RLHF: Reinforcement Learning based on Human Feedback

Lecture 10

Subir Varma

Contents

- RLHF: Reinforcement Learning based on Human Feedback
- PPO: Proximal Policy Optimization

RLHF

RL based on Human Feedback

Training language models to follow instructions with human feedback

Long Ouyang[*] Jeff Wu[*] Xu Jiang[*] Diogo Almeida[*] Carroll L. Wainwright[*]

Pamela Mishkin[*] Chong Zhang Sandhini Agarwal Katarina Slama Alex Ray

John Schulman Jacob Hilton Fraser Kelton Luke Miller Maddie Simens

Amanda Askell[†] Peter Welinder Paul Christiano[*†]

Jan Leike[*] Ryan Lowe[*]

OpenAI

Abstract

Making language models bigger does not inherently make them better at following a user's intent. For example, large language models can generate outputs that are untruthful, toxic, or simply not helpful to the user. In other words, these models are not *aligned* with their users. In this paper, we show an avenue for aligning language models with user intent on a wide range of tasks by fine-tuning with human feedback. Starting with a set of labeler-written prompts and prompts submitted through the OpenAI API, we collect a dataset of labeler demonstrations of the desired model behavior, which we use to fine-tune GPT-3 using supervised learning. We then collect a dataset of rankings of model outputs, which we use to further fine-tune this supervised model using reinforcement learning from human feedback. We call the resulting models *InstructGPT*. In human evaluations on our prompt distribution, outputs from the 1.3B parameter InstructGPT model are preferred to outputs from the 175B GPT-3, despite having 100x fewer parameters. Moreover, InstructGPT models show improvements in truthfulness and reductions in toxic output generation while having minimal performance regressions on public NLP datasets. Even though InstructGPT still makes simple mistakes, our results show that fine-tuning with human feedback is a promising direction for aligning language models with human intent.

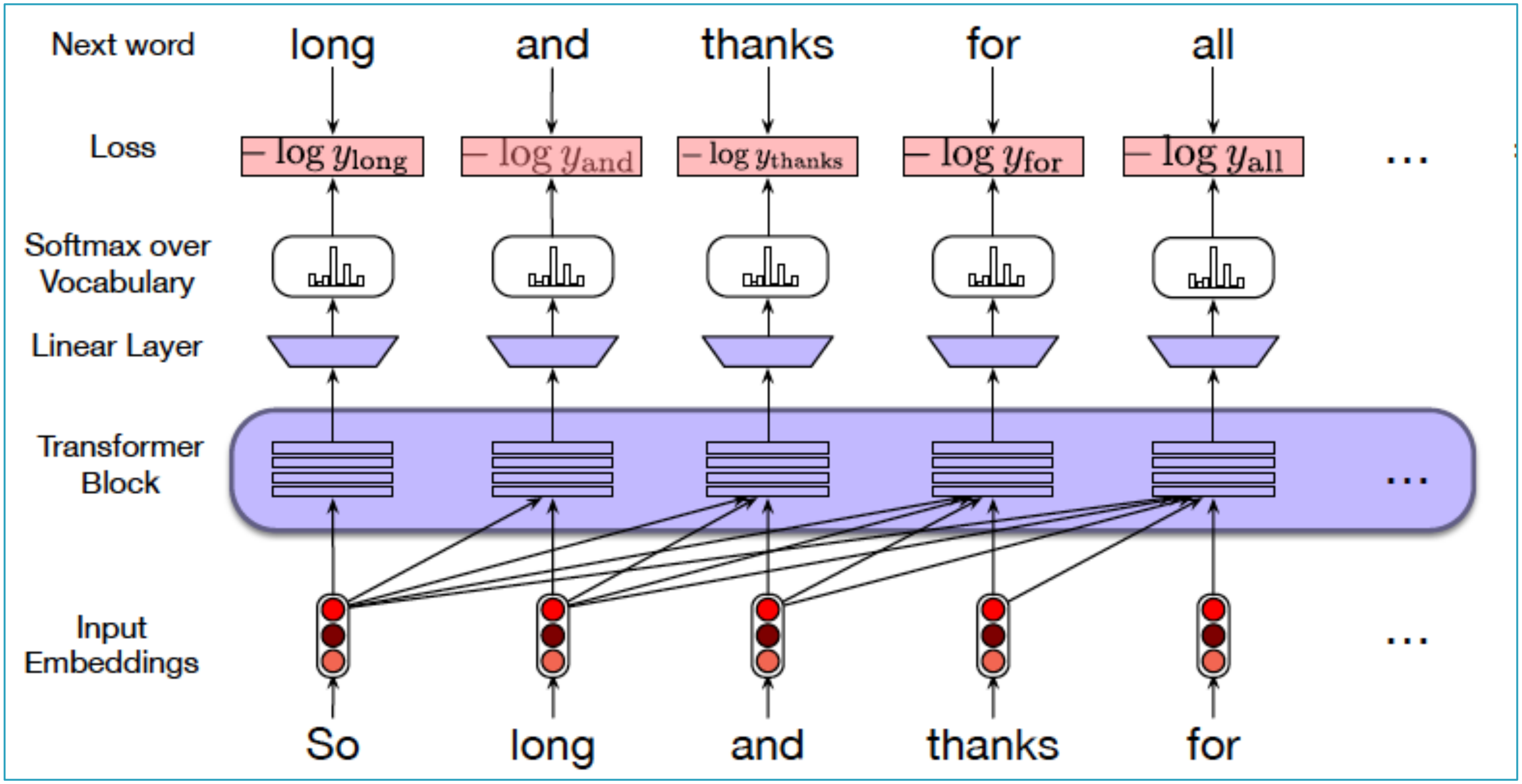
1 Introduction

Large language models (LMs) can be "prompted" to perform a range of natural language processing (NLP) tasks, given some examples of the task as input. However, these models often express unintended behaviors such as making up facts, generating biased or toxic text, or simply not following user instructions (Bender et al., 2021; Bommasani et al., 2021; Kenton et al., 2021; Weidinger et al., 2021; Tamkin et al., 2021; Gehman et al., 2020). This is because the language modeling objective

[*]Primary authors. This was a joint project of the OpenAI Alignment team. RL and JL are the team leads. Corresponding author: lowe@openai.com.

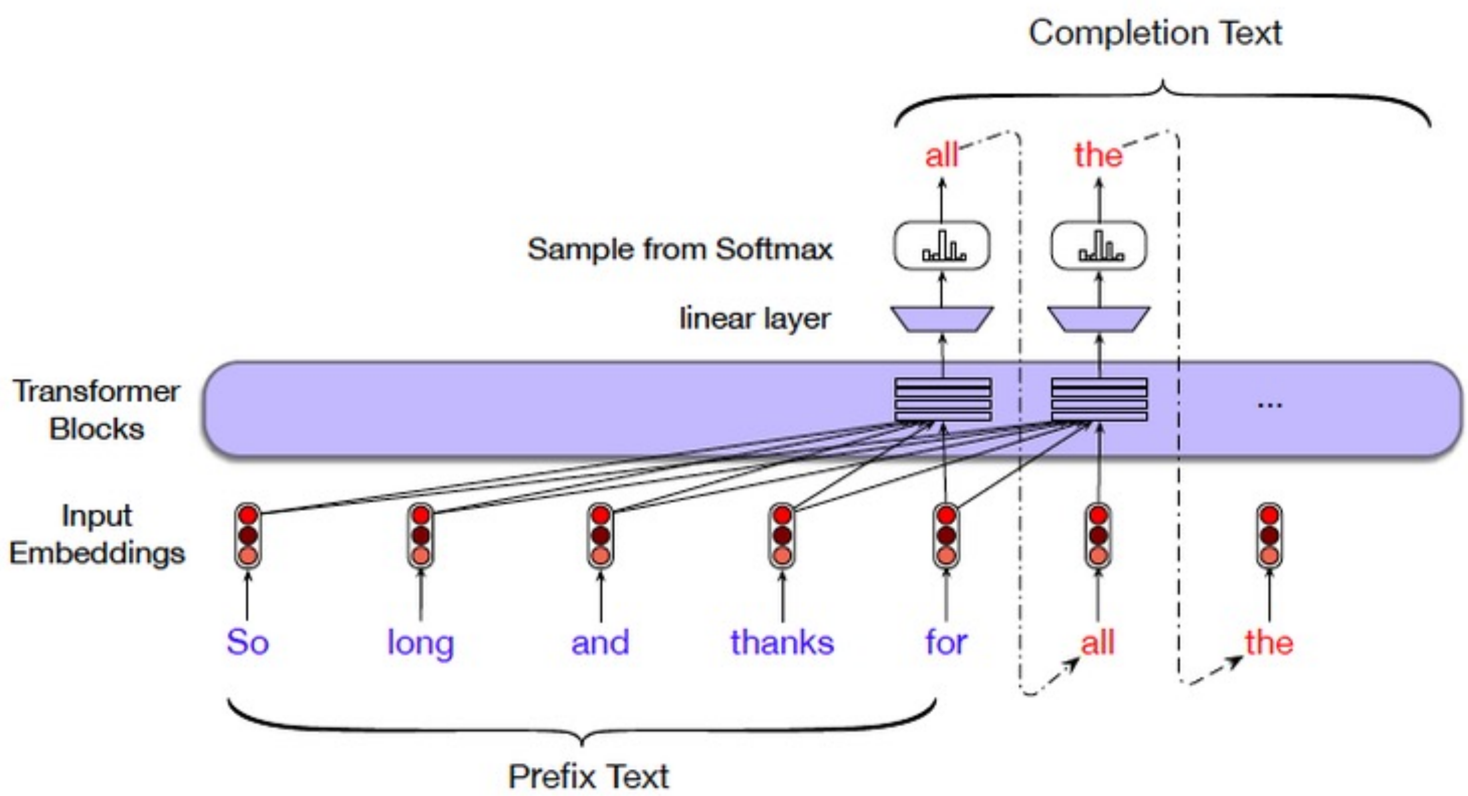
[†]Work done while at OpenAI. Current affiliations: AA: Anthropic; PC: Alignment Research Center.

Large Language Models (LLMs) using Transformers: Training



- Trained to predict the next word in a sentence using the Cross entropy Loss function
- Massive amount of text used for Training
- Some LLMs have hundreds of billions of parameters

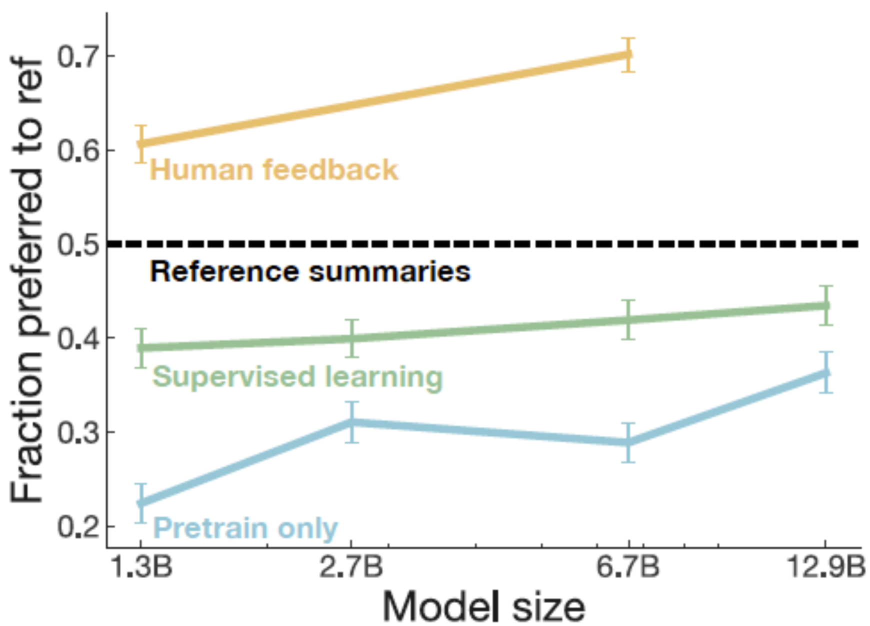
Inference: Text Completion



- Sentences are generated one word at a time by sampling from the output probability distribution
- Its thought that LLMs create some sort of model of the world from the training data, which they use during prediction

Problems with LLMs (Pre Chat GPT)

- Mis-alignment between LLM generated text and generating high-quality text as determined by humans
- A possible reason for mis-alignment: The Cross Entropy Loss function does not differentiate between important errors (like making up facts) and un-important errors (selecting a precise word from a set of synonyms)
- Solution using RLHF: Fine tune models from Human Feedback using Reward Learning



Make the LLM
- Helpful: Help the user
- Honest: No hallucinations
- Harmless: Don't cause harm

RLHF

- RLHF uses methods from Reinforcement Learning to directly optimize a language model with human feedback.
- RLHF has enabled language models to begin to align a model trained on a general corpus of text data to that of complex human values.

ChatGPTs explanation For RLHF!

LE

Can you explain to me how RL with human feedback works for language models like I am five?

Sure! So, imagine you have a robot named Rufus who wants to learn how to talk like a human. Rufus has a language model that helps him understand words and sentences.

First, Rufus will say something using his language model. For example, he might say "I am a robot."

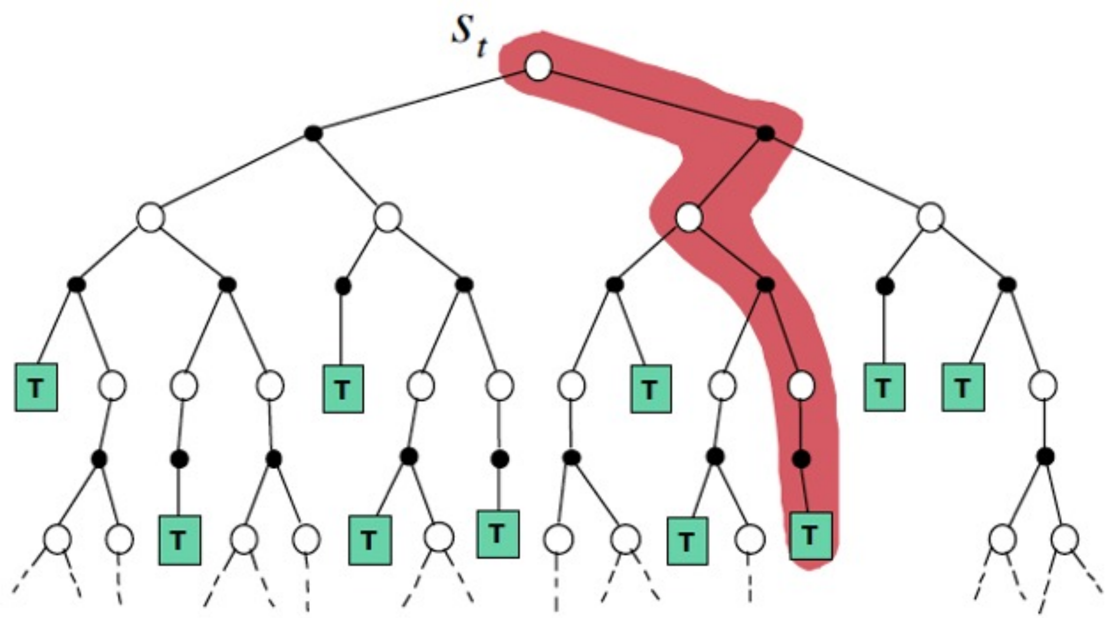
Then, a human will listen to what Rufus said and give him feedback on whether it sounded like a natural sentence a human would say. The human might say, "That's not quite right, Rufus. Humans don't usually say 'I am a robot.' They might say 'I'm a robot' or 'I am a machine.'"

Rufus will take this feedback and use it to update his language model. He will try to say the sentence again, using the new information he received from the human. This time, he might say "I'm a robot."

The human will listen again and give Rufus more feedback. This process will continue until Rufus can say sentences that sound natural to a human.

Over time, Rufus will learn how to talk like a human thanks to the feedback he receives from humans. This is how language models can be improved using RL with human feedback.

Basic Idea behind RLHF Algorithm: Use Policy Gradients on Generated Text



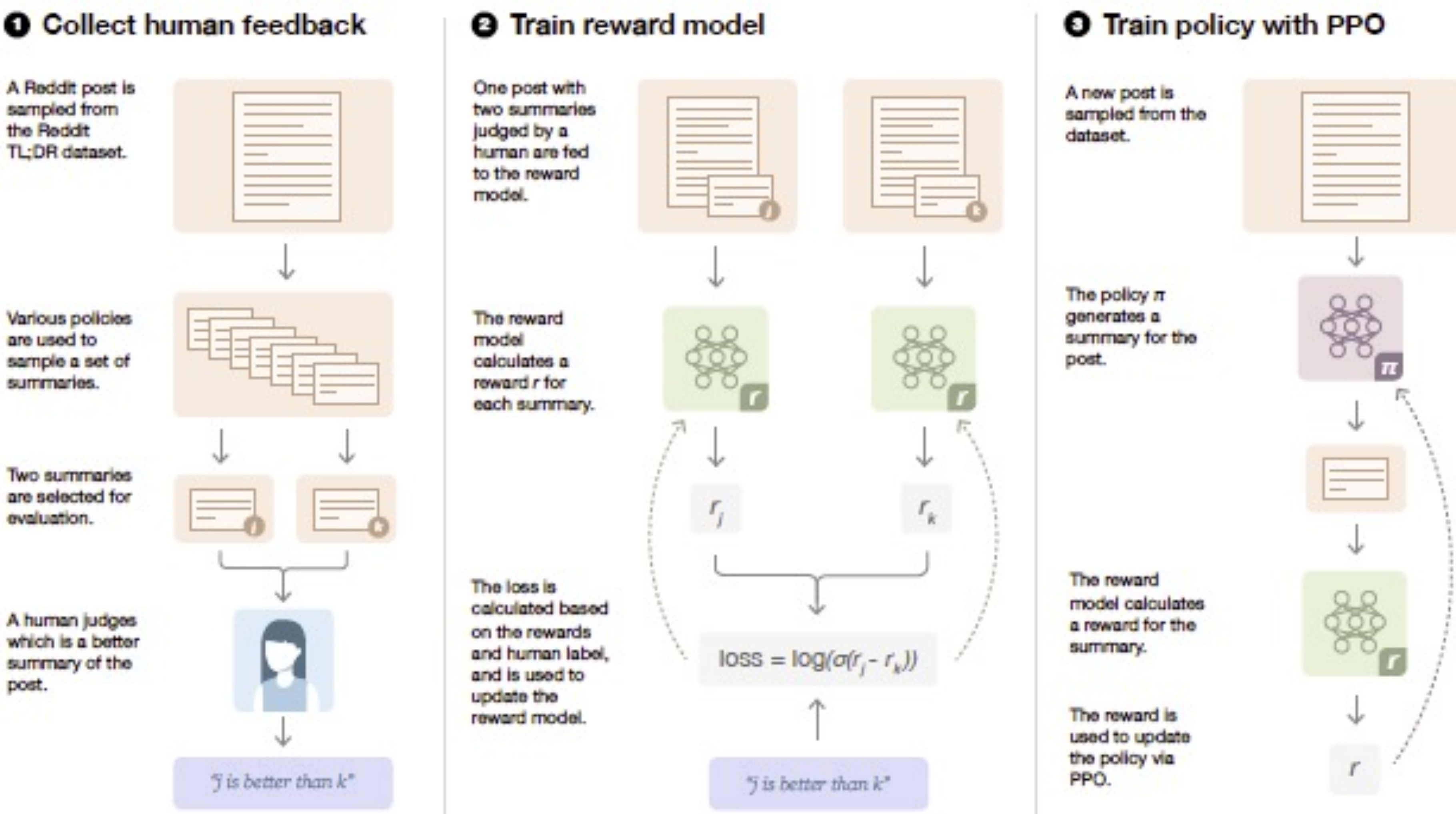
- Treat the LLM as a Policy Function $\pi(S)$ with words as Actions
- An episode corresponds to the text being generated by the LLM
- State S corresponds to the (incomplete) text that has been generated so far
- Action A correspond to the choice of the next word to be added
- Reward R is for the entire text at the end of the episode
- After each episode, modify the Policy (i.e. the LLM) so that in the next iteration it generates text with a higher reward

Where did the reward come from?

RLHF Steps

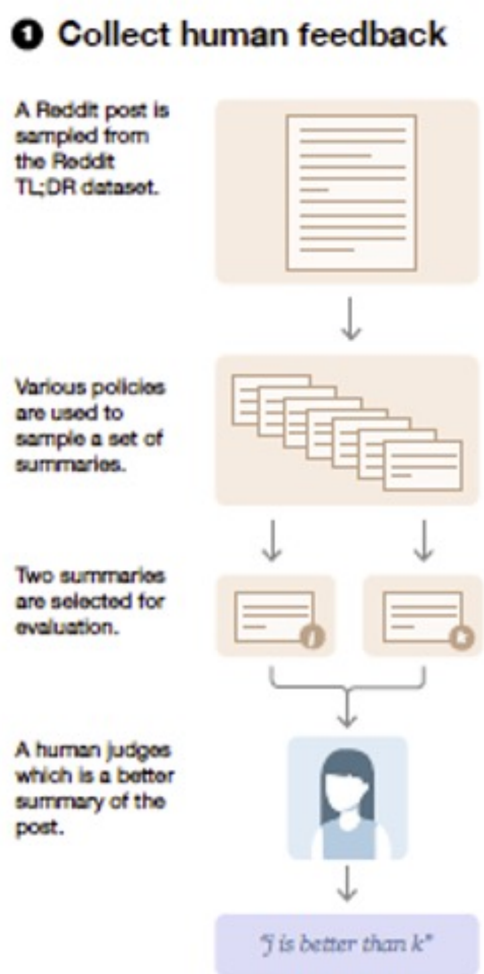
- Step 1: Collect samples from existing policies and send comparisons to humans: For each Reddit post, sample summaries from several sources including the current policy, initial policy, original reference summaries and various baselines. Send a batch of pairs of summaries to human evaluators, who are tasked with selecting the best summary of a given Reddit post.
- Step 2: Learn a reward model from human comparisons: Given a post and a candidate summary, train a reward model to predict the log odds that this summary is the better one, as judged by human labelers.
- Step 3: Optimize a policy against the reward model: Treat the logit output of the reward model as a reward that is used to optimize using Reinforcement Learning, specifically with the PPO algorithm.

RLHF Steps



1 Collect human feedback
A Reddit post is sampled from the Reddit TL;DR dataset.
Various policies are used to sample a set of summaries.
Two summaries are selected for evaluation.
A human judges which is a better summary of the post.
"j is better than k"
2 Train reward model
One post with two summaries judged by a human are fed to the reward model.
The reward model calculates a reward r for each summary.
r_j
r_k
The loss is calculated based on the rewards and human label, and is used to update the reward model.
$\text{loss} = \log(\sigma(r_j - r_k))$
"j is better than k"
3 Train policy with PPO
A new post is sampled from the dataset.
The policy π generates a summary for the post.
The reward model calculates a reward for the summary.
The reward is used to update the policy via PPO.
r

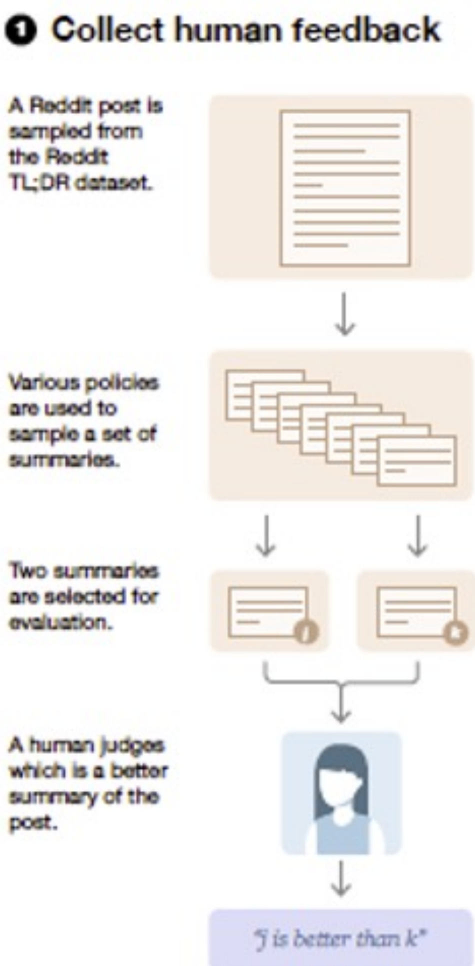
Collect Human Feedback



Objective: Summarize a piece of text

Collect Summaries: These are sampled from the LLM, by varying the temperature etc

Collect Human Feedback (cont)



(a)

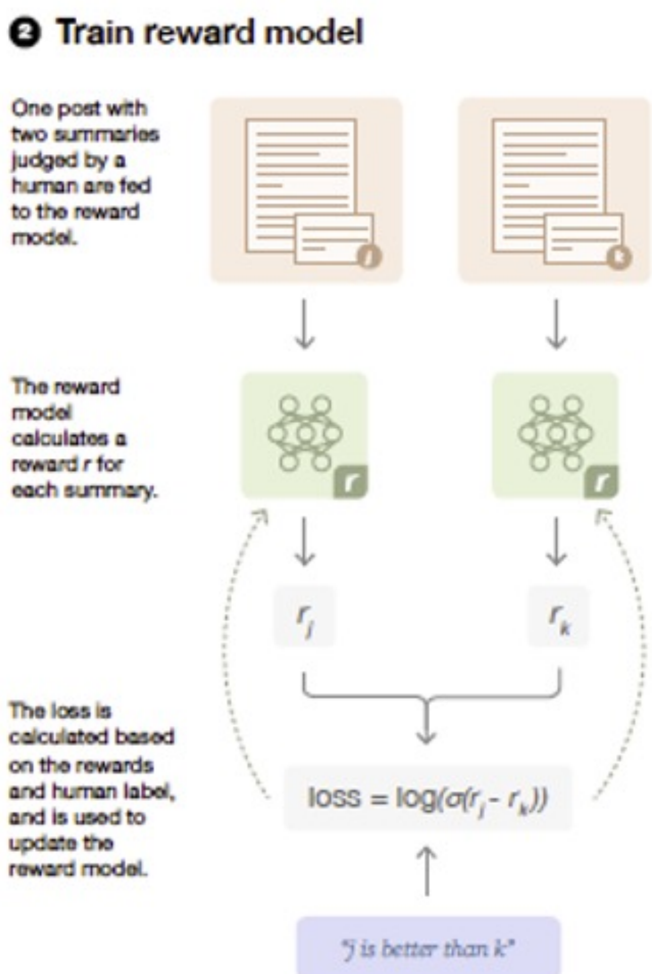
Score the summaries from 1 to 7

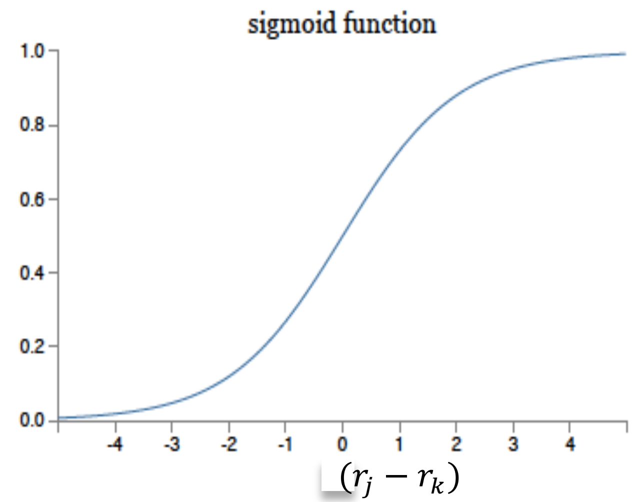
This information is used to rank pairs of summaries.
A>B>C>D results in 6 ranked pairs
A>B, A>C, A>D
B>C, B>D
C>D

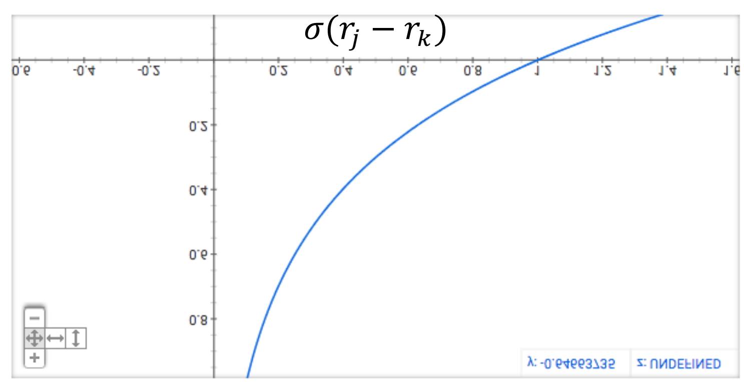
The Reward Model only uses the relative ranking between any two summaries

Train Reward Model

For ChatGPT the Reward model was an LLM with 6B parameters





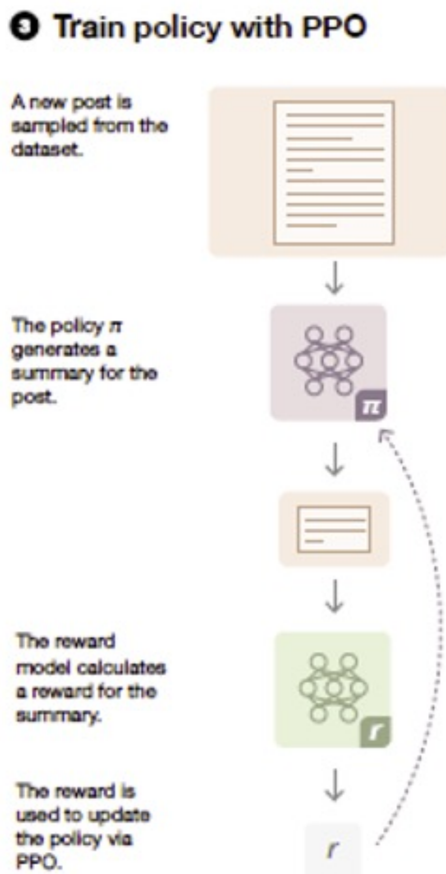


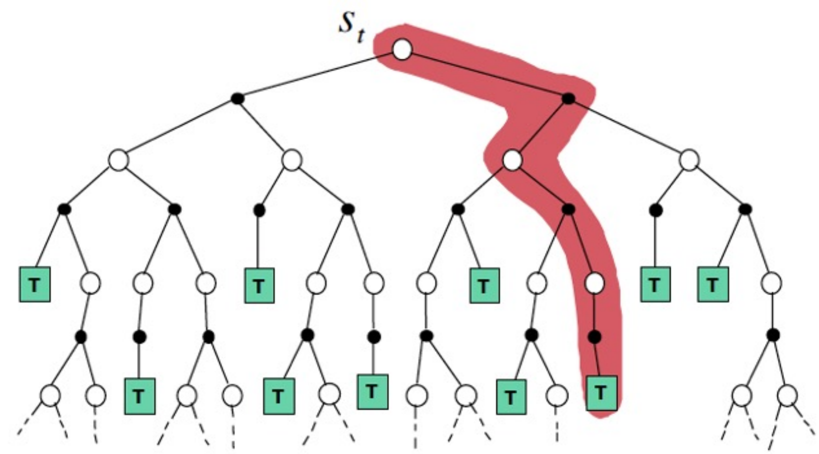
When $r_j > r_k$ then Loss is higher

Hence the Reward model is Trained to predict higher reward for r_j

Objective: Maximize Loss

Train Policy (LLM) with PPO





$$R(x,y) = r_\theta(x,y) - \beta \log[\pi_\phi^{\mathrm{RL}}(y|x)/\pi^{\mathrm{SFT}}(y|x)]$$

This term ensures that the RL model predicted probabilities are not too different from those predicted by the SFT model

PPO

Proximal Policy Optimization

Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov
OpenAI
{joschu, filip, prafulla, alec, oleg}@openai.com

Abstract

We propose a new family of policy gradient methods for reinforcement learning, which alternate between sampling data through interaction with the environment, and optimizing a "surrogate" objective function using stochastic gradient ascent. Whereas standard policy gradient methods perform one gradient update per data sample, we propose a novel objective function that enables multiple epochs of minibatch updates. The new methods, which we call proximal policy optimization (PPO), have some of the benefits of trust region policy optimization (TRPO), but they are much simpler to implement, more general, and have better sample complexity (empirically). Our experiments test PPO on a collection of benchmark tasks, including simulated robotic locomotion and Atari game playing, and we show that PPO outperforms other online policy gradient methods, and overall strikes a favorable balance between sample complexity, simplicity, and wall-time.

1 Introduction

In recent years, several different approaches have been proposed for reinforcement learning with neural network function approximators. The leading contenders are deep Q-learning [Mni+15], "vanilla" policy gradient methods [Mni+16], and trust region / natural policy gradient methods [Sch+15b]. However, there is room for improvement in developing a method that is scalable (to large models and parallel implementations), data efficient, and robust (i.e., successful on a variety of problems without hyperparameter tuning). Q-learning (with function approximation) fails on many simple problems[1] and is poorly understood, vanilla policy gradient methods have poor data effiency and robustness; and trust region policy optimization (TRPO) is relatively complicated, and is not compatible with architectures that include noise (such as dropout) or parameter sharing (between the policy and value function, or with auxiliary tasks).

This paper seeks to improve the current state of affairs by introducing an algorithm that attains the data efficiency and reliable performance of TRPO, while using only first-order optimization. We propose a novel objective with clipped probability ratios, which forms a pessimistic estimate (i.e., lower bound) of the performance of the policy. To optimize policies, we alternate between sampling data from the policy and performing several epochs of optimization on the sampled data.

Our experiments compare the performance of various different versions of the surrogate objective, and find that the version with the clipped probability ratios performs best. We also compare PPO to several previous algorithms from the literature. On continuous control tasks, it performs better than the algorithms we compare against. On Atari, it performs significantly better (in terms of sample complexity) than A2C and similarly to ACER though it is much simpler.

[1] While DQN works well on game environments like the Arcade Learning Environment [Bel+15] with discrete action spaces, it has not been demonstrated to perform well on continuous control benchmarks such as those in OpenAI Gym [Bro+16] and described by Duan et al. [Dua+16].

PPO

The idea with Proximal Policy Optimization (PPO) is that we want to improve the training stability of the policy by limiting the change you make to the policy at each training epoch: **we want to avoid having too large policy updates.**

For two reasons:

- We know empirically that smaller policy updates during training are **more likely to converge to an optimal solution.**
- A too big step in a policy update can result in falling "off the cliff" (getting a bad policy) **and having a long time or even no possibility to recover.**

PPO

Key Equations

PPO-clip updates policies via

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

typically taking multiple steps of (usually minibatch) SGD to maximize the objective. Here L is given by

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

in which ϵ is a (small) hyperparameter which roughly says how far away the new policy is allowed to go from the old.

$$A_\pi(S,a) = R + V_\pi(S',W') - V_\pi(S,W')$$

A is the Advantage Function
From the Actor Critic Algorithm

PPO

An equivalent equation for the Loss Function

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

where

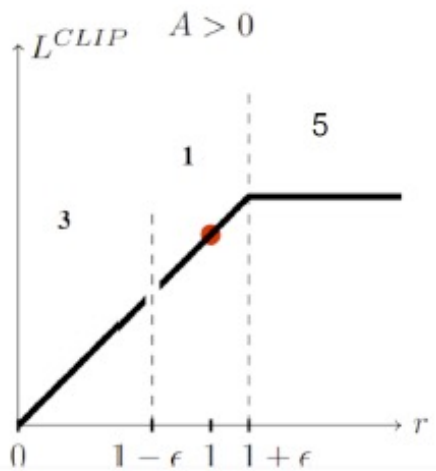
$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0. \end{cases}$$

The Case A > 0

Advantage is positive: Suppose the advantage for that state-action pair is positive, in which case its contribution to the objective reduces to

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

Because the advantage is positive, the objective will increase if the action becomes more likely—that is, if $\pi_\theta(a|s)$ increases. But the min in this term puts a limit to how *much* the objective can increase. Once $\pi_\theta(a|s) > (1+\epsilon)\pi_{\theta_k}(a|s)$, the min kicks in and this term hits a ceiling of $(1+\epsilon)A^{\pi_{\theta_k}}(s, a)$. Thus: *the new policy does not benefit by going far away from the old policy.*



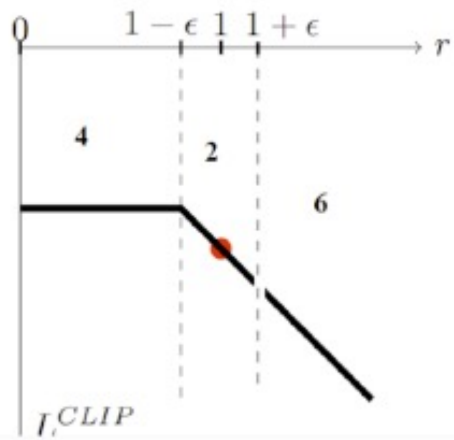
We maximize L by maximizing the min, i.e., by making π_θ larger. However when π_θ becomes larger than $(1+\epsilon)\pi_{\theta_k}$, then the maximization stops

The Case A < 0

Advantage is negative: Suppose the advantage for that state-action pair is negative, in which case its contribution to the objective reduces to

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

Because the advantage is negative, the objective will increase if the action becomes less likely—that is, if $\pi_\theta(a|s)$ decreases. But the max in this term puts a limit to how *much* the objective can increase. Once $\pi_\theta(a|s) < (1-\epsilon)\pi_{\theta_k}(a|s)$, the max kicks in and this term hits a ceiling of $(1-\epsilon)A^{\pi_{\theta_k}}(s,a)$. Thus, again: *the new policy does not benefit by going far away from the old policy*.



We maximize L by minimizing the max, i.e., by making π_θ smaller. However when π_θ becomes smaller than $(1+\epsilon)\pi_{\theta_k}$, then the minimization stops

PPO Algorithm

Algorithm 1 PPO-Clip

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0

2: **for** $k = 0, 1, 2, ...$ **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:

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

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**

Soft Actor Critic

Maximum Entropy RL – Soft Policy Iteration

- Change the Reward Function to incorporate Entropy

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} \left[r(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathcal{H}(\pi(\cdot|\mathbf{s}_t)) \right].$$

- Advantages:
  - Policy is incentivized to explore more widely, while giving up on clearly unpromising avenues
  - The Policy can capture multiple modes of near optimal behavior
  - Improves learning speed

Tabular Soft Policy Iteration

Policy Evaluation

$$\mathcal{T}^\pi Q(\mathbf{s}_t, \mathbf{a}_t) \triangleq r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\mathbf{s}_{t+1}\sim p}\left[V(\mathbf{s}_{t+1})\right],$$

where

$$V(\mathbf{s}_t) = \mathbb{E}_{\mathbf{a}_t\sim\pi}\left[Q(\mathbf{s}_t, \mathbf{a}_t) - \log \pi(\mathbf{a}_t|\mathbf{s}_t)\right]$$

Can also be written as:

$$Q(\mathbf{s}_t, \mathbf{a}_t) \leftarrow r_\pi(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\mathbf{s}_{t+1}\sim p, \mathbf{a}_{t+1}\sim\pi}\left[Q(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})\right]$$

where

$$r_\pi(\mathbf{s}_t, \mathbf{a}_t) \triangleq r(\mathbf{s}_t, \mathbf{a}_t) + \mathbb{E}_{\mathbf{s}_{t+1}\sim p}\left[\mathcal{H}\left(\pi(\,\cdot\,|\mathbf{s}_{t+1})\right)\right]$$

Policy Improvement

$$\pi_{\text{new}} = \arg\min_{\pi'\in\Pi} \mathrm{D}_{\mathrm{KL}}\left(\pi'(\,\cdot\,|\mathbf{s}_t)\,\middle\|\,\frac{\exp\left(Q^{\pi_{\text{old}}}(\mathbf{s}_t,\,\cdot\,)\right)}{Z^{\pi_{\text{old}}}(\mathbf{s}_t)}\right)$$

Functional Soft Policy Iteration: Soft Actor Critic (SAC)

Algorithm 1 Soft Actor-Critic

Initialize parameter vectors ψ, $\bar{\psi}$, θ, ϕ.
for each iteration **do**
  for each environment step **do**
    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
  end for
  for each gradient step **do**
    $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
    $\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$
  end for
end for

$$J_V(\psi) = \mathbb{E}_{\mathbf{s}_t\sim\mathcal{D}}\left[\frac{1}{2}\left(V_\psi(\mathbf{s}_t) - \mathbb{E}_{\mathbf{a}_t\sim\pi_\phi}\left[Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - \log \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)\right]\right)^2\right]$$

$$\hat{\nabla}_\psi J_V(\psi) = \nabla_\psi V_\psi(\mathbf{s}_t)\left(V_\psi(\mathbf{s}_t) - Q_\theta(\mathbf{s}_t, \mathbf{a}_t) + \log \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)\right)$$

$$J_Q(\theta) = \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - \hat{Q}(\mathbf{s}_t, \mathbf{a}_t)\right)^2\right], \tag{7}$$

with

$$\hat{Q}(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma\,\mathbb{E}_{\mathbf{s}_{t+1}\sim p}\left[V_{\bar{\psi}}(\mathbf{s}_{t+1})\right], \tag{8}$$

which again can be optimized with stochastic gradients

$$\hat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\theta(\mathbf{a}_t, \mathbf{s}_t)\left(Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - r(\mathbf{s}_t, \mathbf{a}_t) - \gamma V_{\bar{\psi}}(\mathbf{s}_{t+1})\right).$$

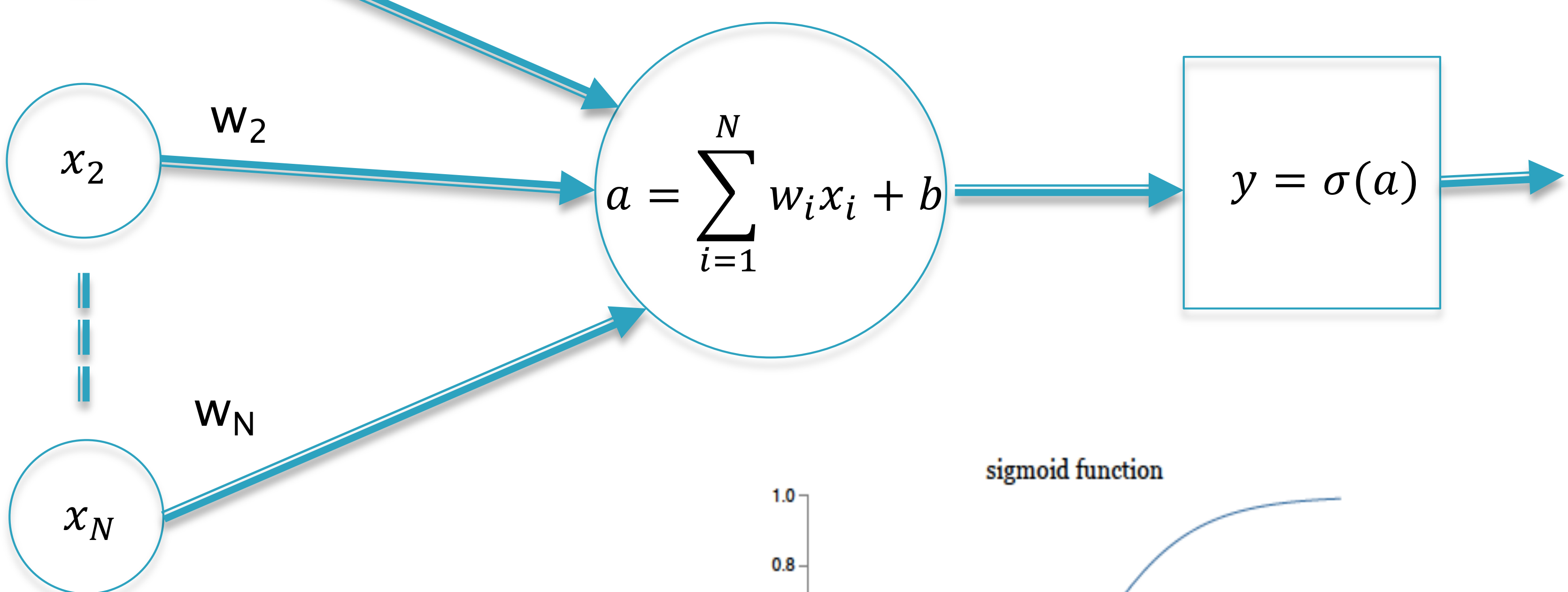
The weights of the Policy Network are changed so that it better matches the Soft Policy computed by the Q Network

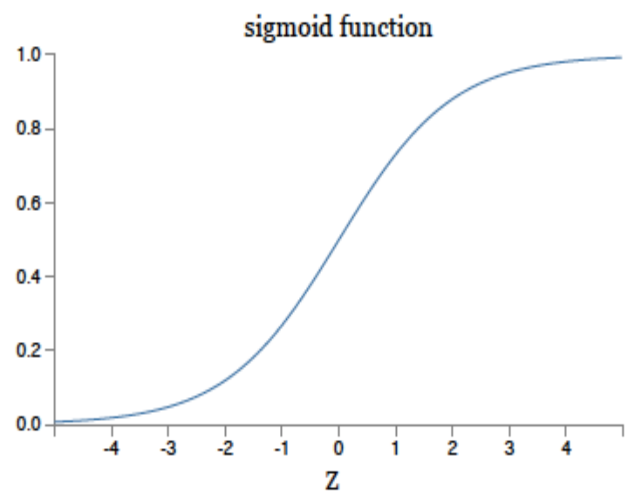
$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}_t\sim\mathcal{D},\epsilon_t\sim\mathcal{N}}\left[\log \pi_\phi(f_\phi(\epsilon_t;\mathbf{s}_t)|\mathbf{s}_t) - Q_\theta(\mathbf{s}_t, f_\phi(\epsilon_t;\mathbf{s}_t))\right],$$

$$\hat{\nabla}_\phi J_\pi(\phi) = \nabla_\phi \log \pi_\phi(\mathbf{a}_t|\mathbf{s}_t) + (\nabla_{\mathbf{a}_t} \log \pi_\phi(\mathbf{a}_t|\mathbf{s}_t) - \nabla_{\mathbf{a}_t} Q(\mathbf{s}_t, \mathbf{a}_t))\nabla_\phi f_\phi(\epsilon_t;\mathbf{s}_t)$$

Convert Scores to Probabilities via the Sigmoid Function

$$y = \sigma(a) = \frac{1}{1 + \exp(-a)}$$





How to Find the Weights?

Given training samples
(X(j),T(j)), j=1,…,M

Find Weights that maximize the Reward Function

$$L(W) = \frac{1}{M}\sum_{j=1}^{M}[t(j)\log y(j) + (1 - t(j)\log(1 - y(j))]$$

$$y(j) = \frac{1}{1 + \exp(-\sum_{i=1}^{n} w_i x_i(j) - b)}$$

No Closed Form solution
Will have to use Numerical Methods

Gradient Computation

$$w_i \leftarrow w_i + \eta \frac{\partial L}{\partial w_i}$$

$$y(j) = \frac{1}{1 + \exp(-\sum_{i=1}^{n} w_i x_i(j) - b)}$$

where

$$L(W) = [t(j) \log y(j) + \left(1 - t(j)\right) \log(1 - y(j))]$$

$$\frac{\partial L}{\partial w_i} = [t(j) - y(j)]x_i(j)$$

$$w_i \leftarrow w_i - \eta x_i(j)[y(j) - t(j)]$$