EECS 182 Deep Neural Networks
Fall 2026 Gireeja Ranade

Homework 12

This homework is due on Wednesday, November 25, 2026, at 10:59PM.

1. Comparing Distributions

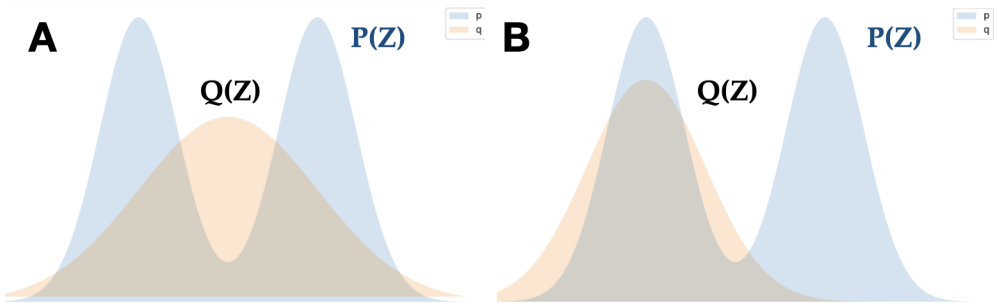
Divergence metrics provide a principled measure of difference between a pair of distributions (P, Q). One such example is the Kullback-Leibler Divergence, that is defined as

$$D_{\mathrm{KL}}(P||Q) = \mathbb{E}_{x\sim P(x)}\Big[\log \frac{P(x)}{Q(x)}\Big]$$

(a) Technically D_{KL} is not a true distance since it is asymmetric, i.e. generally $D_{\mathrm{KL}}(P||Q) \neq D_{\mathrm{KL}}(Q||P)$. **Give an example of univariate distributions P and Q where $D_{\mathrm{KL}}(P||Q) \neq \infty$, $D_{\mathrm{KL}}(Q||P) = \infty$.**

(b) For a fixed target distribution P, we call $D_{\mathrm{KL}}(P||Q)$ the *forward-KL*, while calling $D_{\mathrm{KL}}(Q||P)$ the *reverse-KL*. Due to the asymmetric nature of KL, distributions Q that minimize $D_{\mathrm{KL}}(P||Q)$ can be different from those minimizing $D_{\mathrm{KL}}(Q||P)$.

From the following plots, **identify which of (A, B) correspond to minimizing forward vs. reverse KL. Give brief reasoning.** Here, only the mean and standard deviation of Q are allowed to vary during the minimization.



2. Entropy, Cross-Entropy, Kullback - Leibler (KL)-divergence

Entropy is a measure of expected surprise. For a discrete random variable Y, the surprise of observing $Y = k$ is:

$$\log \frac{1}{p(Y=k)} = -\log[p(Y=k)]$$

As given:

- if $p(Y = k) \to 0$, the surprise of observing k approaches ∞
- if $p(Y = k) \to 1$, the surprise of observing k approaches 0

 All Rights Reserved. This may not be publicly shared without explicit permission.

The Entropy of the distribution of Y is then the expected surprise given by:

$$H(Y) = E_{Y\sim p}\Big[-\log p(Y)\Big] = -\sum_k \Big[p(Y=k)\log[p(Y=k)]\Big]$$

Cross-entropy is the expected surprise under q for samples drawn from p. It is given by:

$$\begin{aligned} H(p,q) &= E_{x\sim p}\Big[-\log q(x)\Big] \\ &= \sum_x \Big[p(x)\log[\frac{1}{q(x)}]\Big] \end{aligned}$$

Relative Entropy also known as KL Divergence measures how much one distribution diverges from another. For two discrete probability distributions, p and q, it is defined as:

$$D_{KL}(p||q) = \sum_x \Big[p(x)\log[\frac{p(x)}{q(x)}]\Big]$$

All logarithms in this problem are base 2, so entropy and KL divergence are measured in bits.

(a) Let's define the following probability distributions given by:

$$p(x) = \begin{cases} 0.5 & \text{when x = 1} \\ 0.5 & \text{when x = -1} \end{cases}, \text{ and } q(x) = \begin{cases} 0.1 & \text{when x = 1} \\ 0.9 & \text{when x = -1} \end{cases}.$$

Note that $H(p) = -\log 0.5 = 1$ bit and $H(q) = -0.1\log 0.1 - 0.9\log 0.9 \approx 0.47$ bits. Show that KL-divergence is not symmetric and hence does not satisfy some intuitive attributes of distances.

(b) Re-write $D_{KL}(p||q)$ in terms of the entropy $H(p)$ and the cross entropy $H(p,q)$.

3. Policy Gradient and the Reparameterization Gradient Estimator

In this question, you are going to derive two gradient estimators for the following objective function:

$$\mathcal{F}(\theta) = \mathbb{E}_{\mathbf{x}\sim p_\theta}\left[f(\mathbf{x})\right], \tag{1}$$

where $\mathbf{x}$ is a random variable that follows the probability distribution of $p_\theta : \mathcal{X} \mapsto \Delta(\mathcal{X})$ that is parameterized by $\theta \in \mathbb{R}^P$, and $f : \mathcal{X} \mapsto \mathbb{R}$ is a function.

(a) Let $\mathbf{x}$ be a K-dimensional multivariate Gaussian random variable that is parameterized by the mean $\mu \in \mathbb{R}^K$ under the distribution,

$$p_\mu(\mathbf{x}) = (2\pi\sigma^2)^{-K/2}\exp\left(-\|\mathbf{x}-\mu\|_2^2/(2\sigma^2)\right), \tag{2}$$

where $\sigma \in \mathbb{R}$ is a scalar constant. **Express $\nabla_\mu\mathcal{F}(\mu)$ as an expectation under** $p_\mu(\mathbf{x})$ (i.e., find $g(\mathbf{x})$ in $\nabla\mathcal{F}(\mu) = \mathbb{E}_{\mathbf{x}\sim p_\mu}\left[g(\mathbf{X})\right]$). *(Hint: $\mathbb{E}_{\mathbf{x}\sim p_\theta}\left[f(\mathbf{x})\right] = \int p_\theta(\mathbf{x})f(\mathbf{x})d\mathbf{x}$)*

(b) **Rewrite the expression in $\mathcal{F}(\mu)$ with an expectation with the distribution over a standard normal $\mathcal{N}(0,\mathbf{I})$?** *(Hint: if $\mathbf{x}$ follows the distribution of $p_\mu(\mathbf{x})$, then $\mathbf{x}-\mu$ follows the distribution of $\mathcal{N}(0,\sigma^2\mathbf{I})$)*

 All Rights Reserved. This may not be publicly shared without explicit permission.

(c) Using the expression you obtained from the previous part, **can you express $\nabla_\mu \mathcal{F}(\mu)$ in a similar way** (i.e., with an expectation with the distribution over $p_{\mu=0}(\mathbf{x})$)? *(Hint: ∇_μ can be safely moved inside the expectation because the expectation no longer follows a distribution that depends on μ and expectations are linear.)*

For the following two parts, we are going to generalize our findings from the specific Gaussian case to arbitrary p_θ's (you may assume p_θ has well-defined gradient $\nabla_\theta p_\theta$ that expresses what change in parameters would make a particular realization more likely).

(d) In the general case, **can you still write $\nabla \mathcal{F}(\theta)$ as an expectation under $p_\theta(\mathbf{x})$?** *(Hint: look at your answer in Part (a) and see how the term inside your expectation relates to $\log p_\theta(\mathbf{x})$).*

(e) Assume there exists a function $g(\mathbf{z}, \theta) : \mathcal{Z} \times \mathbb{R}^P \mapsto \mathcal{X}$ and a distribution over $\mathbf{z}$, $p(\mathbf{z})$, such that $g(\mathbf{z}, \theta)$ has the same distribution as $\mathbf{x} \sim p_\theta(\mathbf{x})$. **Prove that** $\nabla_\theta \mathcal{F}(\theta) = \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\left[\nabla_\theta g(\mathbf{z}, \theta)^\top \nabla_{\mathbf{x}} f|_{\mathbf{x}=g(\mathbf{z},\theta)}\right]$, where $\nabla_\theta g$ is the Jacobian of g with respect to θ. In the previous part (b) and (c), we actually prove a special case of this. **Can you determine what g and $p(z)$ are for the special case?**

These kinds of tricks are practically useful because the spirit of stochastic gradient descent plays nicely whenever we have gradients that we can view as expectations.

4. Honey, Where's My Reward Model?

In the standard Reinforcement Learning from Human Feedback (RLHF) pipeline like in Ziegler et al, there are usually three phases:

1 **SFT:** We perform Supervised Fine-Tuning (SFT) on a pre-trained LM using a dataset of high-quality demonstrations (e.g., summarization, sentiment analysis, etc.) to obtain a model π^{SFT}.

2 **Reward Modeling:** Next, we use the SFT model to generate pairs of answers $(y_1, y_2) \sim \pi^{\text{SFT}}(y \mid x)$ for various prompts x. Human labelers rank these pairs, producing a dataset of preferences $\mathcal{D} = \{(x^{(i)}, y_w^{(i)}, y_l^{(i)})\}_{i=1}^N$, where y_w denotes the preferred (winning) output and y_l the dispreferred (losing) output amongst (y_1, y_2), respectively. Crucially, we assume these preferences are generated by some latent (hidden) reward model $r^*(x, y)$, which we do not have access to. Instead, we parameterize a proxy reward model $r_\phi(x, y)$ to approximate it. However, we face a challenge: How do we model these preferences, which are inherently discrete ($y_w \succ y_l$), using a reward model that outputs continuous scalars?

To bridge this gap, we employ the Bradley-Terry (BT) model. The BT model assumes that the probability of a human preferring one response over another is given by the sigmoid of the difference in their latent rewards:

$$p^*(y_w \succ y_l|x) = \sigma(r^*(x, y_w) - r^*(x, y_l)) = \frac{\exp\big(r^*(x, y_w)\big)}{\exp\big(r^*(x, y_w)\big) + \exp\big(r^*(x, y_l)\big)}$$

This formulation allows us to frame preference learning as a binary classification problem because it treats the reward difference as a logit: if $r^*(x, y_w)$ is significantly higher than $r^*(x, y_l)$, the sigmoid outputs a probability near 1, matching the "label" that y_w is the winner. As a result, we can estimate the parameters ϕ via Maximum Likelihood Estimation (MLE) by minimizing the negative log-likelihood loss:

$$\mathcal{L}_R(\phi) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l))\right]$$

 All Rights Reserved. This may not be publicly shared without explicit permission.

In the context of LMs, $r_\phi(x, y)$ is often initialized from the SFT model $\pi^{\text{SFT}}(y \mid x)$ with the addition of a classification head to produce a single scalar prediction for the reward value. Furthermore, to ensure a reward function with lower variance, prior works normalize the rewards, such that $\mathbb{E}_{(x,y)\sim\mathcal{D}}[r_\phi(x, y)] = 0$ for all x.

3 **RL Fine-Tuning:** Finally, we use learned reward function as feedback to the language model. The optimization problem is formulated as

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D},y\sim\pi_\theta(y|x)} \left[r_\phi(x, y)\right] - \beta\mathbb{D}_{\text{KL}}[\pi_\theta(y \mid x)\|\pi_{\text{ref}}(y \mid x)]$$

where β is a parameter controlling the deviation from the base reference policy π_{ref}, namely the initial SFT model π^{SFT}. In practice, the language model policy π_θ is also initialized to π^{SFT}. Due to the discrete nature of language generation, the objective is not differentiable so it is typically optimized using reinforcement learning. The standard approach has been to construct the reward function $r(x, y) = r_\phi(x, y) - \beta(\log \pi_\theta(y \mid x) - \log \pi_{\text{ref}}(y \mid x))$, and maximize using PPO.

Unfortunately, this three-step process is brittle and computationally expensive. This is where Direct Preference Optimization (DPO) comes in.

In this problem, you will derive **Direct Preference Optimization (DPO)**, a method introduced by Rafailov et al. that provides a simple yet effective approach for policy optimization using preferences directly.

(a) As a warm up, find the optimal p^* for the optimization problem:

$$\min_{p\in\mathcal{P}} \mathbb{D}_{\text{KL}}(p\|q)$$

where

$$\mathbb{D}_{\text{KL}}(p\|q) = \mathbb{E}_{x\sim p(x)} \log\left(\frac{p(x)}{q(x)}\right) = \sum_x p(x) \log\left(\frac{p(x)}{q(x)}\right) \geq 0$$

is the KL divergence of p from q.

Hint: $\mathbb{D}_{\text{KL}}(p\|q) = 0$ *iff* $p = q$

(b) As discussed earlier, we wish to find a language model policy π_θ that maximizes the expected reward while preventing the model from drifting too far from the reference model:

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D},y\sim\pi_\theta(y|x)} \left[r_\phi(x, y)\right] - \beta\mathbb{D}_{\text{KL}}[\pi_\theta(y \mid x)\|\pi_{\text{ref}}(y \mid x)]$$

Show that the optimal solution $\pi_{\theta^*}(y \mid x)$ takes the form:

$$\pi_{\theta^*}(y|x) = \frac{1}{Z(x)}\pi_{\text{ref}}(y \mid x) \exp\left(\frac{1}{\beta}r_\phi(x, y)\right)$$

where $Z(x) = \sum_y \pi_{\text{ref}}(y \mid x) \exp\left(\frac{1}{\beta}r_\phi(x, y)\right)$ is the partition function.

Hint 1: Transform the problem from maximization to minimization by negating the objective and dividing by β. Recall that the optimal solution π_{θ^*} is invariant to scaling or shifting the objective function.

Hint 2: Expand the definition of $\mathbb{D}_{\text{KL}}$ and combine it with the reward term so that the entire objective is a single expectation over $y \sim \pi_\theta(y \mid x)$.

Hint 3: Define the distribution $\pi^*(y \mid x) = \frac{1}{Z(x)}\pi_{\text{ref}}(y \mid x) \exp\left(\frac{1}{\beta}r_\phi(x, y)\right)$. Multiply and divide your objective term by $Z(x)$ so you can substitute π^* into the equation.

 All Rights Reserved. This may not be publicly shared without explicit permission.

Hint 4: Finally, rearrange the objective into the form $\min_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D}}\left[\mathbb{D}_{\mathrm{KL}}(\pi_\theta\|\pi^*) - C\right]$. *From here, your answer from part (a) will be helpful.*

(c) Even though we've obtained the optimal policy $\pi_{\theta^*}(y \mid x)$, this particular representation of it is hard to utilize in practice. Why? *Hint: How large can the output space* $\mathcal{Y}$ *be?*

(d) Using the expression for the optimal policy π_{θ^*} derived in part (b), express the reward function $r_\phi(x, y)$ in terms of π_{θ^*}, π_{ref}, and $Z(x)$.

(e) With this reparametrization of $r_\phi(x, y)$, substitute it into the Bradley-Terry model to obtain the probability of human preference data in terms of the optimal policy. Furthermore, show that the partition function $Z(x)$ cancels out in your computation. Why is this desirable?

Now that we have the probability of human preference data in terms of the optimal policy rather than the reward model, we can formulate the Maximum Likelihood objective for the parameterized policy π_θ

$$\mathcal{L}_{\mathrm{DPO}}(\pi_\theta; \pi_{\mathrm{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_w \mid x)}{\pi_{\mathrm{ref}}(y_w \mid x)} - \beta\log\frac{\pi_\theta(y_l \mid x)}{\pi_{\mathrm{ref}}(y_l \mid x)}\right)\right]$$

which is exactly the result we wanted!

Recall the complex RLHF pipeline we started with: **SFT** $\rightarrow$ **Reward Modeling** $\rightarrow$ **RL Fine-Tuning**. By deriving the equation above, we have bypassed the need for an explicit reward model $r_\phi(x, y)$ and the RL loop entirely. Instead, we are fitting an *implicit* reward model (defined by the log-ratio of our policy and the reference) using a simple binary cross-entropy loss.

Crucially, because we derived this objective by algebraically solving for the reward in part (d), the policy π_θ that minimizes this loss is guaranteed to be the optimal policy for that implicit reward function. We have thus reduced the RLHF pipeline to a standard supervised classification problem!

(f) To verify how this optimization works in practice, let the implicit reward be defined as $\hat{r}_\theta(x, y) = \beta\log\frac{\pi_\theta(y|x)}{\pi_{\mathrm{ref}}(y|x)}$. Derive the gradient of the DPO loss with respect to the parameters θ, i.e., $\nabla_\theta\mathcal{L}_{\mathrm{DPO}}$. Afterwards, interpret the role of the weighting term $\sigma(\hat{r}_\theta(x, y_l) - \hat{r}_\theta(x, y_w))$ in the $\nabla_\theta\mathcal{L}_{\mathrm{DPO}}$. Does the gradient update the model more when it is correct, or when it is incorrect?
Hint: Recall that $\sigma'(z) = \sigma(z)(1 - \sigma(z)) = \sigma(z)\sigma(-z)$.

(g) Throughout this problem, we have worked with the Bradley-Terry model to model human preferences. However, in many real-world scenarios, labelers rank multiple model outputs $y_1, \ldots, y_K$ rather than just comparing pairs. This is modeled by the Plackett-Luce model, which is a generalization of the Bradley-Terry model. In a similar fashion, the PL model stipulates that when presented with a set of possible choices, people prefer a choice with a probability proportional to the value of some latent reward function for that choice. In our context, when presented with a prompt x and a set of K answers $y_1, \ldots, y_K \sim \pi^{\mathrm{SFT}}(y \mid x)$, a user would output a permutation $\tau\colon \{1, \ldots, K\} \to \{1, \ldots, K\}$ indicating the preferred order of the choices. The probability of observing this ranking is given by:

$$p_{\theta^*}(\tau|y_1, \ldots, y_K, x) = \prod_{k=1}^{K} \frac{\exp\Big(r_\phi(x, y_{\tau(k)})\Big)}{\sum_{j=k}^{K}\exp\Big(r_\phi(x, y_{\tau(j)})\Big)}$$

Using the implicit reward parameterization derived in part (d) ($r_\phi(x, y) = \beta\log\frac{\pi_{\theta^*}(y|x)}{\pi_{\mathrm{ref}}(y|x)} + \beta\log Z(x)$), show that the partition function $Z(x)$ cancels out here as well. Write the final Plackett-Luce probability solely in terms of π_{θ^*} and π_{ref}.

 All Rights Reserved. This may not be publicly shared without explicit permission.

Once you've written out that final probability, assuming we have access to a dataset $\mathcal{D} = \{(\tau^{(i)}, x^{(i)}, \{y_j^i\}_{j=1}^K)\}_{i=1}^N$ of prompts and user-specified rankings, we can use a parameterized model and optimize this objective with maximum-likelihood:

$$\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(\tau, y_1, \ldots, y_K, x) \sim \mathcal{D}} \left[\sum_{k=1}^{K} \log \frac{\exp\left(\hat{r}_\theta(x, y_{\tau(k)})\right)}{\sum_{j=k}^{K} \exp\left(\hat{r}_\theta(x, y_{\tau(j)})\right)} \right],$$

with the implicit reward $\hat{r}_\theta(x, y) = \beta \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}$ from part (f). For $K = 2$ this is the DPO loss above.

5. Debugging Generative Models

You are training your first generative model, and you're very excited as your training and validation losses have both dropped to nearly zero. But when you try to generate some samples, only garbage comes out! What is wrong or missing in each of the following implementations, and/or how would you fix it? **Answer briefly. You don't need to give the exact fix in code, just tell us what you need to do.** If you say X is missing, we will understand that the implied fix is to include X.

(a)
```
class CausalAttention(nn.Module):
    def forward(self, x):
        Q = self.W_q(x)  # (batch_size, seq_len, d_model)
        K = self.W_k(x)    # (batch_size, seq_len, d_model)
        V = self.W_v(x)  # (batch_size, seq_len, d_model)
        scores = torch.matmul(Q, K.transpose(-2, -1))
        scores = scores / math.sqrt(self.d_model)
        attention_weights = F.softmax(scores, dim=-1)
        attn_output = torch.matmul(attention_weights, V)
        output = self.W_o(attn_output)
        return output
```

Above, we have what should be a causal attention block used as part of a standard Transformer language model recipe for a GPT-style model pretrained using batches consisting of teacher-forced entire sequences and next-token prediction loss. To sample from the model, we would generate tokens autoregressively. **What is wrong with the above implementation for training?**

(b)
```
class VariationalAutoEncoder(nn.Module):
    def get_loss(self, x):
        # Encode x.
        z_mean, z_log_std = self.encoder(x)

        # Add noise to z.
        noise = torch.randn(z_mean.shape)
        z = z_mean + torch.exp(z_log_std) * noise

        # Decode z back to x, with mean-squared reconstruction loss.
        x_pred = self.decoder(z)
        loss = F.mse_loss(x, x_pred)
        return loss
```

 All Rights Reserved. This may not be publicly shared without explicit permission.

Above, we have what should be the entire loss function for training a variational auto-encoder. To sample from this model, we would sample $z \sim N(0, 1)$, and decode with our learned decoder. **What is wrong with the above implementation for training?**

6. Homework Process and Study Group

Citing sources and collaborators are an important part of life, including being a student!
We also want to understand what resources you find helpful and how much time homework is taking, so we can change things in the future if possible.

(a) **What sources (if any) did you use as you worked through the homework?**

(b) **If you worked with someone on this homework, who did you work with?**
List names and student ID's. (In case of homework party, you can also just describe the group.)

(c) **Roughly how many total hours did you work on this homework?**

Contributors: Gireeja Ranade, Kumar Krishna Agrawal, Jerome Quenum, Anant Sahai, Qiyang Li, Patrick Mendoza, Kevin Frans.

 All Rights Reserved. This may not be publicly shared without explicit permission.