Classic approach: PPO with Preferences instead of Rewards

Before preference-oriented methods were introduced

- PPO was the main method for RL with preferences
- It still is

But it is complex and costly.

- multiple steps

The steps are

- translate preferences to rewards
- run PPO on the translated rewards in the usual manner

The Reward Model

It is difficult and manually-intensive for a human to translate preferences to rewards

- Need to be consistent across examples
- No guiding principles

Instead, we train a *Reward Model* r_ϕ

- Neural Network
- parameterized by ϕ
- satisfying

$$\mathrm{r}_\phi(x, y^+) > \mathrm{r}_\phi(x, y^-)$$

for all preferences

$$(x, y^+, y^-)$$

using per-example Loss function

$$\loss^{\ip}_{\text{reward}}(\phi, x, y^+, y^-) = -\log \sigma\big(\mathrm{r}_\phi(x, y^+) - \mathrm{r}_\phi(x, y^-)\big)$$

The Loss function

- maximizes the spread between the reward for the preferred and non-preferred output
- converts it, via the sigmoid σ function into the range $[0, 1]$

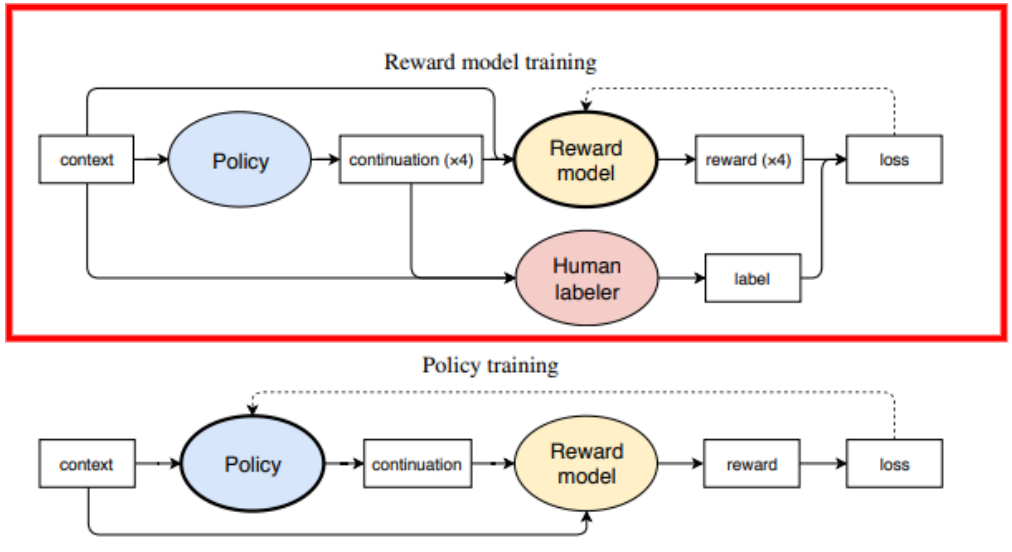
The model is trained on a set of Preference Data

- human preferences
    - costly
- LLM "Judge" preferences

The training causes the reward model to imitate the preferences inherent in the training data.

Reward model: training

- A prompt (context) is fed to the both a human (offline) and the model
- The model creates model responses (continuation)
- The Reward Model and the Human both rank the responses (calculate a reward)
- The Loss function penalizes the model for model rewards that deviate from the human reward



context = prompt; continuation = response

Source: https://arxiv.org/pdf/1909.08593.pdf#page=2

Discussion: Reward model

Typically

- the Reward Model is a scaled-down version of the Policy Model
- which may still have a large number of parameters

$\loss^{\ip}_{\text{reward}}(\phi, x, y^+, y^-)$ interpretation

There is also an interesting interpretation of the per-example Loss

$$\loss^{\ip}_{\text{reward}}(\phi, x, y^+, y^-) = -\log \sigma\big(\text{r}_\phi(x, y^+) - \text{r}_\phi(x, y^-)\big)$$

In Logistic Regression

- we compute a score z
- convert z to a probability $\sigma(z)$
    - probability of "example being Positive" via the sigmoid function
- use Binary Cross Entropy as the Loss
    - sum of terms for
        - Positive examples: $-\log(\sigma(z))$
        - Negative examples: $\log(1 - \sigma(z))$

In our case: the triples are all "Positive" examples so cross entropy collapses to $-\log(\sigma(z))$

So we can view

$$\loss^{\ip}_{\text{reward}}(\phi, x, y^+, y^-) = -\log \sigma\big(\mathrm{r}_\phi(x, y^+) - \mathrm{r}_\phi(x, y^-)\big)$$

as the Cross Entropy Loss of predicting the probability

$$\prc y^+ \text{ preferred to } y^- x$$

where the score z is

$$z = \mathrm{r}_\phi(x, y^+) - \mathrm{r}_\phi(x, y^-)$$

The PPO model

Once example preferences are translated into rewards

- example (x, y) is assigned reward
$$\mathrm{r}_\phi(x, y)$$
- we apply classic PPO

Discussion

PPO for Preferences involves one model

- Reward Model

in addition to the models inherent in PPO

- policy
- reference
- Value/Critic: for advantage computation

Model Type	Typical Size	Role
Policy Model	Large-scale pretrained LM (billions)	Generate responses
Reward Model	Smaller/fine-tuned LM or distilled (hundreds of millions)	Score outputs based on preferences

Group Relative Policy Optimization (GRPO)

GRPO is a policy-based RL method for dealing with Preference Data.

It thus fits nicely within (a slightly simplified) form of our Unified Policy Gradient Formulation

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

In GRPO, the Advantage of a response is defined

- **relative** to a **group** of responses to the **same** prompt
- rather than an *absolute* advantage

The Advantage of two responses in different groups

- are on a similar scale
- all prompts (and their group of responses) have advantages on same scale

Advantage definition

Given a prompt/input x

- a group $g = \mathrm{group}(\mathrm{x})$ of size G sample responses are collected

For each sample response $y_i \in g$

- there is a corresponding reward $\backslash\mathrm{rewseq}(g, y_i)$

The rewards within group g are normalized, giving the advantage for y_i as

$$A_{g,y_i} = \frac{\backslash\mathrm{rewseq}(g, y_i) - \mu_g}{\sigma_g}$$

where (μ_g, σ_g) are the mean and standard deviation of the rewards withing group g

Specializing Unified Policy Gradient Formulation to LLM Next Token prediction task

We need to simplify the formulation to deal with Preference data.

We do this specifically within the context of

- an LLM
- performing Language Modeling
    - Predict the Next Token conditional on the previously generated tokens of the response
- producing output y given input x

A response y is thus a sequence of tokens.

We turn the LLM's `generate` loop into an RL formulation as follows.

Identify iteration of the LLM's `generate` loop

- with a state \stateseq_\tt that has produced the first (-1) tokens of response y

  $$y_{[0:-1]}$$

- and performs action \actseq_\tt

  append next token y_\tt to the sequence $y_{[0:-1]}$

with probability $\pi_\theta(\actseq_t \mid \stateseq_t)$ being defined by the LLM next-token output probability distribution

We can write the probability $\pi_\theta(y|x)$ of the trajectory of response y given input x as

- the chained probabilities of each action given a state

The trajectory probability will suffice for re-writing the Unified Policy Gradient Formulation

- and can be expanded into the chained probability form, if desired
- but is not necessary

The simplified Policy Gradient Formulation is

$$\nabla_\theta J(\theta) = \mathbb{E}_x \left[\sum_{y \in \text{group(x)}} \nabla_\theta \log \pi_\theta(y|x) A_{\text{group(x),y}} \right]$$

where

- x is an input to the LLM
- $\mathrm{group(x)}$ is a set of LLM outputs, given x as input
    - since the LLM actions are probabilistic
- the advantage $A_{\mathrm{group(x),y}}$
    - of a particular response $y \in \mathrm{group(x)}$
    - is *relative* to other members of $\mathrm{group(x)}$

Discussion

The Surrogate Loss for GRPO and PPO

- are similar in form

But GRPO is a major simplification over PPO

- *eliminates* the need for a Value function
    - using trajectory-level rewards
    - compared to token-level rewards for PPO (derived from the Value function)

GRPO differs from PPO

- adds a KL-constraint to limit the policy update

Group-relative, normalized Advantage

- **Normalization**

The advantage of a response y given input x

- is relative to alternate responses to the same inputs x
    - in units of "number of standard deviations of the group"
- across groups:
    - there is a different standard deviation per group
    - but the response to two different inputs x, x' (and hence groups g, g') are in similar units

So a group g' with rewards that are much larger in scale than the rewards in group g

- has advantages that are on the same scale as g

The absence of an absolute reward means

- we can pool responses to different inputs
- without rewards from group g' over-shadowing rewards from group g

Moreover, normalization to mean 0 and unit standard deviation

- reduces variance of gradients
- smoother parameter update

- **Group-relative**

Relative (to the other responses in the group) advantage moves the focus to within-group preferences.

- **Groups**

Having multiple responses per prompt x

- provides multiple updates per prompt, rather than just a single response y

Pseudo code for GRPO

Detailed Surrogate Loss for GRPO

$$
\begin{aligned}
& J_{\mathrm{GRPO}}(\theta) \\
& = \mathbb{E}_{(q,a)\sim D,\{o_i\}_{i=1}^{G}\sim\pi_{\theta_{\mathrm{old}}}(\cdot|q)} \Bigg[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{=1}^{|o_i|} \\
& \min\left(r_{i,}(\theta)\hat{A}_{i,},\ \mathrm{clip}\left(r_{i,}(\theta),1-\epsilon,1+\epsilon\right)\hat{A}_{i,}\right) - \beta \mathrm{D}_{\mathrm{KL}}(\pi_\theta||\pi_{\mathrm{ref}}) \Bigg]
\end{aligned}
$$

Where:

- G is the number of sampled outputs per prompt
- $|o_i|$ is the token length of output o_i
- $r_{i,}(\theta) = \frac{\pi_\theta(o_{i,}|q,o_{i,<})}{\pi_{\theta_{\mathrm{old}}}(o_{i,}|q,o_{i,<})}$ is the per-token importance sampling ratio
- $\hat{A}_{i,}$ is the group-relative normalized advantage for token in sample i
- β is the KL penalty coefficient controlling divergence from

```
# GRPO training for LLM
for prompt in training_prompts:
    outputs = [llm.generate(prompt) for _ in range(group_size)]
    advantages = compute_group_relative_advantages(outputs, prompt) # e.g., usi
ng human rank or scoring function

    # Compute loss for all outputs (favor those with high relative advantage)
    losses = []
    for output, advantage in zip(outputs, advantages):
        logprob = llm.logprob(output, prompt)
        losses.append(-logprob * advantage)

    loss = sum(losses) / group_size
    loss.backward()
    optimizer.step()
```

Key Points:

```
Multiple candidates sampled per prompt; each gets an advantage score; updates i
ncrease likelihood of better completions in the group.
```

DAPO: An Improved GRPO; Case Study of Real-World

The GRPO model was introduced by DeepSeek, which published the algorithm pseudo-code.

However, as is often the case, the "baseline" model that is published is *not sufficient* to replicate the performance results quoted in the paper.

There are substantial "engineering details" that are not disclosed.

- Published DeepSeek results on GRPO
    - perform and incremental 50% higher than baseline GRPO on some benchmarks
- Details matter !

Responding to this, ByteDance introduced a new model:

- *Decoupled Clip and Dynamic sAmpling Policy Optimization (DAPO)*

in the paper: DAPO: An Open-Source LLM Reinforcement Learning System at Scale (https://arxiv.org/pdf/2503.14476).

DAPO is

- fully Open Source
- that provides refinements to the baseline GRPO
- sufficient to achieve similar results to the published DeepSeek paper

This was the result of

- experimentation
- error analysis

to diagnose the failures of baseline GRPO and devise remedies.

DAPO introduces 4 key modifications on baseline GRPO

- Changing the threshold for clipping
- *Eliminating* non-informative samples
- Careful redefinition of Token-level Gradient Loss
- Penalizing overly long responses

DAPO uses a Surrogate Loss that differs from that of GRPO

- in *subtle* ways

To highlight the differences in Surrogate Loss

- we incrementally modify $J_{\mathrm{GRPO}}(\theta)$
- to obtain $J_{\mathrm{DAPO}}(\theta)$

We defer the justification for the modifications until after the final $J_{\mathrm{DAPO}}(\theta)$ has been derived.

Evolution of $J_{\text{DAPO}}(\theta)$

Let's start with the Surrogate Loss for GRPO:

$$
\begin{aligned}
& J_{\text{GRPO}}(\theta) \\
& = \mathbb{E}_{(q,a)\sim D,\{o_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(\cdot|q)} \left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{=1}^{|o_i|} \right. \\
& \left. \min\left(r_{i,}(\theta)\hat{A}_{i,},\ \text{clip}\left(r_{i,}(\theta),1-\epsilon,1+\epsilon\right)\hat{A}_{i,}\right) - \beta \mathrm{D}_{\text{KL}}(\pi_\theta||\pi_{\text{ref}}) \right]
\end{aligned}
$$

where

- $\mathcal{D}$ is the training dataset
    - consisting of examples consisting of question/answer pairs (q, a)

Recall that GRPO

- creates a group of size G answers to q
- by sampling from the LLM with non-zero temperature

Remove KL constraint

First, observe that, relative to GRPO

- DAPO removes the KL-divergence constraint

$$\mathrm{D}_{\mathrm{KL}}\left(\pi_\theta || \pi_{\mathrm{ref}}\right)$$

that limits how far

- new policy π_θ
- can diverge from the *reference* policy π_{ref}.

$$J_{\mathrm{DAPO}^1}(\theta)$$

$$= \mathbb{E}_{(q,a)\sim D,\{o_i\}_{i=1}^{G}\sim\pi_{\theta_{\mathrm{old}}}(\cdot|q)}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{=1}^{|o_i|}\min\left(r_{i,}(\theta)\hat{A}_{i,},\ \mathrm{clip}\left(r_{i,}(\theta),1-\epsilon,1+\epsilon\right.\right.\right.$$

giving us the intermediate $J_{\mathrm{DAPO}^1}(\theta)$

Changing the token-level average

In GRPO

- First averages token losses within each sample by its length $|o_i|$,
- Then averages sample losses uniformly over group G.

DAPO

- Sums all token losses in the group before normalizing by total tokens in all samples.

giving us the intermediate

$$J_{\text{DAPO}^2}(\theta)$$

$$= \mathbb{E}_{(q,a)\sim\mathcal{D},\{o_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(\cdot|q)} \left[\frac{1}{\sum_{i=1}^{G}|o_i|} \sum_{i=1}^{G} \sum_{=1}^{|o_i|} \min \left(r_{i,}(\theta)\hat{A}_{i,},\ \text{clip}\left(r_{i,}(\theta), 1-\epsilon, 1 - \right.\right.\right.$$

This is referred to as the *Token-level Loss* technique in the paper.

Asymmetric Clipping

GRPO's clipping is symmetric with range

$$[1-\epsilon, 1+\epsilon]$$

DAPO adds different clipping values on either side

$$[1-\epsilon_{\text{low}}, 1+\epsilon_{\text{high}}]$$

giving us the intermediate

$$J_{\text{DAPO}^3}(\theta)$$

$$= \mathbb{E}_{(q,a)\sim\mathcal{D},\{o_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(\cdot|q)} \left[\frac{1}{\sum_{i=1}^{G}|o_i|} \sum_{i=1}^{G}\sum_{=1}^{|o_i|} \right.$$

$$\left. \min\left(r_{i,}(\theta)\hat{A}_{i,},\ \text{clip}\left(r_{i,}(\theta), 1-\epsilon_{\text{low}}, 1+\epsilon_{\text{high}}\right)\hat{A}_{i,}\right) \right]$$

This is referred to in the paper as the *Clip Higher* technique.

Dynamic sampling

GRPO's expectation is over

- **all** trajectories in the training dataset

DAPO limits the Group composition

- to ensure that the intra-group trajectory rewards **are not identical**

giving us the final

$$
\begin{aligned}
& J_{\mathrm{DAPO}}(\theta) \\
& = \mathbb{E}_{(q,a)\sim\mathcal{D},\{o_i\}_{i=1}^{G}\sim\pi_{\theta_{\mathrm{old}}}(\cdot|q)} \left[\frac{1}{\sum_{i=1}^{G}|o_i|} \sum_{i=1}^{G} \sum_{=1}^{|o_i|} \right. \\
& \left. \min\left(r_{i,}(\theta)\hat{A}_{i,},\ \mathrm{clip}\left(r_{i,}(\theta), 1-\epsilon_{\mathrm{low}}, 1+\epsilon_{\mathrm{high}}\right)\hat{A}_{i,} \right) \right],
\end{aligned}
$$

subject to:

$$
0 < \left| \{ o_i \mid \mathrm{is\_equivalent}(a, o_i) \} \right| < G
$$

where a is the (single) response in the training dataset $\mathcal{D}$

GRPO vs DAPO Comparison

Feature	GRPO	DAPO				
Loss Aggregation	$ \frac{1}{G}\sum_i \frac{1}{	o_i	} \sum_t $	$ \frac{1} {\sum_i	o_i	} \sum_i \sum_t $
Effect on Long Responses	Weighed less per token (gradient dilution)	Maintains per-token gradient magnitude				
Clipping	Symmetric $(1-\epsilon, 1+\epsilon)$	Asymmetric $(1-\epsilon_{\text{low}}, 1+\epsilon_{\text{high}})$				
Impact	Slower learning on longer tokens	Finer control, better learning on complex outputs				

Aspect	GRPO	DAPO
Value Model Dependency	Removes PPO value model, uses relative group rewards	Same, but more robust and stable optimization
Clipping Mechanism	Basic clip bounds on importance ratios	Asymmetric Clip-Higher for rare, valuable tokens
Sampling	Multi-response batch, potentially redundant	Dynamic Sampling ensures diverse, effective samples
Gradient Weighting	Token-level, gradient diluted in long outputs	Token-Level Gradient Loss corrects signal dilution
Reward for Length	None or truncation-based	Overlong Reward Shaping with soft penalties
Efficiency/Stability	Improved over PPO, but unstable for some scenarios	Superior stability and efficiency; state-of-the-art results

Feature	PPO	GRPO	DAPO
Advantage Source	Value model estimate	Group-relative, reward normalized in batch	Same as GRPO
Clipping	Symmetric $(1-\epsilon, 1+\epsilon)$	Symmetric $(1-\epsilon, 1+\epsilon)$	Asymmetric $(1-\epsilon_{\text{low}}, 1+\epsilon_{\text{high}})$ (Clip-Higher)
Loss Aggregation	Per-sample/token	Per-sample, then averaged over tokens	Per-token, avoids dilution in long outputs
Sampling	Independent samples	Groups of samples per prompt	Dynamic: only informative samples (not all-correct/incorrect)
Reward Shaping	None by default	None by default	Overlong shaping (discourages excessive length)

KL-constraint elimination: why ?

For many RL objectives (e.g., induce long CoT reasoning in the response)

- the *new distribution*
- is necessarily *far* from the reference distribution

So the KL constraint is too limiting for some tasks.

Speculation

DeepSeek initially tried to induce reasoning

- using **only** RL
- **no** SFT step before the RL

As we have seen

- SFT "primes the pump" to make RL more likely to succeed
- by ensuring the initial RL model is able to produce
    - correctly formatted responses

Perhaps the KL-constraint is a vestige of the RL-only attempt

- since the long CoT desired model was far different from the base model

Token-level averaging change: why ?

The GRPO loss aggregation

$$\frac{1}{G}\sum_{i}\frac{1}{|o_i|}\sum_{\dots}$$

creates a loss for sample i in group g

- that is the *average over the tokens* of response o_i

before averaging the sample loss over the G elements of the group.

This introduces a *bias*

- the gradient update of an "important" token in a **long** response o_i
  - important: high gradient
- has less weight in the Gradient of $J_{\text{DAPO}}(\theta)$
- than a token of *equal importance* in a **shorter** response $o_{i'}$

even when the advantages $\hat{A}_i$ and $\hat{A}_{i'}$ are equal.

Reward hacking

This can lead to the undesirable phenomenon known as *Reward Hacking*.

RL learns to produce *short **correct** responses*

- with high trajectory (and thus, high per-token) reward
- to amplify the contribution of important tokens

and *long **incorrect** responses*

- with low trajectory (and thus, low per-token) reward
- to dilute the contribution of important tokens
    - adding gibberish or repetitive content in order to increase length

The result is that

- the impact of important tokens
- on updating the policy
- is distorted

DAPO eliminates this bias by changing the aggregation

$$\frac{1}{\sum_i |o_i|} \sum_i \sum_{\ldots}$$

so that

- *all tokens in all responses in a group*
- have the same contribution to the Gradient of $J_{\mathrm{DAPO}}(\theta)$

Reward shaping: Overlong reward shaping length penalty

DAPO also adds a *length penalty* to directly discourage excessively long responses

$$\tilde{R}_i = R_i - \alpha \cdot \mathrm{LengthPenalty}(o_i)$$

where $\mathrm{LengthPenalty}(o_i)$ measures the undesirable properties of response o_i.

The penalty used in the paper is called *Soft Overlong Punishment*

- two length thresholds
- length penalty is phased in between the thresholds

In addition

- long responses are truncated to a maximum length
- the truncated elements of the response are masked in the loss calculation

This is referred to as *Overlong Filtering* in the paper.

Asymmetric Clipping: why ?

Recall the *probability ratio*

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

expresses how much

- the *new probability* $\pi_\theta(a_t|s_t)$ of an action (given a state)
- can differ from
- the *old probability* $\pi_{\theta_{\text{old}}}(a_t|s_t)$
- in *proportional terms*

This ratio is *clipped* in GRPO and DAPO.

In the GRPO loss

- the clipping range is
$$[1-\epsilon, 1+\epsilon]$$

Typically:

$$\epsilon = .20$$

In DAPO

- the clipping range is adjusted to
$$1 - \epsilon_{\text{low}}, 1 + \epsilon_{\text{high}}$$

Typically

- $\epsilon_{\text{low}} = \epsilon = .2$
- $\epsilon_{\text{high}} = .28$

Thus, the probability in DAPO is allowed to increase

- more (proportionally) than it is allowed to decrease

The reason for doing so is a simple consequence of

- translating proportional increase to absolute increase

Given two actions/tokens at step

$$a_{t,\text{low}}, a_{t,\text{high}}$$

where

$$\pi_{\theta_{\text{old}}}(a_{t,\text{low}}|s_t) < \pi_{\theta_{\text{old}}}(a_{t,\text{high}}|s_t)$$

the absolute increase in probability for $a_{t,\text{low}}$ is less than for $a_{t,\text{high}}$

- since the initial probability $\pi_{\theta_{\text{old}}}(a_{t,\text{low}}|s_t)$ is lower
- even if $\epsilon_{\text{high}} = \epsilon$

When both actions/tokens are potentially high importance *exploration* (rather than exploitation) actions

- the potential *reward* benefit for $a_{t,\text{low}}$
- is lower than for $a_{t,\text{high}}$

The RL may learn to favor the (potentially less important in terms of global goal) $a_{t,\text{high}}$ over $a_{t,\text{low}}$.

This has the negative effect of

- diminishing the reward benefit of some exploration actions
- that *could* be high value in moving the policy in the optimal direction

thus restricting profitable exploration.

Raising the upper clipping bound has the effect of increasing the entropy of the policy

- more exploration vs exploitation

Dynamic sampling: why ?

If all samples in a group have the same trajectory reward

- the Advantage of each sample in the group
- is mathematically equal to 0

Groups with samples having identical reward are typically

- all samples are correct
- all samples are incorrect

Because the advatange of each sample is 0, these groups

- do not contribute to the Gradient of $J_{\text{DAPO}}(\theta)$

When computing the Gradient of a batch of N questions

- each with G sample responses

the groups with 0 advantage reduce the effective batch size.

This results in such batches having *noisier* gradient updates than unaffected batches.

Moreover the learning signal

- is stronger
- when there is a *contrast* between samples
    - high reward vs. low reward
- whether within a group or within a batch

Dynamic sampling promotes contrastive groups.

On the topic of *contrastive examples*

- without the initial SFT of RFT
- when the behavior to learn via RL is very different than the behavior of the base LLM
- all examples and samples are likely to have the same low reward
    - because of incorrect formatting or logic

So the initial SFT will hopefully create some high reward examples

- by "moving the distribution" of RL training examples
- closer to the ultimate RL goal
- than the distribution of examples from the base (non-SFT tuned) model

Contribution of each technique to the improvement of DAPO vs GRPO

Technique	Description	Commentary	Accuracy Improvement (AIME 2024 avg@32)
Naive GRPO	Baseline group relative policy optimization without enhancements	Starting point with relatively low accuracy	30
Overlong Filtering	Filters out truncated (overlong) samples from training loss	Reduces reward noise caused by forced truncation, stabilizes training	36
Clip-Higher	Decouples lower and upper clipping range to allow higher increase for low-probability tokens	Enhances policy entropy and exploration, avoids early collapse of exploration	38
Soft Overlong Punishment	Length-aware penalty on excessively long responses	Prevents reward noise from overly penalizing valid but long reasoning chains	41
Token-Level Loss	Aggregates loss over tokens normalized by total token count (not per sample average)	Improves training stability and healthier growth in output length	42
Dynamic Sampling	Oversamples and filters batches to keep effective gradient signals by excluding zero-advantage samples	Significantly improves training stability and performance speed	50

Pseudo code for DAPO

Detailed Surrogate Loss for DAPO

$$J_{\mathrm{DAPO}}(\theta)$$

$$= \mathbb{E}_{(q,a),\{o_i\}\sim\pi_{\theta_{\mathrm{old}}}} \left[\frac{1}{\sum_{i=1}^{G}|o_i|} \sum_{i=1}^{G}\sum_{=1}^{|o_i|} \min \left(r_{i,}(\theta)\hat{A}_{i,},\ \mathrm{clip}(r_{i,}(\theta), 1-\epsilon_{\mathrm{low}}, 1+\epsilon_{\mathrm{high}} \right.\right.$$

```
# Given:
# batch_size = N
# num_samples = K
# sequence_lengths = [T_gi for each response i in group g]
# importance_scores = array of same shape as tokens, default = 1

for group_id in range(N):
    for sample_id in range(K):
        T = sequence_lengths[group_id][sample_id]
        importance_sum = 0
        # Compute sum of importance scores in sample
        for t in range(T):
            importance_sum += importance_scores[group_id][sample_id][t]
        # Calculate alpha for each token
        for t in range(T):
            alpha = importance_scores[group_id][sample_id][t] / importance_sum
            # Compute token-wise policy gradient component:
            grad = alpha * w[group_id][sample_id] \
                   * clip(r[group_id][sample_id][t], 1-eps_low, 1+eps_high) \
                   * advantage[group_id][sample_id][t]
```

References for GRPO to DAPO

- DAPO: An Open-Source LLM Reinforcement Learning System at Scale (arXiv) (https://arxiv.org/abs/2503.14476)
- Mathematics of DAPO, PPO, and GRPO (SSRN) (https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5205449)
- From GRPO to DAPO and GSPO: What, Why, and How (Hugging Face blog) (https://huggingface.co/blog/NormalUhr/grpo-to-dapo-and-gspo)
- The Evolution of GRPO: DAPO (Towards AI) (https://towardsai.net/p/l/the-evolution-of-grpo-dapo)

Direct Preference Optimization (DPO)

We now present a **Supervised Training** method for Preference data

- **not** Reinforcement Learning
- similar to Binary Classification
    - compute probability that y^+ is preferred
- no reward model

Direct Preference Optimization (DPO) is a policy optimization method where supervision

- comes from *Preference Data*
- rather than rewards

$$(x, y^+, y^-)$$

The policy is trained to assign a higher likelihood to the preferred outcome y^+ than the non-preferred outcome y^-.

Relation to the Unified Gradient Formulation

DPO also uses a surrogate loss to guide the derivation of the policy.

$$L_{\mathrm{DPO}}(\theta) = -\log \sigma \left(\log \frac{\pi_\theta(y^+|x)}{\pi_\theta(y^-|x)} \right) = -\log \sigma \left(\Delta \right)$$

where

- $\pi_\theta(y^+|x)$
- $\pi_\theta(y^-|x)$

are the probabilities of the *trajectories* resulting in outputs y^+ and y^-.

- The ratio
$$\frac{\pi_\theta(y^+|x)}{\pi_\theta(y^-|x)}$$

is the relative probability of the preferred output, compared to the non-preferred output._

- We take the log of the ratio
    - resulting in log-probabilities, as in the Universal Formulation
- The sigmoid σ converts the relative probability to the range $[0, 1]$.

We can compute the gradient of $L_{\mathrm{DPO}}(\theta)$

We first simplify $L_{\mathrm{DPO}}(\theta)$ by defining

$$\Delta = \log \frac{\pi_\theta(y^+|x)}{\pi_\theta(y^-|x)} = \log \pi_\theta(y^+|x) - \log \pi_\theta(y^-|x)$$

Substituting into $L_{\mathrm{DPO}}(\theta)$

$$L_{\mathrm{DPO}}(\theta) = -\log \sigma\left(\Delta\right)$$

The gradient of this simplified $L_{\mathrm{DPO}}(\theta)$ is $$ \begin{array}{l} \nabla_\theta L_{\mathrm{DPO}}(\theta) & = & -(1 - \sigma(\Delta)) \cdot \nabla_\theta \Delta \\ & = & -(1 - \sigma(\Delta)) \cdot \big(\nabla_\theta \log \pi_\theta(y^+|x) \nabla_\theta \log \pi_\theta(y^-|x) \big) \end{array} $$

This follows from basic rules of calculus

The term

$$(1 - \sigma(\Delta))$$

is interpreted as the *Advantage* of y^+ over y^-.

This advantage is small

- when $\sigma(\Delta) \approx 1$
    - i.e., the model is confident: assigning high probability to the preferred output

Conversely, it is large when the model is uncertain.

So the advantage term adjusts the Gradient update step size depending on how far the probability of the preferred output is from 100%.

The gradient can be interpreted as adjusting the policy

- so as to increase the (log) likelihood
- adjusted by the advantage

Discussion

Supervised Training vs Reinforcement Learning

You will notice that, in DPO

- the classic elements of RL are missing
    - supervision via Loss function, not rewards
    - direct preferences rather than implicit/explicit rewards
        - reward sparsity not an issue; just trajectory preference
    - no reference to states/actions/rewards

Fine-tuning a model with DPO is Supervised Fine Tuning rather than Reward Tuning

$$L_{\mathrm{DPO}}(\theta) = -\log \sigma\left(\log \frac{\pi_\theta(y^+|x)}{\pi_\theta(y^-|x)}\right) = -\log \sigma\left(\Delta\right)$$ **interpretation**

We can interpret $L_{\mathrm{DPO}}(\theta)$

- in an identical manner
- to how we interpreted $\loss^{\ip}_{\mathrm{reward}}(\phi, x, y^+, y^-)$

for the Reward model of PPO.

That is:

$L_{\mathrm{DPO}}(\theta)$ is equivalent to the Binary Cross Entropy loss for the problem predicting the probability

$$\mathrm{prc} y^{+} \text{ preferred to } y^{-} x$$

where the score z is

$$z = \log \frac{\pi_\theta(y^+|x)}{\pi_\theta(y^-|x)} = \log \pi_\theta(y^+|x) - \log \pi_\theta(y^-|x)$$

From the perspective of $L_{\mathrm{DPO}}(\theta)$

- we are performing Binary Classification

DPO vs PPO

PPO for Preferences involves one sub-model

- Reward Model

in addition to the sub-models inherent in PPO

- policy
- reference
- Value/Critic: for advantage computation

All of these models are roughly the same size

- so PPO is more expensive than DPO
    - training data for each PPO sub-model
    - similar number of parameters for each sub-model

DPO, by contrast, is a single model

- more accessible in terms of reduced compute/memory footprint

Approach	Data Used	Training Steps	Model Copies Required	Key Challenge
PPO + Reward Model	Preference → Scalar rewards	Train reward model, then PPO optimization	Policy, reference, value, reward	Reward modeling complexity, instability
DPO	Preference pairs directly	Single-stage policy gradient optimization	Only policy	Requires careful pairing, but simpler overall

Pseudo code for DPO

```
# DPO training for LLM
for prompt in training_prompts:
    outputs = [llm.generate(prompt) for _ in range(2)]

    # outputs: [output_0, output_1]
    # preference: 0 if output_0 is preferred, 1 if output_1 is preferred
    preferred_idx = compare_outputs(outputs) # Human or synthetic comparison

    logit_pref = llm.score(outputs[preferred_idx], prompt)
    logit_nonpref = llm.score(outputs[1 - preferred_idx], prompt)

    # DPO loss: maximize difference so preferred > non-preferred
    loss = -logsigmoid(logit_pref - logit_nonpref)

    loss.backward()
    optimizer.step()
```

Comparison of methods for Preference Data

Aspect	PPO (Proximal Policy Optimization)	DPO (Direct Preference Optimization)	GRPO (Group Relative Policy Optimization)
Stability	Moderate stability, uses clipped objective to limit policy updates and prevent divergence. Can still be sensitive to reward noise and hyperparameters.	High stability due to supervised-learning style objective on preference pairs. Does not rely on policy gradient RL steps.	Higher stability than PPO due to normalized group rewards reducing gradient noise; does not require a value function critic which reduces instability.
Variance	High variance in gradient estimates caused by sparse rewards and stochastic policy sampling. Requires variance reduction techniques (e.g., baseline/critic).	Low variance because gradients come from direct supervised preference comparisons without sampling or policy gradients.	Moderate variance—variance is reduced by reward normalization within groups but still involves sampling multiple outputs, so more variance than DPO but less than PPO.
Sample Efficiency	Moderate to low—needs many environment interactions/samples due to sparse reward signal and on-policy updates. Sampling multiple sequences per prompt increases cost.	Very high—trains directly on labeled preference pairs with no complex sampling or reward modeling.	Higher than PPO—requires multiple samples per prompt for group comparison but gains efficiency from relative advantage normalization and critic-free updates.

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