Policy Gradient Algorithms

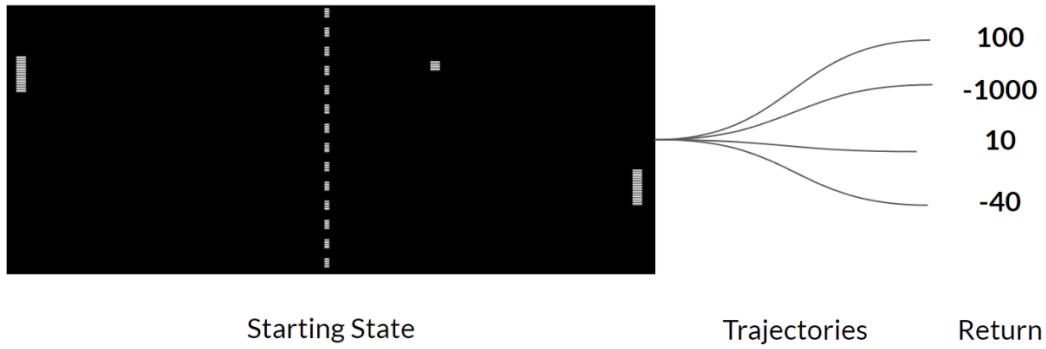
1. Run an episode of your environment and collect all states, actions, and rewards
2. For each timestep, compute G_t and gradient
3. Update parameters based on computed gradients

Algorithm 5 REINFORCE Algorithm

Input: Differentiable policy parameterization $\pi_\theta(a|s)$
Hyperparameters: Learning rate α, discount factor γ
Initialize policy parameters θ randomly
repeat
    Generate an episode $s_0, a_0, r_1, s_1, a_1, \ldots, s_T$ following π_θ
    for $t = 0$ to T **do**
        Compute return $G_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k+1}$
        Compute gradient: $\nabla_\theta \log \pi_\theta(a_t|s_t) \cdot G_t$
    end for
    Update policy parameters: $\theta \leftarrow \theta + \alpha \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot G_t$
until convergence

Issues with REINFORCE:
High Variance: what if you get (un)lucky in your episode?

We only collect one episode's worth of information



Algorithm 5 REINFORCE Algorithm

Input: Differentiable policy parameterization $\pi_\theta(a|s)$
Hyperparameters: Learning rate α, discount factor γ
Initialize policy parameters θ randomly
repeat
    Generate an episode $s_0, a_0, r_1, s_1, a_1, \ldots, s_T$ following π_θ
    for $t = 0$ to T **do**
        Compute return $G_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k+1}$
        Compute gradient: $\nabla_\theta \log \pi_\theta(a_t|s_t) \cdot G_t$
    end for
    Update policy parameters: $\theta \leftarrow \theta + \alpha \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot G_t$
until convergence

Issues with REINFORCE:
High Variance: what if you get (un)lucky in your episode?

We only collect one episode's worth of information

How can we fix it?
Option 1: Collect more than one episode

Algorithm 5 REINFORCE Algorithm

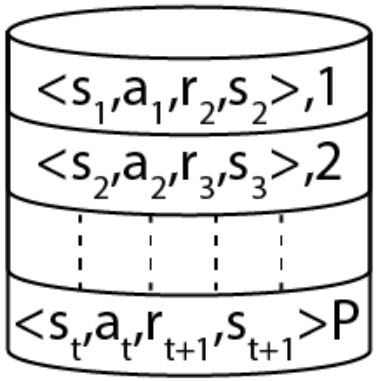
Input: Differentiable policy parameterization $\pi_\theta(a|s)$
Hyperparameters: Learning rate α, discount factor γ
Initialize policy parameters θ randomly
repeat
    Generate an episode $s_0, a_0, r_1, s_1, a_1, \ldots, s_T$ following π_θ
    for $t = 0$ to T **do**
        Compute return $G_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k+1}$
        Compute gradient: $\nabla_\theta \log \pi_\theta(a_t|s_t) \cdot G_t$
    end for
    Update policy parameters: $\theta \leftarrow \theta + \alpha \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot G_t$
until convergence

Issues with REINFORCE:
High Variance: what if you get (un)lucky in your episode?

We only collect one episode's worth of information

How can we fix it?
Option 1: Collect more than one episode

Replay Buffer:
Store state, action, reward, next state pairs for multiple episodes



Algorithm 5 REINFORCE Algorithm

Input: Differentiable policy parameterization $\pi_\theta(a|s)$
Hyperparameters: Learning rate α, discount factor γ
Initialize policy parameters θ randomly
repeat
    Generate an episode $s_0, a_0, r_1, s_1, a_1, \ldots, s_T$ following π_θ
    for $t = 0$ to T **do**
        Compute return $G_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k+1}$
        Compute gradient: $\nabla_\theta \log \pi_\theta(a_t|s_t) \cdot G_t$
    end for
    Update policy parameters: $\theta \leftarrow \theta + \alpha \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot G_t$
until convergence

Issues with REINFORCE:
High Variance: what if you get (un)lucky in your episode?

We only collect one episode's worth of information

How can we fix it?
Option 1: Collect more than one episode
Option 2: Reduce Variance of return estimate

Algorithm 5 REINFORCE Algorithm

Input: Differentiable policy parameterization $\pi_\theta(a|s)$
Hyperparameters: Learning rate α, discount factor γ
Initialize policy parameters θ randomly
repeat
    Generate an episode $s_0, a_0, r_1, s_1, a_1, \ldots, s_T$ following π_θ
    for $t = 0$ to T **do**
        Compute return $G_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k+1}$
        Compute gradient: $\nabla_\theta \log \pi_\theta(a_t|s_t) \cdot G_t$
    end for
    Update policy parameters: $\theta \leftarrow \theta + \alpha \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot G_t$
until convergence

Baseline Functions

We derived the following policy gradient theorem: $$\nabla J(\theta) \propto \sum_s \sum_a q_\pi(s,a) \nabla \pi(a|s,\theta)$$

A Baseline function is any function that only depends on s

Adding a "Baseline Function" doesn't change the gradient: $$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a \left(q_\pi(s,a) - b(s)\right) \nabla \pi(a|s,\theta)$$

By linearity of sum and gradient operators $$\sum_a b(s) \nabla \pi(a|s,\theta) = b(s) \nabla \sum_a \pi(a|s,\theta)$$

$\pi(a|s,\theta)$ are probabilities, so they must add to 1 $$b(s) \nabla \sum_a \pi(a|s,\theta) = b(s) \nabla 1 = 0$$

The Advantage Function

- When learning policies, we are learning stochastic policies
- There will be some probability, we take an action other than the "best" action

$$\pi(s,a) \propto e^{Q^{\pi}(s,a)}$$

$$V^{\pi}(s) = r + \gamma \sum_{a} \pi(s,a) Q(s,a)$$

The advantage function tells us how much better an action was than our expected value.

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

What's the best baseline function to use?

Do you know of any functions that only depend on state?

Algorithm 6 REINFORCE Algorithm with Baseline

Input: Differentiable policy parameterization $\pi_\theta(a|s)$ and baseline function $V_w(s)$
Hyperparameters: Learning rates α_θ, α_w, discount factor γ
Initialize policy parameters θ and baseline parameters w randomly
repeat
    Generate an episode $s_0, a_0, r_1, s_1, a_1, \ldots, s_T$ following π_θ
    for $t = 0$ to T **do**
        Compute return $G_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k+1}$
        Compute advantage: $A_t = G_t - V_w(s_t)$
        Update baseline: $w \leftarrow w + \alpha_w \nabla_w [V_w(s_t) - G_t]^2$
        Compute policy gradient: $\nabla_\theta \log \pi_\theta(a_t|s_t) \cdot A_t$
    end for
    Update policy parameters: $\theta \leftarrow \theta + \alpha_\theta \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot A_t$
until convergence

Actor Critic Algorithm: Train both a policy and an estimate of Q at the same time

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 $t = 1 \ldots T$: **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:

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

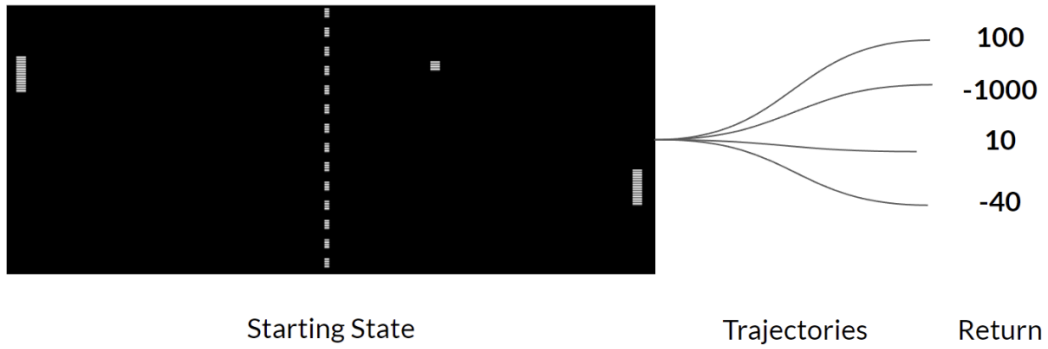
Move to a $\leftarrow a'$ and s $\leftarrow s'$

end for

Addressing High Variance of REINFORCE

- Collecting additional data reduces the variance of our estimates
- Adding in a Baseline (critic) can reduce variance

Instead of fixing the cause, what if we fixed the effect of high variance?



Why is high variance bad? Because it might cause large updates to our model for rare events.

Clipping

- Large updates in weights can cause the training process to be very unstable
- So what if we just put a cap on the size of these updates?
- We can *clip* the size of the updates (the gradient)

One final insight: clip not based on how large a parameter update is, but rather how big of a change that update causes in our **policy**.

numpy.clip

numpy.clip(*a, a_min=<no value>, a_max=<no value>, out=None, *, min=<no value>, max=<no value>, **kwargs*) [source]

Clip (limit) the values in an array.

Given an interval, values outside the interval are clipped to the interval edges. For example, if an interval of `[0, 1]` is specified, values smaller than 0 become 0, and values larger than 1 become 1.

Equivalent to but faster than `np.minimum(a_max, np.maximum(a, a_min))`.

PPO

- Keep track of current policy and policy at previous iteration $\pi_{\theta_i}, \pi_{\theta_{i-1}}$
- Collect experiences (i.e., run some episodes)
- Update policy (i.e., compute π_{θ_i+1}) based on the ratio $r_i = \frac{\pi_{\theta_i}(s,a)}{\pi_{\theta_{i-1}}(s,a)}$
- If r_i is too large (or negative), use a clipped version to be capped at some ϵ

PPO is (basically) state of the art

Algorithm 7 Proximal Policy Optimization (PPO) with Clipped Objective

Input: Initial policy parameters θ_0, value function parameters ϕ_0
Hyperparameters: Clipping parameter ϵ, learning rates α_θ, α_ϕ, discount factor γ, epochs K, mini-batch size M
for $i = 0, 1, 2, \ldots$ **do**
    Collect set of trajectories $\mathcal{D}_i = \{\tau_j\}$ by running policy π_{θ_i} in the environment
    Compute rewards-to-go $\hat{R}_t$
    Compute advantage estimates $\hat{A}_t$
    Compute policy ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_i}(a_t|s_t)}$
    for $k = 0, 1, 2, \ldots, K-1$ **do**
        Sample mini-batch of size M from $\mathcal{D}_i$
        Compute policy objective: $L^{\text{CLIP}}(\theta) = \frac{1}{|M|} \sum_{(s_t,a_t)\in M} \min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)$
        Compute value function loss: $L^{\text{VF}}(\phi) = \frac{1}{|M|} \sum_{(s_t)\in M} (V_\phi(s_t) - \hat{R}_t)^2$
        Update policy parameters: $\theta_{i+1} = \theta_i + \alpha_\theta \nabla_\theta L^{\text{CLIP}}(\theta)|_{\theta=\theta_i}$
        Update value function parameters: $\phi_{i+1} = \phi_i - \alpha_\phi \nabla_\phi L^{\text{VF}}(\phi)|_{\phi=\phi_i}$
    end for
end for

Step 1

Collect demonstration data and train a supervised policy.

A prompt is sample from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3.5 with supervised learning.

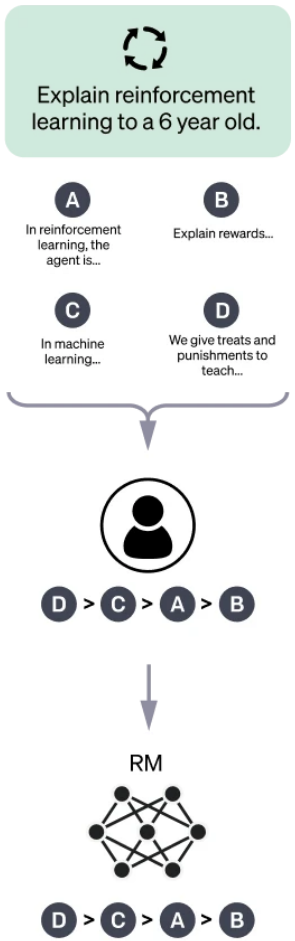
Step 2

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 the PPO reinforcement learning algorithm.

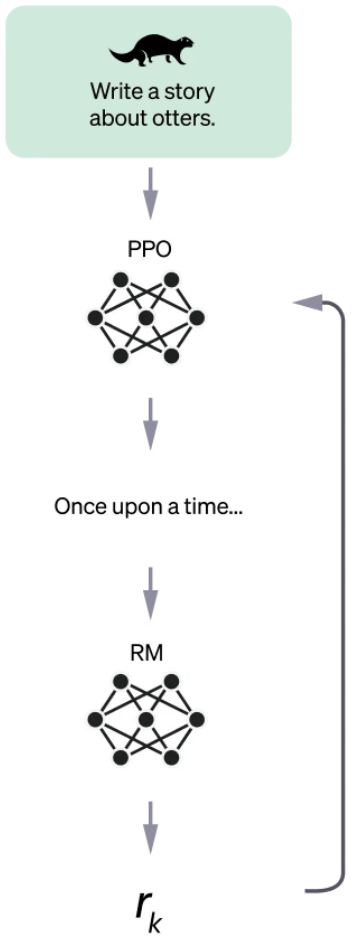
A new prompt is sampled from the dataset.

The PPO model is initialized from the supervised policy.

The policy generates an output.

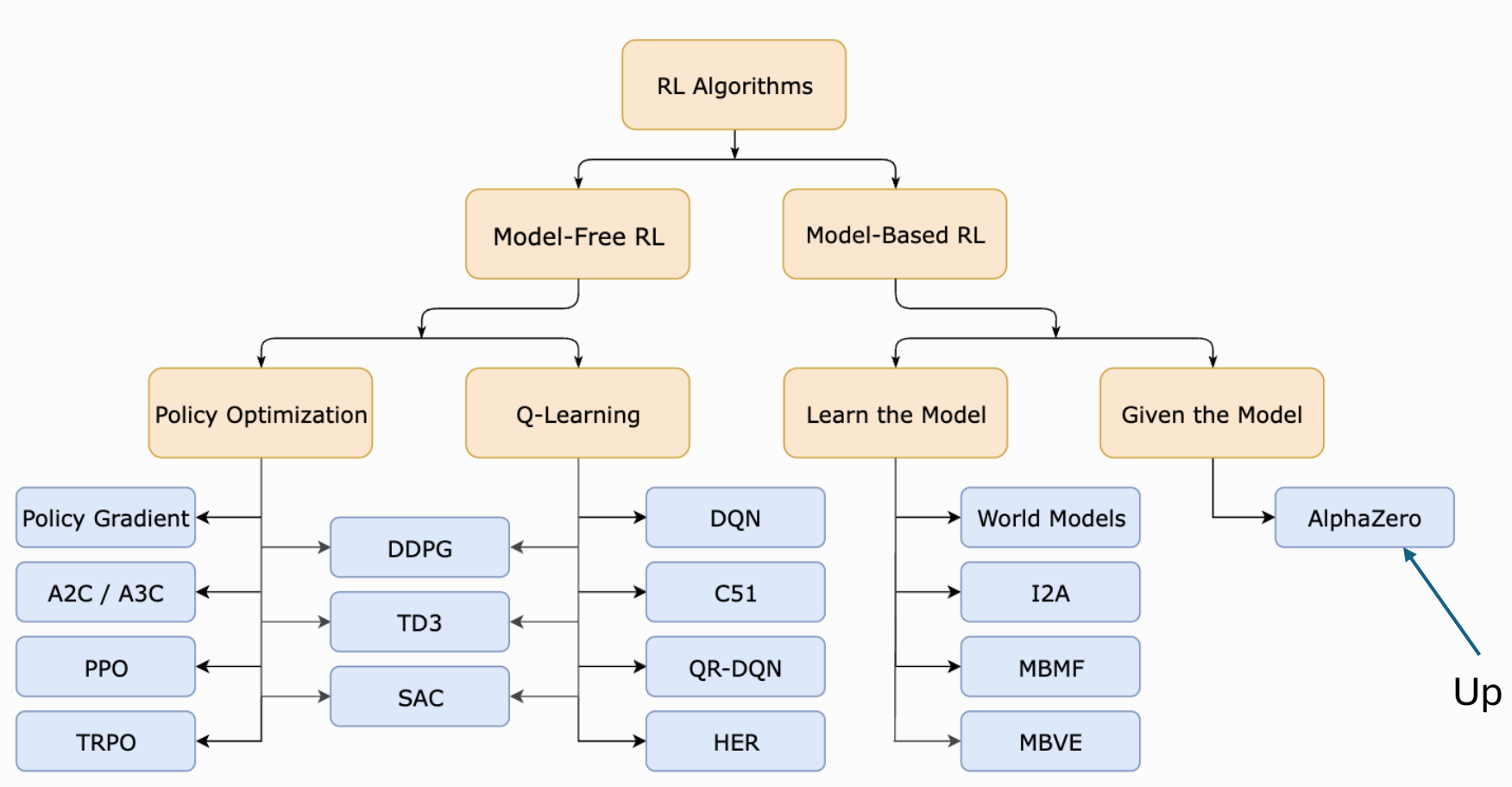
The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.



Source: OpenAI

RL Algorithms



Source: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

Final Project: Go Playing Agents

Why Go?

- Simple rules, complex strategy
- Difficulty of search problem can be controlled (play on a smaller board)
- At the forefront of research in AI in the past 10 years