CSCI 1470

Deep Learning

Eric Ewing

Day 21: TRPO and PPO

Tuesday, 11/18/25

DL Day Info

Will take place Thursday, December 11th

Poster sessions will take place in the third floor atrium of the CIT

Two separate sections:

1) 9:30-10:45
2) 11:00-12:15

If you can't make an entire section, I can visit your poster for the part of the section when you are there.

Actor Critic Algorithms Review

Idea: Learn policy (actor) and value function (critic) side-by-side

Compute Temporal Difference Error (TD-Error) to train Value network

Compute policy gradient update to train actor

One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} I \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

Fig 13.3. *One step actor critic*

Source: Sutton and Barto, Reinforcement Learning

Advantage Actor Critic

Instead of using V(s), use A(s, a) = Q(s, a) – V(s)

A(s, a): What is the advantage of taking action *a* over the average value of the state

Learn estimate of V(s)

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

One-step Actor–Critic (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} I \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

Fig 13.3. *One step actor critic*

Actor Critic Algorithms

Note: Wikipedia uses R_i for returns

- $\sum_{0 \leq i \leq T}(\gamma^i R_i)$.
- $\gamma^j \sum_{j \leq i \leq T}(\gamma^{i-j} R_i)$: the **REINFORCE** algorithm.
- $\gamma^j \sum_{j \leq i \leq T}(\gamma^{i-j} R_i) - b(S_j)$: 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: Wikipedia

On-Policy and 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

Review: On + Off-Policy Learning

	On-Policy	Off Policy
Summary	Learns policy/value function based on policy used during training	Learns policy independent of policy used to collect experiences during training
Algorithms	SARSA, Policy Gradient, Actor Critic, PPO	Q-Learning, Off-policy Actor-Critic, Deep Deterministic Policy Gradient (DDPG)

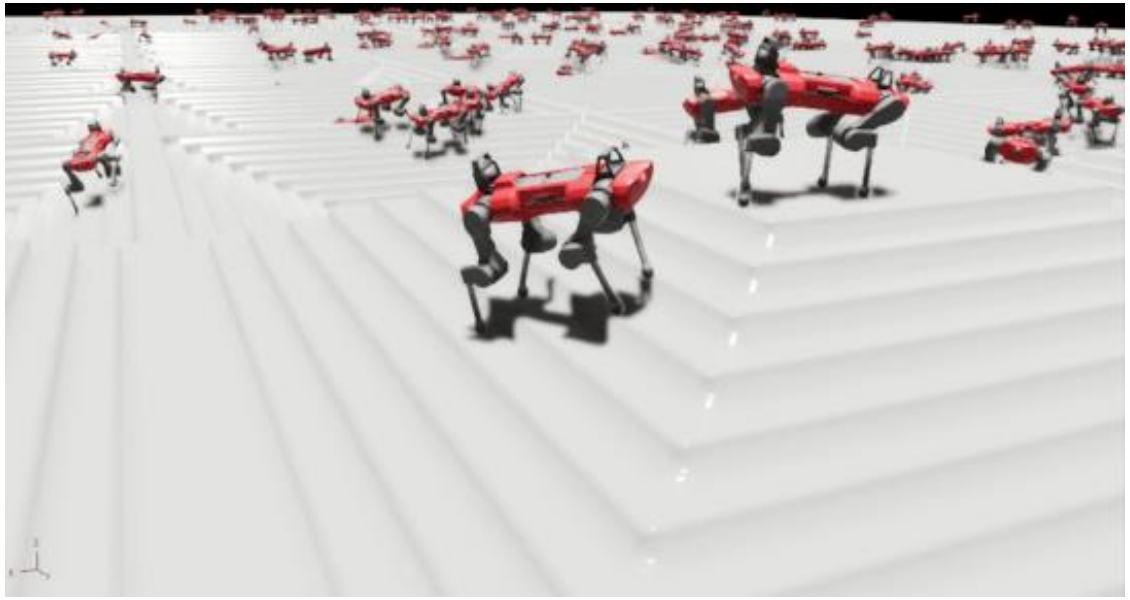
Off-Policy Learning

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)

Maybe we shouldn't throw away useful data immediately…



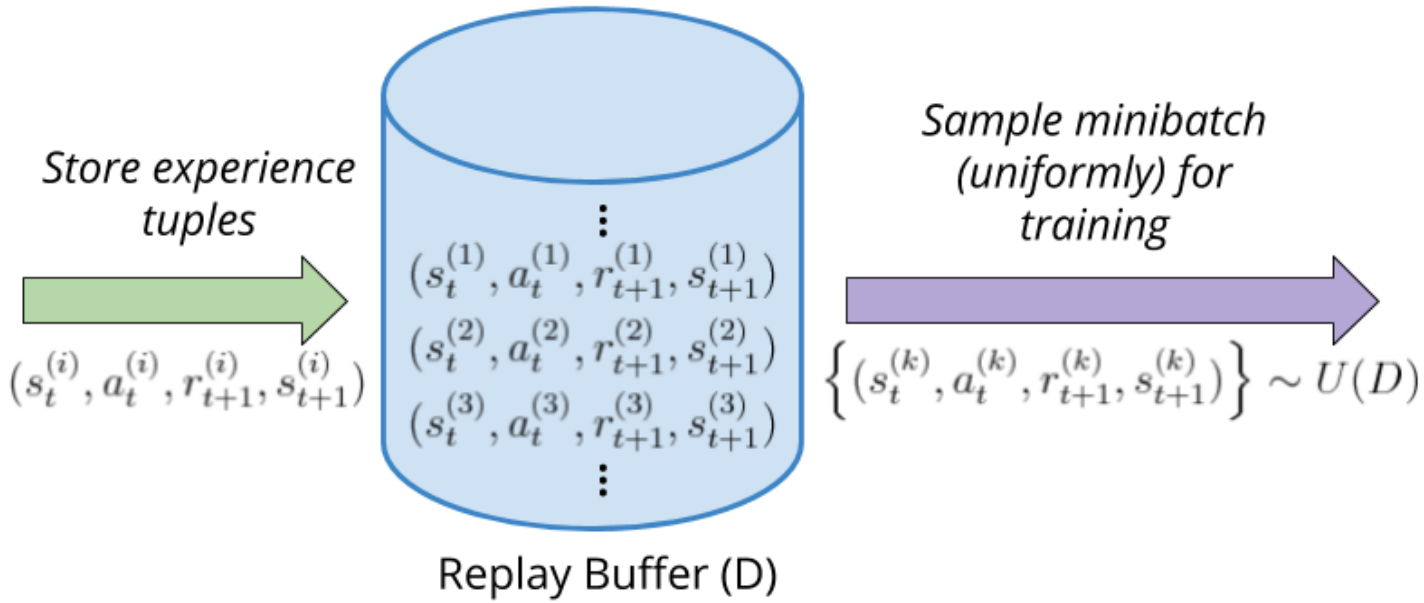
Isaac Gym

Experience Replay and Replay Buffers

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

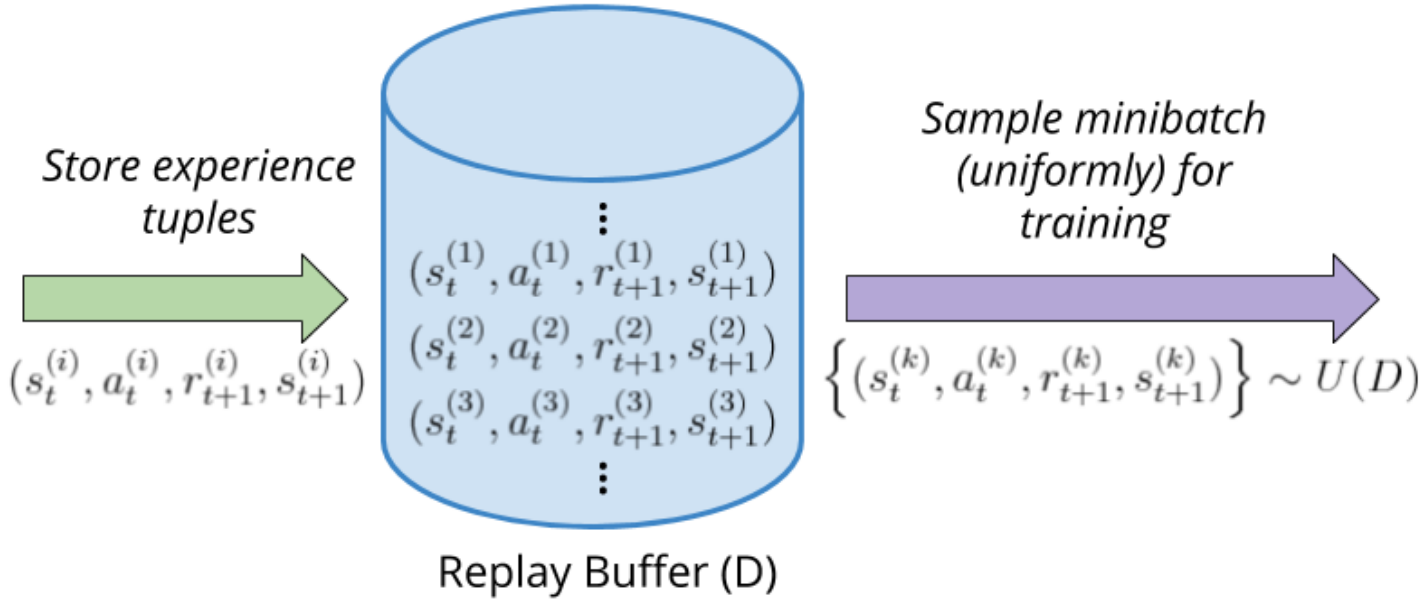
As you collect new experiences, remove oldest experiences from buffer

To train model, sample batch of data from buffer



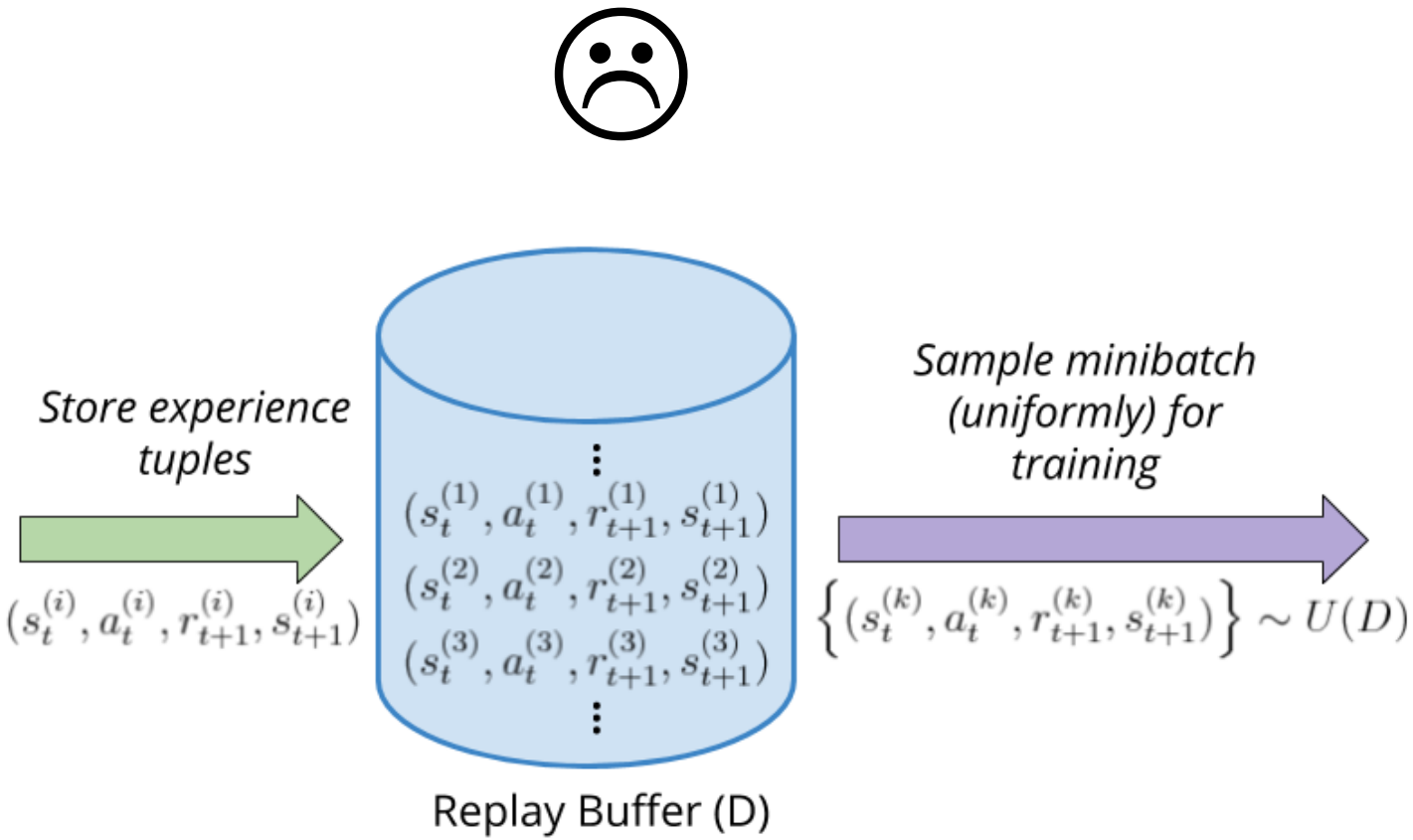
On-Policy Learning

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.)?



Not Really! 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:

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

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

How much should we weigh each experience?

Actor-Critic with Importance Sampling

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

Importance Sampling

We'd like to calculate the expected value of f(x), with x drawn from some probability distribution *p*. *However,* distribution p may be hard to sample from.

$$E_{(x\sim p)}[f(x)]$$

$$E_{(x\sim p)}[f(x)] = \int f(x)p(x)\,dx$$

$$E_{(x\sim p)}[f(x)] = \int f(x)\frac{p(x)}{q(x)}q(x)\,dx = E_{(x\sim q)}[p(x)/q(x)\cdot f(x)]$$

Importance Sampling

We can compute our original objective by including an importance sampling ratio that corrects for the fact that x is drawn from a different distribution.

$$E_{(x\sim p)}[f(x)] = E_{(x\sim q)}[p(x)/q(x) \cdot f(x)]$$

A (short) Journey Into Convex Optimization Theory

Source and further reading: Parikh and Boyd, Proximal Algorithms. 2013.

Proximal Optimization

Let f be a convex function:

$$f: \boldsymbol{\mathcal{R}} \to \boldsymbol{\mathcal{R}}$$

The *proximal operator* $\mathbf{prox}_f : \mathbf{R}^n \to \mathbf{R}^n$ of f is defined by

$$\mathbf{prox}_{\lambda f}(v) = \underset{x}{\operatorname{argmin}} \left(f(x) + (1/2\lambda)\|x - v\|_2^2 \right).$$

Takes in a vector

Outputs a vector

Compromises between minimizing $f(x)$ and the right term

What's the intuition?

Limit the size of the step

Minimize our objective

$$\mathbf{prox}_{\lambda f}(v) = \underset{x}{\operatorname{argmin}} \left(f(x) + (1/2\lambda)\|x - v\|_2^2 \right).$$

Our original objective is to minimize $f(x)$, we'd like to take a step towards an optimal solution

However, *f* may not be differentiable and we may have constraints on the domain of *f*.

Idea: take a (small) step in the direction of your objective or towards the feasible region

The Proximal Operator

$$\mathbf{prox}_{\lambda f}(v) = \underset{x}{\operatorname{argmin}} \left(f(x) + (1/2\lambda)\|x - v\|_2^2 \right).$$

Blue points: *u's*
Red points: *x's*

Applying the proximal operator moves from blue to red

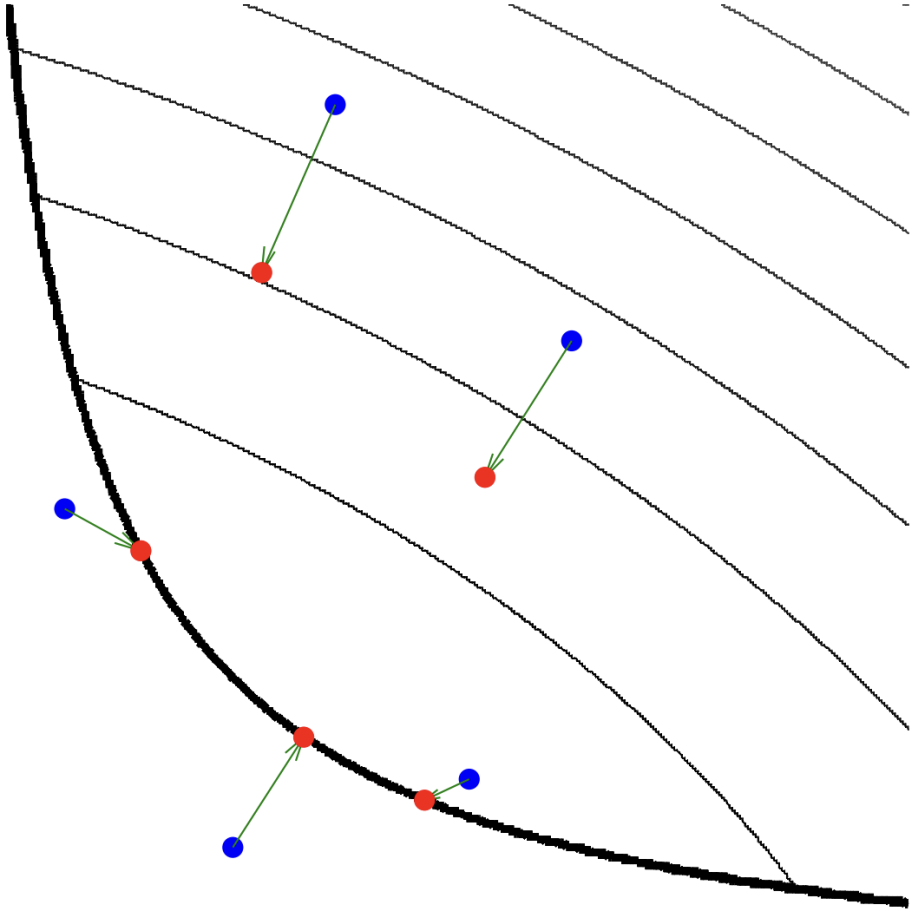


Figure 1.1: Evaluating a proximal operator at various points.

Relationship to Gradient Descent

If f is differentiable and λ is small, then the proximal operator is approximately a step of gradient descent

$$\mathbf{prox}_{\lambda f}(v) \approx v - \lambda \nabla f(v)$$

Another Example Operator

$$\mathbf{prox}_{\lambda f}(v) = \underset{x}{\operatorname{argmin}} \left(f(x) + (1/2\lambda)\|x - v\|_2^2 \right).$$

proximal operator of an indicator function of a convex set is projection:

$$I_{\mathcal{C}}(x) = \begin{cases} 0 & x \in \mathcal{C} \\ +\infty & x \notin \mathcal{C} \end{cases}$$

$$\mathbf{prox}_{\lambda I_{\mathcal{C}}}(v) = \Pi_{\mathcal{C}}(v) = \underset{x \in \mathcal{C}}{\operatorname{argmin}} \|x - v\|_2$$

Proximal Optimization Algorithm

A Proximal Optimization algorithm is one that repeatedly applies the proximal operator

$$\mathbf{prox}_{\lambda f}(v) = \underset{x}{\operatorname{argmin}} \left(f(x) + (1/2\lambda)\|x - v\|_2^2 \right).$$

Trust Region Problems

A *trust region problem* has the form

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & \|x - v\|_2 \leq \rho, \end{array}$$

Where ρ is the radius of the *trust region*

These problems typically arise when *f(x)* is an approximation to some more complex function

f(x) can be a Taylor approximation of some “hard-to-optimize” function
Taylor approximations are good approximations **locally**.

The proximal problem

$$\text{minimize} \quad f(x) + (1/2\lambda)\|x - v\|_2^2$$

An Ideal RL Algorithm

So long as π_i and π_{i+1} are "close", this algorithm provides a **guarantee** that π_{i+1} has better expected returns than π_i

How do we solve that maximization problem?

Algorithm 1 Policy iteration algorithm guaranteeing non-decreasing expected return η

Initialize π_0.

for $i = 0, 1, 2, \ldots$ until convergence **do**

Compute all advantage values $A_{\pi_i}(s, a)$.

Solve the constrained optimization problem

$$\pi_{i+1} = \arg\max_{\pi} \left[L_{\pi_i}(\pi) - C D_{\mathrm{KL}}^{\max}(\pi_i, \pi) \right]$$

where $C = 4\epsilon\gamma/(1-\gamma)^2$

and $L_{\pi_i}(\pi) = \eta(\pi_i) + \sum_s \rho_{\pi_i}(s) \sum_a \pi(a|s) A_{\pi_i}(s, a)$

end for

$V^{\pi}(s_0)$ under new policy

Probability of reaching a state s Under new policy

Probability of taking an action with old policy

Advantage of that action under old policy

Trust Region Policy Optimization

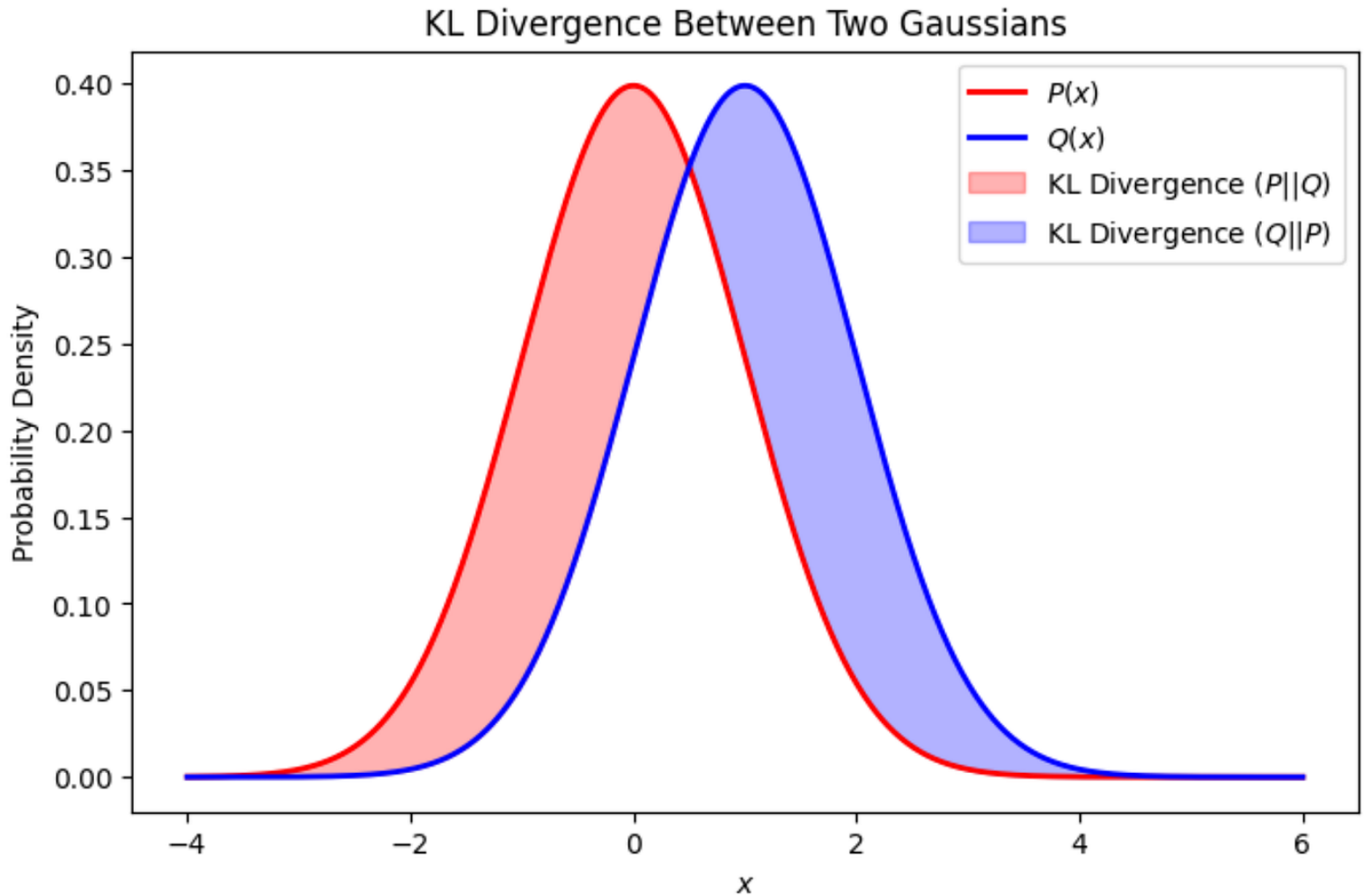
Want to solve:

$$\underset{\theta}{\text{maximize}} \left[L_{\theta_{\text{old}}}(\theta) - C D_{\text{KL}}^{\max}(\theta_{\text{old}}, \theta) \right].$$

Which is equivalent to a hard constraint problem for *some* δ

$$\underset{\theta}{\text{maximize}}\, L_{\theta_{\text{old}}}(\theta)$$
$$\text{subject to } D_{\text{KL}}^{\max}(\theta_{\text{old}}, \theta) \leq \delta.$$

KL Divergence



Source: https://medium.com/@yian.chen261/introduction-to-kullback-leibler-divergence-2d76979d1d8c

$$\underset{\theta}{\text{maximize}}\, L_{\theta_{\text{old}}}(\theta)$$

$$\text{subject to } D_{\text{KL}}^{\max}(\theta_{\text{old}}, \theta) \leq \delta.$$

The maximum KL divergence limits how different the two policies can be at **any** state.

That is a constraint for every state in the state space… (that's a lot)

It's easier to work with the expected KL Divergence as an approximation for the true constraint

$$\overline{D}_{\text{KL}}^{\rho}(\theta_1, \theta_2) := \mathbb{E}_{s\sim\rho}\left[D_{\text{KL}}(\pi_{\theta_1}(\cdot|s) \parallel \pi_{\theta_2}(\cdot|s))\right]$$

Trust Region Policy Optimization

$$\underset{\theta}{\text{maximize}} \sum_s \rho_{\theta_{\text{old}}}(s) \sum_a \pi_\theta(a|s) A_{\theta_{\text{old}}}(s,a)$$

$$\text{subject to } \overline{D}_{\text{KL}}^{\rho_{\theta_{\text{old}}}}(\theta_{\text{old}}, \theta) \leq \delta.$$

Unfortunately, solving this constrained optimization (with the conjugate gradient algorithm) problem involves computing the Hessian matrix (matrix of second derivatives), which is very expensive.

TRPO

	B. Rider	*Breakout*	*Enduro*	*Pong*	*Q*bert*	*Seaquest*	*S. Invaders*
Random	354	1.2	0	−20.4	157	110	179
Human (Mnih et al., 2013)	7456	31.0	368	−3.0	18900	28010	3690
Deep Q Learning (Mnih et al., 2013)	4092	168.0	470	20.0	1952	1705	581
UCC-I (Guo et al., 2014)	5702	380	741	21	20025	2995	692
TRPO - single path	1425.2	10.8	534.6	20.9	1973.5	1908.6	568.4
TRPO - vine	859.5	34.2	430.8	20.9	7732.5	788.4	450.2

Competitive with Deep-Q Learning and Human level performance in some Atari domains

Proximal Policy Optimization

Let $r_t(\theta)$ denote the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$, so $r(\theta_{\text{old}}) = 1$. TRPO maximizes a "surrogate" objective

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right] = \hat{\mathbb{E}}_t\left[r_t(\theta)\hat{A}_t\right]. \tag{6}$$

The main objective we propose is the following:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

The original objective of TRPO

Clipped probability ratio

PPO Optimizes a lower bound of the TRPO objective without the need for constrained optimization methods!

Source: Schulman et al., "Proximal Policy Optimization Algorithms"

PPO

Algorithm 1 PPO, Actor-Critic Style

for iteration=$1, 2, \ldots$ **do**
  for actor=$1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

PPO

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

When $r_t(\theta) > 1$, then the new policy is **more likely** to select the current action than the old policy

When $0 \leq r_t(\theta) \leq 1$, then the new policy is **less likely** to select the current action than the old policy

$r_t(\theta)$ estimates the divergence between the old and new policy

When that divergence is large (i.e., far from 1), we are no longer confident in our estimates of $\hat{A}_t$.

Clipping makes our updates conservative.

PPO and TRPO are on-policy learning algorithms!

Do not be fooled by the presence old and new policies, the old policy refers to the policy that collected the data and the new policy is the decision variable of the optimization problem

PPO is an approximation of TRPO, but outperforms it!

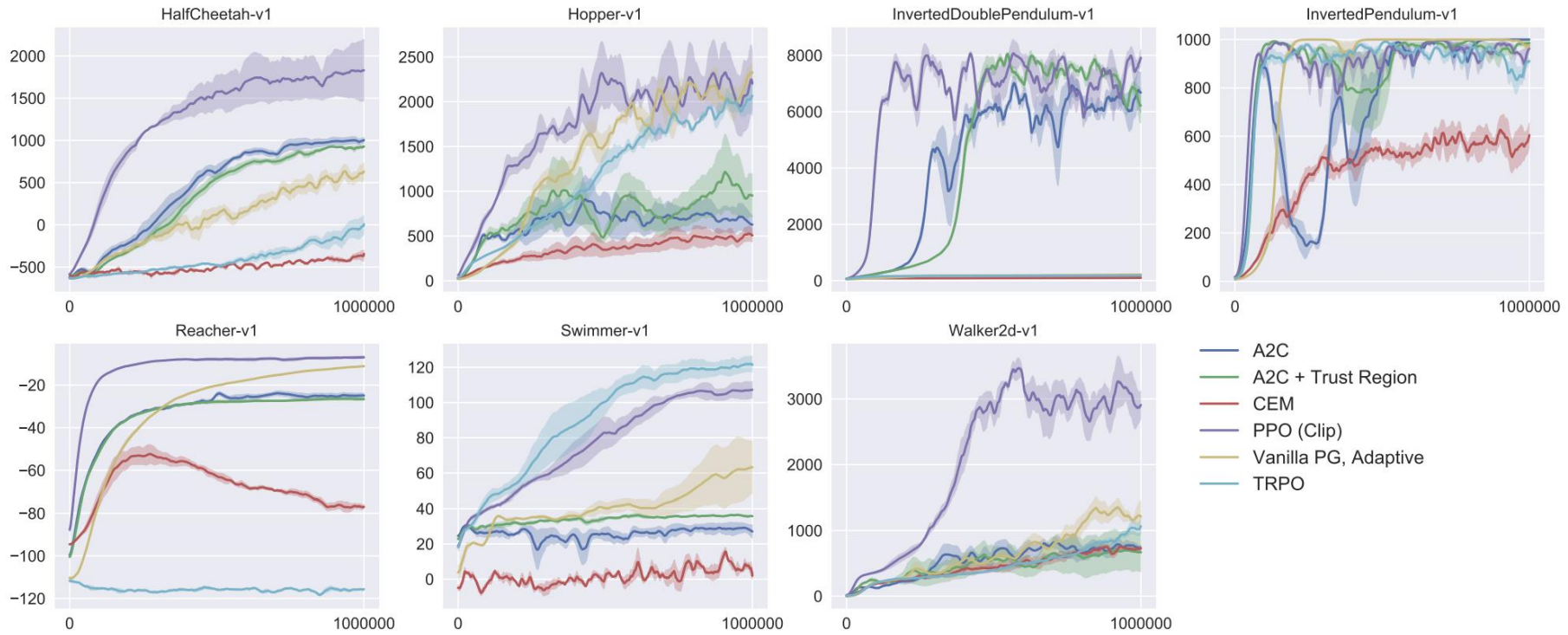
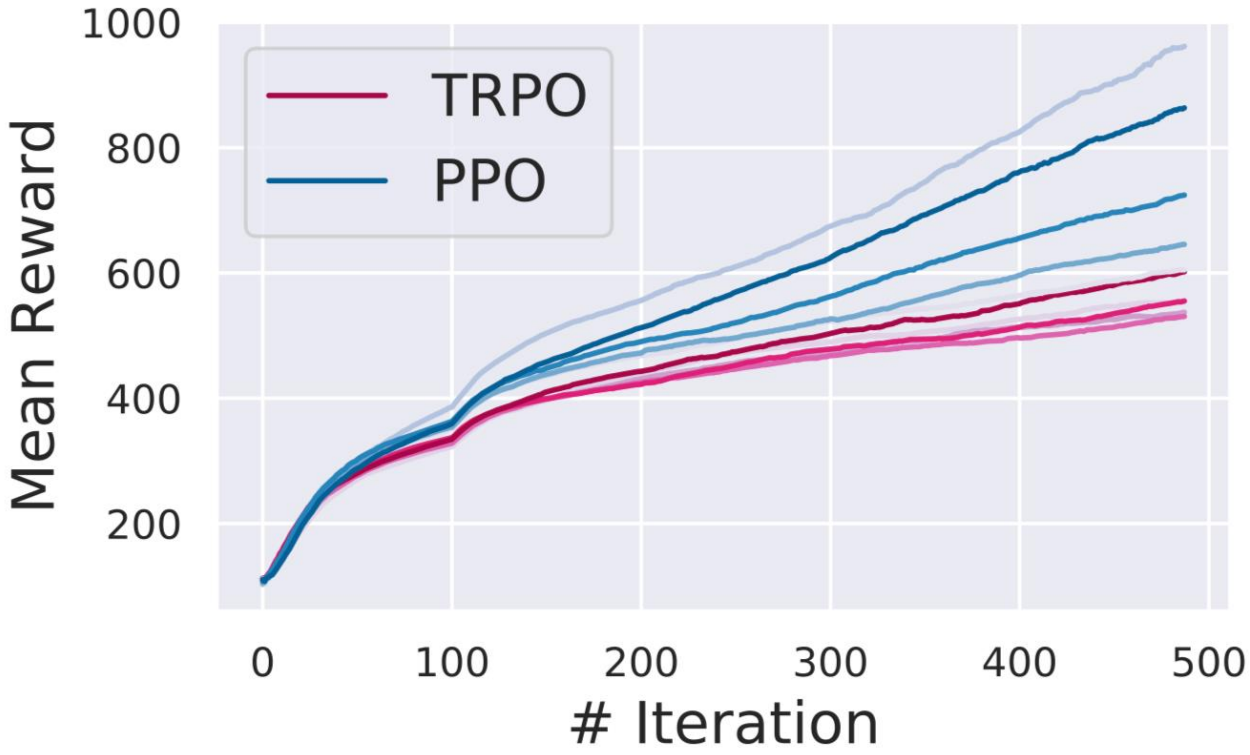


Figure 3: Comparison of several algorithms on several MuJoCo environments, training for one million timesteps.

PPO

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

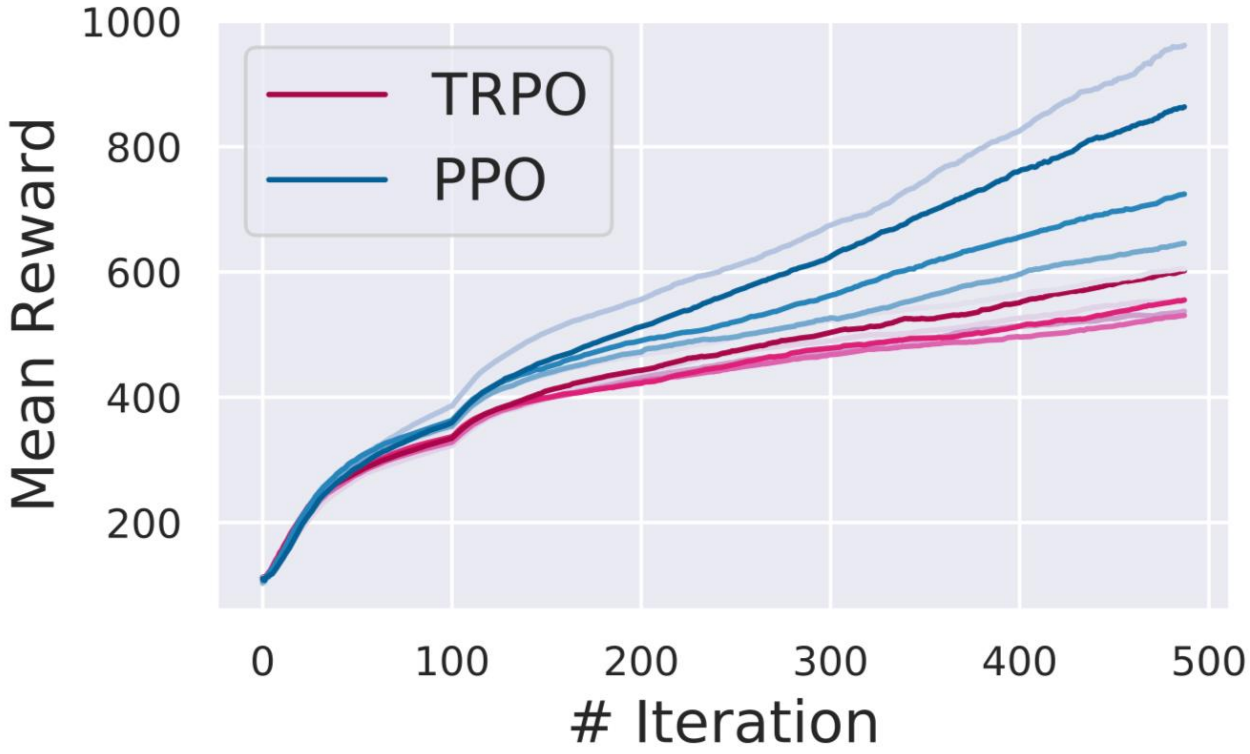
Provides **fast**, **sample-efficient**, and **stable** training



PPO

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

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



PPO: OpenAI5

OpenAI Five: team of bots trained to play DOTA 2.

Defeated professional teams when first released.

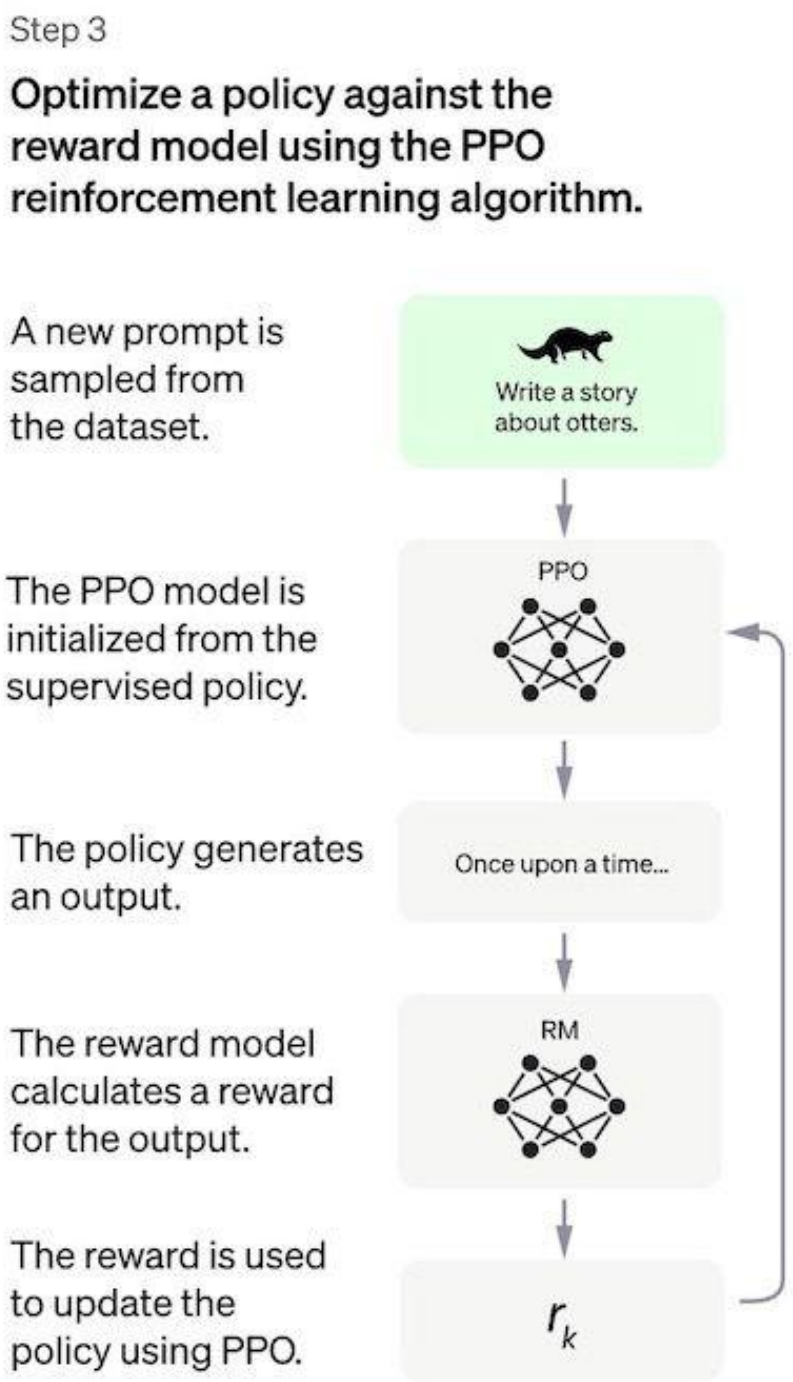
PPO

For tips and tricks to get PPO to train well: https://costa.sh/blog-the-32-implementation-details-of-ppo.html

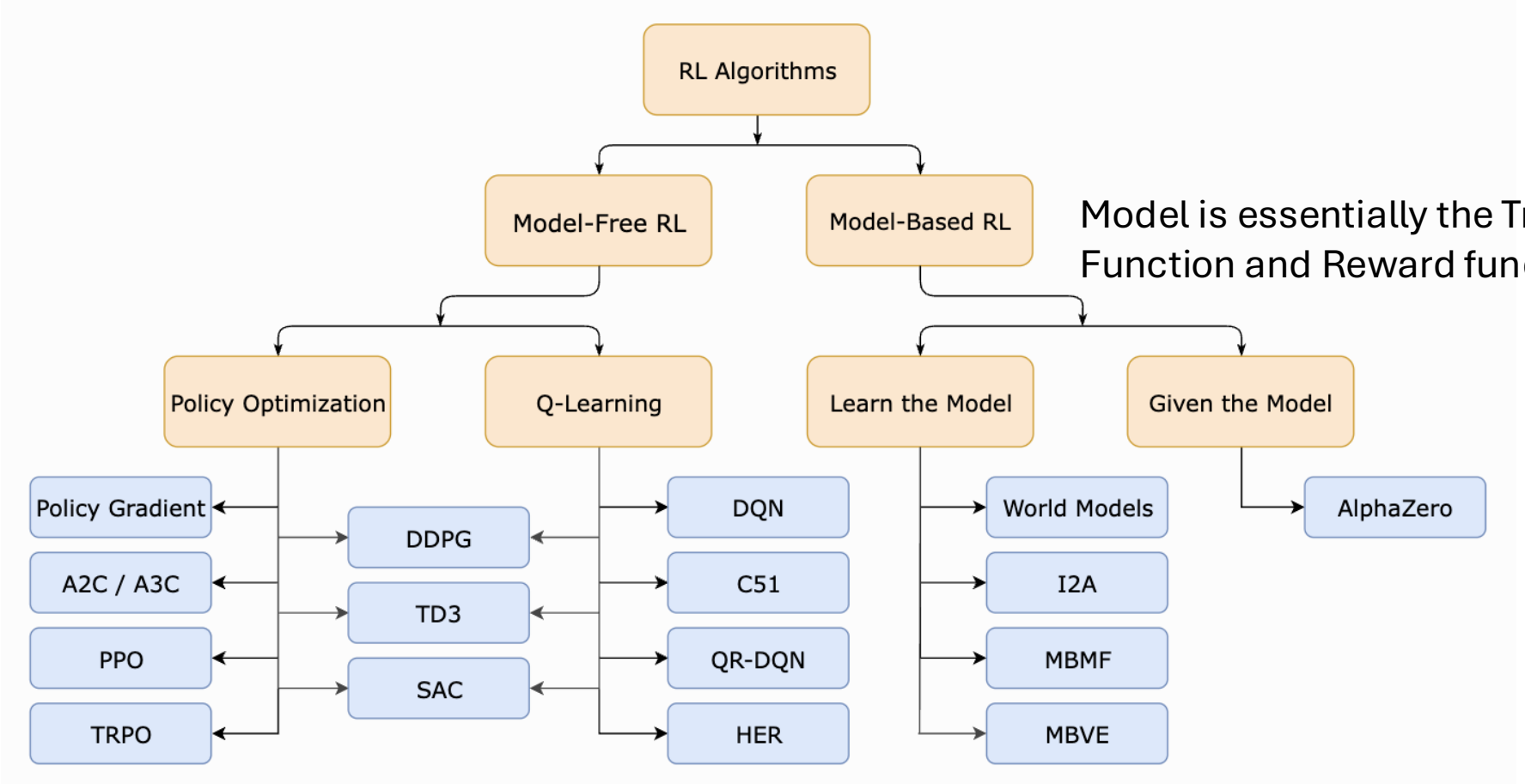
PPO

Final phase of training ChatGPT

Was the default RL algorithm used by OpenAI and most researchers for many years.



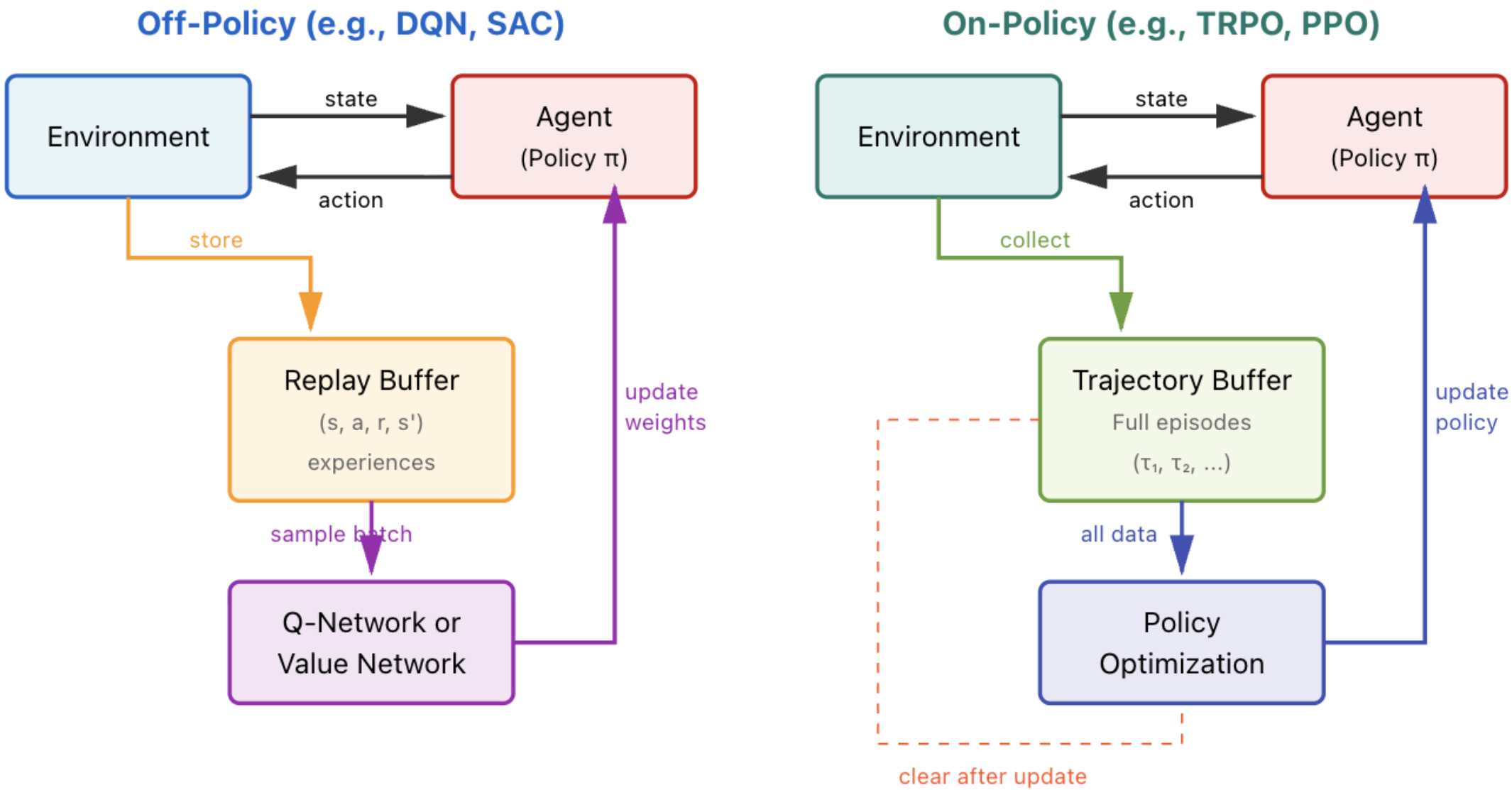
RL Hierarchy



Model is essentially the Transition Function and Reward function

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

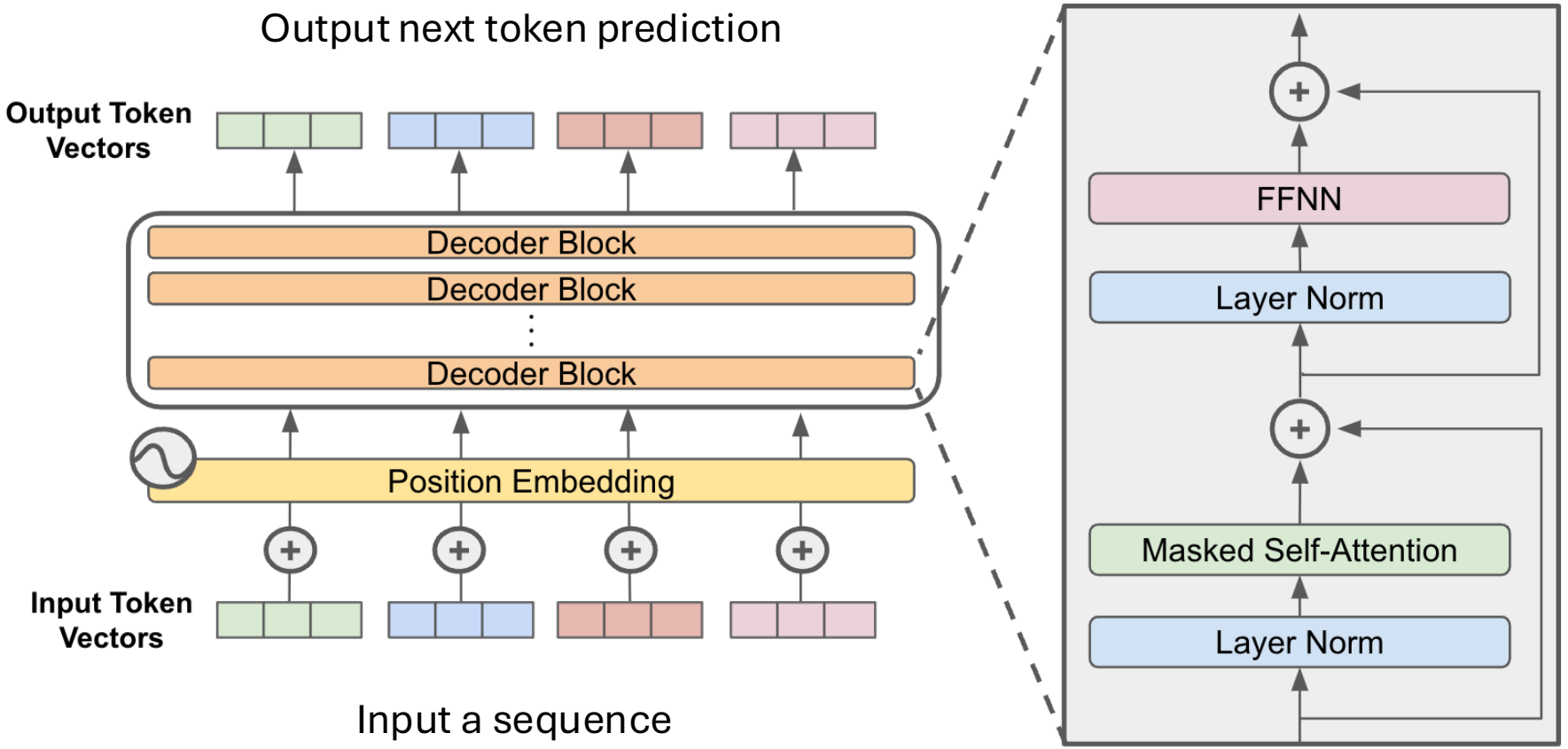
RL Agent Training Architecture



Language Modelling Revisited

Typically framed as supervised learning-style problem:

1. Given some context (e.g., a question)
2. Predict the next token.



Turning Language modelling into an MDP

MDP: <S, A, P, R, γ>

States: Each state is a sequence of tokens

Actions: LLM adds the next token

Transition Function: Transitions are deterministic, given a state and next token, the next state is just the token appended to the previous state

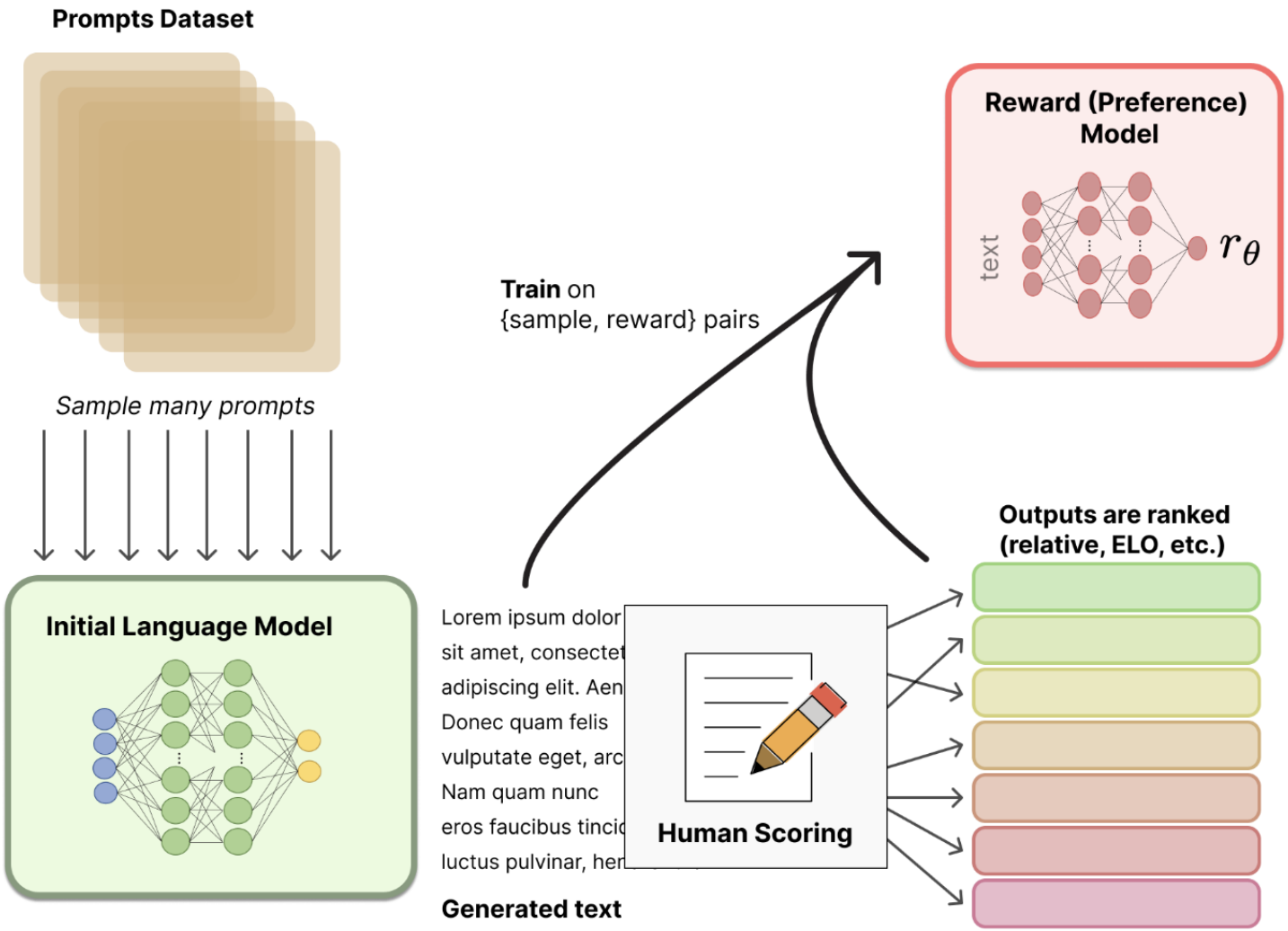
Reward Function: The LLM should be rewarded for good responses, but how do we know what the quality of response is?

Reward Modeling

In MDPs, the reward function is a mapping from states to rewards

Reward Modeling: Learn a reward function

Reward Modeling



Source: https://huggingface.co/blog/rlhf

Bradley-Terry Preference Modeling

Human labelers output a ranking of potential responses from the language model

Our reward model needs to produce a (scalar) reward

How can we figure out scalar rewards from (many) rankings?

Bradley-Terry Preference Modeling

Bradley-Terry model:

$$\mathrm{P}(\mathrm{i} > \mathrm{j}) = \frac{p_i}{p_i + p_j}$$

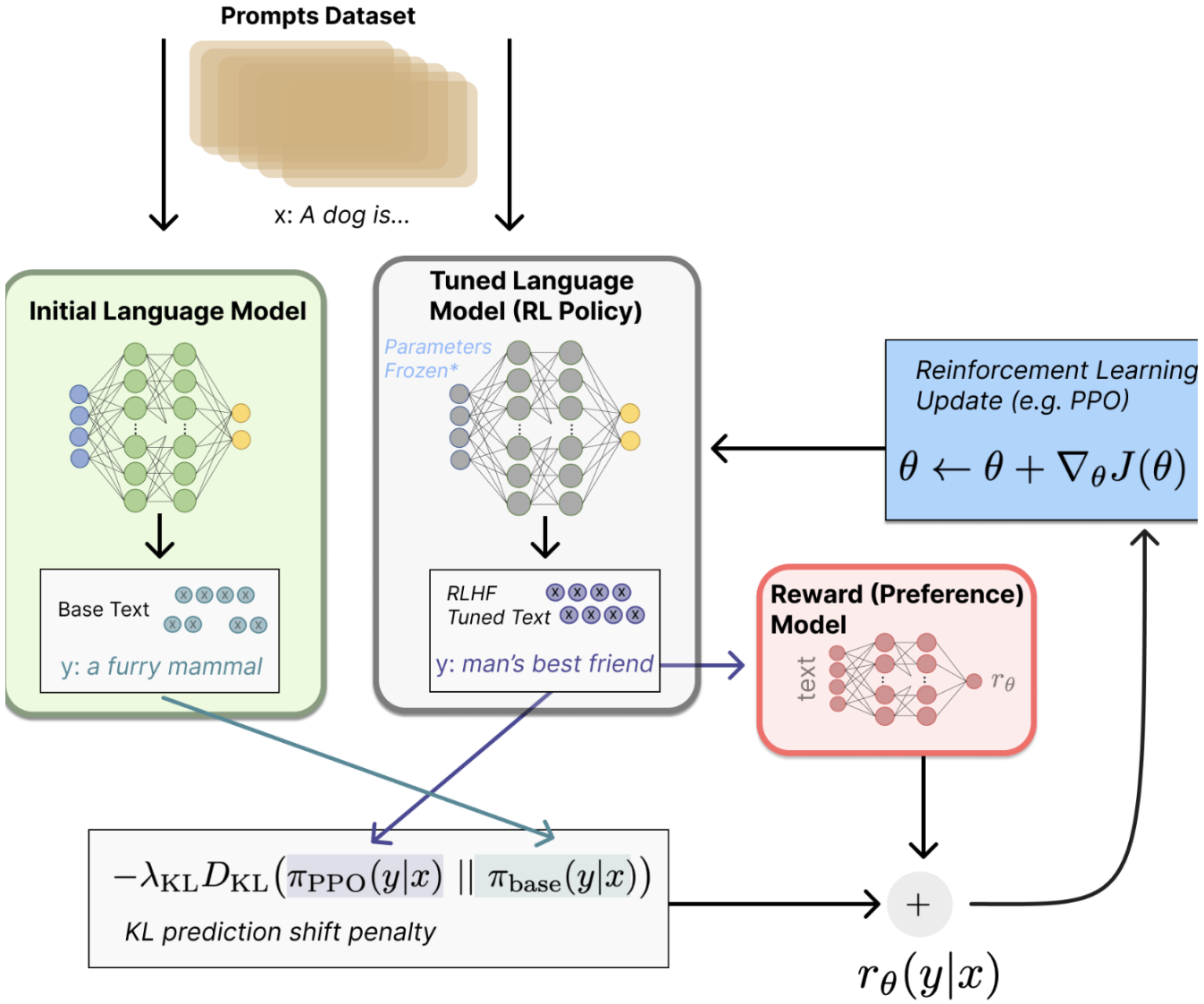
The probability that response *i* will be ranked higher than response *j*

Our model is trained to predict a score for each response.

For every pair of responses, you can calculate the probability of each response being chosen from those scores.

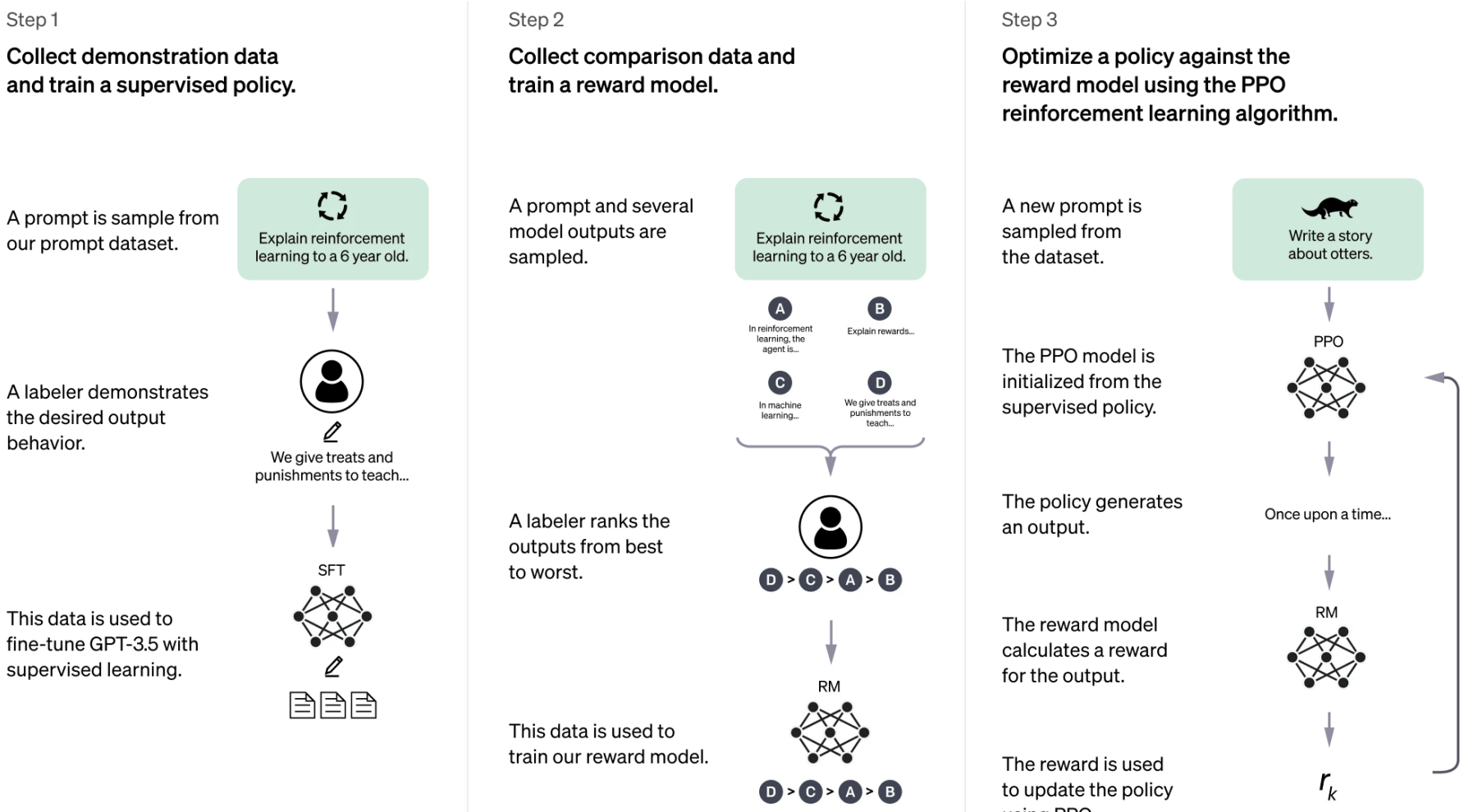
The loss function is ensuring your predicted $\mathrm{P}(\mathrm{i} > \mathrm{j})$ aligns with human rankings

RL+Human Feedback (RLHF)



Source: https://huggingface.co/blog/rlhf

Chat-GPT Training Revisited

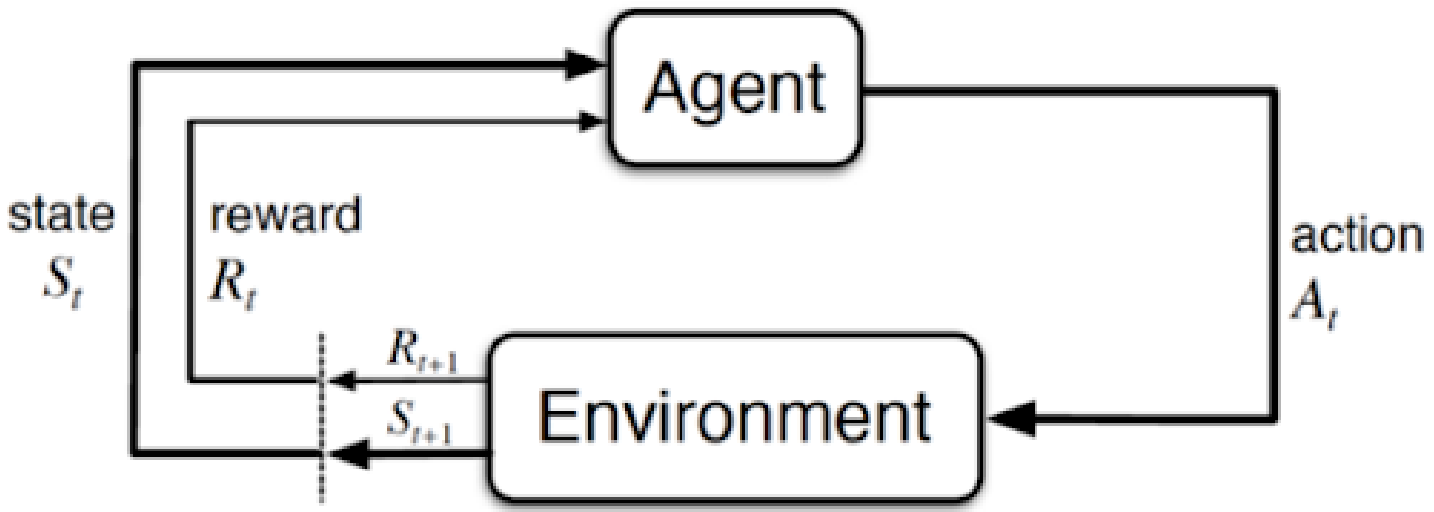


Source: https://openai.com/index/chatgpt/

Robots!

Robots are the most concrete example of autonomous agents

So where are all of the robots trained with RL?



Don’t specify algorithm, but have PPO examples in unitree_rl_gym

Challenges in RL and Robotics

- Simulation environment and real world won't match perfectly (Sim2Real Gap)
  - Hard to collect enough data in the real world
  - Impossible to simulate physics perfectly
- No guarantees of safe policies
  - If you follow a learned and cause an accident, that's very expensive
- Sparse/Delayed rewards
  - It is challenging for a robot to know if it is doing well until a task is complete
- Partial Observability in the real world
  - Robots do not have access to the entire world state, just what they can observe with their sensors.

Why don’t we see more RL in deployed robots?

Why don’t we see more RL in deployed robots?

Deep Learning is not the answer to every problem

We already know optimal-control algorithms for certain types of problems, Deep RL cannot be better than optimal solutions…