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

Yaoshiang Ho YAOSHIANG@GMAIL.COM

Abstract

Direct Preference Optimization (DPO) is a popular approach to aligning large language models with human preferences. In this paper, we analyze the underlying math and propose a new algorithm which we call Supervised Learning Preference Optimization (SLPO).

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:

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

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, 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.*

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.

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. The term $\pi_{\text{ref}}(y_t \mid x, y_{<t})$ 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 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:

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

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

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

which simplifies to:

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

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.

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 $y_{\text{true}} = [1, 0]$. This is well known but derived in Appendix A. This allows us to rewrite the DPO objective as follows:

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

where

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

$$S_l = \sum_{t=1}^{T_l} \left(\mathrm{g}_\theta(y_{l,t} \mid x, y_{l,<t}) - \mathrm{g}_{\text{ref}}(y_{l,t} \mid x, y_{l,<t})\right).$$

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 value, it's a sum of differences of values. That difference, and that sum, are the remaining nuances to resolve.

Key Observation #2: All the language model's winning and losing token logits are scaled to zero at the start of training. Since the language model is initialized by the reference model, the difference starts at zero for the logits of all the differences of logits $\mathrm{g}_\theta - \mathrm{g}_{\text{ref}}$. This means that at the start of training, all the gradients on all the winning tokens in the language model will be the same (and likewise for the losing tokens). This difference is the vestige of the *log odds ratio of the language model* in the original DPO formulation.

Key Observation #3: The softmax of a sum of inputs is equivalent to mean of softmax on each input with a very low temperature, strongly disincentivizing material updates during backpropagation. The final remaining nuance is that the softmax is applied to the sum of the inputs, where the inputs were originally softmax logits during next token prediction pretraining. What happens when the input to a softmax with two outputs is a sum? First, we make a few reasonable assumptions.

- The softmax's $y_{\text{true}} = [1, 0]$.
- The softmax's first input S_w and its subcomponents, the shifted logits, starts at zero and increases during optimization, remaining non-negative.
- The softmax's second input S_l and its subcomponents, the shifted logits, starts at zero and decreases during optimization, remaining non-positive.
- The shifted winning logits will move in sync with each other. This would certainly be true if the logits were free variables, but the logits are entangled since they are the result of the complex nonlinear function g.

- The shifted losing logits will move in sync with each other. This would certainly be true if the logits were free variables, but the logits are entangled since they are the result of the complex nonlinear function g.

We show in Appendix B that the softmax where an input is the sum of subcomponents under the above assumptions is approximated by optimizing each input directly via CCE(softmax) with temperature scaling to a very low temperature, introduced in Hinton et al. (2015).

Key Observation #4: Although optimization of the shifted logits are strongly decelerated, the final objective is still $+inf$ and $-inf$ for each shifted 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 which may have been perfectly fungible as alternative winning sequences.

We have been able to remove much of the machinery of the DPO objective, first through algebraic manipulation, and then through some assumptions. can we take these learning and fix the final issue of the DPO loss?

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.

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

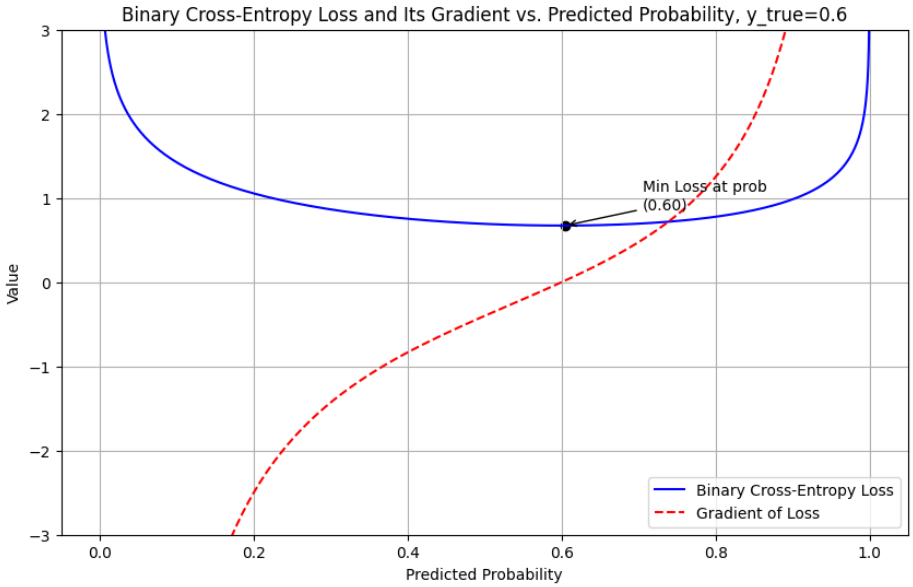

Figure 1: Setting y_{true} to a value other than 0 or 1 is a valid use of the crossentropy loss function and perfectly consistent with both KL divergence and MLE.

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 unexplained ratios, logs, or Bradley-Terry models.
- Instead of having an unconstrained objective and then heavily decelerating it, specify an objective with a natural "bowl shape" to assist in optimization.
- Explicitly stabilize the model's predictions for all other sequences other than the winning and losing sequences.

The SLPO objective is defined as follows:

$$L_{\text{SLPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim D}$$
$$\left(\left(\underbrace{w_w * \log(\pi_\theta(y_w \mid x))}_{1}\right) + \left(\underbrace{0 * \log(\pi_\theta(y_l \mid x))}_{2}\right) + \left(\underbrace{(1-w_w) * \log(\pi_\theta(y_o \mid x))}_{3}\right)\right) \quad (6)$$

where

$$w_w = \cdot \left(\pi_{\text{ref}}(y_w \mid x) + \pi_{\text{ref}}(y_l \mid x)\right).$$

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 - This is essentially the SLPO's version of the KL divergence regularization term in traditional RLHF (underbrace 3).

Alternatively, a softer version of the SLPO objective would be

$$L_{\text{SLPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim D} \\ ((w_w * \log(\pi_\theta(y_w \mid x))) + (w_l * \log(\pi_\theta(y_l \mid x))) + ((1 - w_w - w_l) * \log(\pi_\theta(y_o \mid x)))) \quad (7)$$

where

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

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

and

$$0.5 < \alpha <= 1.0.$$

6 Implementation

The y_o set of sequences grows intractably as vocabsize$^{\text{sequencelength}}$.

6.1 Efficient Implementation of the SLPO Objective

However, we can optimize the SLPO objective by recognizing that the output of a model with a vocabulary size of v (i.e., the number of possible tokens) and a sequence length of t (i.e., the number of tokens per sequence) produces a tensor of shape:

$$(B, T, V)$$

where:

- B is the batch size,
- T is the sequence length,
- V is the vocabulary size.

This tensor represents $v \times t$ logits per sample, corresponding to the unnormalized probabilities for each possible token at each timestep.

Of these $v \times t$ logits, we can mask y_w as a vector of length t, extracting the log probabilities for the winning tokens while treating the sum over all other logits (i.e., those corresponding to y_o and y_l) as the probability of $P(y_o + y_l)$.

For numerical stability, instead of directly summing softmax probabilities, we use the **logsumexp trick**, which ensures stable computation by preventing underflow and overflow when summing probabilities in the log domain:

$$\log P(y_o + y_l) = \text{logsumexp}_{y_o, y_l} \pi_\theta(y_o \mid x).$$

We can implement this loss function efficiently in PyTorch as follows:

```
import torch
import torch.nn.functional as F

import torch
import torch.nn.functional as F

def slpo_loss(logits, y_w, y_l, w_w):
    """Computes the SLPO loss.

    Args:
        logits: Model output logits of shape (B, T, V)
        y_w: Indices of winning tokens (B, T)
        y_l: Indices of losing tokens (B, T)
        w_w: Weight for winning sequence (B, T)

    Returns:
        Tensor: SLPO loss
    """
    # Compute log probabilities using log_softmax for numerical stability
    log_probs = F.log_softmax(logits, dim=-1)  # (B, T, V)

    # Extract log probabilities for the winning tokens y_w
    log_prob_w = log_probs.gather(dim=-1, index=y_w.unsqueeze(-1)).squeeze(-1)  # (B, T)

    # Create a mask to exclude y_w and y_l from logsumexp computation
    mask = torch.ones_like(log_probs, dtype=torch.bool)  # Create a mask of ones (B, T, V)
    mask.scatter_(-1, y_w.unsqueeze(-1), False)  # Mask out y_w
    mask.scatter_(-1, y_l.unsqueeze(-1), False)  # Mask out y_l

    # Apply the mask to log_probs, replacing masked entries with a large negative value
    log_probs_masked = log_probs.masked_fill(~mask, float('-inf'))

    # Compute logsumexp over all vocabulary tokens except y_w and y_l (P(y_o))
    log_prob_o = torch.logsumexp(log_probs_masked, dim=-1)  # (B, T)

    # Compute final loss: weighted sum of log probabilities
    loss = -torch.mean(w_w * log_prob_w + (1 - w_w) * log_prob_o)

    return loss
```

7 Results

TBD

8 Conclusion

TBD

Acknowledgments and Disclosure of Funding

The author thanks Chiara Cerini, Peter Tran, David Wang, and Sam Wookey for their invaluable reviews of early drafts 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. 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:

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

for all k. Rewriting this with the definition of softmax

$$\frac{\partial \text{CCE}}{\partial \alpha_k} = \frac{e^{T\times\alpha_k}}{e^{T\times\alpha_k} + e^0} - y_i. \tag{8}$$

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 $\log(\text{temp})$ 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

$$\frac{\partial \mathcal{C}}{\partial z_i} = \left(\frac{e^{z_1 \cdot \text{temp}^{-1}}}{e^{z_1 \cdot \text{temp}^{-1}} + \text{C}} - y_1 \right) \tag{9}$$

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 8 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 B 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.

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`.

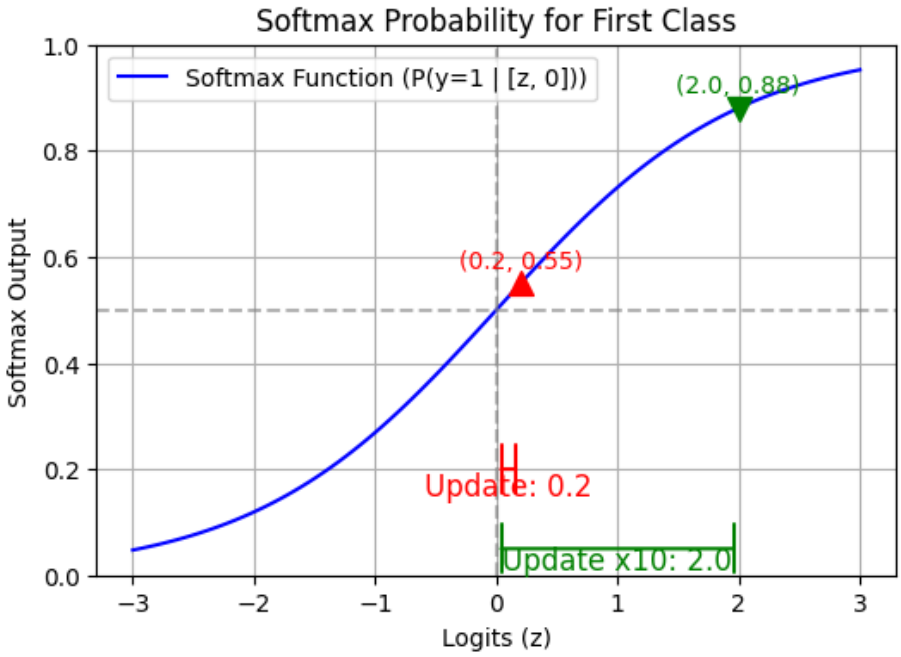

Figure 2: The red update represent the actual update to a logit. But on the next itertion of backprop, the gradient is not calculated based on the sigmoid of logit itself, but the sigmoid of the sum of the logit and nine others, all of which are approximately equal. This pushes the gradient calculation 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.

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.

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.