

Can you thoroughly answer question 1 as well?

Solution to Problem 1: Debugging Transformers

The Bug (Brief Description)

The bug is in the initialization of the word embedding weights (line 9), which are set to a normal distribution with standard deviation 1. This is problematic because `d_model` is large in transformer models (typically 512 or more), leading to embedding vectors with excessively large magnitudes. Since the language model head (`lm_head`) shares these weights (line 23), this causes the output logits to have high variance, resulting in a high initial cross-entropy loss and unstable training, preventing the model from converging properly.

The Fix (Code)

Change line 9 (and optionally line 10 for consistency, though the head is primarily affected by line 9):

```
text  
  
self.emb_word.weight.data.normal_(mean=0, std=d_model ** -0.5)  
self.emb_pos.weight.data.normal_(mean=0, std=d_model ** -0.5)
```

Why the Fix Should Work (Brief Explanation)

With $\text{std}=1$, the expected norm of each embedding vector is approximately $\sqrt{d_{\text{model}}}$, which is large. When computing logits as the dot product between the hidden states (post-layer norm, with unit variance per dimension) and the embedding weights, the variance of each logit becomes approximately $d_{\text{model}} * 1 * 1 = d_{\text{model}}$, leading to logits with $\text{std} \approx \sqrt{d_{\text{model}}}$ (very large). This causes an excessively high initial cross-entropy loss (far above $\log(n_{\text{words}})$) and gradient instability. Initializing with $\text{std}=1/\sqrt{d_{\text{model}}}$ ensures embedding norms are around 1, keeping logit variance around 1, which stabilizes training and allows the loss to decrease effectively, as seen in your friend's model. This is a standard practice (similar to Xavier/Glorot initialization) to maintain signal variance across layers in deep models with large dimensions.

→ This is the critical insight... it isn't just knowing that $\text{std}=1$ is bad, but rather noticing that the line $[self.lm_head.weight = self.emb_word.weight]$.

This matters b/c if the embeddings were only used at the input, the subsequent LayerNorm would likely normalize the signal, dampening the impact of bad initialization.

Grok recognized that because the weights are tied to the output head, the raw, un-normalized large weights are used in the final dot product. This directly causes the logits to explode, which LayerNorm cannot save.

Can you provide a detailed solution for problem 2 in the attached file?

→ This is Grok's response to part 2(a). Grok suggested Uniform distributions $P = U[0,1]$ and $Q = U[0,2]$.

↳ This might be pattern retrieval, as $U[0,1]$ and $U[0,2]$ seem to be standard textbook counter-example to KL-symmetry.

Grok correctly verified the "Support condition". This shows it understands the boundary conditions of the integrals, not just the formula.

Thought: The model might be stronger when it can anchor the problem to a standard pedagogical example. If I had maybe asked for a non-standard distribution like a mixture of Laplacians, it might have struggled more. It very well could've succeeded here b/c the problem could've perfectly matched to its training data.

The model correctly linked the mathematical definition of KL divergence to its geometric behavior. It correctly explained that minimizing Forward KL ($P \parallel Q$) forces Q to 'cover' P , while Reverse KL ($Q \parallel P$) allows Q to focus on a single mode to avoid high penalties where $P \approx 0$.

Solution to Problem 2: Comparing Distributions

Part (a): Example of Univariate Distributions Where $D_{KL}(P\|Q) \neq \infty$ but $D_{KL}(Q\|P) = \infty$

The Kullback-Leibler (KL) divergence is defined as:

$$D_{KL}(P\|Q) = \mathbb{E}_{x \sim P(x)} \left[\log \frac{P(x)}{Q(x)} \right] = \int P(x) \log \frac{P(x)}{Q(x)} dx$$

(for continuous distributions; a similar form holds for discrete ones).

This integral diverges to infinity if there exists any x where $P(x) > 0$ but $Q(x) = 0$ (since $\log(P(x)/Q(x)) \rightarrow \infty$ as $Q(x) \rightarrow 0$). However, it can remain finite even if $Q(x) > P(x)$ in some regions, as long as $Q(x) > 0$ wherever $P(x) > 0$ (i.e., the support of P is contained within the support of Q).

For the reverse, $D_{KL}(Q\|P)$ diverges if there is any x where $Q(x) > 0$ but $P(x) = 0$.

To satisfy the condition $D_{KL}(P\|Q) < \infty$ but $D_{KL}(Q\|P) = \infty$, we need distributions where the support of P is a **proper subset** of the support of Q (so Q has "extra" support where $P = 0$, causing the reverse KL to blow up, but P 's support is fully covered by Q , keeping the forward KL finite).

Example:

Let P be the uniform distribution on $[0, 1]$: $P(x) = 1$ for $x \in [0, 1]$, and 0 otherwise.

Let Q be the uniform distribution on $[0, 2]$: $Q(x) = 1/2$ for $x \in [0, 2]$, and 0 otherwise.

Computation of $D_{KL}(P\|Q)$:

$$D_{KL}(P\|Q) = \int_0^1 1 \cdot \log \frac{1}{1/2} dx = \int_0^1 \log 2 dx = \log 2 \approx 0.693 < \infty.$$

This is finite because $Q(x) = 1/2 > 0$ everywhere that $P(x) > 0$ (i.e., on $[0, 1]$).

Computation of $D_{KL}(Q\|P)$:

$$D_{KL}(Q\|P) = \int_0^2 \frac{1}{2} \log \frac{1/2}{P(x)} dx.$$

On $[0, 1]$, $P(x) = 1$, so the integrand is $(1/2) \log(1/2)$. But on $(1, 2]$, $P(x) = 0$, so $\log((1/2)/P(x)) \rightarrow \infty$, and the integral over a positive-length interval diverges: $D_{KL}(Q\|P) = \infty$.

This example works because Q assigns positive probability to regions where P has zero density (the interval $(1, 2]$), causing the reverse KL to infinite, while the forward KL is finite.

(Other examples could work, e.g., P as a Dirac delta at 0 and Q as a Gaussian centered at 0 with variance 1, but the uniform example is simple and univariate as requested.)

Part (b): Identifying Which Plots Correspond to Minimizing Forward vs. Reverse KL

Background on Forward vs. Reverse KL Minimization:

Forward KL ($D_{KL}(P\|Q)$): Minimizing this over Q (here, parameterized as a univariate Gaussian with variable mean and std. dev.) encourages Q to **cover all modes of P** . Why? The penalty is high if $Q(x)$ is small where $P(x)$ is large (since $P(x) \log(P(x)/Q(x))$ blows up when $Q(x) \rightarrow 0$ in high- P regions). Thus, Q spreads out to avoid underestimating P 's mass in any region. This is "mode-covering" or "zero-avoiding" behavior.

Reverse KL ($D_{KL}(Q\|P)$): Minimizing this encourages Q to **seek a single mode of P** . Why? The penalty is high if $Q(x)$ is large where $P(x)$ is small (since $Q(x) \log(Q(x)/P(x))$ blows up when $P(x) \rightarrow 0$ in high- Q regions). Thus, Q concentrates on one high-density region of P to avoid "wasting" mass where P is low. This is "mode-seeking" or "zero-forcing" behavior.

The target P is bimodal (two peaks, like a mixture of two Gaussians). We're fitting Q as a single Gaussian.

Identification Based on the Plots:

Plot A: Q is a broad Gaussian centered between the two modes of P , with high variance to cover both peaks (though not perfectly matching either). This spreads out to ensure no part of P 's mass is in a low- Q region.

This corresponds to minimizing **forward KL ($D_{KL}(P\|Q)$)**: Mode-covering behavior.

Plot B: Q is a narrow Gaussian tightly fitting one of the modes of P , ignoring the other mode entirely.

This corresponds to minimizing **reverse KL ($D_{KL}(Q\|P)$)**: Mode-seeking behavior.

Brief Reasoning:

In forward KL minimization, Q must assign probability everywhere P does to avoid infinite penalties, leading to a wide, averaging distribution (Plot A).

In reverse KL minimization, Q avoids assigning probability where P is low, so it focuses sharply on one mode (Plot B), as spreading to the other mode would incur high cost in the low-density valley between modes.

(If the plots are swapped in your view, the key is: broad/averaging Q = forward KL; sharp/single-mode Q = reverse KL.)

Now can you provide a detailed solution for problem 3?

Solution to Problem 3: Variational Information Bottleneck

Part (a): Block Diagram for Obtaining Z from μ , σ , and V

The encoder network (parameterized by θ_e) outputs the mean μ and standard deviation σ (since variance σ^2 , but we use σ for the reparameterization) as functions of the input x . The independent randomness V is drawn from a standard normal distribution $N(0, 1)$. To sample Z from the conditional Gaussian $p_{\theta_e}(Z|x) = N(Z | \mu, \sigma^2)$, we use the reparameterization trick to make the sampling differentiable for backpropagation.

The formula is:

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

This ensures that Z is a scalar Gaussian with the desired mean and variance, and gradients can flow through μ and σ during training (while V is fixed noise).

Block Diagram:

Input: V (from $N(0,1)$)

Multiplier: Multiply V by σ (scalar multiplication)

Adder: Add μ to the result of the multiplication

Output: Z

Visually, this can be represented as:

This block diagram correctly described
the data flow of the reparameterization
trick.

text

$V \rightarrow [Multiply \ by \ \sigma] \rightarrow [Add \ \mu] \rightarrow Z$

(Where arrows denote data flow, the multiplier is a \otimes symbol or equivalent, and the adder is a $+$ symbol.)

This structure allows the randomness to be external (V), while the learnable parameters (μ, σ from the encoder) control the distribution, enabling end-to-end differentiation.

Part (b): Loss Terms Contributing to Gradients for Encoder and Decoder Parameters

The overall loss to minimize is:

$$L = \frac{1}{N} \sum_i^N \mathbb{E}_{z \sim p_{\theta_e}(z|x_n)} [-\log q_{\theta_d}(y_n|z)] + \beta \cdot D_{KL}(p_{\theta_e}(Z|x_n)||r(Z))$$

The **task loss** term ($-\log q_{\theta_d}(y_n|z)$) depends on z (sampled via the encoder) and then passed through the decoder to predict y_n . During backpropagation (via SGD), gradients from this term flow through the decoder network (to θ_d) and then back through z to the encoder network (to θ_e), since z is a function of θ_e via the reparameterization.

The **latent regularizer** term ($\beta * D_{KL}$) is the KL divergence between the encoder's posterior $p_{\theta_e}(Z|x_n)$ and the prior $r(Z) = N(0,1)$. This term only involves the parameters of the encoder (μ and σ from θ_e) and does not depend on the decoder or y_n . Thus, its gradients only update θ_e .

Since we treat V as external input and draw fresh V each time, the stochasticity is handled correctly for SGD.

Selections:

For encoder parameters θ_e :

task loss

latent regularizer

For decoder parameters θ_d :

task loss

latent regularizer

Part (c): Curve Representing Validation Error with Increasing β

The regularization strength β controls the trade-off between the task loss (which encourages Z to carry useful information from X to predict Y) and the KL regularizer (which pushes the latent distribution $p_{\theta_e}(Z|x)$ close to the uninformative prior $N(0,1)$).

Grok correctly analyzed the computational graph. It recognized that the encoder parameters θ_e determine the latent z , which feeds into both the KL term and the Decoder, so it receives gradients from both loss terms.

At $\beta \approx 0$ (weak regularization), the model prioritizes minimizing the task loss, allowing Z to encode rich information about X , leading to low validation error (good generalization if not overfitting).

As β increases, the KL term dominates, forcing Z to be nearly independent of X (close to $N(0,1)$), which discards task-relevant information from X , degrading performance and increasing validation error.

This results in a monotonically increasing trend in validation error with β , as more regularization strips away useful information.

From Figure 3 (validation error profiles for different β values):

- (a) Shows error starting low and increasing with β (matches the expected trend).
- (b) Shows decreasing error (opposite, as if more regularization helps, which it doesn't here).
- (c) Shows a U-shape (error decreases then increases, like under/over-regularization, but in VIB for this setup, it's not typical since $\beta=0$ is already "optimal" for task but risks overfitting; however, the problem implies held-out validation, and the trend is increasing).
- (d) Shows an inverted U (error increases then decreases, not matching).

Thus, the best representation is:

- (a)

Part (d): Guessing β Values and Better Latent Space

Figure 4 shows 2D latent embeddings for MNIST classification (three digits, so three class labels with different symbols) under different β values. The hint emphasizes axis labels for scale, which indicate the spread/variance of the latents (smaller scale means more collapse due to stronger KL regularization).

Small β (e.g., 10^{-6}): Weak regularization allows Z to capture rich, task-relevant structure from X , leading to well-separated clusters by class (good separability for classification). The latents spread out widely (large axis scale, e.g., -10 to 10).

Medium β (e.g., 10^{-3}): Moderate regularization compresses the space somewhat, with partial separation but more overlap (medium scale, e.g., -0.5 to 0.5).

Large β (e.g., 100): Strong regularization forces Z close to $N(0,1)$ regardless of X , collapsing points to the origin with minimal separation (tiny scale, e.g., -0.01 to 0.01), losing class information.

Assuming the figures are labeled (a), (b), (c) from left to right or as described:

- (a): Wide spread, separated clusters \rightarrow small β .
- (b): Medium spread, some overlap \rightarrow medium β .
- (c): Collapsed, mixed \rightarrow large β .

i. Guesses:

- (a) $\beta = 10^{-6}$ $\beta = 10^{-3}$ $\beta = 100$
- (b) $\beta = 10^{-6}$ $\beta = 10^{-3}$ $\beta = 100$
- (c) $\beta = 10^{-6}$ $\beta = 10^{-3}$ $\beta = 100$

ii. Better latent space for prediction: The one with clear class separation (linearly separable clusters), as it preserves more task-relevant information for the decoder to classify accurately. This is the small β case.

- (a) (b) (c)

for (c) and (d), Grok showed strong intuition for the VIB objective

$L = \text{TaskLoss} + \beta \cdot I(Z; X)$. It correctly reasoned that low β results in a representation close to the data while high β forces the posterior to match the prior.

Now can you answer question 5 parts (a) and (b) as they are written parts of the homework?

Solution to Problem 5: Meta-learning for Learning 1D Functions

Part (a)

We have one training point (x, y) where $y = \phi_t(x)$, and $\phi_t(x) = \phi_a(x) = \phi(x)$. For simplicity, assume $\phi(x) = 1$ and $y = 1$ (this is consistent with the normalization and the form given in the problem, as the scaling generalizes).

The learned function is $f_{\hat{\beta}, c}(x) = \hat{\beta}_0 c_0 \phi_t(x) + \hat{\beta}_1 c_1 \phi_a(x)$, but since $\phi_t(x) = \phi_a(x) = 1$, the constraint for minimal norm regression is:

$$\hat{\beta}_0 c_0 + \hat{\beta}_1 c_1 = 1$$

We minimize $\|\hat{\beta}\|_2^2 = \hat{\beta}_0^2 + \hat{\beta}_1^2$ subject to this constraint.

This is equivalent to solving $\mathbf{c}^T \hat{\beta} = 1$ where $\mathbf{c} = [c_0, c_1]^T$. The minimum-norm solution is:

$$\hat{\beta} = \frac{\mathbf{c}}{\|\mathbf{c}\|^2} = \frac{1}{c_0^2 + c_1^2} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix}$$

This matches the required form.

Part (b)

The expected test error is:

$$L = \mathbb{E}_{x_{\text{test}}, y_{\text{test}}} \left[\frac{1}{2} \left(y - \hat{\beta}_0 c_0 \phi_t(x_{\text{test}}) - \hat{\beta}_1 c_1 \phi_a(x_{\text{test}}) \right)^2 \right]$$

where $y_{\text{test}} = \phi_t(x_{\text{test}})$, and $\hat{\beta}_0 = \frac{c_0}{D}$, $\hat{\beta}_1 = \frac{c_1}{D}$ with $D = c_0^2 + c_1^2$.

Using orthonormality ($\mathbb{E}[\phi_t^2] = 1$, $\mathbb{E}[\phi_a^2] = 1$, $\mathbb{E}[\phi_t \phi_a] = 0$):

$$\mathbb{E}[(y - \text{pred})^2] = 1 - 2\hat{\beta}_0 c_0 + (\hat{\beta}_0 c_0)^2 + (\hat{\beta}_1 c_1)^2 = 1 - 2\frac{c_0^2}{D} + \left(\frac{c_0^2}{D}\right)^2 + \left(\frac{c_1^2}{D}\right)^2$$

Let $r = \frac{c_0^2}{D}$, so $1 - r = \frac{c_1^2}{D}$. Then:

$$\mathbb{E}[(y - \text{pred})^2] = 2 - 4r + 2r^2 \implies L = 1 - 2r + r^2 = (1 - r)^2 = \left(\frac{c_1^2}{D}\right)^2$$

The gradients are:

$$\frac{\partial L}{\partial c_0} = -\frac{4c_0 c_1^4}{(c_0^2 + c_1^2)^3}, \quad \frac{\partial L}{\partial c_1} = \frac{4c_0^2 c_1^3}{(c_0^2 + c_1^2)^3}$$

Grok correctly solved the constrained optimization problem for minimum norm regression.

It correctly identified the constraints $\hat{\beta}_0 c_0 + \hat{\beta}_1 c_1 = 1$ and used the standard pseudo-inverse/projection logic to find the weights are proportional to feature strengths c .

Solves the problem perfectly!