Special Participation A: Gemini Flash on HW12

Jincheng Ou

December 7

I directly uploaded the HW12 file and used this prompt to have Gemini generate the complete LaTeX code for the answer directly.

Part 1: Prompt

User Prompt

You are a deep learning expert. Your goal is to provide solutions and explanations to my deep learning homework 12 (which I have attached) about the written parts (Q1, Q2, Q3), demonstrating intermediate mathematical reasoning. Avoid hallucinating or making up definitions if information is missing; explicitly say so. Answer with clear math notation and concise justification. Do not skip derivation steps. If the problem contains multiple subparts, answer each clearly. Output the final solutions in LaTeX code for me.

Part 2: Output Solutions by Gemini Flash

Gemini correctly identified the bug in line 9 (std=1) and the compounding factor of weight sharing between the embedding and the head.

1. Debugging Transformers

The Bug: The issue lies in the initialization of the embedding weights. The code initializes the weights with a standard normal distribution ($\mu = 0, \sigma = 1$). Since d_{model} is large (as per the hint), the magnitude of the embedding vectors will be very large (expected squared norm $\approx d_{\text{model}}$). Because the language modeling head weights are tied to these embeddings (`self.lm_head.weight = self.emb_word.weight`), the logits produced by the final linear layer will have extremely high variance. This leads to saturated softmax probabilities, vanishing or exploding gradients, and the training instability observed in Model 1.

The Fix: We need to scale the standard deviation of the initialization to account for the large dimensionality. A common practice (similar to Xavier/Glorot initialization) is to use a standard deviation of $1/\sqrt{d_{\text{model}}}$.

Corrected Code (Line 9):

The proposed code change (std=d_model**-0.5) is the standard "Xavier/Glorot" style correction required to stabilize variance.

```
# Original Buggy Line:
# self.emb_word.weight.data.normal_(mean=0, std=1)

# Fixed Line:
self.emb_word.weight.data.normal_(mean=0, std=d_model**-0.5)
```

Why the fix works: By scaling the standard deviation by $1/\sqrt{d_{\text{model}}}$, we ensure that the variance of the dot products in the linear layers (and specifically the tied LM head) remains effectively constant (close to 1) regardless of the model dimension. This prevents the logits from starting with massive values, allowing the optimizer to traverse the loss landscape smoothly from the beginning, resulting in a loss curve similar to Model 2.

---

2. Comparing Distributions

Gemini provided a standard counter-example using Uniform distributions (P~U[0,1], Q~U[0,2]) to prove D_KL(P||Q) < infty while D_KL(Q||P) = infty

(a) Asymmetry of KL Divergence

We need an example where $D_{KL}(P||Q) \neq \infty$ but $D_{KL}(Q||P) = \infty$. This occurs when the support of P is a subset of the support of Q, but the support of Q is *not* a subset of the support of P.

Let P and Q be uniform distributions:

- Let $P \sim \text{Uniform}(0,1)$. Density $P(x) = 1$ for $x \in [0,1]$, else 0.
- Let $Q \sim \text{Uniform}(0,2)$. Density $Q(x) = 0.5$ for $x \in [0,2]$, else 0.

Calculating $D_{KL}(P||Q)$:

$$D_{KL}(P||Q) = \int_{-\infty}^{\infty} P(x) \log\left(\frac{P(x)}{Q(x)}\right) dx = \int_{0}^{1} 1 \cdot \log\left(\frac{1}{0.5}\right) dx = \log(2) \approx 0.693 < \infty$$

Calculating $D_{KL}(Q||P)$:

$$D_{KL}(Q||P) = \int_{-\infty}^{\infty} Q(x) \log\left(\frac{Q(x)}{P(x)}\right) dx$$

The integral includes the range $(1,2]$ where $Q(x) = 0.5$ but $P(x) = 0$. In this region, the term inside the log is $\frac{0.5}{0} \to \infty$. Thus, the integral diverges to infinity.

It correctly articulated the "zero-forcing" vs. "zero-avoiding" nature of the denominators in the KL integral, explaining that Reverse KL penalizes Q heavily if it puts mass where P has none.

(b) Forward vs. Reverse KL

- **Plot A corresponds to Forward-KL ($D_{KL}(P||Q)$).**
  Minimizing Forward-KL requires $P(x) > 0 \implies Q(x) > 0$. If Q drops to zero where P has mass, the penalty is infinite. Therefore, the unimodal Q must stretch to cover **both modes** of the bimodal P (mean-seeking behavior).
- **Plot B corresponds to Reverse-KL ($D_{KL}(Q||P)$).**
  Minimizing Reverse-KL weights the cost by $Q(x)$. If $P(x)$ is near zero, $Q(x)$ must also be near zero to avoid a high cost. Therefore, Q tends to fit just **one mode** of P sharply and ignore the other to avoid the low-probability valley between them (mode-seeking behavior).

---

3. Variational Information Bottleneck

(a) Reparameterization Trick Diagram

To sample $Z \sim \mathcal{N}(\mu, \sigma^2)$ using the reparameterization trick, we use an auxiliary noise variable $V \sim \mathcal{N}(0,1)$. The operation is:

$$Z = \mu + \sigma \odot V$$

Block Diagram Description:

1. **Inputs:** Encoder outputs μ and σ; Random noise generator output V.
2. **Operation 1 (Multiplier):** Compute product $\sigma \times V$.
3. **Operation 2 (Adder):** Compute sum $\mu + (\sigma \times V)$.
4. **Output:** Latent variable Z.

Gemini correctly described the Reparameterization Trick. While it provided a text description of the operations (Multiplier and Adder) rather than drawing the image, the description describes the correct circuit required for backpropagation.

The model correctly identified that the **Encoder** receives gradients from both the Task Loss and the Latent Regularizer (via the reparameterization trick), while the **Decoder** only receives gradients from the Task Loss.

(b) Loss Terms Contributing to Gradients

The total loss is $\mathcal{L} = \text{Task Loss} + \beta \cdot \text{KL Regularizer}$.

$$\mathcal{L} = -\log q_{\theta_d}(y|z) + \beta D_{KL}(p_{\theta_e}(z|x)||r(z))$$

- **Encoder parameters θ_e:** These parameters determine μ and σ.
  - They affect the **KL Regularizer** directly (as it depends on μ, σ).
  - They affect the **Task Loss** indirectly because z is sampled using μ and σ (via the reparameterization trick), and backpropagation flows through z to θ_e.

  Gradient contributors: Task Loss AND Latent Regularizer.

- **Decoder parameters θ_d:** These parameters are only present in the likelihood function $q_{\theta_d}(y|z)$.
  - They affect the **Task Loss**.
  - They do **not** affect the KL Regularizer (which depends only on the encoder and the fixed prior).

  Gradient contributors: Task Loss only.

It correctly identified the trade-off: low beta causes overfitting (low bottleneck), and high beta causes underfitting (posterior collapse to prior), meaning the optimal validation error is in the middle.

(c) Validation Error Trend

Selection: Curve (b)

Reasoning:

- **Small β:** The model overfits (low training error, high validation error) because the latent space is not regularized enough.
- **Large β:** The model underfits (high validation error) because the information bottleneck is too tight, forcing Z to be noise ($Z \sim \mathcal{N}(0, 1)$) and ignoring the input X.
- **Intermediate β:** There is an optimal "sweet spot" where the model learns useful features while generalizing well. This results in a U-shaped curve (or V-shaped) for validation error, which matches Curve (b).

(d) Latent Space Visualization

i. Guessing β values: The parameter β controls how close the posterior is to the unit Gaussian prior $\mathcal{N}(0, 1)$.

- **Figure (b):** The clusters are very spread out (axis scale ≈ 6). This implies weak regularization. **Result: $\beta = 10^{-6}$.**
- **Figure (c):** The points are collapsed tightly around 0 (axis scale ≈ 0.15). This implies extremely strong regularization. **Result: $\beta = 10^{0}$.**
- **Figure (a):** The points form distinct clusters but are reasonably compact (axis scale ≈ 1.5). This implies balanced regularization. **Result: $\beta = 10^{-3}$.**

ii. Better Latent Space: Selection: Figure (a) ($\beta = 10^{-3}$)

Reasoning: Figure (a) represents the best trade-off. The classes are clearly separated (unlike the collapsed state in Figure c) but the variance is controlled (unlike the spread out state in Figure b). This structure facilitates better generalization for the downstream prediction task.

Gemini correctly estimated the beta values by looking at the axis scales and cluster tightness. It identified Plot (b) as beta=10^-6 (spread out), Plot (c) as beta=10^0 (collapsed), and Plot (a) as the balanced beta=10^-3

Part 3: Executive Summary

Gemini Flash demonstrated a perfect one-shot performance on the non-coding written portions of Homework 12. The model exhibited strong domain knowledge in Deep Learning, specifically regarding Transformer initialization stability, KL Divergence, and Variational Information Bottlenecks.

Key Observations:

- **Accuracy:** The model answered all sub-parts of the three questions correctly, matching the logic found in the reference solutions.
- **Reasoning Capability:** Gemini Flash went beyond simple answers by providing mathematical justification. For instance, in Q1, it explicitly reasoned about the expected squared norm of the embeddings, and in Q3, it correctly interpreted the trade-off between task loss and regularization strength to analyze the validation error curve.
- **Visual Interpretation:** The model successfully interpreted the unlabelled plots in Q2 (distribution shapes) and Q3 (scatter plots of latent spaces), correctly mapping visual characteristics (variance, clustering) to hyperparameters.
- **Hallucinations:** There were no hallucinations or misunderstoods found in this output. The model adhered strictly to standard definitions and deep learning theory.