

# Special Participation A

## Special Participation A

### Question 1

**Me:** Copied the text of the question and code. Included a picture of just the two graphs.

**Gemini 3 Pro:**

### The bug

The embedding weights (and, due to weight tying, the output head weights) are initialized with a standard deviation of 1. Because the Transformer's hidden dimension (`d_model`) is typically large (e.g., 512 or 768), the dot product operation in the final linear layer sums up hundreds of values with high variance. This results in output logits with extremely large magnitudes, causing the Softmax function to produce a highly "peaked" (confident but wrong) distribution rather than a uniform distribution at the start of training. This leads to the massive initial loss seen in Model 1.

### The fix (code)

You need to scale the standard deviation by the inverse square root of `d_model` (often called Xavier or Glorot initialization scaling).

Python

```
# Change line 9 to:  
self.emb_word.weight.data.normal_(mean=0, std=d_model**-0.5)
```

### Why the fix should work

The final projection (the `lm_head`) calculates a dot product between the decoder output and the embedding weights. The variance of this sum grows linearly with the number of elements being summed (`d_model`).

By scaling the weights by  $\frac{1}{\sqrt{d_{model}}}$ , we ensure that the variance of the output logits remains close to 1 (unit variance). This keeps the logits in a reasonable range (e.g., -2 to 2), ensuring the Softmax output is roughly uniform across the vocabulary at initialization. A uniform prediction results in an initial loss of  $\ln(\text{vocab\_size})$  (roughly 10-11 for standard vocabulary sizes), which matches your friend's loss curve.

Me: This looks like the correct change.

## Question 2

Me: Copied the text of the question and math in latex form. Included a picture of just the two graphs.

Based on the provided image and problem statement, here are the answers.

Gemini 3 Pro:

### Part (a): Example of Asymmetric Infinity

The Kullback-Leibler divergence  $D_{KL}(P||Q)$  blows up to infinity if the support of distribution  $P$  is not contained within the support of distribution  $Q$ . In simpler terms, if there is any region where  $P(x) > 0$  but  $Q(x) = 0$ , the ratio inside the logarithm ( $\frac{P(x)}{Q(x)}$ ) becomes infinite, causing the divergence to be infinite.

To satisfy the condition where forward KL is finite but reverse KL is infinite, we need  **$P$ 's support to be inside  $Q$ 's support**, but  **$Q$  to have support where  $P$  is zero**.

Example:

Let  $P$  and  $Q$  be Uniform distributions:

- Let  $P = \text{Uniform}(0, 1)$ . It has density 1 for  $x \in [0, 1]$  and 0 elsewhere.
- Let  $Q = \text{Uniform}(0, 2)$ . It has density 0.5 for  $x \in [0, 2]$  and 0 elsewhere.

1. Calculate Forward KL  $D_{KL}(P||Q)$ :

The integral is only evaluated where  $P(x) > 0$  (the interval  $[0, 1]$ ). In this region,  $Q(x) = 0.5$ , which is non-zero.

$$D_{KL}(P||Q) = \int_0^1 1 \cdot \log\left(\frac{1}{0.5}\right) dx = \log(2) \approx 0.693$$

Result: Finite.

## 2. Calculate Reverse KL $D_{KL}(Q||P)$ :

The integral is evaluated where  $Q(x) > 0$  (the interval  $[0, 2]$ ).

Consider the region  $x \in (1, 2]$ . Here,  $Q(x) = 0.5$  but  $P(x) = 0$ .

The term inside the integral becomes  $0.5 \cdot \log\left(\frac{0.5}{0}\right)$ . Since division by zero approaches infinity:

Result:  $\infty$ .

---

## Part (b): Identify Plots for Forward vs. Reverse KL

- **Plot A:** Corresponds to minimizing **Forward KL ( $D_{KL}(P||Q)$ )**.
- **Plot B:** Corresponds to minimizing **Reverse KL ( $D_{KL}(Q||P)$ )**.

### Reasoning:

#### 1. Forward KL (Plot A) is "Zero-Avoiding" (Mass-Covering):

When minimizing  $D_{KL}(P||Q) = \mathbb{E}_{x \sim P}[\log P(x) - \log Q(x)]$ , we are effectively maximizing  $\mathbb{E}_{x \sim P}[\log Q(x)]$ . This means that wherever  $P(x)$  is high (has probability mass),  $Q(x)$  must also be high. If  $P(x) > 0$  and  $Q(x)$  is near zero, the penalty is massive.

- **Visual Evidence:** In Plot A, the approximation  $Q$  (beige) stretches out to cover **both** modes of the target  $P$  (blue). To avoid being zero where  $P$  is non-zero, the Gaussian  $Q$  accepts a wider variance and centers itself between the two modes. This is often called "moment matching."

#### 2. Reverse KL (Plot B) is "Zero-Forcing" (Mode-Seeking):

When minimizing  $D_{KL}(Q||P) = \mathbb{E}_{x \sim Q}[\log Q(x) - \log P(x)]$ , the term  $-\log P(x)$  acts as a cost function. If  $Q(x)$  places any mass where  $P(x)$  is near zero (like the valley between the two peaks), the cost explodes.

- **Visual Evidence:** To minimize this cost,  $Q$  "squeezes" itself into an area where  $P$  is high and strictly avoids areas where  $P$  is low. Consequently,  $Q$  locks onto a single mode (peak) of  $P$  and ignores the other to avoid crossing the low-probability valley. This is why Plot B shows  $Q$  fitting just one of the peaks perfectly.

## Question 3

**Me:** Copied the text of the question and math in latex form. Included a picture of just the graph. First put in parts a and b.

**Gemini 3 Pro:**

Based on the description of the Variational Information Bottleneck (VIB) and the standard mechanics of Variational Auto-Encoders (VAEs), here are the solutions to the problems.

### Part (a): Reparameterization Trick Block Diagram

To allow backpropagation through the random sampling step, we use the **reparameterization trick**. Instead of sampling  $Z$  directly from a distribution parameterized by the encoder, we express  $Z$  as a deterministic function of the parameters  $(\mu, \sigma)$  and an independent source of noise ( $V$ ).

The equation for the reparameterization is:

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

**Block Diagram:**

Code snippet

```
graph LR
    V((V)) --> Mult[x Multiplier]
    sigma[sigma] --> Mult
    Mult --> Add[+ Adder]
```

```

mu[μ] --> Add
Add --> Z((Z))

```

### Description of the flow:

1. **Inputs:** You have the standard Gaussian noise  $V$  and the standard deviation  $\sigma$  (from the encoder).
  2. **Multiplier:** These two enter a multiplier node to compute the "noise scaled by variance" ( $\sigma \cdot V$ ).
  3. **Adder:** The output of the multiplier enters an adder node along with the mean  $\mu$  (from the encoder).
  4. **Output:** The result of the addition is the latent variable  $Z$ .
- 

## Part (b): Gradient Contributions

We need to determine which parameters are updated by which parts of the loss function.

The total Loss  $L$  is:

$$L = \underbrace{\frac{1}{N} \sum E[-\log q_{\theta_d}(y_n|z)]}_{\text{Task Loss}} + \underbrace{\beta D_{KL}(p_{\theta_e}(Z|x_n)||r(Z))}_{\text{Latent Regularizer}}$$

1. For encoder parameters  $\theta_e$ :

The encoder parameters determine  $\mu$  and  $\sigma$ .

- **Latent Regularizer:** The KL divergence term explicitly compares the distribution defined by  $\theta_e$  (via  $\mu$  and  $\sigma$ ) to the prior  $r(Z)$ . Therefore, gradients flow directly from this term to  $\theta_e$ .
- **Task Loss:** Because of the reparameterization trick ( $Z = \mu_{\theta_e} + \sigma_{\theta_e} \cdot V$ ), the value of  $z$  fed into the decoder depends on  $\theta_e$ . To minimize the classification error, the encoder must produce "good"  $z$  coordinates. Gradients flow from the output, through the decoder, back through  $z$ , and finally to  $\mu$  and  $\sigma$  (and thus  $\theta_e$ ).

Selection:

For encoder parameters  $\theta_e$ : [x] task loss [x] latent regularizer

## 2. For decoder parameters $\theta_d$ :

The decoder parameters are used to calculate the likelihood of  $y$  given  $z$ .

- **Task Loss:** The term  $-\log q_{\theta_d}(y_n|z)$  explicitly contains the decoder parameters  $\theta_d$ . Minimizing this term trains the decoder to classify correctly.
- **Latent Regularizer:** The KL term is  $D_{KL}(p_{\theta_e}||r)$ . It compares the encoder's output to a fixed Gaussian prior. The decoder  $\theta_d$  does not appear in this formula at all. Therefore, it contributes zero gradient to the decoder.

Selection:

For decoder parameters  $\theta_d$ : [x] task loss [ ] latent regularizer

**Me:** Then put in parts c and d.

## Gemini 3 Pro:

Based on the principles of the Variational Information Bottleneck (VIB) and the provided graphs, here are the answers.

## Part 1: Validation Error Trend (Figure 3)

### Correct Curve: (b)

Reasoning:

The parameter  $\beta$  controls the trade-off between the Prediction Task (minimizing classification error) and the Bottleneck Constraint (compressing the representation  $Z$  to look like the Gaussian prior).

- **Low  $\beta$  (Left side of x-axis):** The model focuses almost entirely on maximizing prediction accuracy. It behaves like a standard neural network. While accuracy is good, it may slightly overfit the training data.
- **Intermediate  $\beta$  (Middle):** As  $\beta$  increases, the term acts as a regularizer. It forces the model to ignore "nuisance" information in the input  $X$  that isn't necessary for the target  $Y$ . This improves generalization, causing the validation error on held-out data to **decrease** (the "dip" in curve b).
- **High  $\beta$  (Right side of x-axis):** If  $\beta$  becomes too large, the regularization term dominates. The model is forced to compress the latent variable  $Z$  so much that it

loses information required to classify the digits.  $Z$  collapses to the prior (noise), and the validation error **spikes drastically** as the model effectively starts guessing.

Curve **(b)** is the only one showing this characteristic regularization behavior: a "sweet spot" of improved performance followed by a collapse when the bottleneck becomes too tight.

---

## Part 2: Latent Space Embeddings (Figure 4)

*Note: Figure 4 was not included in the uploaded images, but based on VIB theory, I can guide you to the correct identification of the plots.*

### i. Guessing the values of $\beta$ :

You need to look at the "spread" and "clustering" of the points in the scatter plots.

#### 1. $\beta = 100$ (High Regularization):

- **Visual Characteristic:** All data points (regardless of class) will be squashed into a single, dense blob centered at  $(0, 0)$ . The classes will likely overlap significantly because the encoder is forced to match the standard Gaussian prior  $\mathcal{N}(0, 1)$  almost perfectly, destroying class information.
- **Matches Graph:** The one that looks like a round blob/noise.

#### 2. $\beta = 10^{-6}$ (Low Regularization):

- **Visual Characteristic:** The points will likely be spread out widely to maximize the distance between classes. The clusters might be far from the origin or have irregular shapes, as the constraint to look like a Gaussian is very weak.
- **Matches Graph:** The one with the most spread out or "exploded" clusters.

#### 3. $\beta = 10^{-3}$ (Balanced/Sweet Spot):

- **Visual Characteristic:** You will see distinct, well-separated clusters (good for classification) that are nevertheless compact and centered around the origin (constrained by the prior).
- **Matches Graph:** The one that looks structured but compact.

### ii. Which results in a better latent space for the prediction task?

**Answer:** The experiment corresponding to  $\beta = 10^{-3}$ .

Reasoning:

This choice corresponds to the "sweet spot" identified in Figure 3(b) where the validation error is lowest.

- At  $\beta = 100$ , the information is lost (collapsed), so prediction is poor.
- At  $\beta = 10^{-6}$ , the space is good for the training data but might be less robust (overfitted).
- At  $\beta = 10^{-3}$ , the VIB has successfully compressed the input to the most "useful" information, creating a latent space that generalizes best to new data.