```
In [1]: %run Latex_macros.ipynb
```

```
$$ \newcommand{\x}{\mathbf{x}} \newcommand{\tx}{\tilde{\x}} \newcommand{\y}
{\mathbf{y}} \newcommand{\b}{\mathbf{b}} \newcommand{\c}{\mathbf{c}}
\newcommand{\e}{\mathbf{e}} \newcommand{\z}{\mathbf{z}} \newcommand{\h}
{\mathbf{h}} \newcommand{\u}{\mathbf{u}} \newcommand{\v}{\mathbf{v}}
\newcommand{\w}{\mathbf{w}} \newcommand{\V}{\mathbf{V}} \newcommand{\W}
{\mathbf{W}} \newcommand{\X}{\mathbf{X}} \newcommand{\KL}{\mathbf{KL}}
\newcommand{\E}{{\mathbb{E}}} \newcommand{\Reals}{{\mathbb{R}}}
\newcommand{\ip}{\mathbf{{(i)}}} % % Test set \newcommand{\xt}{\underline{\x}}
\newcommand{\yt}{\underline{\y}} \newcommand{\Xt}{\underline{\X}}
\newcommand{\perfm}{\mathcal{P}} % % \ll indexes a layer; we can change the actual
letter \newcommand{\ll}{l} \newcommand{\llp}{{(\ll)}} % \newcommand{Thetam}
{\Theta_{-0}} % CNN \newcommand{\kernel}{\mathbf{k}} \newcommand{\dim}{d}
\newcommand{\idxspatial}{{\text{idx}}} \newcommand{\summaxact}{\text{max}}
\newcommand{idxb}{\mathbf{i}} % % % RNN % \tt indexes a time step
\newcommand{\tt}{t} \newcommand{\tp}{{(\tt)}} % % % LSTM \newcommand{\g}
{\mathbf{g}} \newcommand{\remember}{\mathbf{remember}} \newcommand{\save}
{\mathbf{save}} \newcommand{\focus}{\mathbf{focus}} % % % NLP
\newcommand{\Vocab}{\mathbf{V}} \newcommand{\v}{\mathbf{v}}
\newcommand{\offset}{o} \newcommand{\o}{o} \newcommand{\Emb}{\mathbf{E}} % %
\newcommand{\loss}{\mathcal{L}} \newcommand{\cost}{\mathcal{L}} % %
\newcommand{\pdata}{p_\text{data}} \newcommand{\pmodel}{p_\text{model}} % %
SVM \newcommand{\margin}{{\mathbb{m}}} \newcommand{\lmk}{\boldsymbol{\ell}} %
% % LLM Reasoning \newcommand{\rat}{\mathbf{r}} \newcommand{\model}
{\mathcal{M}} \newcommand{\bthink}{\text{}} \newcommand{\ethink}{\text{}} % % %
Functions with arguments \def\xsy#1#2{#1^#2} \def\rand#1{\tilde{#1}}
\def\randx{\rand{\x}} \def\randy{\rand{\y}} \def\trans#1{\dot{#1}}
```

\def\transx{\trans{\x}} \def\transy{\trans{\y}} % \def\argmax#1{\underset{#1}{\operatorname{argmax}} } \def\argmin#1{\underset{#1}{\operatorname{argmin}} } \def\max#1{\underset{#1}{\operatorname{max}} } \def\min#1{\underset{#1}{\operatorname{min}} } % \def\pr#1{\mathcal{p}(#1)} \def\prc#1#2{\mathcal{p}(#1 \; | \; #2)} \def\cnt#1{\mathcal{count}_{#1}} \def\node#1{\mathbb{#1}} % \def\loc#1{{\text{##} {#1}}} % \def\OrderOf#1{\mathcal{O}\left({#1} \right)} % % Expectation operator \def\Exp#1{\underset{#1}{\operatorname{\mathbb{E}}} } % % VAE \def\prs#1#2{\mathcal{p}_{#2}(#1)} \def\qr#1{\mathcal{q}(#1)} \def\qrs#1#2{\mathcal{q}_{#2}(#1)} % % Reinforcement learning \newcommand{\Actions}{{\mathcal{A}}} \newcommand{\actseq}{A} \newcommand{\act}{a} \newcommand{\States}{{\mathcal{S}}} \newcommand{\stateseq}{S} \newcommand{\state}{s} \newcommand{\Rewards}{{\mathcal{R}}} \newcommand{\rewseq}{R} \newcommand{\rew}{r} \newcommand{\transp}{P} \newcommand{\statevalfun}{v} \newcommand{\actvalfun}{q} \newcommand{\disc}{\gamma} \newcommand{\advseq}{\mathbb{A}} % % \newcommand{\floor}[1]{\left\lfloor #1 \right\rfloor} \newcommand{\ceil}[1]{\left\lceil #1 \right\rceil} % % $$

Policy based methods

Recall the Policy Gradient Theorem

- formulated with single reward $\mathbf{\backslash rewseq}(\tau)$ for the entire trajectory

$$\nabla_\theta J(\theta) = \mathbf{\backslash Expr}\tau \sim \mathbf{\backslash pr}\theta \sum_{=0}^{|\tau|} \nabla_\theta \log \pi(\mathbf{\backslash actseq}_{\tau,} | \mathbf{\backslash stateseq}_{\tau,})) \;\; \mathbf{\backslash rewseq}(\tau)$$

- formulated with intermediate rewards

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{=0}^{T-1} \nabla_\theta \log \pi_\theta(\mathbf{\backslash actseq}_{\tau,} | \mathbf{\backslash stateseq}_{\tau,t}) G_{\tau,}\right]$$

Policy based methods update parameters θ

- in the direction (gradient)
- that increases expected return

Advantage vs Return/Reward: Baselines

Suppose all returns/rewards are positive.

The Policy Gradient Theorem would suggest that we update parameters

- to favor *all* actions
- with positive gradients

But consider two actions at some step

- one with a *very large* positive return/reward
- one with a *very small* positive return/reward

We probably want to favor the action with large reward.

Thus, we want to *relativize* the return/reward by comparing it to a *baseline* for the state.

The relativized value of an action is called the *advantage*.

Often, the advantage is derived from the action's return/reward by

- subtracting a *baseline*
- where the baseline value
    - is a function of *all* possible actions from the state

For example: consider a baseline that is the average return/reward from the state

- above average returns have positive advantage
- below average returns have a negative advantage

Thus, in the case of all returns/rewards being positive

- we favor actions that result in positive advantages

Subtracting a baseline from the return/reward has other advantages

- reduces the *magnitude* of the parameter update
    - smoother training
- reduces "noise"
    - the baseline value for a state is common to *all actions* from the state
    - so the advantage is relative to the signal *only* from the action itself

Aspect	Effect of Subtracting Baseline
Noise cancellation	Removes expected (common) return, reducing fluctuations
Bias introduction	None, baseline does not depend on action
Zero-centering	Advantages centered around zero, stabilizing updates

Unified Policy Gradient Formulation: Advantage Definitions

Using the concept of Advantage, we can rewrite the Policy Gradient Theorem

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{\text{=0}}^{T-1} \nabla_\theta \log \pi_\theta(\backslash\mathrm{actseq}_{\tau,} | \backslash\mathrm{stateseq}_{\tau,t})\backslash\mathrm{advseq}_{\tau,}\right]$$

The form of the advantage might vary depending on whether there is

- a single per-trajectory reward
- or intermediate rewards.

We can characterize many policy-based methods

- by defining their advantage calculation

$$\backslash\mathrm{advseq}_{\tau,}$$

We thus refer to the above formulation as the Unified Policy Gradient Formulation.

The policy-based methods we present will typically be variations of this "vanilla" gradient.

Unified Policy Gradient Formulation: Advantage Definitions

All methods optimize:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{=0}^{T-1} \nabla_\theta \log \pi_\theta(\backslash\mathrm{actseq}_{\tau,} | \backslash\mathrm{stateseq}_{\tau,t})\backslash\mathrm{advseq}_{\tau,}\right]$$

where the advantage

$$\backslash\mathrm{advseq}_{\tau,}$$

has the following forms:

Method	Intermediate Reward Advantage $\backslash\mathbf{advseq}_{\tau,}$	Single Trajectory Reward Advantage $\backslash\mathbf{advseq}_{\tau}$
REINFORCE	$\backslash\mathrm{advseq}_t = G_t - b_t$	$\backslash\mathrm{advseq} = R(\tau) - b$
PPO	$\backslash\mathrm{advseq}_t = \hat{A}_t$ (e.g., GAE estimator)	$\backslash\mathrm{advseq} = R(\tau) - b$ applied per step
GRPO	Relative advantages per candidate	Same per-candidate advantage applied per step

This unified view clarifies that the choice of advantage estimate adapts to the available feedback type.

Policy based methods

We will show several common policy-based methods.

- REINFORCE
- PPO
- GRPO (relatively new: 2024)

Details of each follow.

As a preview, we show the definition of the Advantage for each.

Surrogate Loss

Rather than

- maximization of return

Policy methods often switch to

- minimization of a *surrogate loss*

The surrogate loss often imposes constraints on the derived policy and on the solution process.

Some of these constraints constrain the policy, for example

- so that it is "close" to a reference policy

Other constraints try to promote stability and convergence in the solution process

- by limiting how far each policy update can diverge from the previous policy
- limiting the magnitude of a gradient update
- promoting low variance of the gradient updates

Constraints

Reason	Explanation
Trust Region Constraints	Avoid overly large policy updates that destabilize learning
Regularization	KL divergence or clipping adds penalty to maintain stability
Stability & Convergence	Ensures smooth, incremental improvement of the policy
Computational Tractability	Easier to optimize surrogate than raw return objective
Exploration-Exploitation Tradeoff	Clipping controls how much policy can change per step

The Policy Gradient Theorem provides

- the *theoretical* basis for an optimal policy

The Surrogate Loss provides

- the *practical* objective for finding the optimal policy
- subject to practical constraints: stability, convergence

The actual implementation of our models will usually revert to the Minimization of Surrogate Loss forumulation.

Relationship between Policy Gradient Theorem and Surrogate Loss

Aspect	Policy Gradient Theorem	Surrogate Loss
Role	Theoretical formula for expected return gradient	Practical approximation/objective for policy update
Gradient	Directly gives unbiased gradient of $J(\theta)$	Its gradient approximates policy gradient but includes stabilizing terms
Constraints	None intrinsic; pure gradient formula	Includes clipping, penalties to enforce trust region and avoid large steps
Use in Algorithms	Foundation for policy gradient methods	Used in PPO-style algorithms to compute safe gradient steps
Goal	Maximize expected return $J(\theta)$	Provide stable, incremental improvement proxy

Policy-based methods: Summary Table

Here is a preview of the policy-based methods we will explore.

Method	Surrogate Return/Objective Used for Gradient	Credit Assignment
REINFORCE	G_t (return-to-go from t)	Monte Carlo trajectory return
PPO	Clipped ratio times A_t (advantage)	Advantage-based, stable
GRPO	Relative advantages over candidate	

REINFORCE

REINFORCE is a policy-based method.

It uses a minor variation of the Policy Gradient Theorem to implement the gradient

- subtracts a "baseline" from the reward

Method	Intermediate Reward Advantage $A_{\tau,}$	Single Trajectory Reward Advantage A_τ
REINFORCE	\advseq_\tt = G_\tt - b_\tt	\advseq $= R(\tau) - b$

The purpose of subtracting a baseline is to

- reduce the magnitude of the gradients
- reduces the variance of the gradients
- and hence: the change in policy parameters θ

With a single trajectory reward

$$\backslash\mathrm{advseq}_{\tau,} = (\backslash\mathrm{rewseq}(\tau) - b)$$

- b is often the moving average (over trajectories τ) of $\backslash\mathrm{rewseq}(\tau)$
- advantage is the same for every step

With intermediate rewards

\advseq_{\tau,\tt} = \left(G_{\tau, \tt} - b_\tt \right)

- where b_\tt is often a proxy for the *expected* Value function $\backslash\mathrm{statevalfun}_{\pi}(\backslash\mathrm{stateseq}_{)}$ of state \stateseq_\tt

It uses a Monte-Carlo method to estimate the gradient

- based on a sample of trajectories

Discussion

REINFORCE is very close to the vanilla Policy Gradient.

It attempts to reduce Gradient variance by subtracting a baseline.

But it is still considered high variance.

Subsequent methods will take explicit steps to reduce variance.

Pseudo code for REINFORCE

```
# REINFORCE training for LLM
for prompt in training_prompts:
    output = llm.generate(prompt)
    reward = evaluate_output(output) # Human or automated score
    logprob = llm.logprob(output, prompt)

    # Monte Carlo policy gradient update (no critic)
    baseline = compute_baseline() # Optional: running mean for variance reducti
on
    loss = -logprob * (reward - baseline)
    loss.backward()
    optimizer.step()
```

PPO

Intutition

PPO optimizes a **clipped surrogate** loss

- which can be interpretted as variant of the standard policy gradient theorem.

The goal of the surrogate objective is to control how rapidly the policy changes from epoch to epoch of training.

- avoid large policy shifts
- variance reduction
- monotonic improvement of policy

We express the relative change in policy with the *probability ratio*

$$r_{(}\theta) = \frac{\pi_{\theta}(\backslash\mathrm{actseq}_{|}\backslash\mathrm{stateseq}_{)}}{\pi_{\theta_{\mathrm{old}}}(\backslash\mathrm{actseq}_{|}\backslash\mathrm{stateseq}_{)}}$$

where $\pi_{\theta_{\mathrm{old}}}$ is the *reference policy*

The reference policy is the policy in effect at the start of each epoch of training

- before this epoch modifies the policy parameters

During an epoch of training

- trajectories are generated the trajectories using the reference policy
- the policy parameters are updated based on the surrogate loss for these trajectories
- the updated policy becomes the reference policy for the next epoch

By keeping the probability ratio close to 1, we constrain the Gradient Ascent update step to a small change in policy.

How to we keep the probability ratio close to 1 ?

- by using clipping to constrain it to the range $[1 - \epsilon, 1 + \epsilon]$ for small ϵ

this is the *clipped* part of the clipped surrogate objective.

Formally

The *surrogate loss* in PPO is:

```
L_{\text{sur}}(\theta) = \mathbb{E}_\tt \left[ r_\tt(\theta) \hat{\advseq}_\tt \right]
= \mathbb{E}_\tt \left[ \frac{\pi_\theta(\actseq_\tt|\stateseq_\tt)}{\pi_{\theta_{\rm old}}(\actseq_\tt|\s
```

where

- $r_{(}\theta)$ is the probability ratio,
- $\hat{\backslash\mathrm{advseq}}_t$ is the advantage estimate at step t.

The *clipped surrogate loss* is

L_{\text{clip}}(\theta) = \mathbb{E}_\tt \left[\min{} \left(r_\tt(\theta) \hat{\advseq}_\tt, \; \mathrm{cli

By minimizing the clipped surrogate loss

- we maximize $J(\theta)$ (our goal in Gradient Ascent)
- in a controlled manner

Relation to the Unified Gradient Formulation

On the surface, the term

$$r_{(}(\theta) = \frac{\pi_\theta(\backslash\mathrm{actseq}_{|}\,\backslash\mathrm{stateseq}_{)}}{\pi_{\theta_{\mathrm{old}}}(\backslash\mathrm{actseq}_{|}\,\backslash\mathrm{stateseq}_{)}}$$

does not resemble the corresponding term

$$\log \pi_\theta(\backslash\mathrm{actseq}_{\tau,}|\backslash\mathrm{stateseq}_{\tau,t})$$

that we see we see in our unified formulation

But, these terms appear inside the Expectation of the Gradient and, by using the Likelihood Ratio trick

$$\begin{aligned}
\nabla_\theta r_{(}\theta) &= r_{(}\theta)\,\nabla_\theta \log \mathbf{r}_{(}\theta) \\
&= r_{(}\theta)\,\nabla_\theta \left(\log \pi_\theta (\backslash\text{actseq}_{|}\, \backslash\text{stateseq}_{)} - \log \pi_{\text{old}} \backslash\text{actseq}_{|}\, \backslash\text{stateseq}_{)}\right) \\
&= r_{(}\theta)\,\nabla_\theta \log \pi_\theta (\backslash\text{actseq}_{|}\, \backslash\text{stateseq}_{)}
\end{aligned}$$

Note

We still have an extra $r_{(}\theta)$ multiplicative term (relative to the Unified Gradient Formulation).

So we don't obtain an *identical* expression

- but we note that, with clipping, $r_{(}\theta) \approx 1$

Advantage for PPO

Reward Setting	**Advantage Estimate**
Intermediate Rewards	$\hat{\backslash\mathrm{advseq}}_t = G_t$ or GAE $- \backslash\mathrm{statevalfun}_\pi(\backslash\mathrm{stateseq}_{)}$
Single Total Reward	$\hat{\backslash\mathrm{advseq}}_t = R(\tau) - b$

Common ways to compute Advantage Estimate $\hat{\backslash\mathrm{advseq}}_t$ for Intermediate Rewards case

- **Using Return-to-go and value estimate:**

$$\hat{\backslash\mathrm{advseq}}_t = G_t - V(s_t)$$

where G_t is the discounted sum of rewards (return-to-go) starting from time step t, and $V(s_t)$ is the estimated state value.

- **Generalized Advantage Estimation (GAE):**

$$\hat{\backslash\mathrm{advseq}}_t = \sum_{l=0}^{\infty}(\gamma\lambda)^l\delta_{t+l}$$

where

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

with

- γ as the discount factor,
- $\lambda \in [0, 1]$ controlling the bias-variance trade-off.

- When $\lambda = 0$, GAE reduces to the **1-step Temporal Difference (TD) advantage**:
- When $\lambda = 1$, GAE reduces to the **Monte Carlo advantage**:

GAE smoothly interpolates between high-bias low-variance and low-bias high-variance advantage estimates, making it very effective in practice.

Discussion

PPO takes explicit steps

- clipped probability ratio

to reduce variance of Gradient estimates.

It is robust and is a very common method when performing RL Tuning.

Pseudo code for PPO

Detailed Surrogate Loss for PPO

```
J_{\mathrm{PPO}}(\theta) = \mathbb{E}_\tt \left[
\min{} \left(
r_\tt(\theta) \hat{A}_\tt,\;
\mathrm{clip}\left(r_\tt(\theta), 1 - \epsilon, 1 + \epsilon\right) \hat{\advseq}_\tt
\right)
\right]
```

```
# PPO training for LLM
for prompt in training_prompts:
    output = llm.generate(prompt)
    reward = evaluate_output(output)
    logprob_old = llm.logprob(output, prompt) # From previous policy
    value = critic(output, prompt) # Critic gives value baseline

    # Calculate advantage
    advantage = reward - value

    # Compute importance ratio
    logprob_new = llm_new.logprob(output, prompt)
    ratio = exp(logprob_new - logprob_old)

    # Clipped surrogate objective for stability
    clip_epsilon = 0.2
    loss1 = ratio * advantage
    loss2 = clip(ratio, 1-clip_epsilon, 1+clip_epsilon) * advantage
    loss = -min(loss1, loss2)
    loss.backward()
```

PPO with batches

```
for epoch in range(num_epochs):
    for batch in data_loader:
        # Generate trajectories
        contexts = batch["contexts"]
        responses = model.sample(contexts)  # generate outputs

        # Score trajectories with reward model
        rewards = reward_model.score_batch(contexts, responses)

        # Compute value estimates and advantages (using GAE or similar)
        values = value_model.predict(contexts, responses)
        advantages = compute_advantages(rewards, values)

        # Get old policy log-probs for PPO ratio calculation
        old_log_probs = model.log_prob(contexts, responses).detach()

        # Forward pass to get new log probabilities
        new_log_probs = model.log_prob(contexts, responses)

        # Calculate PPO ratio
```

Note that all the mathematical operations

- are performed in parallel on each example in the batch
- the per-example loss
    - is reduced to a single scalar
    - via the `.mean()`

```
loss = -torch.min(surrogate1, surrogate2).mean()
```

Trust Region Policy Optimization (TRPO) Theory (Brief)

We stated that, constraining the probability ratio

$$r_t(\theta) = \frac{\pi_\theta(\backslash\mathrm{actseq}_{|}\ \backslash\mathrm{stateseq}_{)}}{\pi_{\theta_{\text{old}}}(\backslash\mathrm{actseq}_{|}\ \backslash\mathrm{stateseq}_{)}}$$

to the small range $[1-\epsilon, 1+\epsilon]$

is the key to the monotonic improvement of the optimization objective.

This is based on *Trust Region Policy Optimizaton (TRPO)*.

We state TRPO briefly.

TRPO formulates policy optimization as a constrained optimization problem to ensure stable and monotonic policy improvement.

It maximizes a surrogate objective subject to a constraint on how much the new policy can deviate from the old policy:

$$\max_{\theta} \quad \mathbb{E}_{s,a\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A^{\pi_{\theta_{\text{old}}}}(s,a)\right]$$

subject to the trust region constraint:

$$\mathbb{E}_{s\sim\pi_{\theta_{\text{old}}}}\left[D_{\text{KL}}\left(\pi_{\theta_{\text{old}}}(\cdot|s) \parallel \pi_\theta(\cdot|s)\right)\right] \leq \delta$$

where:

- $\pi_{\theta_{\text{old}}}$ is the old (reference) policy,
- π_θ is the new policy parameterized by θ,
- $A^{\pi_{\theta_{\text{old}}}}(s,a)$ is the advantage function with respect to the old policy,
- $D_{\text{KL}}(\cdot\|\cdot)$ is the Kullback-Leibler (KL) divergence measuring the difference between the old and new policies,
- δ is a small positive constant controlling the maximum allowed policy update.

The trust region constraint (the KL divergence term)

- **limits the size of the policy update**
- to ensure the new policy does not differ drastically from the old policy
- preventing performance collapse.

This facilitates **monotonic improvement** in policy performance.

PPO can be interpreted as a practical approximation of TRPO

- that replaces the hard KL constraint
- with a clipped probability ratio

in the objective.

```
In [2]: print("Done")
```

```
Done
```