Study Note: Proximal Policy Optimization

Yanqing Wu

Advisor: Prof. Hengshuai Yao

Viwistar Robotics

Update: November 30, 2021

Since some parts are not addressed directly from the literature, the explanations in this note are the best of my understanding.

1 Background

In normal policy gradient, training can be unstable due to sparse reward and sampled target, causing dramatic changes on the policy; this eventually collapses performance. To enhance training stability, we want to limit the parameter updates that would change the policy too much at one step. With this goal, True Region Policy Optimization (TRPO) Schulman et al. (2017a) was introduced. TRPO delimit the KL-divergence with a constraint. More specifically, TRPO updates policies by taking the largest step possible to improve performance while satisfying the constraint on how close the new and old policies are allowed to be. However, TRPO is complicated to implement and is computational-heavy (from the computation of second-order derivatives of KL-divergence). Proximal Policy Optimization (PPO) Schulman et al. (2017b) simplifies it by using a *clipped surrogate objective* while retaining similar constraints and similar performance. Instead of enforcing a hard constraint as in TRPO, PPO formalized the constraint as a penalty in the objective function.

There are two primary variants of PPO: PPO-Clip and PPO-Penalty.
PPO-Clip does not have a KL-divergence term in the objective and does not have a constraint at all. Instead relies on specialized clipping in the objective function to remove incentives for the new policy to drift from the old policy, PPO does hard clipping the policy ratio to be within a small range around 1.0, where 1.0 means the new policy is the same as old.
PPO-Penalty approximately solves a KL-constrained update like TRPO, but penalizes the KL-divergence in the objective function instead of making it a hard constraint, and automatically adjusts the penalty coefficient throughout training so that it is scaled appropriately. PPO-Penalty performs worse than PPO-Clip Schulman et al. (2017b).

2 Implementation

PPO uses the Actor-Critic approach. It uses two models, Actor Model and Critic Model, as shown in Figure 1.

The Actor model (policy π) learns what action to take under a particular observed state of the environment. The action predicted by the actor is sent to the environment. Afterwards, observation is made and reward is given from the environment and fed into the Critic Model.

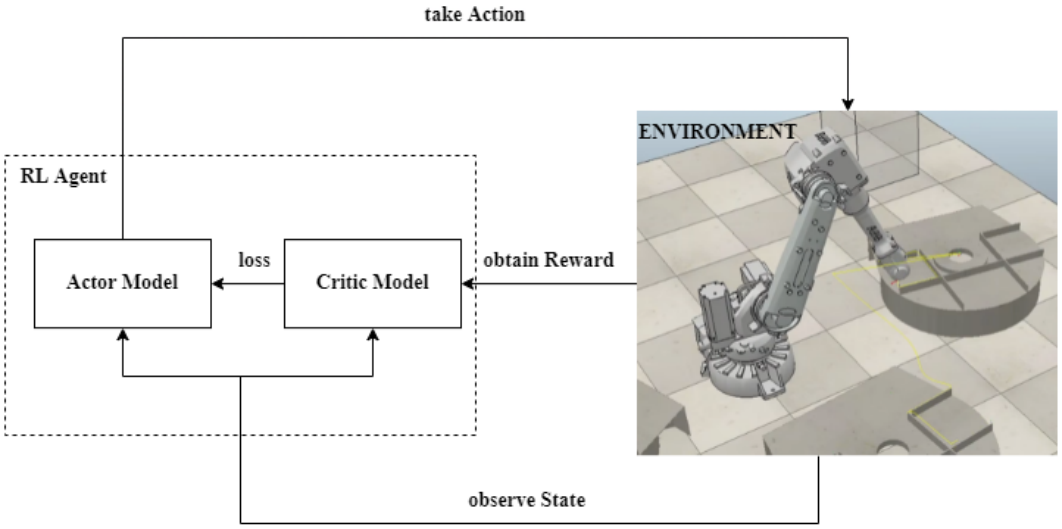


Figure 1: PPO agent

The Critic model (value $V(s)$) learns to evaluate changes in the environment after action is taken by the Actor, and gives its feedback to the Actor. It outputs a real number indicating a rating (Q-value) of the action taken in the previous state. By comparing this rating obtained from the Critic, the Actor compares its current policy with a new policy and decides how it wants to improve itself to take better actions.

2.1 OpenAI 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

Figure 2: Screenshot of OpenAI PPO algorithm

2.1.1 Key components

Policy loss, value function loss, KL loss (optional) and entropy regularization make up the loss for PPO.

Probability Ratio $r_t(\theta)$ calculates how much the policy has changed. This ratio decides the tolerance of a change in policy. A clipping parameter epsilon ϵ is used to ensure only the maximum of ϵ change is made to the policy at a time. $r_t(\theta) > 1$ means the action is more

probable for the current policy than the old policy.

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} = \frac{\pi_{new}(\theta)}{\pi_{old}(\theta)} \overset{\text{computational convenience}}{=} e^{log(\pi_{new}(\theta)) - log(\pi_{old}(\theta))}$$

Advantage $\hat{A}$ measures how much better or worse by taking a particular action in a particular state. Advantage is computed using Generalized Advantage Estimator (GAE(λ)). Truncated version of advantage estimator $\hat{A}_t$ is given by

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1}$$
$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

Actor Loss, or **Policy (Gradient) loss** L^{CLIP} is defined as a minimum of two functions in PPO. The first function p_1 is a surrogate loss function from conservative policy iteration (L^{CPI}), which is from TRPO Schulman et al. (2017a). The second function p_2 is the clipped probability ratio, which $r_t(\theta)$ is limited within the interval $[1-\epsilon, 1+\epsilon]$ (ϵ is clipping range). The minimum of the two is taken so that the final objective is a lower bound (i.e. a *pessimistic bound*) on the unclipped objective.

$$p_1 = \text{ratio} \cdot \text{advantage} = r_t(\theta) \cdot \hat{A}_t$$
$$p_2 = \text{clip}(\text{ratio}, 1-\epsilon, 1+\epsilon) \cdot \text{advantage} = clip(r_t(\theta), 1-\epsilon, 1+\epsilon) \cdot \hat{A}_t$$
$$L^{CLIP}(\theta) = min(p_1, p_2) = \hat{\mathbb{E}}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$$

To illustrate why the minimum of the two functions is chosen, see Figure 3 below. Think of

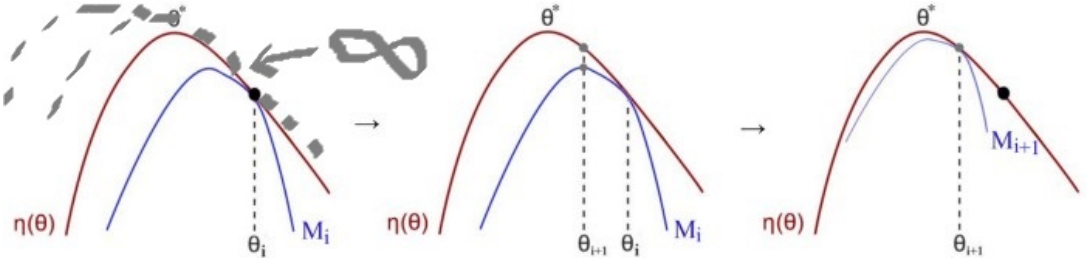


Figure 3: Minorize-Maximization (MM) Algorithm

the red line as a true objective function, and the blue line as a local approximation. If we choose to maximize all the way, the approximation will go to infinity. If we take local approximation and subtract a penalty, we receive a lower bound on the true objective function. When we make progress on this pessimistic version of the objective, we are guaranteed to make progress and eventually converge on the true objective function Schulman (2021). The idea behind this is Minorize-Maximization Algorithm.

Policy loss correlates to how much the policy (process for deciding actions) is changing. The magnitude of this should decrease during a successful training session. These values will oscillate during training. Generally they should be less than 1.0.

Another version of L^{CLIP} is as follows:

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

where $g(\epsilon, A) = (1 + \epsilon)A$ when $A \geq 0$ and $g(\epsilon, A) = (1 - \epsilon)A$ when $A < 0$. Through some simplifications, we see that probability ratio $\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}$ is clipped at $1+\epsilon$ $(A > 0)$ and $1-\epsilon$ $(A < 0)$. The hyperparameter ϵ specifies how much the new policy is allowed to change from the old policy (while still profiting the objective). Therefore, the clipping acts as a regularizer by discouraging the policy to change dramatically (see Figure 4).

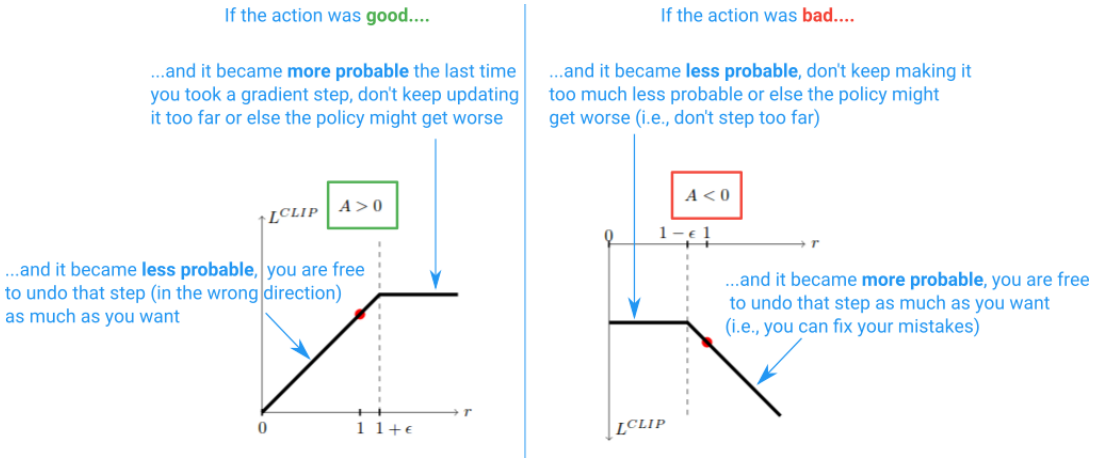


Figure 4: Effect of L^{CLIP} function matwilso (2021)

Critic Loss, or **Value (Function) Loss** L^{VF} is defined as mean squared error between critic values and returns (or V_t^{target} in the paper). Target value is computed using TD(λ) estimator. Value Loss correlates to how well the model is able to predict the value of each state. This should increase while the agent is learning, and then decrease once the reward stabilizes. These values will increase as the reward increases, and then should decrease once reward becomes stable.

$$L^{VF} = (V(s) - R)^2 = (V_\theta(s_t) - V_t^{target})^2$$

Total Loss, or **Objective Loss** $L_t^{CLIP+VF+S}$ is defined as[1][2]:

$$\text{total loss} = \text{actor loss} + \text{critic loss} - \text{entropy}$$

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF} + c_2 S[\pi_\theta](s_t)]$$

[1]The plus/minus sign of each loss component differs from source to source. The sign of the first (text) equation is align with PPO2 source code and some online post; the sign of the second (letter) equation is align with the paper.
[2]Based on discussion with the Advisor, the plus/minus sign of the first (text) equation should be the correct version. We must maximize entropy to minimize loss, hence the minus sign before entropy.

where c_1, c_2 are coefficients, θ is a vector of policy parameters, S is entropy bonus that acts as a regularizer to encourage policy exploration (see Appendix A for more information of *Entropy*).

Note that actor loss (L_t^{CLIP}) and critic loss (L_t^{VF}) correspond to exploitation, and entropy regularization (S) corresponds to exploration. The total loss equation here is a trade-off problem between exploitation and exploration, which is one of the key features and challenges in RL problem.

2.1.2 Practical considerations

There are two alternating threads in PPO. In the first thread, policy interacts with the environment, collects data and computes advantage estimates (using fitted baselines estimates); in the second thread, it collects all the experiences and runs Stochastic Gradient Descent (SGD) to optimize the policy using the clipped objective.

The surrogate loss L is optimized with minibatch SGD on NT timesteps of data, where N is the number of parallel actors and T is timesteps.

2.2 DeepMind PPO

DeepMind paper provides a more detailed pseudocode than OpenAI. According to DeepMind pseudo-code of PPO (5)

Algorithm 1 Proximal Policy Optimization (adapted from [8])

for $i \in \{1, \cdots, N\}$ **do**
  Run policy π_θ for T timesteps, collecting $\{s_t, a_t, r_t\}$
  Estimate advantages $\hat{A}_t = \sum_{t'>t} \gamma^{t'-t} r_{t'} - V_\phi(s_t)$
  $\pi_{\text{old}} \leftarrow \pi_\theta$
  for $j \in \{1, \cdots, M\}$ **do**
    $J_{PPO}(\theta) = \sum_{t=1}^{T} \frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)} \hat{A}_t - \lambda \text{KL}[\pi_{old}|\pi_\theta]$
    Update θ by a gradient method w.r.t. $J_{PPO}(\theta)$
  end for
  for $j \in \{1, \cdots, B\}$ **do**
    $L_{BL}(\phi) = -\sum_{t=1}^{T} (\sum_{t'>t} \gamma^{t'-t} r_{t'} - V_\phi(s_t))^2$
    Update ϕ by a gradient method w.r.t. $L_{BL}(\phi)$
  end for
  if $\text{KL}[\pi_{old}|\pi_\theta] > \beta_{high} \text{KL}_{target}$ **then**
    $\lambda \leftarrow \alpha\lambda$
  else if $\text{KL}[\pi_{old}|\pi_\theta] < \beta_{low} \text{KL}_{target}$ **then**
    $\lambda \leftarrow \lambda/\alpha$
  end if
end for

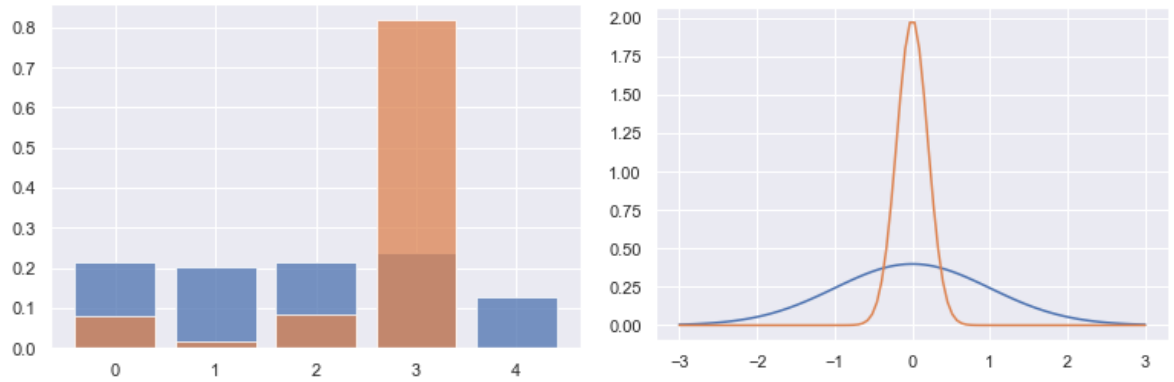
Figure 5: Screenshot of DeepMind PPO algorithm

In this algorithm, KL_{target} is the desired change in the policy per iteration. The Actor maximizes J_{PPO}, and Critic minimizes L_{BL}.

Appendices

Appendix A Entropy

Entropy was first recognized in classical thermodynamics and was brought to information theory by Claude Shannon in the mid 20th century. It is now widely used and is most commonly associated with a state of disorder, randomness, or uncertainty. In the realm of machine learning, entropy is related to randomness in the information being processed in a system. In other words, high entropy means that the randomness in the system is high, meaning it is difficult to predict the state in it. More specifically, in reinforcement learning (RL), entropy refers to the unpredictability of the actions taken of an agent. The more uncertain of which action will be taken by the agent, the higher the entropy. An example can be seen below in Figure 6.



(a) Discrete probability distribution $p(a|s)$ **(b)** Continuous probability distribution $p(a|s)$

Figure 6: Orange shows low-entropy distributions. Blue shows high-entropy distributions.

A.1 Terminology

Based on my understanding of RL literature, 'entropy' technique in RL normally refers to 'entropy regularization'. 'entropy regularization' technique includes but not limited to adding 'entropy bonus' in objective function and using 'maximum-entropy'. 'entropy bonus' includes but not limited to 'one-step (naive) entropy bonus' and 'proper entropy bonus'. The term 'entropy bonus' is usually a constant or a vector of real numbers. 'one-step entropy bonus' only applies to the current state, while 'maximum-entropy' optimizes over the long-term sum of entropy. There is also a technique to maximize over both long-term rewards and long-term entropy. This is referred to as maximum entropy reinforcement learning.

A.2 Why use entropy in RL

It is a common practice to use entropy regularization to ensure policy exploration in policy gradient methods O'Donoghue et al. (2017). An agent might be stuck in a local optimum or never finding the global optimum because of not exploring the behavior of other actions. Entropy encourages exploration, avoiding the situations where agent might fall into a local optimum. This is critical for tasks with sparse reward because the agent seldomly receives feedback for its action, and therefore might overestimate some reward received and repeat the actions that led to that reward. With improved exploration, the agent will be more robust to abnormal or rare events while developing a task.

Other benefits of using entropy includes augmenting RL objective Ziebart (2010), encouraging longer episodes Schulman et al. (2018), fine-tuning policies and better adaptability to new environments Haarnoja et al. (2017).

A.3 How to use entropy in RL

In information theory, entropy for a discrete random variable x is defined as:

$$H(X) = -\sum_{x \in X} P(x) log P(x) \tag{1}$$

In RL, the formula becomes Equation 2 as we calculate the entropy of the policy $\pi(a|s_t)$ for discrete action space (replace sum with integral for continuous action space).

$$H(\pi(\cdot|s_t)) = -\sum_{a \in \mathcal{A}} P(x) log P(x) \tag{2}$$

It is typical to add an 'entropy bonus' term to the loss function, which encourages the agent to take actions more unpredictably so as to counteract the tendency of falling into local optimum O'Donoghue et al. (2017). For example, in A3C Mnih et al. (2016)

$$\nabla_{\theta'} log \pi(a_t|s_t;\theta')(R_t - V(s_t;\theta_v)) + \beta \nabla_{\theta'} H(\pi(s_t;\theta'))$$

where $H(\pi)$ is the entropy bonus term, and the hyperparameter β controls the strength of the entropy regularization term. Mnih et al. (2016) found that adding the entropy of the policy π to the objective function improved exploration by discouraging premature convergence to suboptimal deterministic policies.

For one more example, Schulman et al. (2018) discussed naive and proper entropy bonuses for A2C (advantage actor critic; a well-tuned version of A3C). The entropy bonus term is added in following way:

naive / one-step entropy bonus:

$$\nabla log \pi_\theta(a_t|s_t) \left(\sum_{d=0}^{n-1} \gamma^d r_{t+d} - V(s_t) \right) - \tau \nabla_\theta \underbrace{D_{KL}[\pi_\theta || \bar{\pi}](s_t)}_{\text{entropy bonus}}$$

proper entropy bonus:

$$\nabla log \pi_\theta(a_t|s_t) \left(\sum_{d=0}^{n-1} \gamma^d (r_{t+d} - \tau \underbrace{D_{KL}[\pi_\theta || \bar{\pi}](s_{t+d})}_{\text{entropy bonus}}) - V(s_t) \right) - \tau \nabla_\theta \underbrace{D_{KL}[\pi_\theta || \bar{\pi}](s_t)}_{\text{entropy bonus}}$$

A side note for training: entropy should slowly and consistently decrease during a successful training process. If it decreases too quickly, the entropy coefficient hyperparameter should be increased.

A.4 Further reading

Application of entropy (regularization) in RL is a well-studied topic, some notable paper in this area:

- Ahmed et al. (2019) Understanding the impact of entropy on policy optimization
- Schulman et al. (2018) Equivalence Between Policy Gradients and Soft Q-Learning
- Haarnoja et al. (2017) Reinforcement Learning with Deep Energy-Based Policies (soft (entropy-regularized) Q-learning)
- Nachum et al. (2017) Bridging the Gap Between Value and Policy Based Reinforcement Learning (entropy bonus)
- O'Donoghue et al. (2017) Combining policy gradient and q-learning (entropy bonus)
- Mnih et al. (2016) Asynchronous Methods for Deep Reinforcement Learning (A3C)
- Ziebart (2010) Modeling purposeful adaptive behavior with the principle of maximum causal entropy
- Williams (1992) Simple statistical gradient-following algorithms for connectionist reinforcement learning

Related topics include one-step entropy vs. maximum entropy, and entropy vs. KL-divergence as regularizer.

Appendix B PPO1 vs. PPO2

PPO2 is made for GPU by OpenAI. It uses vectorized environments for multi-processing while PPO1 uses MPI. PPO2 contains several modifications from the original algorithm which are not documented by OpenAI: value function is also clipped and advantages are normalized Raffin (2021).

PPO1 is not gpu-optimized: it uses one environment per MPI worker. In other words, when running PPO1 with multiple MPI processes, each process creates its own copy of the environment, and its own neural net (NN). The gradients for NN updates are aggregated across the workers by virtue of using MpiAdamOptimizer class. PPO2 implementation (while with recent updates it can use MPI as well) uses a different version of parallelism. Head process with a single neural net creates a bunch of subprocesses that run separate environments (run environments means that take actions and produce next observations and rewards). The observations and rewards from these multiple environments in subprocesses are batched together in the head process. For visual observations, that creates a big enough batch so that computation of NN gradients on a GPU starts making sense. By default, multiple environments in subprocesses are only used for atari and retro video games, but not, for instance, for mujoco (because observations there are not visual) pzhokhov (2021).

In conclusion, PPO1 is obsolete at the time of writing this note, and its functionality is fully covered by PPO2.

B.1 OpenAI source code of PPO

- PPO1: https://github.com/openai/baselines/blob/master/baselines/ppo1/pposgd_simple.py
- PPO2: https://github.com/openai/baselines/blob/master/baselines/ppo2/model.py

References

Ahmed, Z., Roux, N. L., Norouzi, M., and Schuurmans, D. (2019). Understanding the impact of entropy on policy optimization.

Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. (2017). Reinforcement learning with deep energy-based policies.

matwilso (2021). *What is the way to understand Proximal Policy Optimization Algorithm in RL?* https://stackoverflow.com/a/50663200 [Accessed: 2021-11-26].

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning.

Nachum, O., Norouzi, M., Xu, K., and Schuurmans, D. (2017). Bridging the gap between value and policy based reinforcement learning.

O'Donoghue, B., Munos, R., Kavukcuoglu, K., and Mnih, V. (2017). Combining policy gradient and q-learning.

pzhokhov (2021). *[PPO2] What is the difference between PPO1 and PPO2?* https://github.com/openai/baselines/issues/485 [Accessed: 2021-11-26].

Raffin, A. (2021). *PPO2 Stable Baselines 2.10.2 documentation*. https://stable-baselines.readthedocs.io/en/master/modules/ppo2.html [Accessed: 2021-11-26].

Schulman, J. (2021). *University of California, Berkeley CS294-112 11/20/17*. https://www.youtube.com/watch?v=gqX8J38tESw [Accessed: 2021-11-28].

Schulman, J., Chen, X., and Abbeel, P. (2018). Equivalence between policy gradients and soft q-learning.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. (2017a). Trust region policy optimization.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017b). Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256.

Ziebart, B. D. (2010). *Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy*. PhD dissertation, Carnegie Mellon University.