Supervised Learning Preference Optimization: Rethinking RLHF and DPO as Supervised Learning

Yaoshiang Ho YAOSHIANG@GMAIL.COM

Abstract

We propose *Supervised Learning Preference Optimization* (SLPO), a simpler approach to aligning language models with human preferences without resorting to reinforcement learning concepts. SLPO reframes alignment as a standard supervised learning problem, shifting probabilities to preferred responses while preserving the reference distribution. Our analysis and implementation demonstrate how SLPO parallels and simplifies Direct Preference Optimization (DPO), opening avenues for more transparent and efficient alignment training.

Keywords: Reinforcement Learning from Human Feedback (RLHF), Direct Preference Optimization (DPO)

1 Introduction

Alignment is the task of ensuring that the behavior of a Large Language Model (LLM) is consistent with human preferences.

A key difference between the alignment phase and other phases of training an LLM is that the alignment phase considers full sequences of text, rather than simply predicting the next token, as in the pretraining and supervised fine-tuning (SFT) phases.

The alignment approach popularized by the commercial success of ChatGPT was Reinforcement Learning from Human Feedback (RLHF, Ouyang et al. (2022)). Despite its effectiveness, RLHF requires training a second model, called a reward model, as well as Proximal Policy Optimization (PPO), resulting in a technique that is more complex than the basic supervised learning. It also requires a Kullback-Leibler (KL) divergence term to regularize the changes to the LLM during alignment training.

Direct Preference Optimization (DPO) is a simpler approach to alignment which does not require a secondary reward model. In the paper introducing DPO, the authors examine the underlying approach of RLHF and propose the DPO objective to align the target LLM directly using maximum likelihood estimation (MLE). The key insight from the DPO paper is that an LLM's outputs can be reparameterized into a reward model using ratios, logs, and the Bradley-Terry model (Bradley and Terry (1952)).

The specific contribution of this paper is to reframe the alignment phase away from reward modeling entirely and treating it simply as a pure supervised learning problem by training a model to align to a directly modified probability distribution. We call this approach Supervised Learning Preference Optimization (SLPO).

2 Related Work

Related work...

3 Preliminaries: DPO

We review and continue the analysis of the DPO objective by its authors.

The DPO objective is defined as follows:

$$L_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = \underbrace{-\mathbb{E}_{(x,y_w,y_l)\sim D} \log}_{1} \left[\underbrace{\sigma}_{2} \left(\underbrace{\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)}}_{3} - \underbrace{\beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)}}_{4} \right) \right]. \quad (1)$$

where

π_θ is the language model to be aligned,

π_{ref} is the reference language model,

D is the dataset of training examples,

x is the input context,

y_w is the winning sequence,

y_l is the losing sequence,

σ is the sigmoid function,

β is a hyperparameter.

In the underbraced section 1, we see the standard negative log likelihood (NLL) objective. In the underbraced section 2, we see the Bradley-Terry model [1]. In the underbraced sections 3 and 4, we see how the reference and language model's predictions are reparameterized into a winning and losing score: they are the log of the ratio of the language model to the reference model, for the winning and losing completion, respectively. This score is later described as the reward function.

With simple algebraic manipulation, we can rewrite this objective as:

$$L_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = \underbrace{-\mathbb{E}_{(x,y_w,y_l)\sim D} \log}_{1} \left[\underbrace{\sigma}_{2} \left(\underbrace{\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_\theta(y_l \mid x)}}_{3} - \underbrace{\beta \log \frac{\pi_{\text{ref}}(y_w \mid x)}{\pi_{\text{ref}}(y_l \mid x)}}_{4} \right) \right].$$

We can interpret the undercomponents as follows: The first and second underbraces remain unchanged. The third underbrace is the log of the ratio language models probability of the winner divided by the loser. This ratio is optimized to be bigger, given that the log,

1. A ranking method that is mathematically equivalent and perhaps more widely understood is the ELO score, used to rank Chess players, and, LLMs in the Chatbot Arena (Elo (1978); Chiang et al. (2024)). Both ELO and Bradley-Terry assign scores to players, and pass the difference through a sigmoid function to assess the probability of the minued (aka LHS) player of winning and subrahend (aka RHS) player of losing.

sigmoid, and log from underbraces 1, 2, and 3 are all monotonic functions. Since a ratio of probabilities is an odds ratio, we will refer to this as the *language model's log-odds ratio.*

The fourth underbrace is reference model's log-odds ratio. This is a constant per y_w and y_l and is not differentiated. However, these values and their ratio vary across different y_w and y_l - that is, each row of training data will have a different value for this constant. More specifically, since π_θ is initialized as π_{ref}, the difference between underbraces 3 and 4 starts at zero during training, and the shape of the sigmoid function (underbrace 2) and its gradient are known. During training, as the language model's log-odds ratio increases, the sigmoid function will naturally regularize and decelerate the increase by reducing the magnitude of the gradient (a useful version of the vanishing gradient problem). This is the exact outcome described by the DPO authors in their analysis of the gradient of DPO. *But we have developed a different intuition the DPO loss: rather than reparameterizing the language model's output into a reward model, the DPO objective optimizes the language model's log-odds ratio, with a constant shift so that the shifted value starts at zero during optimization.*

Let's continue reworking the DPO objective into mathematically equivalent forms to better understand it by looking at tokens and their logits rather than sequences [2].

The variable y is a sequence of tokens, defined as

$$\pi_{\text{ref}}(y \mid x) = \prod_{t=1}^{T} \pi_{\text{ref}}(y_t \mid x, y_{<t}), \tag{2}$$

where

$y = (y_1, y_2, \ldots, y_T)$ is the output sequence,

x is the input context, and

$y_{<t} = (y_1, y_2, \ldots, y_{t-1})$ represents the tokens generated prior to time step t.

$\pi_{\text{ref}}(y_t \mid x, y_{<t})$ denotes the conditional probability of generating token y_t given the input x and the previously generated tokens $y_{<t}$.

Since both π language model are LLMs, they are activated using the softmax function. The softmax operation can be mathematically cumbersome due to its dependence on all other logits for normalization. For simplicity, we assume that the logits are transformed using a log-softmax function, which computes the logarithm of the softmax probabilities in a numerically stable way. This transformation results in log-probs, which are easier to reason about since they can be exponentiated to recover probabilities. Importantly, we do

2. Technically, a logit is the log-odds ratio and it is the output of a feature extractor before a sigmoid activation function in binary classification. The term logit is also used for categorical classification, when a feature extractor's outputs are activated with the softmax function. However, these softmax logits cannot be exponentiated to calculate an odds ratio. They do have an unfortunate property however: they are shift invariant, leading to numerical instability and the need for log-softmax and its use of the log-sum-exp trick to improve numerical stability. In this paper, since we are dealing with both sigmoid and softmax functions, we will refer to the value passed into a softmax function as a softmax logit.

not lose any generality because log-probs can also be passed through a softmax function to recover probabilities, ensuring equivalent behavior.

Let us establish the term g for the layers of the model up to the softmax activation, namely, the feature extractor and log-softmax normalization:

$$\mathrm{g}(y \mid x) = \mathrm{logsoftmax}(\mathrm{f}(y \mid x)) \tag{3}$$

such that $\pi(y \mid x) = \exp(\mathrm{g}(y \mid x))$. Plugging Equations 2 and 3 into the DPO objective 1,

$$L_{\mathrm{DPO}}(\pi_\theta; \pi_{\mathrm{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim D} \log \left[\sigma\left(\beta \log \frac{\prod_{t=1}^{T_w} \exp(\mathrm{g}_\theta(y_{w,t} \mid x, y_w{<}\, t))}{\prod_{t=1}^{T_w} \exp(\mathrm{g}_{\mathrm{ref}}(y_{w,t} \mid x, y_w{<}\, t))} - \beta \log \frac{\prod_{t=1}^{T_l} \exp(\mathrm{g}_\theta(y_{l,t} \mid x, y_w{<}\, t))}{\prod_{t=1}^{T_l} \exp(\mathrm{g}_{\mathrm{ref}}(y_{l,t} \mid x, y_w{<}\, t))}\right)\right]$$

which simplifies to:

$$L_{\mathrm{DPO}}(\pi_\theta; \pi_{\mathrm{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim D} \log \left[\sigma\left(\beta \left(\sum_{t=1}^{T_w} [\mathrm{g}_\theta(y_{w,t} \mid x, y_{w,<t}) - \mathrm{g}_{\mathrm{ref}}(y_{w,t} \mid x, y_{w,<t})] - \sum_{t=1}^{T_l} [\mathrm{g}_\theta(y_{l,t} \mid x, y_{l,<t}) - \mathrm{g}_{\mathrm{ref}}(y_{l,t} \mid x, y_{l,<t})]\right)\right)\right] \tag{4}$$

We notice another possibility to rewrite the DPO objective towards something more familiar. Optimizing the sigmoid of a difference $a-b$ through BCE is equivalent to optimizing the softmax of a and b through CCE towards a one-hot encoded target of $y_{\mathrm{true}} = [1, 0]$. This is well known but derived in Appendix A. This allows us to rewrite the DPO objective as follows:

$$L_{\mathrm{DPO}}(\pi_\theta; \pi_{\mathrm{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim D}\, \mathrm{CCE}\left[y_{\mathrm{true}} = \begin{bmatrix}1\\0\end{bmatrix}\right], y_{\mathrm{pred}} = \mathrm{softmax}\left(\begin{bmatrix}\beta S_w\\ \beta S_l\end{bmatrix}\right) \tag{5}$$

where

$$S_w = \underbrace{\sum_{t=1}^{T_w} \mathrm{g}_\theta(y_{w,t} \mid x, y_{w,<t})}_{A} \underbrace{- \sum_{t=1}^{T_w} \mathrm{g}_{\mathrm{ref}}(y_{w,t} \mid x, y_{w,<t})}_{B}$$

$$S_l = \underbrace{\sum_{t=1}^{T_l} \mathrm{g}_\theta(y_{l,t} \mid x, y_{l,<t})}_{A} \underbrace{- \sum_{t=1}^{T_l} \mathrm{g}_{\mathrm{ref}}(y_{l,t} \mid x, y_{l,<t})}_{B}$$

Key Observation #1: Much of the DPO objective can be simplified to the form of a standard classification task. This formulation of the DPO objective provides our first key observation. What started off as a complex reparameterization of the language model's output into a reward model and a Bradley-Terry model, has been simplified into a standard categorical crossentropy loss applied to the softmax, in Equation 5. However, the

input to the softmax is not a single logit, but rather, the summation of all the winning (or losing) logits in the language model (underbraces A), subtracted by the same summation from the reference model (underbraces B). Our next two observations will interpret those. Our reformulation is a strength, as it allows to apply more familiar tools to understand the DPO objective, removing reinforcement learning and Bradley-Terry concepts.

Key Observation #2: The difference terms simply zero-align the input to the softmax function, across all examples during training, allowing the training to equally optimize all examples. Since the language model is initialized by the reference model, the difference introduced in underbraces B of Equation 5 simply zero-aligns the input to softmax, allowing the training process to send identical gradients to all examples, regardless of the relative probabilities $\pi(y_w)$ and $\pi(y_l)$ for that example. This difference is the vestige underbraces 3 and 4 in the original DPO objective in Equation 1 was refered to as a "reward function", but we now consider it simply a zero-aligning of examples. We have successfully eliminated reinforcement learning concepts from the DPO objective. This is a strength of the DPO objective.

Key Observation #3: DPO optimizes a joint probability The summation in unbraces A of Equation 5 may not be familiar. In fact, the softmax of a sum optimizes a joint probability. See Appendix B for a derivation. This should not be surprising since the original DPO objective optimizes the joint probability of the winning and losing sequences. Equation 2 expresses this as the joint probability of tokens, represented as a product. This, in turn, corresponds to the product of probabilities, which can be rewritten as the sum of log probabilities. This is a strength of the DPO objective.

Key Observation #4: Behind the summations and shifts in unbraces A and B of Equation 5, the final objective is still $+inf$ or $-inf$ for each token logit. This is obvious from Equation 5. An equivalent statement is that the DPO objective does not consider any sequence other than the winning and losing, and if left unchecked, would ultimately force the model to predict the winning sequence with probability 1 and all other sequences, including the loser, with probability 0. This is obviously destabilizing, e.g. other sequences may have been perfectly reasonable as winning sequences. This is a weakness of the DPO objective.

We have been able to remove much of the machinery of the DPO objective through algebraic manipulation and equivalences to more common deep learning concepts. We have also developed key observations on the strength and weaknesses of the DPO objective. Can we take these learnings and develop a superior objective?

4 Preliminaries: Probabilistic Classification

A basic task for deep learning has been classification, both binary and categorical.

- Look at this image and decide which of the ten digits it is(LeCun (1998)).
- Classify every pixel in this image as a foreground class or background.
- Decide if this movie review's sentiment is positive or negative(Pang and Lee (2004)).

Modern LLMs auto-regressively predict a probability distribution for the next token among a finite set of possibilities, conditioned on the previous tokens. Hence, an LLM also solves a classification problem.

Given the prevalence of classification, deep learning oftens assumes target labels can be described as either ordinal numbers or equivalently one-hot encoded vectors. When target labels are intermediate values between one and zero, they were called "soft targets", treated as an regularization technique or even called dark knowledge (Hinton et al. (2015); Szegedy et al. (2016); Hinton et al. (2014)).

But the two roots of the crossentropy loss functions, KL divergence and Maximum Likelihood Estimation (MLE) applied to the probability distribution function of a Multi-Bournoulli distribution, are not limited to binary and categorical outcomes. For example, the crossentropy loss function can be used directly to condition a model to predict to be true 60% of the time and false 40% of the time, as demonstrated in Figure 4.

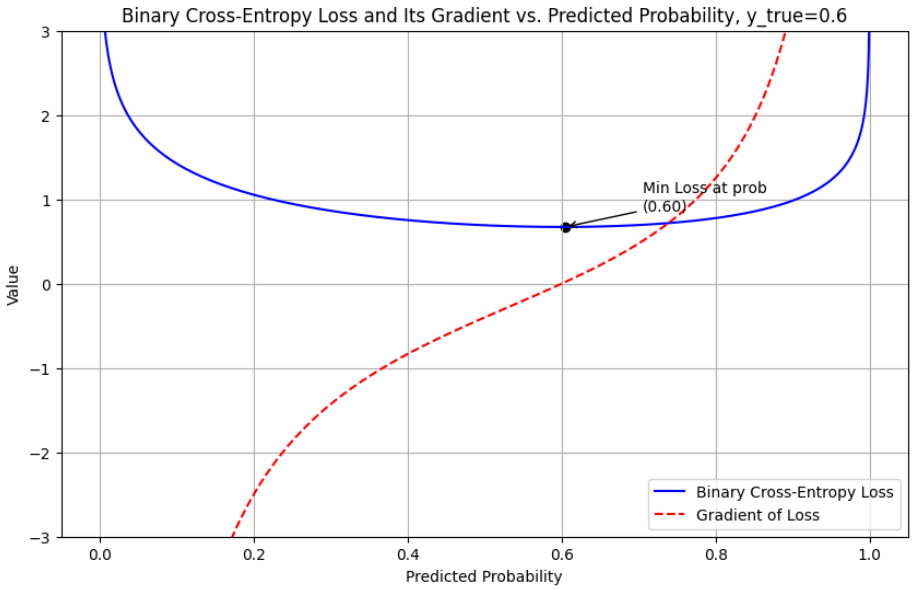

Figure 1: Setting y_{true} to a value other than 0 or 1 is a valid use of the crossentropy loss function and perfectly consistent with both KL divergence and MLE.

The term we will use for the idea of predicting a probability distribution is *probabilistic classification*.

5 Supervised Learning Preference Optimization

Armed with intuition of the DPO loss and probabilistic classification, we can now turn to a simpler, supervised learning approach to alignment. Our goals are:

- Apply only MLE under a probabilistic classification approach. That is, we only want to use a CCE loss function.

- Avoid any RL concepts like rewards.
- Avoid any extra ratios, logs, or the Bradley-Terry model.
- Continue optimizing sequences rather than individual tokens.
- Continue giving comparable weight to all examples during optimization.
- Continue stabilizing the optimization to prevent divergence.
- Instead of having an unconstrained objective, specify an objective with a natural "bowl shape" at a target, to assist in optimization.

The SLPO objective achieves these goals by directly optimizing probablities for the winning, losing, and all other sequences. It is defined as follows:

$$L_{\text{SLPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim D}\Bigg(\underbrace{w_w \cdot \log \pi_\theta(y_w \mid x)}_{1} + \underbrace{0 \cdot \log \pi_\theta(y_l \mid x)}_{2} + \underbrace{(1 - w_w) \cdot \log \pi_\theta(y_{\overline{w \cup l}} \mid x)}_{3} \Bigg) \tag{6}$$

where

x is the input context,

y_w is the winning token sequence,

y_l is the losing token sequence,

π_θ is the language model to be aligned,

π_{ref} is the reference language model,

$w_w = \pi_{\text{ref}}(y_w \mid x) + \pi_{\text{ref}}(y_l \mid x)$, and

$y_{\overline{w \cup l}}$ represents all other possible token sequences.

Intuitively, the goal is to take the probabilities of the winning and losing sequence and reallocate them to the winning sequence (underbrace 1). The losing sequence is optimized to zero probability (underbrace 2). Importantly, the other tokens are directly optimized to maintain the same probability as the reference model (underbrace 3). This is essentially the SLPO's version of the KL divergence regularization term in traditional RLHF.

Alternatively, a softer version of the SLPO objective would shift the probabilities in a probabilistic matter:

$$L_{\text{SLPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim D}\Big(w_w \cdot \log \pi_\theta(y_w \mid x) + w_l \cdot \log \pi_\theta(y_l \mid x) + (1 - w_w - w_l) \cdot \log \pi_\theta(y_{\overline{w \cup l}} \mid x) \Big) \tag{7}$$

where

$w_w = \alpha \cdot (\pi_{\text{ref}}(y_w \mid x) + \pi_{\text{ref}}(y_l \mid x))$,

$w_l = (1 - \alpha) \cdot (\pi_{\text{ref}}(y_w \mid x) + \pi_{\text{ref}}(y_l \mid x))$, and

$0.5 < \alpha <= 1.0$.

6 Implementation

For a standard language model, the vocabulary size v is typically in the range of 10,000 to 100,000 tokens. The sequence length T is typically in the range of 128 to 1024 tokens. So the $y_{\overline{w \cup l}}$ set of sequences is intractably large to enumerate.

Let us consider y_w independently of y_l. Then we can rewrite the SLPO Equation 6 in two halves and consider them separately:

$$L_{\text{SLPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim D}\Big(\underbrace{w_w \cdot \log \pi_\theta(y_w \mid x)}_{1} + \underbrace{(1 - w_w) \cdot \log \pi_\theta(y_{\overline{w}} \mid x)}_{2} + \underbrace{0 \cdot \log \pi_\theta(y_l \mid x)}_{3} + \underbrace{1 \cdot \log \pi_\theta(y_{\overline{l}} \mid x)}_{4} \Big) \tag{8}$$

The term focused on the winning sequence (underbrace 1) is simple: it is just the sum of the logits of the winning token sequences, passed through softmax and CCE with the target weight.

For the term focused on the losing sequence (underbrace 3), the loss at every token position is also simple: like underbrace 1, we sum the logits before we pass them into softmax and assign a target value of zero in the CCE loss. The terms in underbrace 2 and underbrace 4 are the tricky ones, since they deal with complementary sets. However, we can see a recursive pattern by visualizing a tree. For the first position in the sequence, there are essentially two options: the winning token (or losing token) and everything else. The target odds of everything else is simply $1 - w_w$, and the predicted odds is the

$$\sum \exp\left(\text{logsoftmax}(z_{\overline{w}})\right).$$

Obviously, we can calculate this with the logsumexp trick for both numerical stability and to stay in log space. Importantly, *we do not need to recursively explore sequence position 2 and beyond for sequences that start with a non-winning token.*

Similarly, for position two, we can directly calculate the predicted probability as

$$\pi_\theta(y_{w,1}) \cdot \pi_\theta(y_{\overline{w},2}).$$

Adding all such terms to our running total results in

$$\log \pi_\theta(y_{\overline{w}} \mid x),$$

which we compute efficiently in a vectorizable way.

Formally, the term $\log \pi_\theta(y_{\overline{w}} \mid x)$ is calculated as:

$$\log \sum_{j=1}^{T} \pi_\theta(y_{\overline{w},t} \mid y_{w:t-1}, x) \prod_{k=1}^{j} \pi_\theta(y_{w,k} \mid x, y_{w:<k}).$$

6.1 Code

```
See
  """Defines the SLPO loss function."""

  import torch

  from torch import Tensor

  from torch.nn import functional as F
  from torch.nn.modules.loss import _Loss


  class SLPO(_Loss):
      """The SLPO loss function.

      This loss function is used to align LLMs.

       Args:
          size_average (bool, optional): Deprecated (see :attr:‘reduction‘). By default,
              the losses are averaged over each loss element in the batch. Note that for
              some losses, there are multiple elements per sample. If the field :attr:‘size
              is set to ‘‘False‘‘, the losses are instead summed for each minibatch. Ignore
              when :attr:‘reduce‘ is ‘‘False‘‘. Default: ‘‘True‘‘
          reduce (bool, optional): Deprecated (see :attr:‘reduction‘). By default, the
              losses are averaged or summed over observations for each minibatch depending
              on :attr:‘size_average‘. When :attr:‘reduce‘ is ‘‘False‘‘, returns a loss per
              batch element instead and ignores :attr:‘size_average‘. Default: ‘‘True‘‘
          reduction (str, optional): Specifies the reduction to apply to the output:
              ‘‘’none’‘‘ | ‘‘’mean’‘‘ | ‘‘’sum’‘‘. ‘‘’none’‘‘: no reduction will be applied
              ‘‘’mean’‘‘: the sum of the output will be divided by the number of
```

```
            elements in the output, ‘‘’sum’‘‘: the output will be summed. Note: :attr:‘si
            and :attr:‘reduce‘ are in the process of being deprecated, and in the meantim
            specifying either of those two args will override :attr:‘reduction‘.

    Shape:
      - Input: :math:‘(N, V, T)‘ where :math:‘N‘ is the batch size,
      :math:‘V‘ is the vocabulary size, and :math:‘T‘ is the sequence length.
      - Target: Dictionary of ’pi_ref_w’ holding :math:‘(N)‘ of the winning sequence’s re
        ’pi_ref_l’ holding :math:‘(N)‘ of the losing sequence’s reference probability,
        ’winner’ holding :math‘(B, T)‘ representing the winning sequence, and
        ’loser’ holding :math‘(B, T)‘ representing the losing sequence.

    """
    __constants__ = ["reduction"]

    def __init__(self, size_average=None, reduce=None, reduction: str = "mean") -> None:
        super().__init__(size_average, reduce, reduction)

    def forward(self, input: Tensor, target: Tensor) -> Tensor:
        return slpo_loss(input, target)

import torch
import torch.nn.functional as F

def slpo_loss_hard_no_loop(input: torch.Tensor, target: dict) -> torch.Tensor:
    r"""
    Calculates the "hard" SLPO loss:
      - E_n [
          w_w * log p_theta(y_w)
        + (1 - w_w) * log p_theta(\overline{y_w})
        + 0 * log p_theta(y_l)
        + 1 * log p_theta(\overline{y_l})
      ]

    Args:
        input: Float tensor of shape (N, T, V) or (N, V, T).
               Raw logits from the LM for each batch element n,
               each time-step t, each vocab token v.
        target: A dict with entries:
            - ’pi_ref_w’: shape (N,). The reference prob for the winning sequence.
            - ’pi_ref_l’: shape (N,). The reference prob for the losing sequence.
            - ’winner’:   shape (N, T). The winning token IDs for each batch example.
            - ’loser’:    shape (N, T). The losing token IDs for each batch example.

    Returns:
```

```
        A scalar Tensor (mean over the batch).
    """
    # If input is (N, V, T), we transpose to (N, T, V) so that dim=-1 is vocabulary.
    if input.shape[1] != target['winner'].shape[1]:
        # Likely means shape is (N, V, T). We'll transpose so we have (N, T, V).
        input = input.transpose(1, 2)
    # Now input is (N, T, V).

    # Convert to log-probs
    log_probs = F.log_softmax(input, dim=-1)  # shape (N, T, V)

    # For convenience, define the "weight" for the winning path:
    #   w_w = pi_ref_w + pi_ref_l
    # We'll broadcast to shape (N,) automatically.
    w_w = target['pi_ref_w'] + target['pi_ref_l']

    # We'll get p_theta(y_w) and p_theta(\overline{y_w})
    log_p_yw, log_p_not_yw = _compute_log_p_and_log_p_not_y(
        log_probs, target['winner']
    )
    # Similarly for y_l
    log_p_yl, log_p_not_yl = _compute_log_p_and_log_p_not_y(
        log_probs, target['loser']
    )

    # Hard-SLPO objective, per example n:
    #  - [ w_w * log_p_yw
    #      + (1 - w_w) * log_p_not_yw
    #      + 1 * log_p_not_yl
    #      + 0 * log_p_yl
    #    ]
    # => negative sign outside
    loss_per_example = -(
        w_w * log_p_yw
        + (1.0 - w_w) * log_p_not_yw
        + 1.0 * log_p_not_yl
    )

    return loss_per_example.mean()


def _compute_log_p_and_log_p_not_y(log_probs: torch.Tensor,
                                   y_tokens: torch.Tensor) -> (torch.Tensor, torch.Tensor
    """
    Given:
```

```
  log_probs: shape (N, T, V) = log softmax outputs
  y_tokens: shape (N, T) = each row is a sequence of token-IDs
Returns:
  (log_p_y, log_p_not_y): each is shape (N,)

  Where log_p_y = log probability of exactly matching y_tokens,
        log_p_not_y = log probability of any sequence that differs
                      from y_tokens in at least one position.
"""
# 1) Gather the log-prob of the chosen token at each step => shape (N,T)
#    sum across T => log p(y).
#    But we also need partial prefix sums for the "first mismatch" trick.
#
# We'll gather along dim=-1, which is the vocab dimension:
#   y_tokens[n,t] is in [0..V-1].
# We must reshape y_tokens to (N,T,1) to gather from (N,T,V).
chosen_logp = log_probs.gather(dim=-1, index=y_tokens.unsqueeze(-1)).squeeze(-1)  # (

# log_p_y: sum over the T dimension (each example independently)
log_p_y = chosen_logp.sum(dim=1)  # => shape (N,)

# 2) For the complement, we do the prefix trick:
#
#    prefix[t] = sum_{k=1..t} log p(y_k)
#    mismatch[t] = prefix[t-1] + log( sum_{v != y_t} p(v_t) )
#    log_p_not_y = logsumexp_{t=1..T} mismatch[t].
#
# We'll build a prefix array of shape (N, T+1):
prefix = torch.zeros(log_probs.size(0), log_probs.size(1) + 1,
                     device=log_probs.device)
# prefix[:, t] = sum_{k=0..(t-1)} chosen_logp[:, k], if we do 0-based indexing
prefix[:, 1:] = torch.cumsum(chosen_logp, dim=1)

# log_all_t = logsumexp of all tokens at step t => shape (N,T)
log_all_t = torch.logsumexp(log_probs, dim=-1)  # (N,T)

# log_excl_t = log( sum_{v != y_t} p(v_t) ) = log_all_t + log(1 - exp(log_p(y_t) - lo
# => must do a stable "1 - e^{x}" approach. We'll define a helper:
def _safe_log_diff_exp(log_a, log_b, eps=1e-10):
    # compute log( exp(log_a) - exp(log_b) ),
    # with a clamp to avoid negative or zero inside the log.
    # We'll do log_a as the larger one. Here we want:
    #   log( sum_{v != y_t} p(v_t) )
    # = log_all_t + log(1 - exp(chosen_logp[t] - log_all_t)).
    diff = log_b - log_a   # chosen_logp[t] - log_all_t
```

```
        # clamp exp(diff) to avoid going above 1 or below 0
        e_diff = torch.clamp(torch.exp(diff), max=1.0 - eps)
        out = torch.log1p(- e_diff)  # log(1 - e^(diff))
        return out

    # Now do that for each t:
    # log_excl_t = log_all_t + log(1 - exp( chosen_logp[t] - log_all_t ))
    # shape => (N,T)
    log_excl_t = log_all_t + _safe_log_diff_exp(log_all_t, chosen_logp)

    # mismatch[t] = prefix[t-1] + log_excl_t at step t-1
    # We can shift them by 1 index:
    prefix_tminus1 = prefix[:, :-1]       # shape (N,T)
    log_excl_tminus1 = log_excl_t         # shape (N,T)

    mismatch = prefix_tminus1 + log_excl_tminus1  # shape (N,T)

    log_p_not_y = torch.logsumexp(mismatch, dim=1)  # shape (N,)

    return log_p_y, log_p_not_y
```

7 Future Work

Most importantly, for future work we will follow the experimental setup outlines by Rafailov et al. (2023).

We will also explore custom trainers that can bypass the autograd of PyTorch. Much of the reasoning of the DPO and SLPO objectives were in the gradient space, and only brought back into loss space at the end.

8 Conclusion

In this paper, we have analyzed the DPO objective, found its strengths and weaknesses, and proposed a new objective that is a pure supervised learning approach to alignment.

Given the fact that DPO appeared to be superior to RLHF, if either a formal proof or empirical experiments can prove that the SLPO is superior to DPO, we would consider whether Reinforcement Learning concepts were ever necessary in alignment.

Acknowledgments and Disclosure of Funding

The author thanks Chiara Cerini for her invaluable reviews of of this work.

Appendix A. Equivalence of Cross-Entropy on Sigmoid of Difference and Categorical Cross-Entropy on Softmax for Two Classes

Logistic regression uses a single logit α, transforming it through the *sigmoid* function to obtain a probability $\sigma(\alpha) = \frac{1}{1+e^{-\alpha}}$. The *binary cross-entropy* (BCE) loss for a label y and predicted probability $\hat{y} = \sigma(\alpha)$ is:

$$\text{BCE}(y, \hat{y}) \;=\; -\left[y \log(\hat{y}) \;+\; (1-y) \log(1-\hat{y})\right].$$

Alternatively, single-label multiclass classification of two outcomes uses two softmax logits z_1 and z_2 (one per class) and applies the softmax function to obtain probabilities:

$$p(y=1 \mid z_1, z_2) \;=\; \frac{e^{z_1}}{e^{z_1}+e^{z_2}}, \qquad p(y=2 \mid z_1, z_2) \;=\; \frac{e^{z_2}}{e^{z_1}+e^{z_2}}.$$

The associated categorical cross-entropy (CCE) loss is:

$$\text{CCE}(y, \hat{y}) \;=\; -\sum_{k=1}^{2} y_k \log(\hat{y}_k),$$

where $\hat{y}_1 = \frac{e^{z_1}}{e^{z_1}+e^{z_2}}$, $\hat{y}_2 = \frac{e^{z_2}}{e^{z_1}+e^{z_2}}$, $y_1 = 1$, and $y_2 = 0$.

If we set

$$\alpha = z_1 - z_2,$$

then

$$\frac{e^{z_1}}{e^{z_1}+e^{z_2}} = \frac{1}{1+e^{z_2-z_1}} = \sigma(\alpha).$$

Therefore, the predicted probability $\hat{y_1}$ from softmax matches the $\sigma(\alpha)$ in logistic regression. This means that the BCE loss on the sigmoid of a difference of two numbers $(z_1 - z_2)$ is equivalent to the CCE loss on the softmax of the first class (z_1) (Goodfellow et al. (2016)).

Appendix B. Softmax as a sum as a joint probability

We prove this for the special case of two examples, each with N possible classes, where softmax probabilities are given by:

$$\hat{y}_j^1 = \frac{e^{z_j^1}}{\sum_m e^{z_m^1}}, \quad \hat{y}_k^2 = \frac{e^{z_k^2}}{\sum_n e^{z_n^2}}$$

The joint probability of selecting class j for the first example and class k for the second is:

$$P(j,k) = \hat{y}_j^1 \hat{y}_k^2$$

We define the target joint probability as:

$$y_{\text{true},j,k} = C, \quad \sum_{(m,n)\neq(j,k)} y_{\text{true},m,n} = 1 - C.$$

We aim to show that the cross-entropy loss:

$$L = -C \log(\hat{y}_j^1 \cdot \hat{y}_k^2) - (1 - C) \log(\Omega)$$

where Ω represents all other possible outcomes, a set with cardinality $C^2 - 1$.
can be rewritten as the categorical cross-entropy of a softmax over two terms.
Define the logits for two terms:

$$s_1 = \log \hat{y}_j^1 + \log \hat{y}_k^2 = z_j^1 - \log \sum_m e^{z_m^1} + z_k^2 - \log \sum_n e^{z_n^2},$$

$$s_2 = \log(\Omega).$$

Applying the softmax function to these logits:

$$p_1 = \frac{e^{s_1}}{e^{s_1} + e^{s_2}}, \quad p_2 = \frac{e^{s_2}}{e^{s_1} + e^{s_2}}.$$

Since $e^{s_1} = \hat{y}_j^1 \hat{y}_k^2$ and $e^{s_2} = \Omega$, we obtain:

$$p_1 = \frac{\hat{y}_j^1 \hat{y}_k^2}{\hat{y}_j^1 \hat{y}_k^2 + \Omega}, \quad p_2 = \frac{\Omega}{\hat{y}_j^1 \hat{y}_k^2 + \Omega}.$$

The categorical cross-entropy loss with target distribution $[C, 1 - C]$ is:

$$\begin{aligned} L &= -C \log p_1 - (1 - C) \log p_2 \\ &= -C \log \left(\frac{\hat{y}_j^1 \hat{y}_k^2}{\hat{y}_j^1 \hat{y}_k^2 + \Omega} \right) - (1 - C) \log \left(\frac{\Omega}{\hat{y}_j^1 \hat{y}_k^2 + \Omega} \right). \end{aligned}$$

Expanding the logarithms:

$$\begin{aligned} L = &-C \left(\log(\hat{y}_j^1 \hat{y}_k^2) - \log(\hat{y}_j^1 \hat{y}_k^2 + \Omega) \right) \\ &- (1 - C) \left(\log(\Omega) - \log(\hat{y}_j^1 \hat{y}_k^2 + \Omega) \right) \\ = &-C \log(\hat{y}_j^1 \hat{y}_k^2) - (1 - C) \log(\Omega). \end{aligned}$$

Thus, we have rewritten the original loss function as the categorical cross-entropy of a softmax over two terms:

$$L = \text{CCE}(\text{softmax}([s_1, s_2]), y_{\text{true}} = [C, 1 - C]).$$

Appendix C. Approximate Equivalence of Softmax of Sum and Temperature Scaling

Here we show that the two-class softmax of a sum of inputs $\alpha_0..\alpha_T$, holding all other inputs constants, is appromixated by temperature scaling, under simplifying assumptions:

- We consider the case where the first class is true and the second class is false.

- we assume the output of softmax is passed to Categorical Crossentropy (CCE) loss.
- We assume the second logit to the softmax is zero.
- We assume the inputs are approximately equal to each other.
- We assume the many inputs, such as 100.

The softmax of the sum of T inputs is:

$$\text{softmax}([\alpha_1 + \alpha_2 + \ldots + \alpha_T, 0]) = \frac{e^{\alpha_1+\alpha_2+\ldots+\alpha_T}}{e^{\alpha_1+\alpha_2+\ldots+\alpha_T} + e^0}$$

The gradient of the softmax function with a CCE loss is well known to be:

$$\frac{\partial \text{CCE}}{\partial z_i} = \hat{y_i} - y_i.$$

Equivalently, since $\hat{y_i}$ is the softmax of the sum of the inputs,

$$\frac{\partial \text{CCE}}{\partial z_i} = \text{Softmax}(z_i) - y_i,$$

where z_i is the input to softmax. Applying this to our unusual softmax of a sum formulation by setting $z_i = \sum_{i=1}^{T} \alpha_i$:

$$\frac{\partial \text{CCE}}{\partial z_i} = \text{Softmax}([\sum_{i=1}^{T} \alpha_i, 0]) - y_i.$$

The nuance here is that due to the chain rule, the gradient on the sum of the inputs is passed to each input independently. This is the heart of the "multiplication" effect which we will soon see is equivalent to temperature scaling. For every input α_k, the gradient is:

$$\frac{\partial \text{CCE}}{\partial \alpha_k} = \text{Softmax}([\sum_{i=1}^{T} \alpha_i, 0]) - y_i.$$

But since we assumed all the inputs are approximately equal to each other, this is equivalent to:

$$\frac{\partial \text{CCE}}{\partial \alpha_k} = \text{Softmax}([T \cdot \alpha_k, 0]) - y_i.$$

for all k. Rewriting this with the definition of softmax

$$\frac{\partial \text{CCE}}{\partial \alpha_k} = \frac{e^{T \cdot \alpha_k}}{e^{T \cdot \alpha_k} + e^0} - y_i. \tag{9}$$

shows a multipler on the input α_k of T, representing the cardinality of the elements of the sum (e.g. tokens in a sequence).

Now let's turn to the temperature scaling. The canonical definition of temperature scaling per (Hinton et al., 2015, Equation 2) is the following, where we use the term temp^{-1} to distinguish it from our use of T as the number of inputs:

$$\frac{\partial \mathcal{C}}{\partial z_i} = \text{temp}^{-1} \left(\frac{e^{z_i \cdot \text{temp}^{-1}}}{\sum_j e^{z_j \cdot \text{temp}^{-1}}} - \frac{e^{v_i \cdot \text{temp}^{-1}}}{\sum_j e^{v_j \cdot \text{temp}^{-1}}} \right)$$

where v_i are logits of the teacher and z_i are the logits of the student. Equivalently:

$$\frac{\partial \mathcal{C}}{\partial z_i} = \left(\frac{e^{z_i \cdot \text{temp}^{-1} - \log(\text{temp})}}{\sum_j e^{z_j \cdot \text{temp}^{-1} - \log(\text{temp})}} - \frac{e^{v_i \cdot \text{temp}^{-1} - \log(\text{temp})}}{\sum_j e^{v_j \cdot \text{temp}^{-1} - \log(\text{temp})}} \right).$$

Considering we only have two classes, and the second class has a fixed logit of zero, we can simplify this to:

$$\frac{\partial \mathcal{C}}{\partial z_1} = \left(\frac{e^{z_1 \cdot \text{temp}^{-1} - \log(\text{temp})}}{e^{z_1 \cdot \text{temp}^{-1} - \log(\text{temp})} + e^{-\log(\text{temp})}} - \frac{e^{v_1 \cdot \text{temp}^{-1} - \log(\text{temp})}}{e^{v_1 \cdot \text{temp}^{-1} - \log(\text{temp})} + e^{-\log(\text{temp})}} \right),$$

As the logits are optimized up to reasonable non-zero values such as 1.0, and with the assumption that the term temp^{-1} is large (e.g. the number of tokens in a sequence such as 100), the terms log(temp) is dominated by the term temp^{-1} and the term . We can replace the terms that use v with y_i since we are using hard targets, not soft labels from a teacher. Hence, we can approximate the gradient under KD as

$$\frac{\partial \mathcal{C}}{\partial z_i} = \left(\frac{e^{z_1 \cdot \text{temp}^{-1}}}{e^{z_1 \cdot \text{temp}^{-1}} + \text{C}} - y_1 \right) \tag{10}$$

where C is a small constant that is dominated by the term $e^{z_1 \cdot \text{temp}^{-1}}$. For example, at reasonable values of $z_1 = 1.0$ and $\text{temp}^{-1} = 100$, the term $e^{z_1 \cdot \text{temp}^{-1}} = e^{100}$, while the term $e^{-\log(\text{temp})} = 1/temp = 0.01$.

This exactly of the form of Equation 9 with the temperature scaling term temp^{-1} replacing the number of inputs (such as the number of tokens in a sequence) T. Hence, optimization proceeds, a softmax of a sum of many inputs is equivalent to optimizing the softmax of each input with a low temperature.

See Figure C for an illustration of this concept.

References

R. A. Bradley and M. E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952. doi: 10.2307/2334029. URL `https://doi.org/10.2307/2334029`.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E Gonzalez, et al. Chatbot arena: An open platform for evaluating llms by human preference. *arXiv preprint arXiv:2403.04132*, 2024.

Arpad E. Elo. *The Rating of Chessplayers, Past and Present*. Arco Publishing, New York, 1978. ISBN 9780668047210.

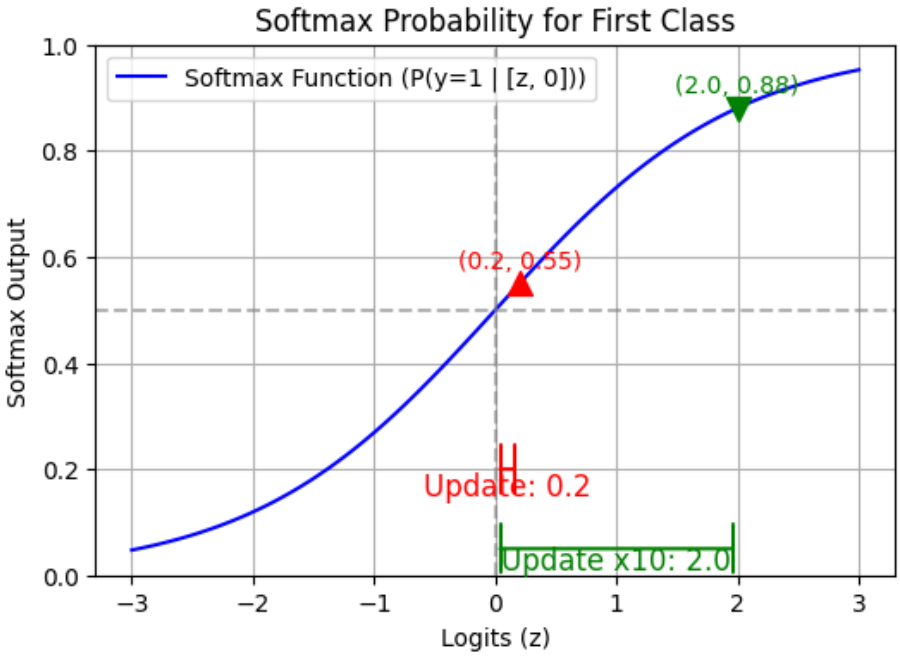

Figure 2: The red update represent the actual update to a logit. But on the next itertion of backprop, the gradient is not calculated based on the sigmoid of logit itself, but the sigmoid of the sum of the logit and nine others, all of which are approximately equal. This pushes the gradient calculation out to a less steep part of the curve. This "step function" like behavior is similar to a low temperature scaling of a logit during training.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL http://www.deeplearningbook.org.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Dark knowledge. Presented at the University of Chicago, 2014. URL https://www.ttic.edu/dl/dark14.pdf.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. URL https://arxiv.org/abs/1503.02531.

Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Bo Pang and Lillian Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the ACL*, 2004.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 53728–53741. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/a85b405ed65c6477a4fe8302b5e06ce7-Paper-Conference.pdf.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.