Proximal Policy Optimization (PPO): The Key t LLM Alignment

Modern policy gradient algorithms and their application to language models...

CAMERON R. WOLFE, PH.D.

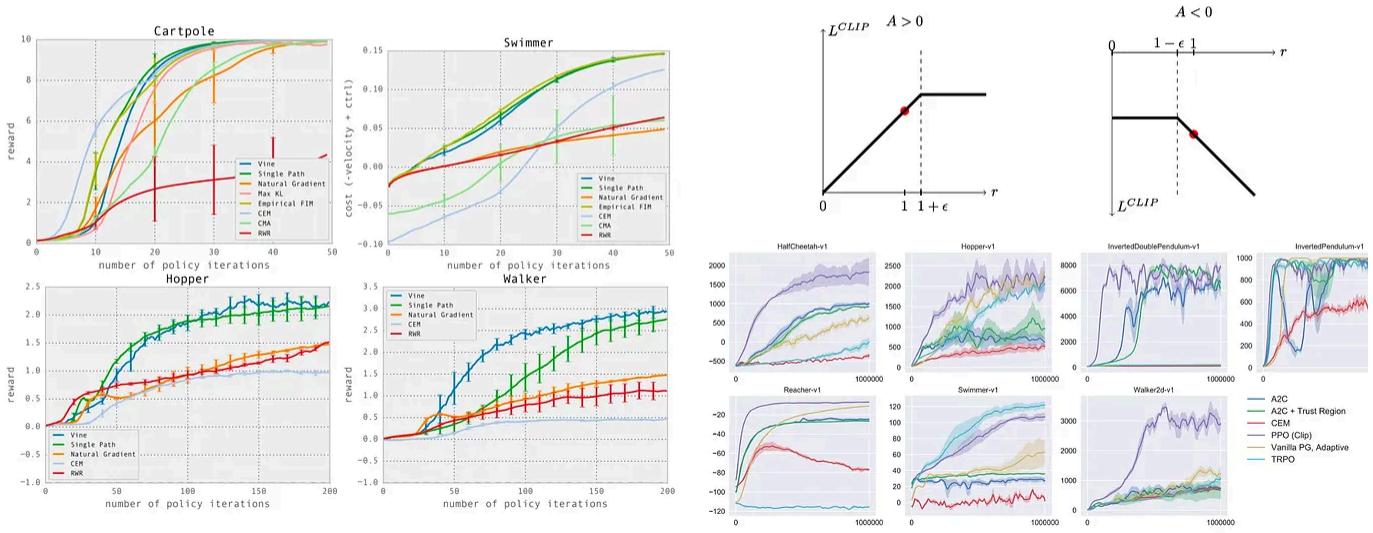
OCT 23, 2023

This newsletter is presented by Rebuy, the commerce AI company.

If you like the newsletter, feel free to get in touch or follow me on Medium, X, a LinkedIn. I try my best to produce useful and informative content.

DEEP (LEARNING) FOCUS

Proximal Policy Optimization (PPO): The Key to LLM Alignment



(from [1, 2])

Recent AI research has revealed that reinforcement learning (RL)—*reinforcemer learning from human feedback (RLHF) in particular*—is a key component of trainin large language models (LLMs). However, many AI practitioners (admittedly) avo the use of RL due to several factors, including a lack of familiarity with RL or preference for supervised learning techniques. There are valid arguments again the use of RL; e.g., the curation of human preference data is expensive and RL c be data inefficient. However, *we should not avoid using RL simply due to a lack of understanding or familiarity*! These techniques are not difficult to grasp and, as shown by a variety of recent papers, can massively benefit LLM performance.

This overview is part three in a series that aims to demystify RL and how it is u to train LLMs. Although we have mostly covered fundamental ideas related to l up until this point, we will now dive into the algorithm that lays the foundation for language model alignment—*Proximal Policy Optimization (PPO)* [2]. As we wi see, PPO works well and is incredibly easy to understand and use, making it a

desirable algorithm from a practical perspective. For these reasons, PPO was originally selected in the implementation of RLHF used by OpenAI to align InstructGPT [6]. Shortly after, the popularization of InstructGPT's sister model ChatGPT—led both RLHF and PPO to become highly popular.

Background Information

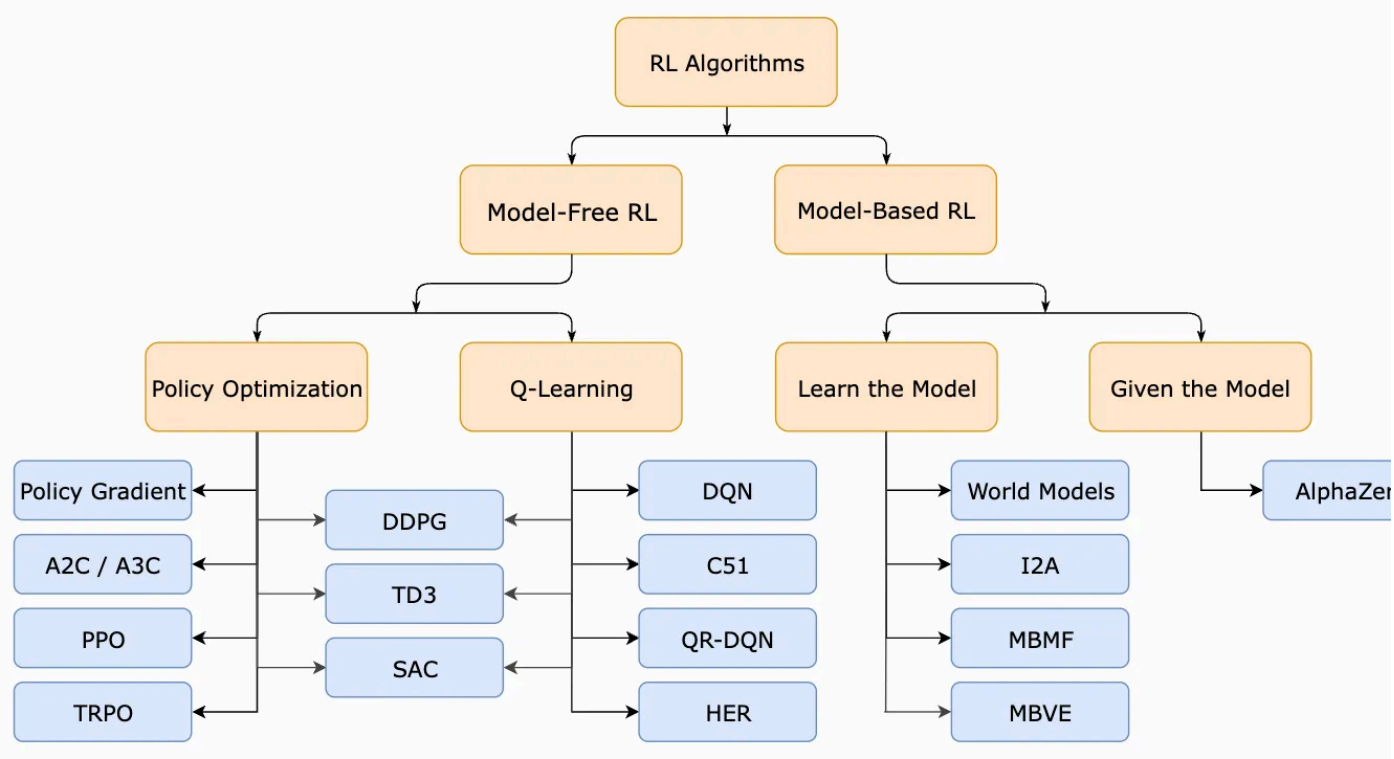
In this series, we are currently learning about reinforcement learning (RL) fundamentals with the goal of understanding the mechanics of language model alignment. More specifically, we want to learn exactly how reinforcement learn from human feedback (RLHF) works. Given that many AI practitioners tend to avoid RL due to being more familiar with supervised learning, deeply understanding RLHF will add a new tool to any practitioner's belt. Plus, researc has demonstrated that RLHF is a pivotal aspect of the alignment process [8]—*ju using supervised fine-tuning (SFT) is not enough* [1]; see below.

Figure 11: Evolution of Llama 2-Chat. We show the evolution after multiple iterations fine-tuning for win-rate % of Llama 2-Chat compared to ChatGPT. *Left*: the judge is our reward model, which may fa our model, and *right*, the judge is GPT-4, which should be more neutral.

(from [8])

In this section, we will briefly cover the RL algorithms we've learned about in t series so far, focusing upon their limitations and the primary reasons why bette algorithms are needed. Then, we will (once again) overview the problem setup c RL for language model alignment, which we should use as relevant context whe learning about new algorithms. Finally, we'll learn about the KL divergence, wh is a useful concept for both RL and machine learning in general.

What we have learned so far?

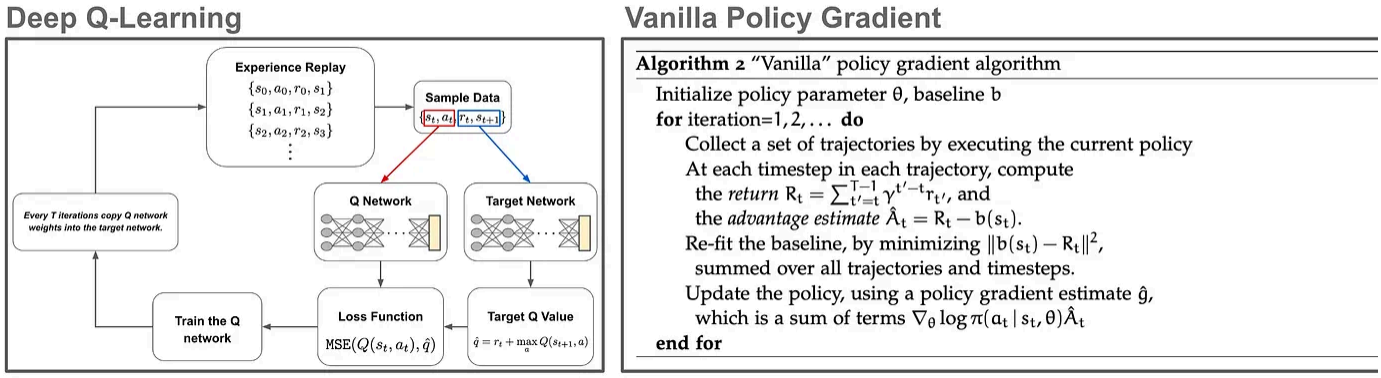


Taxonomy of RL algorithms (from [9])

Until this point, our overviews within this series have mostly focused on fundamental concepts within RL, including:

- *Basics of RL for LLMs* [link]: problem setup (with extensions to LLMs) and basic algorithms like (Deep) Q-Learning.
- *Policy Optimization* [link]: understanding policy gradients—the class of optimization techniques used by RLHF—and basic algorithms in this space

Within this post, we will build on these basic concepts by diving into two RL algorithms that are more directly related to RLHF: Trust Region Policy Optimization (TRPO) [1] and Proximal Policy Optimization (PPO) [2]. Similarly the vanilla policy gradient algorithm that we saw in a prior overview, both of th algorithms are based upon policy gradients. However, PPO, which is an extensi of TRPO, is the most commonly used RL algorithm for RLHF [6]!



(from [10])

Why do we need a new algorithm? So far, we have seen two main RL algorithm that can be used to train neural networks (see above):

- Deep Q-Learning (DQL) [link]
- Vanilla Policy Gradient Algorithm (VPG) [link]

However, these algorithms have notable limitations when used to solve comple problems. DQL can only be applied in relatively simple environments (e.g., gam environments like Atari). Despite being effective for problems with discrete act spaces, DQL struggles to generalize to more realistic (continuous action space environments, where it is known to fail at solving even simple problems. Going further, VPG has poor data efficiency and robustness, meaning that we must collect tons of data from our environment to eliminate noise within the policy gradient estimate and, in turn, effectively train the underlying policy.

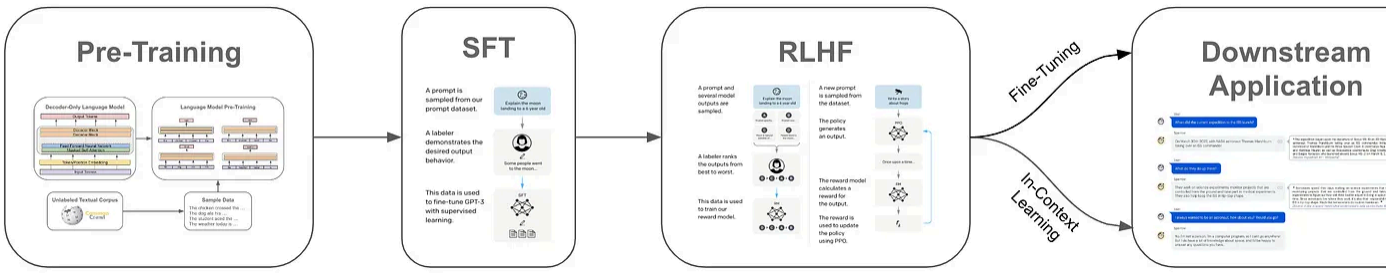
> *"There is room for improvement in developing a method that is scalable (to large models and parallel implementations), data efficient, and robust (i.e., successful on a variety of problems without hyperparameter tuning)."* - from [2]

With this in mind, the motivation behind TRPO and PPO is to improve upon these issues. Namely, we want to derive an RL algorithm that is:

- Generally applicable (i.e., to both discrete and continuous problems)
- Data efficient
- Robust (i.e., works without too much tuning)
- Simple (i.e., not too difficult to understand/implement)

TRPO satisfies the first two points outlined above, while PPO satisfies all four. Due to its simplicity and effectiveness, PPO is widely used across domains and has become the go-to choice for aligning language models via RLHF.

Aligning Language Models with RL



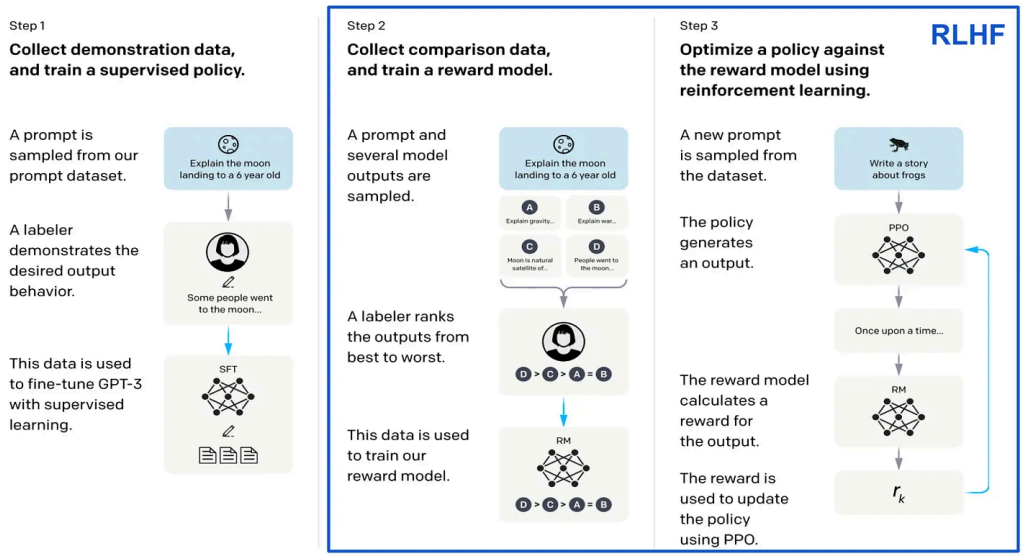
(from [6, 11])

Most modern language models are trained in several phases; see above. First, we perform pretraining, which is the most computationally expensive component of the training process. After pretraining, the LLM can accurately perform next token prediction, but its output may be repetitive, uninteresting, or not useful.

solve this, we can finetune the model to improve its *alignment*, or ability to generate text that aligns with the desires of a human user.

> *"Making language models bigger does not inherently make them better at following 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 aligned with their users."* - from [6]

Typically, we perform alignment by first selecting several alignment criteria (e.g. follow instructions, avoid harmful output, avoid hallucination, produce interesting/creative output, etc.), then finetuning the model—via SFT and RLHF (shown below)—to satisfy these criteria. Once the alignment process is complete the final model can further finetuned and used to solve a downstream application via prompting (or in-context learning).

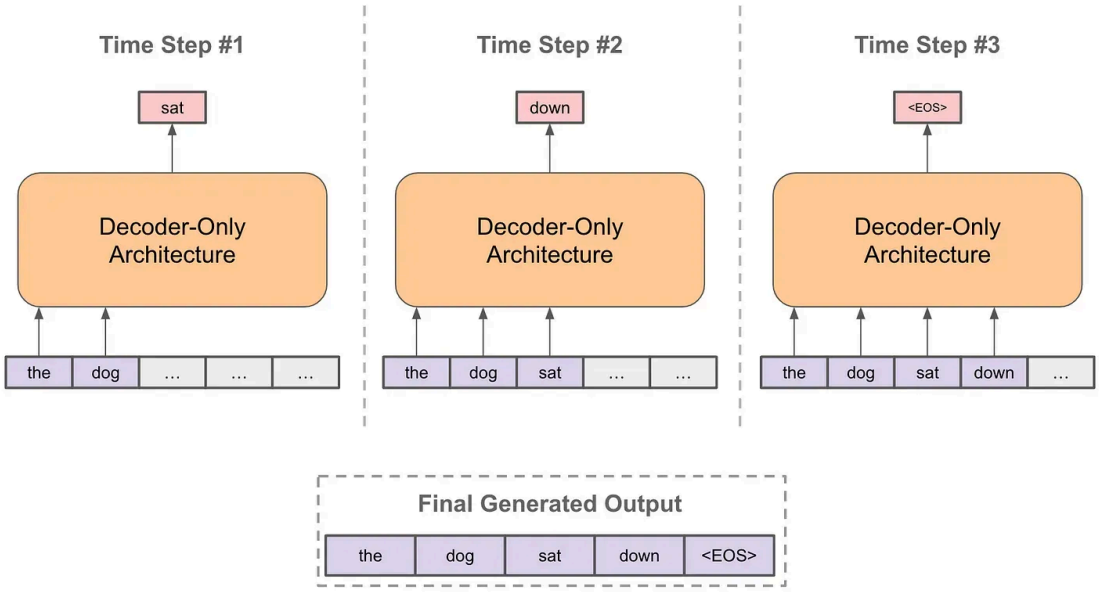


(from [6])

Applying RLHF. As its name indicates, RLHF (depicted above) relies upon RL to train a language model from human feedback. We start with a set of prompts and generate several outputs for each prompt with the language model. From here, we ask a group of human annotators to rank/score the responses to each prompt

according to our alignment criteria. Using these ranked responses, we can trai reward model that predicts a human preference score from a language model's response. Then, we can use PPO [3] to finetune our language model to maximize human preferences scores (predicted by the reward model) of its outputs.

Alignment with RL. Obviously, the language model alignment domain is slight different from the typical RL setup that we have learned about. However, the components of RL actually generalize quite well to language models! To genera text, language models follow an autoregressive process of iteratively predicting the next token and adding this predicted token to the input sequence; see belov



Generating text with a language model

From the lens of RL, our language model is the policy in this case. Given the current state (the textual input sequence), the language model produces an acti —*the next token*—that modifies the current state by adding a token to the curre sequence. Once a full textual sequence has been produced, we can obtain a rew by rating the quality of the language model's output with the reward model. To finetune the model with RL, we simply alternate between collecting data from t environment—*done by generating text with the language model then scoring it with t*

reward model—and updating the policy according to the update rule defined by RL algorithm of choice (e.g., VPG, TRPO, or PPO).

Kullback–Leibler (KL) Divergence

$$H = -\mathbb{E}\left[\log p(x)\right] \quad \text{(Continuous Case)}$$

$$H = -\sum_{i=1}^{N} p(x_i) \cdot \log p(x_i) \quad \text{(Discrete Case)}$$

Continuous and discrete formulations of entropy

At the highest level, the Kullback-Leibler (KL) Divergence is just a method of comparing two probability distributions. The idea of KL divergence has its roo in information theory and is highly related to the concept of entropy. In the equation above, we can see common formulations of entropy for a probability distribution p. Intuitively, the entropy value captures how much information is stored within a probability distribution—*a lower entropy means that you would ne fewer bits to encode the information stored within* p.

KL divergence formulation. Instead of a single probability distribution p, the K divergence considers two probability distributions: p and q. Then, mirroring th above entropy formulation, we compute KL divergence by finding the expected difference in log probabilities between these two distributions; see below.

$$D_{\text{KL}}(p||q) = \mathbb{E}\left[\log p(x) - \log q(x)\right] \quad \text{(Continuous Case)}$$

$$D_{\text{KL}}(p||q) = \sum_{i=1}^{N} p(x_i) \cdot \left(\log p(x_i) - \log q(x_i)\right) \quad \text{(Discrete Case)}$$

The KL divergence is commonly explained in the context of approximations. Namely, if we approximate p with q, the KL divergence is the number of bits w would expect to lose by making this approximation. For more details on the information theory perspective of KL divergence, see the article below.

Applications to AI/ML. KL divergence is heavily used across different domains AI/ML research. For example, it is commonly used in loss functions for training neural networks, either as the core loss or as an added regularization term. For example, Variational Autoencoders (VAEs) use the KL divergence to encourage similarity between the predicted latent distribution and a prior distribution.

> *The final reward function we use during optimization contains a [KL divergence] penalty term ... we find this constraint is useful for training stability, and to reduce reward hacking.*" - from [8]

Furthermore, KL divergence is used heavily within RL research. Here, we will s this concept used in the definition of TRPO, as well as to explain the intuition behind PPO's update rule. Additionally, most implementations of RLHF add an extra KL divergence term to their loss function, which helps to prevent reward hacking and ensures updates to the language model's policy are not too large.

Better Algorithms for Reinforcement Learnin

Within this section, we will learn about two new RL algorithms, called Trust Region Policy Optimization [1] and Proximal Policy Optimization (PPO) [2] tha improve upon the algorithms we have learned about so far. Notably, TRPO and PPO both have drastically improved data efficiency, allowing us to train an effective policy faster and with less data. Going further, PPO is quite simple an robust compared to TRPO, leading to its use in a variety of popular domains.

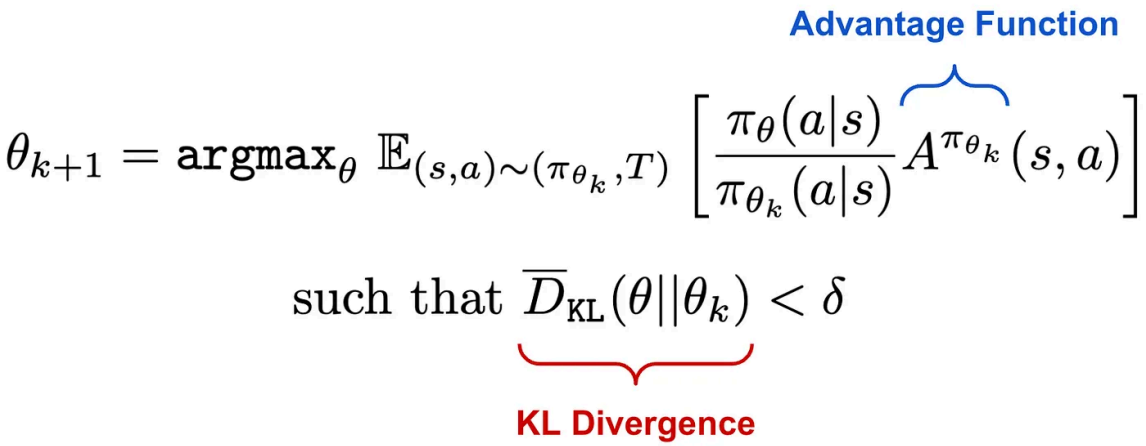
Trust Region Policy Optimization (TRPO) [1]

We should recall that VPG is limited by the fact that it can only perform a sing policy update for each estimate of the policy gradient that is derived. Given tha VPG is notoriously data inefficient, meaning that we have to sample a lot of da when deriving a policy update, performing multiple (or larger) updates may see enticing. However, such an approach is not justified theoretically and, in practi leads to policy updates that are too large, thus damaging performance.

> *"While it is appealing to perform multiple steps of optimization on this loss using th same trajectory, doing so is not well-justified, and empirically it often leads to destructively large policy updates"* - from [2]

Trust Region Policy Optimization (TRPO) [1] aims to solve the problem describ above using an approach that is similar to VPG. At each step of the optimizatic process, however, we find the largest possible policy update that still improves performance. Put simply, TRPO allows us to learn faster by finding a reliable w to make larger policy updates that do not damage performance.

TRPO formulation. More specifically, we update the policy under a constraint- *based on the KL divergence*—that captures the distance between policies before a after the current update. Considering this constraint allows us to find a balance between update size and the amount of change to the underlying policy.



TRPO update rule

To make this discussion a bit more concrete, the theoretical update rule used b TRPO is shown within the equation above. Here, the advantage function is computed using some advantage estimation technique, such as Generalized Advantage Estimation (GAE) [3].

$$\theta_{k+1} = \theta_k + \alpha \left(\mathbb{E}_{(s,a)\sim(\pi_{\theta_k},T)} [\nabla_{\theta_k} \log \pi_{\theta_k}(a|s) A^{\pi_{\theta_k}}(s,a$$

Vanilla policy gradient update rule

Intuitively, this formulation looks quite similar to the update rule for the VPG algorithm (copied above for reference) with a few important differences:

- The terms in the expectation are modified slightly to express the probabilit of a given action `a` as a ratio between old and updated policies.
- The update has an added constraint based on the KL divergence between o and updated policies.
- Instead of performing gradient ascent, we are solving a constrained maximization problem to generate each new policy [5].

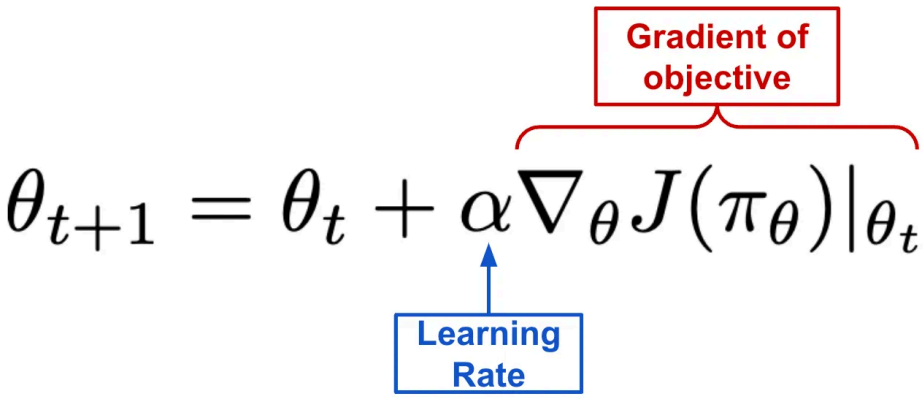
How do we compute TRPO's update in practice? Well, working with the analytical update rule shown above is difficult. But—*like many techniques in AI/ML*—we ca find an approximation to this equation that works quite well and can be compu efficiently! The details of computing this approximation are beyond the scope c this post. However, the high-level steps we take in this process are:

1. Approximate the objective (and constraint) with a Taylor expansion.
2. Solve this approximate objective function (using ideas from optimization research like Lagrangians and duality).
3. Use the conjugate gradient algorithm to avoid inverting large matrices whe solving the problem above (this is computationally intractable otherwise!).
4. Perform post-processing to ensure that the updated policy both satisfies th KL divergence constraint and results in an improvement to the objective.

For more details on his this update rule is derived, check out the article below.

Using TRPO in practice. The implementation of TRPO is similar to that of VP We allow our current policy to interact with the environment and collect data [6]. From this observed data, we can compute the approximate update for TRPO as described above. Then, we can continue the process of collecting data and performing an update until we arrive at a policy that performs quite well. Becau we are using the actual policy being trained to collect the data used to train it, TRPO is an *on-policy* reinforcement learning algorithm.

KL Divergence Constraint. Before moving on, let's briefly consider the KL divergence constraint used by TRPO to better understand its role in the learnir process. As mentioned previously, the VPG algorithm is based upon gradient ascent, which—by nature—ensures that updates to the policy's parameters θ ar not too large. In particular, we use a learning rate to perform updates with VPG which can control the size of the update in the parameter space; see below.



Policy optimization with gradient ascent

Here, only the size of the update to θ is controlled—*the old and updated policies c close in the parameter space*. Despite this fact, small tweaks to the parameters θ r completely change the underlying policy! Even a relatively small update can drastically change the policy's performance (or even lead to collapse). Because small changes to θ can drastically alter the policy, ensuring that policy updates small in the parameter space does not provide much of a guarantee on changes the resulting policy. As a result, we are constrained to relatively small updates within the VPG algorithm—*larger or multiple updates could be harmful*.

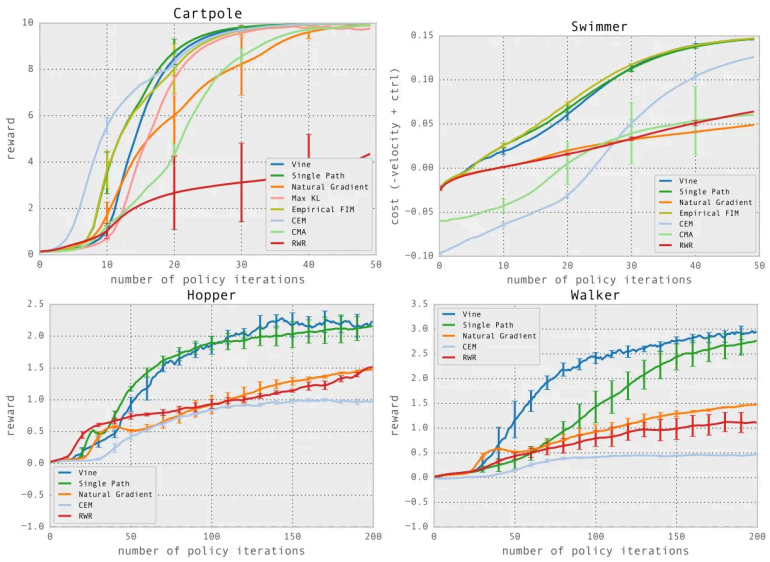


Figure 4. Learning curves for locomotion tasks, averaged across five runs of each algorithm with random initializations. Note that for the hopper and walker, a score of -1 is achievable without any forward velocity, indicating a policy that simply learned balanced standing, but not walking.

(from [1])

TRPO sidesteps this issue by considering the size of our policy update from an alternative viewpoint. Namely, we compare updated and old policies using the divergence, which measures the difference in probability distributions over the action space produced by the two policies. Such an approach compares policies based upon the actions they take rather than their underlying parameters θ. In this way, we can perform large policy updates while ensuring that the new poli does not produce actions that are significantly different from the old policy. Su an approach allows us to drastically speed up the learning process as shown in figure above (single path and vine are both based upon TRPO).

Proximal Policy Optimization (PPO) [2]

> *"We [introduce] proximal policy optimization, a family of policy optimization meth that use multiple epochs of stochastic gradient ascent to perform each policy updat These methods have the stability and reliability of trust-region methods but are muc

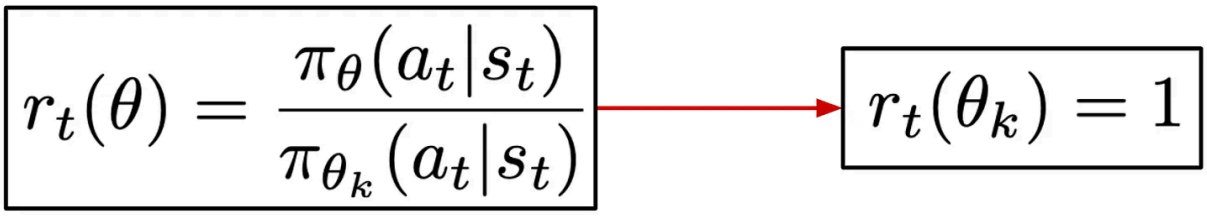
> *simpler to implement ... applicable in more general settings, and have better overall performance.*"- from [2]

TRPO has improved data efficiency, stability, and reliability compared to the V algorithm, but there are still limitations that need to be addressed. Namely, the algorithm is complicated, can only perform a single update each time new data sampled from the environment, and is only applicable to certain problem setup Aiming to develop a better approach, authors in [2] propose Proximal Policy Optimization (PPO), another policy gradient algorithm that alternates between collecting data from the environment and performing several epochs of trainin over this sampled data. PPO shares the reliability of TRPO and is *i)* much simp *ii)* more data efficient, and *iii)* more generally applicable.

$$
\begin{aligned}
L(\theta) &= \mathbb{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t) \right] \\
&= \mathbb{E}_t \left[r_t(\theta) A_t \right]
\end{aligned}
$$

Reformulation of TRPO surrogate objective

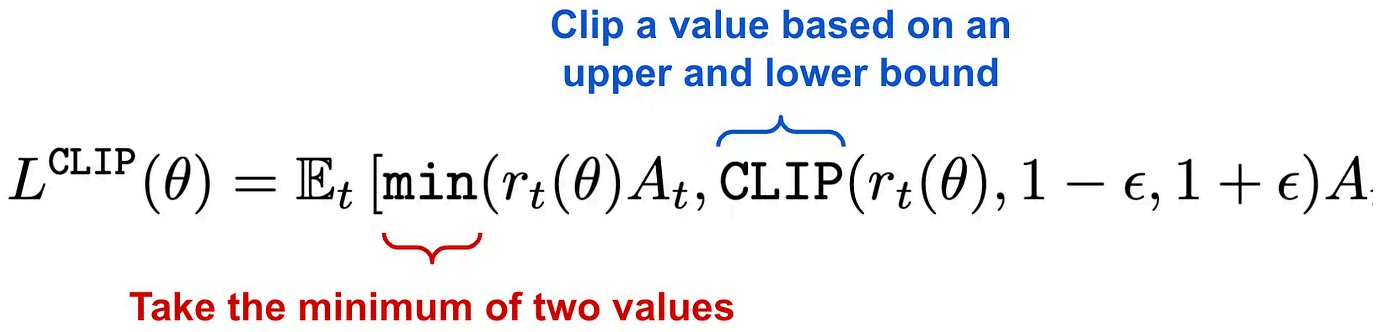
Reformulating the TRPO update rule. Recall that, during each policy update, TRPO maximizes a surrogate objective to get the new policy. The surrogate objective being solved by TRPO can be reformulated as shown above. Here, we simplify our original expectation over actions/states sampled from a policy witl the subscript `t`, which represents time steps along different trajectories sample from the environment. This objective has two terms: the probability ratio and t advantage function. The expression for the probability ratio is shown below.



The probability ratio in TRPO

If we were to maximize this objective without any constraints, it would lead to policy update that is too large (and potentially destructive)—*this is why we levera the KL divergence constraint*. Put simply, adding a constraint to the policy update penalizes policy updates that move the probability ratio too far away from 1.

PPO surrogate objective. Similar to TRPO, we perform policy updates in PPO according to a surrogate objective. However, this surrogate objective has a "clipped" probability ratio, as shown in the equation below.



PPO surrogate objective

The surrogate objective for PPO is expressed as a minimum of two values. The first value is the same surrogate objective from TRPO, while the second value i "clipped" version of this objective that lies within a certain range. In practice, t expression is formulated such that there is no reward for moving the probabilit ratio beyond the interval `[1 - ϵ, 1 + ϵ]`; see below.

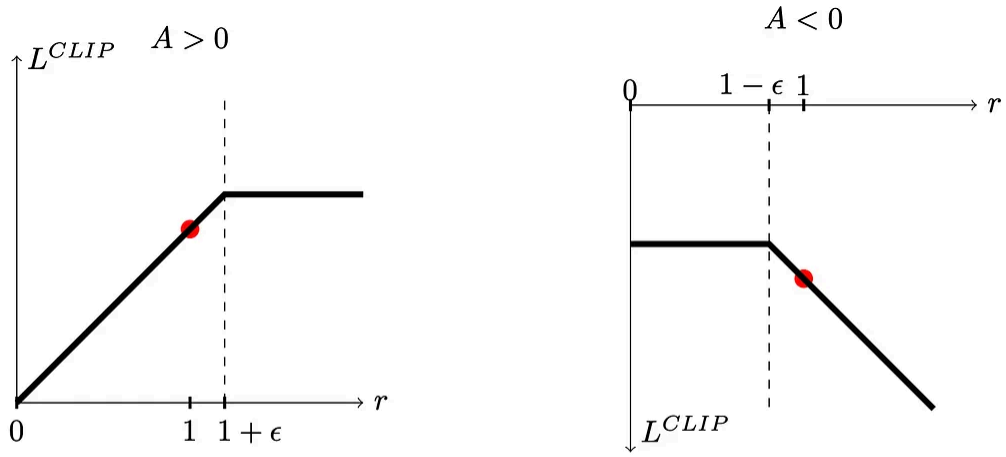


Figure 1: Plots showing one term (i.e., a single timestep) of the surrogate function L^{CLIP} as a functio the probability ratio r, for positive advantages (left) and negative advantages (right). The red circle on ϵ plot shows the starting point for the optimization, i.e., $r = 1$. Note that L^{CLIP} sums many of these ter

(from [2])

In other words, *PPO has no incentive for excessively large policy updates.* Plus, by taking the minimum of the clipped and unclipped version of the surrogate objective, we only ignore excessive changes to the probability ratio if they impr the underlying objective. In the figure above, we see a basic depiction of this tr for both positive and negative values of the advantage function.

To understand PPO's surrogate objective more intuitively, we should look at th figure below, which plots several objective functions as we interpolate between old and updated policy obtained via PPO. In this figure, we see the KL diverger the TRPO surrogate objective (labeled as `CPI`), the clipped surrogate objective, and the full PPO surrogate objective. From these plots, *we can see that the PPO surrogate objective is a pessimistic/lower bound for the TRPO surrogate objective, whe penalty is incurred for having too large of a policy update.*

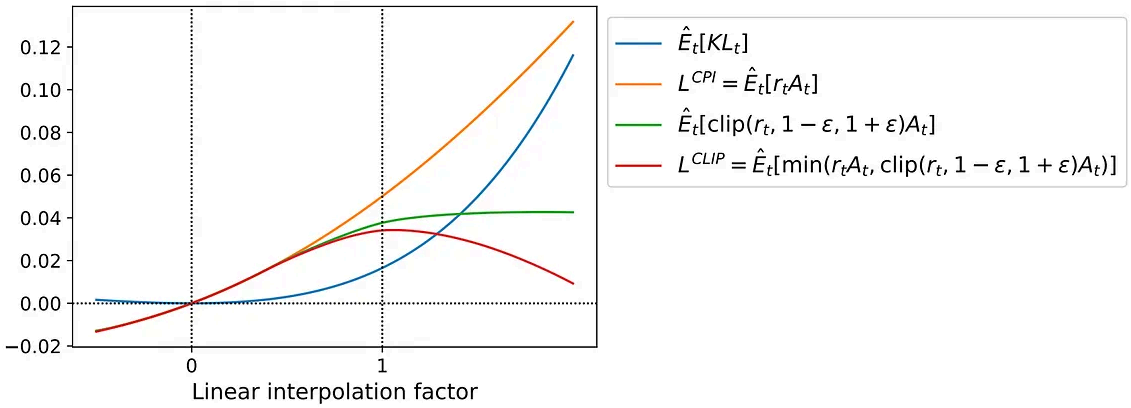


Figure 2: Surrogate objectives, as we interpolate between the initial policy parameter θ_{old}, and the upd policy parameter, which we compute after one iteration of PPO. The updated policy has a KL divergenc about 0.02 from the initial policy, and this is the point at which L^{CLIP} is maximal. This plot corresp to the first policy update on the Hopper-v1 problem, using hyperparameters provided in Section 6.1.

(from [2])

While TRPO sets a hard constraint to avoid policy updates that are too large, P simply formulates the surrogate objective such that a penalty is incurred if the divergence is too large. Such an approach is much simpler, as we no longer hav solve a difficult, constrained optimization problem. Rather, we can compute PPO's surrogate loss with only minor tweaks to the VPG algorithm.

The PPO algorithm and its benefits. So, *how do we use this surrogate objective to train a neural network?* PPO operates similarly to VPG and TRPO by alternating between collecting data from the environment and updating the underlying pol (i.e., it is an on-policy RL algorithm). If we are using a package like PyTorch tha supports automatic differentiation, we can perform a policy update by simply constructing a loss function corresponding to the surrogate objective outlined above [8] and performing several iterations/epochs of stochastic gradient ascent using this loss and samples from the observed data; see below.

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 $M \leq N$
  $\theta_{\text{old}} \leftarrow \theta$
end for

(from [2])

PPO has several benefits compared to TRPO. First, the implementation of PPC much simpler compared to TRPO, as we can use automatic differentiation and gradient-based optimization techniques [9] instead of deriving an (approximate) solution for a complex, constrained objective function. Additionally, while TRP makes only a single policy update each time new data is collected, PPO perforn multiple epochs of optimization via stochastic gradient ascent over the surroga objective, which improves data efficiency. See the link below for an accessible a well-documented implementation of PPO.

Finally, computing estimates of the advantage function (e.g., via GAE) typically requires that we learn a corresponding value function. In TRPO, we must learn this state-value function with a separate neural network. However, PPO—*due t compatibility with a wider scope of architectures (including those with parameter shari* —can train a joint network for policy and value functions by just adding an ext term to the loss function that computes the mean-squared error (MSE) between estimated and actual value function values; see below.

$$L^{\text{CLIP+VF}}(\theta) = \mathbb{E}_t \left[L^{\text{CLIP}}(\theta) - c_1 L^{\text{VF}}(\theta) \right]$$

Coefficient to balance loss values

PPO surrogate objective

MSE loss for value function estimates

Combined surrogate and value function loss for PPO

Does it perform well? When PPO is compared to a variety of state-of-the-art R algorithms on problems with continuous action spaces, we see that it tends to learn faster and outperforms prior techniques on nearly all tasks; see below.

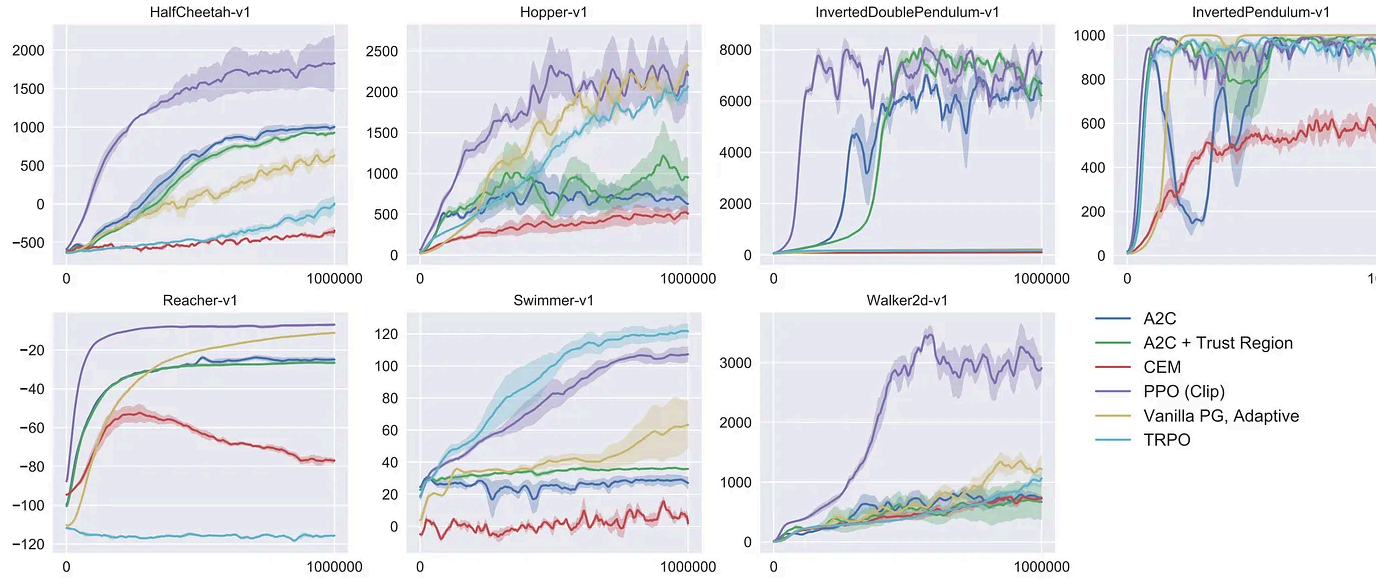


Figure 3: Comparison of several algorithms on several MuJoCo environments, training for one mi timesteps.

(from [2])

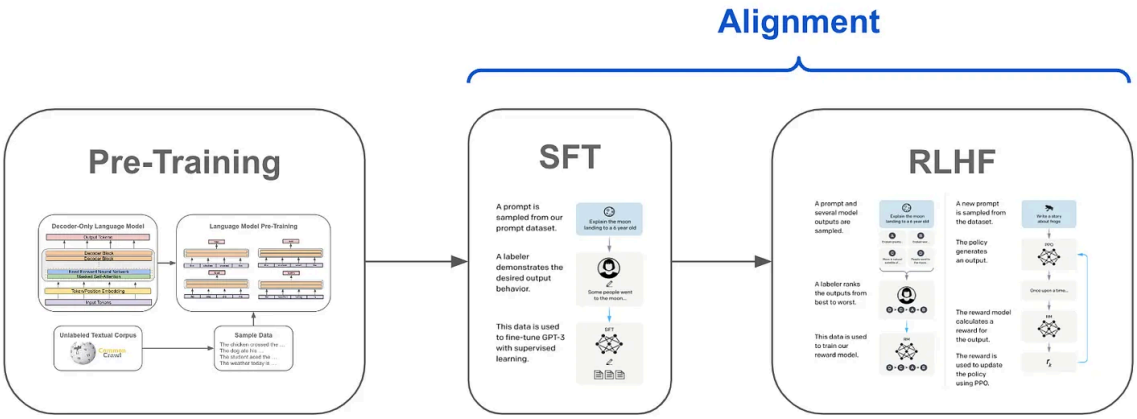
In the Atari domain, PPO is found to perform comparably to algorithms like A [4] and ACER [5], despite these algorithms being tuned extensively for this dom The table below outlines the number of games won by each algorithm.

	A2C	ACER	PPO	Tie
(1) avg. episode reward over all of training	1	18	**30**	0
(2) avg. episode reward over last 100 episodes	1	**28**	19	1

Table 2: Number of games "won" by each algorithm, where the scoring metric is averaged across three tr

(from [2])

The Role of PPO in RLHF



Steps of training a language model (from [6])

Although PPO was a useful advancement in mainstream RL research, this algorithm also had a massive impact on the space of language modeling. More specifically, InstructGPT [6]—the predecessor to ChatGPT—was aligned (i.e., trained to produce output that aligns with human expectations) via a three-part framework (shown above) that includes both supervised fine-tuning (SFT) and reinforcement learning from human feedback (RLHF). Although such an appro was explored previously for text summarization tasks [7], InstructGPT populari this framework for training language foundation models, leading to its use in tl creation of a variety of popular language models; e.g., ChatGPT, GPT-4, LLaM and Sparrow.

How does this relate to PPO? Due to its ease of use, PPO was the RL algorithn that was originally selected for use in RLHF by InstructGPT. The alignment

strategy used by InstructGPT later became standardized and, though alternativ have been explored, PPO remains a popular choice for RLHF even today. As su PPO is a pivotal aspect of language model alignment, and any AI practitioner w an interest in understanding or implementing the alignment process would benefit from a working understanding of PPO.

Takeaways

Within this overview, we expanded our understanding of basic policy gradient algorithms, such as VPG, to include more recent RL algorithms like TRPO and PPO. Compared to prior algorithms, TRPO and PPO have improved data efficiency and better reliability/stability. Notably, PPO is the primary RL algorithm used by RLHF, making it a key component of the language model alignment process. Although many factors led to the use of PPO in this domain *PPO's ease-of-use is undoubtedly a key aspect of its success*. In particular, PPO inhe the data efficiency and reliability of TRPO with several added benefits:

- Improved robustness (i.e., not much tuning required)
- Better data efficiency
- Simplicity (i.e., only requires small tweaks to VPG)
- More general (i.e., applies to a wider class of model architectures)

Due to these (many) factors, PPO is a popular choice of RL algorithm among practitioners. As the learning algorithm used for RLHF, PPO is the foundation language model alignment. With a deeper understanding of this algorithm, we gain a new level of insight into how language models learn and behave.

New to the newsletter?

Hi! I'm Cameron R. Wolfe, deep learning Ph.D. and Director of AI at Rebuy. Th is the Deep (Learning) Focus newsletter, where I help readers understand AI research via overviews of relevant topics from the ground up. If you like the newsletter, please subscribe, share it, or follow me on Medium, X, and LinkedI

Bibliography

[1] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. "Trust region policy optimization". In: CoRR, abs/1502.05477 (2015).

[2] Schulman, John, et al. "Proximal policy optimization algorithms." *arXiv prep arXiv:1707.06347* (2017).

[3] Schulman, John, et al. "High-dimensional continuous control using generali advantage estimation." *arXiv preprint arXiv:1506.02438* (2015).

[4] V. Mnih, et al. "Asynchronous methods for deep reinforcement learning". In: arXiv preprint arXiv:1602.01783 (2016).

[5] Z. Wang et al. "Sample Efficient Actor-Critic with Experience Replay". In: arXiv preprint arXiv:1611.01224 (2016).

[6] Ouyang, Long, et al. "Training language models to follow instructions with human feedback." *Advances in Neural Information Processing Systems* 35 (2022): 27730-27744.

[7] Stiennon, Nisan, et al. "Learning to summarize with human feedback." *Advances in Neural Information Processing Systems* 33 (2020): 3008-3021

[8] Touvron, Hugo, et al. "Llama 2: Open foundation and fine-tuned chat models." *arXiv preprint arXiv:2307.09288* (2023).

[9] Achiam, Josh. *Spinning Up in Deep RL*. OpenAI, 2018: https://spinningup.openai.com/en/latest/index.html

[10] Schulman, John. *Optimizing expectations: From deep reinforcement learning to stochastic computation graphs*. Diss. UC Berkeley, 2016.

[11] Glaese, Amelia, et al. "Improving alignment of dialogue agents via targeted human judgements." *arXiv preprint arXiv:2209.14375* (2022).

---

1 Despite this fact, many open-source LLMs are aligned using solely SFT instead of a combination of SFT and RLHF. LLaMA-2 breaks this trends by heavily investing int curating a massive human preference dataset for alignment via RLHF.

2 By continuous domain, we mean that actions outputted by the agent are continuous, rather than discrete. For example, driving a car is a continuous domain, as we can adjust the steering wheel and pedals according to a continuous output (i.e., angle or pressure). In contrast, chess would be a discrete domain, as there are a fixed number actions capable of being taken at each state.

3 We can also use other algorithms, but PPO is the original (and most common) choic RL algorithm for RLHF. Recent research has begun to explore alternative choices.

4 Language models have a discrete action space comprised of all tokens that can be outputted by the model. However, this action space is still quite large. Most languag models have a vocabulary size of tens to hundreds of thousands of tokens!

5 In discussions of TRPO, we will see this referred to as solving the “surrogate” objec This surrogate objective simply refers to the maximization problem that we solve during each update step of TRPO. More generally, surrogate functions refer to functions that approximate other functions. In TRPO, the surrogate objective is a problem that we solve in pursuit of achieving an optimal policy!

6 This process is often referred to as *experience replay* and the data that we collect from this process is stored within a *replay buffer*.

7 Notably, TRPO cannot be used for neural network architectures that include noise (e dropout) or perform any form of parameter sharing (e.g., between the policy and the value function).

8 The objective that we use in practice is different from what we see in the equations. This is because we cannot compute an expectation exactly, as this would require tak an average over all possible trajectories from our current policy. Instead, we collect from the environment and take a sample mean to approximate this objective.

9 These are the same exact tools that we use for training neural networks according to common objectives such as supervised of self-supervised learning!

25 Likes · 3 Restacks

Discussion about this post

Comments Restacks

Write a comment...

Michael Lux Umbra Dei Oct 23, 2023 *Edited*

Liked by Cameron R. Wolfe, Ph.D.

Cameron, could you recommend a SOTA textbook aimed at a graduate level audience on this tc know it is a dynamic area and a moving target, but a foundational text would be great.

LIKE (2) REPLY

1 reply by Cameron R. Wolfe, Ph.D.

1 more comment...