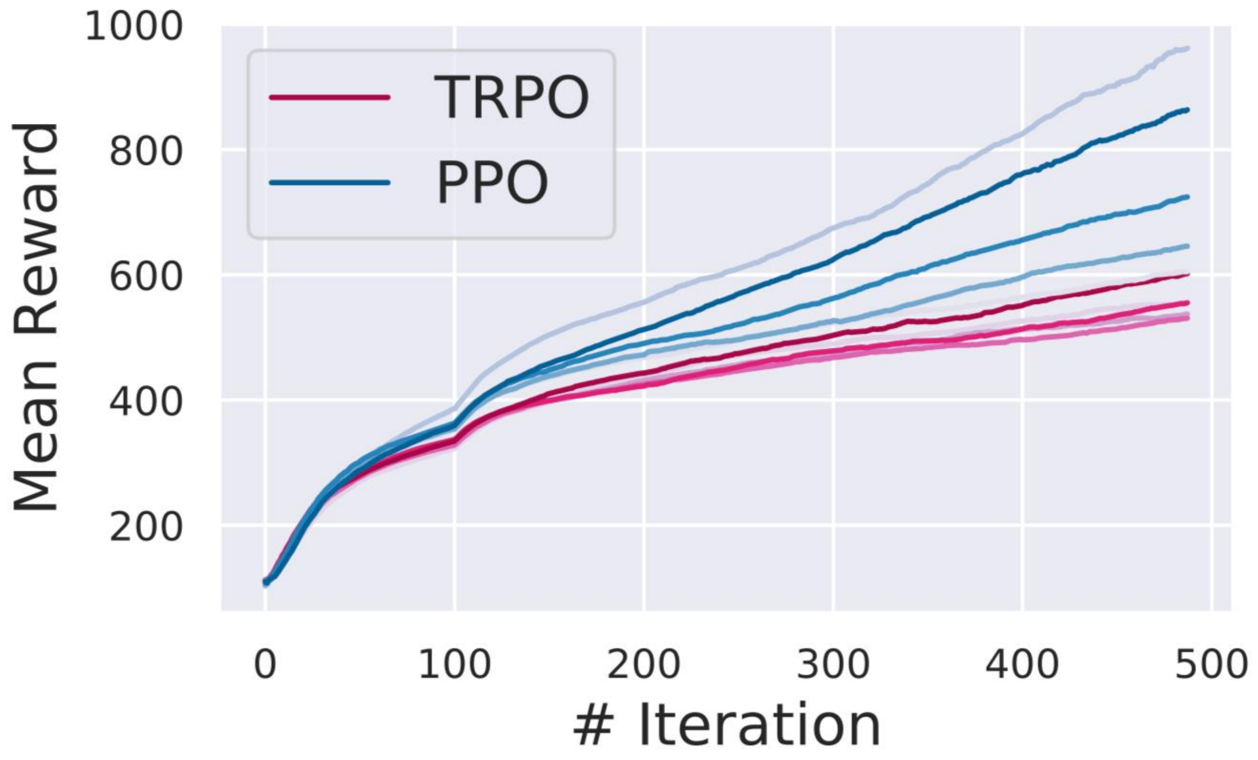
$$J^{PPO}(\theta) = \mathbb{E}[\min(\rho_{clipped} \cdot \left(r + \gamma V^{\pi^{old}}(s') - V^{\pi^{old}}(s)\right), \rho\left(r + \gamma V^{\pi^{old}}(s') - V^{\pi^{old}}(s)\right)]$$

Spinning Up PPO: https://spinningup.openai.com/en/latest/algorithms/ppo.html

PPO

PPO is (basically) State-Of-The-Art (SOTA)

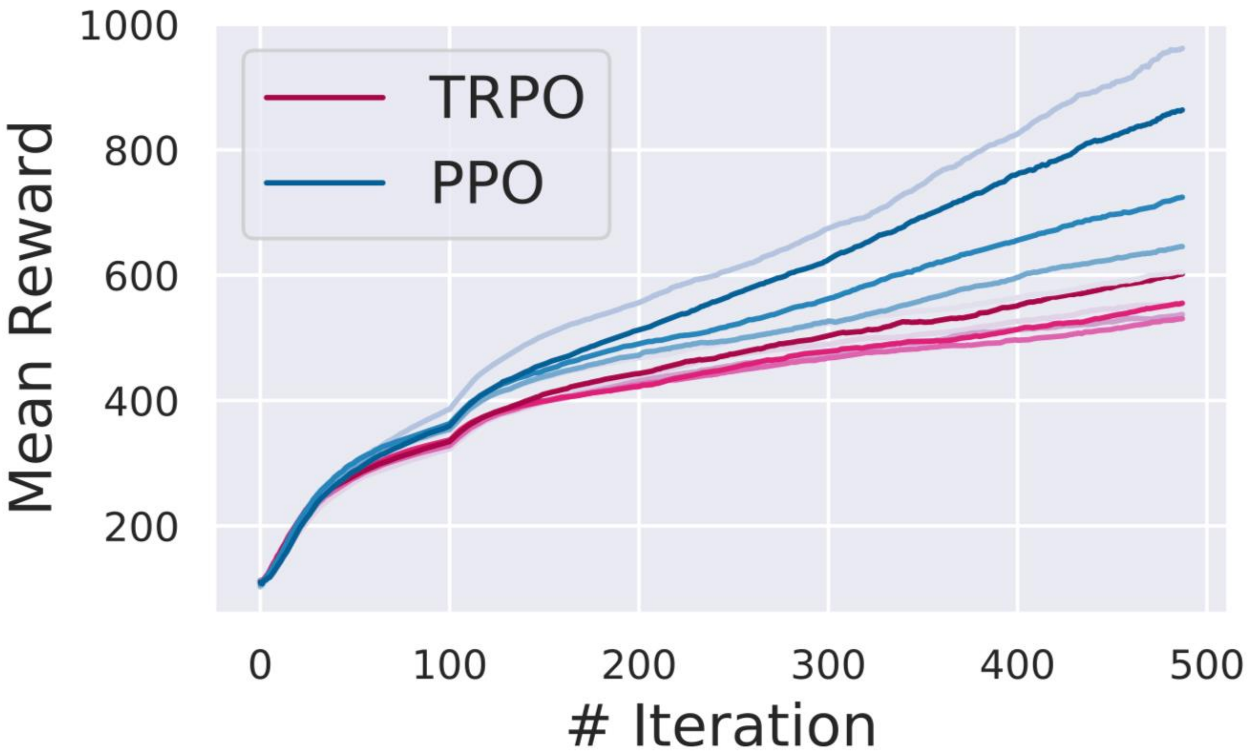
Provides **fast**, **sample-efficient**, and **stable** training



PPO

PPO is (basically) State-Of-The-Art (SOTA)

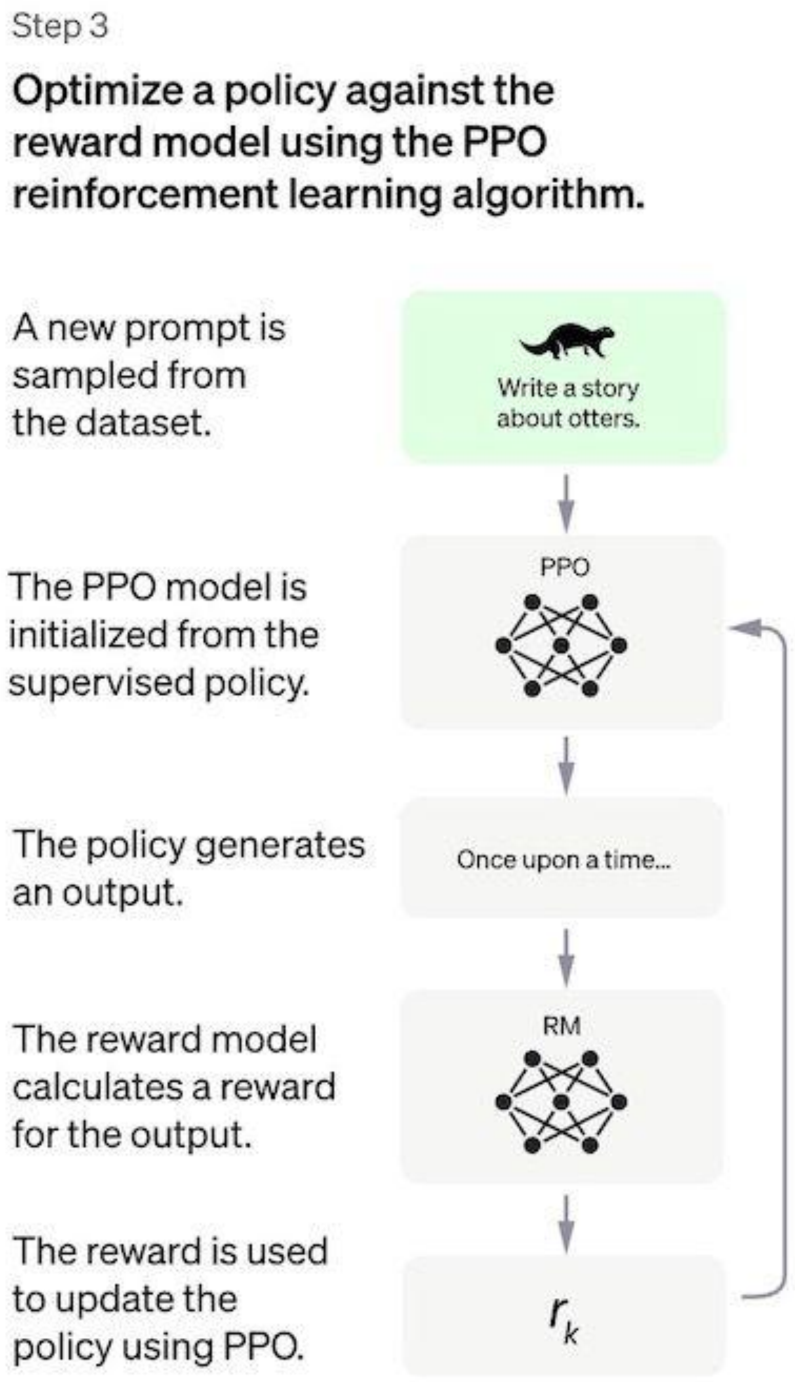
Provides **fast**, **sample-efficient**, and **stable** training



PPO: OpenAI5

PPO

Final phase of training ChatGPT



Group Relative Policy Optimization

PPO still needs a value estimate V^{π}, what if we had another way of estimating V^{π}

More specifically, for each question q, GRPO samples a group of outputs $\{o_1, o_2, \cdots, o_G\}$ from the old policy $\pi_{\theta_{old}}$ and then optimizes the policy model by maximizing the following objective:

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}[q \sim P(Q), \{o_i\}_{i=1}^{G} \sim \pi_{\theta_{old}}(O|q)]$$
$$\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\left\{\min\left[\frac{\pi_\theta(o_{i,t}|q, o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t}|q, o_{i,<t})}\hat{A}_{i,t}, \text{clip}\left(\frac{\pi_\theta(o_{i,t}|q, o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t}|q, o_{i,<t})}, 1-\varepsilon, 1+\varepsilon\right)\hat{A}_{i,t}\right] - \beta\mathbb{D}_{KL}\left[\pi_\theta||\pi_{ref}\right]\right\}, \tag{3}$$

where ε and β are hyper-parameters, and $\hat{A}_{i,t}$ is the advantage calculated based on relative rewards of the outputs inside each group only

Shao et al., DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models, 2023