Quiz 5

601.471/671 NLP: Self-Supervised Models

Johns Hopkins University Fall 2026

Name: _____ **Instructor version**

Total: 100 required points + 15 optional bonus points.

Time: 20 minutes.

Coverage: normalization, KV caching, and programming with padded-batch losses. Write short answers and show the requested calculations. One or two sentences suffice for each explanation. No calculator is needed. Bonus parts are optional: leaving them blank costs no required points; bonus scores are added beyond 100. The time limit includes bonus work.

Q1. Normalization *40 + 5 bonus points*

(a) **20 required points + 5 bonus points.** Consider two sublayer layouts, where N is LayerNorm and S is a differentiable sublayer. Both map a token vector x to a vector of the same width:

$$\text{A: } y = N(x + S(x)), \qquad \text{B: } y = x + S(N(x)).$$

(i) **4 points.** Label A and B as pre-norm or post-norm.

Solution: A is **post-norm**: normalization follows the residual addition. B is **pre-norm**: normalization occurs inside the sublayer branch.
Grading: 2 points per correct label.

(ii) **6 points.** For layout A, set $r = x + S(x)$, so $y = N(r)$. Assume the local Jacobians $J_S(u) = \partial S(u)/\partial u$ and $J_N(u) = \partial N(u)/\partial u$ are given; rows index outputs and columns index inputs. Use column gradients and leave these Jacobians symbolic. Given $g_y = \nabla_y \ell$ for a scalar loss ℓ, what gradient $g_r = \nabla_r \ell$ reaches r through N?

Solution: Since $y = N(r)$, backpropagation through this local map gives

$$g_r = J_N(r)^\top g_y = J_N(x + S(x))^\top g_y.$$

The transpose maps the upstream column gradient to the input coordinates of N; the Jacobian is evaluated at r.
Grading: 6 points for the correct local backward expression.

(iii) **6 points.** Still in A, use g_r from (ii) to write the two contributions to $g_x = \nabla_x \ell$: one through the direct x term in r, and one through $S(x)$. Add them to obtain g_x.

Solution: The direct path from x to $r = x + S(x)$ contributes g_r. The path through S contributes $J_S(x)^\top g_r$. Add the contributions:

$$g_x^{\text{A}} = g_r + J_S(x)^\top g_r = \big(I + J_S(x)\big)^\top J_N(x + S(x))^\top g_y,$$

where I is the identity matrix of the hidden width.
Grading: 2 points per path contribution and 2 for their sum. Accept correct follow-through using an earlier expression for g_r.

(iv) **Optional bonus: 5 points.** For layout B, set $n = N(x)$, so $y = x + S(n)$. First write $g_n = \nabla_n \ell$ in terms of g_y (2 bonus points). Then combine the direct residual contribution and the contribution through N to obtain g_x (3 bonus points). Indicate the Jacobians' evaluation inputs.

Solution: For $n = N(x)$ and $y = x + S(n)$, the branch receives upstream gradient g_y:

$$g_n = J_S(n)^\top g_y.$$

The direct residual path contributes g_y to x, and the normalization branch contributes $J_N(x)^\top g_n$. Thus

$$g_x^{\text{B}} = g_y + J_N(x)^\top g_n = g_y + J_N(x)^\top J_S(N(x))^\top g_y.$$

The sublayer Jacobian is evaluated at $n = N(x)$; the normalization Jacobian is evaluated at x.
Grading: 2 bonus points for g_n and 3 for the summed g_x. Accept equivalent expressions and consistent local backward steps.

(v) **4 points.** Which layout has a direct residual path outside normalization, and what contribution does it make to g_x? Answer from the block formulas; the bonus derivation is not required.

Solution: B, shown by its standalone additive g_y term in $g_x^{\mathrm{B}} = g_y + J_N(x)^\top J_S(N(x))^\top g_y$. This term bypasses normalization. In A, the direct residual contribution is instead $J_N(x + S(x))^\top g_y$. B's branch contribution can still cancel the direct term; the identity path does not guarantee a nonzero total gradient.
Grading: 2 points for B and 2 for its direct g_y term.

(b) **20 points.** Compare affine LayerNorm and bias-free RMSNorm on one token $x \in \mathbb{R}^D$. Normalize over its D features, using population variance and fixed ϵ.

(i) **8 points.** Write the formula for output coordinate j under each normalization (4 points each). Define any input statistics and learned parameters in your formulas.

Solution: For $x \in \mathbb{R}^D$, define the feature mean, population variance, and uncentered second moment:

$$\mu = \frac{1}{D}\sum_{k=1}^{D} x_k, \qquad v = \frac{1}{D}\sum_{k=1}^{D}(x_k - \mu)^2, \qquad m_2 = \frac{1}{D}\sum_{k=1}^{D} x_k^2.$$

For coordinate j,

$$\mathrm{LayerNorm}(x)_j = \gamma_j \frac{x_j - \mu}{\sqrt{v + \epsilon}} + \beta_j,$$
$$\mathrm{RMSNorm}(x)_j = \widetilde{\gamma}_j \frac{x_j}{\sqrt{m_2 + \epsilon}}.$$

Here $\gamma, \beta \in \mathbb{R}^D$ are LayerNorm's learned gain and bias; $\widetilde{\gamma} \in \mathbb{R}^D$ is RMSNorm's separate learned gain. RMSNorm neither centers the input nor adds a bias. Its denominator uses $m_2 = v + \mu^2$, not the centered variance v.
Grading: 4 points per formula: 2 for normalization and input statistics, 2 for the correct learned affine terms. Accept equivalent notation.

(ii) **4 points.** Give each layer's learnable scalar count in terms of D (2 points each).

Solution: LayerNorm has D gains and D biases, totaling $2D$ learnable scalars. Bias-free RMSNorm has only D gains, totaling D scalars. Input-derived statistics and the fixed ϵ are not trainable parameters.
Grading: 2 points per correct count.

(iii) **4 points.** Now set $D = 2$, $x = [0, 2]$, and $\epsilon = 0$. Set all learned gains to 1 and all LayerNorm biases to 0. Compute this vector's feature mean, population variance, and mean of squared features.

Solution: For $x = [0, 2]$, the statistics are

$$\mu = \frac{0+2}{2} = 1, \qquad v = \frac{(0-1)^2 + (2-1)^2}{2} = 1, \qquad m_2 = \frac{0^2 + 2^2}{2} = 2.$$

The first is the feature mean, the second the population variance, and the third the mean of squared features.
Grading: 1 point for $\mu = 1$, 2 for $v = 1$, and 1 for $m_2 = 2$.

(iv) **2 points.** Using the statistics from (iii), compute $\mathrm{LayerNorm}(x)$.

Solution: With unit gains, zero biases, and $\epsilon = 0$,

$$\mathrm{LayerNorm}(x) = \left[\frac{0-1}{\sqrt{1}}, \frac{2-1}{\sqrt{1}}\right] = [-1, 1].$$

Grading: 2 points for the output vector. Accept consistent substitution and arithmetic using earlier statistics.

(v) **2 points.** Using the same settings, compute $\mathrm{RMSNorm}(x)$. Exact square roots are acceptable.

Solution: With unit gains and $\epsilon = 0$,

$$\mathrm{RMSNorm}(x) = \left[\frac{0}{\sqrt{2}}, \frac{2}{\sqrt{2}}\right] = [0, \sqrt{2}].$$

No mean is subtracted. Both normalization denominators in this example are nonzero, so setting $\epsilon = 0$ is well defined.
Grading: 2 points for the vector; accept $[0, 2/\sqrt{2}]$ or consistent follow-through from earlier statistics.

Q2. KV Caching *20 + 5 bonus points*

(a) **20 required points + 5 bonus points. KV caching.** A decoder generates one token at a time with fixed parameters and no dropout. In every layer, attention permits each position to read only itself and earlier positions; all other operations act separately on each position. Assume earlier token embeddings and position assignments do not change.

(i) **4 points.** Which two kinds of projected vectors from earlier tokens are stored in a KV cache?

Solution: Store the projected **keys** and **values** from earlier tokens at each layer. These are activation tensors, not the learned projection weight matrices.
Grading: 2 points each for keys and values.

(ii) **4 points.** When a new token is appended, can an earlier position attend to it under the stated attention restriction? Explain.

Solution: No. The appended token is later than every earlier position, so the stated attention restriction prevents those earlier positions from attending to it. Their permitted attention prefixes remain the same.
Grading: 2 points for no and 2 for the attention restriction.

(iii) **Optional bonus: 5 points.** Using (ii) and the fixed-input/parameter assumptions, explain why earlier cached vectors can be reused at every layer.

Solution: Earlier token embeddings and position assignments are unchanged. At the first layer, each earlier position reads the same permitted prefix as before; the remaining operations act separately on each position. Its output therefore stays unchanged. Repeating this reasoning layer by layer shows that all earlier representations, and hence their projected keys and values, stay unchanged with fixed parameters and no dropout. Reuse those entries and append entries for the new token; the current query still has to compute attention over the available cache.
Grading: 2 bonus points for unchanged inputs/prefixes and 3 for preservation of earlier representations and K/V through successive layers.

(iv) **6 points.** One cached tensor has shape $[2, 2, 8, 4]$ (batch, head, cached position, head width). How many scalar entries does it contain? Show the product.

Solution: Multiply the four axis sizes:

$$2 \cdot 2 \cdot 8 \cdot 4 = 128 \quad \text{scalar entries per tensor.}$$

Grading: 3 points for the product and 3 for 128.

(v) **6 points.** Each of 3 layers caches two tensors of the shape in (iv). At 2 bytes per scalar, how many bytes do all these tensors occupy? Use your count from (iv); exclude all other memory.

Solution: There are 128 scalars per tensor, two tensors per layer, three layers, and two bytes per scalar. Therefore,

$$128 \cdot 2 \cdot 3 \cdot 2 = 1536 \text{ bytes.}$$

In general, storage is $2LBHTd_h s$ bytes for L layers, batch size B, H heads, cached length T, head width d_h, and s bytes per scalar. This count excludes allocation overhead and all other tensors.
Grading: 3 points for the remaining tensor/layer/byte factors and 3 for 1536 bytes. Accept any factor order and correct arithmetic follow-through from (iv).
Source: Transformer slides 94–104 (KV cache).

Q3. Programming: Loss for a Padded Batch *40 + 5 bonus points*

The code below should compute the **mean next-token cross-entropy over non-PAD targets**. There are **two independent bugs**. Parts (i)–(iii) are required; (iv) is optional bonus work. No full model implementation is needed.

Use these assumptions:

- `F` is the imported alias for `torch.nn.functional`. `logits` contains raw floating-point scores of shape $[B, T, V]$; `targets` is an integer `torch.long` tensor of shape $[B, T]$. Axes are batch, position, and vocabulary class. Devices and dtypes are compatible; arithmetic is well behaved.
- $B = 2$, $T = 4$, $V = 3$, and `pad_id = 0`. The target rows are `[1, 2, 0, 0]` and `[2, 1, 2, 1]`; ID 0 marks padding. Logits and targets are already aligned for next-token prediction.
- All model computations and attention masks are correct.

API reference: `F.cross_entropy` (selected arguments)

- `input`: raw, unnormalized class scores (logits); for this question, shape $[N, V]$.
- `target`: integer class IDs, shape $[N]$.
- `ignore_index`: target ID excluded from the loss; its positions have zero loss in the unreduced output.
- `reduction="none"`: returns one loss per target position, without summing or averaging.

Here N is the number of flattened target positions and V the class count.
Adapted from PyTorch 2.14 documentation; class-index targets, default class weights and label smoothing.

```
1 probabilities = logits.softmax(dim=-1)
2 token_losses = F.cross_entropy(
3     probabilities.reshape(-1, V),
4     targets.reshape(-1),
5     ignore_index=pad_id, reduction="none",
6 ).reshape(B, T)
7 loss = token_losses.mean()
```

(i) **8 points. Count positions.** How many total target positions are in this batch, including PAD (4 points)? How many are non-PAD (4 points)?

Solution: There are $2 \cdot 4 = 8$ total target positions. The first target row has 2 non-PAD targets and the second has 4, giving $2 + 4 = 6$ non-PAD positions.
Grading: 4 points per correct count.

(ii) **12 points. Check the averaging.** For this part only, use `token_losses` $= \begin{bmatrix} 1 & 3 & 0 & 0 \\ 2 & 2 & 2 & 2 \end{bmatrix}$ (values supplied). Compute (a) line 7's result (6 points) and (b) the mean over non-PAD targets (6 points). Show each denominator.

Solution: The supplied entries sum to $1 + 3 + 0 + 0 + 2 + 2 + 2 + 2 = 12$. Line 7 averages all eight stored entries, returning $12/8 = 1.5$. The mean over non-PAD targets is instead $12/6 = 2$. Zero PAD entries still count toward an ordinary tensor mean's denominator.
Grading: 6 points per average: 3 for the denominator and 3 for the value. Allow consistent arithmetic follow-through from (i).

(iii) **20 points. Check the cross-entropy input.** Which line supplies the wrong kind of class scores? Give its line number, a correction, and a one-sentence reason. Assume the averaging bug is fixed.

Solution: Line 3: replace `probabilities.reshape(-1, V)` by `logits.reshape(-1, V)`. The API requires raw class scores (logits). Cross-entropy applies log-softmax internally; passing already normalized probabilities changes the intended loss in general. Line 1 is then unused and may be deleted; it is not a third independent bug. Replacing line 1 with `probabilities = logits` is an equivalent fix, although the variable name would then be misleading.
Grading: 4 points for identifying line 3 or the equivalent input route, 8 for the correction, and 8 for explaining the raw-score requirement or internal log-softmax. Grade independently of (i)–(ii).

(iv) **Optional bonus: 5 points. Fix the averaging.** Replace line 7 using `token_losses` and `targets` to average only non-PAD targets in any batch with at least one valid target. Explain why zero PAD losses do not make the original `mean()` correct. Assume the input bug is fixed.

Solution: One valid replacement is `loss = token_losses.sum() / targets.ne(pad_id).sum()`. Ignored positions already contribute zero to the numerator. The denominator counts only valid targets: 6 here, rather than all 8 stored entries. An ordinary `mean()` counts every entry, even zero entries, so zero PAD losses do not remove PAD positions from its denominator. Accept also `token_losses[targets.ne(pad_id)].mean()`. At least one non-PAD target ensures a nonzero denominator.
Equivalent combined fix for (iii)–(iv): `F.cross_entropy(logits.reshape(-1, V), targets.reshape(-1), ignore_index=pad_id, reduction="mean")` directly returns the intended scalar. It must replace the unreduced-loss computation and line 7, without reshaping the scalar to $[B, T]$. Ignoring PAD targets removes positions from the loss, not the PAD class from the vocabulary softmax at valid positions.
Grading: 3 bonus points for a correct general expression and 2 for explaining the denominator. Accept the combined fix when both issues are explained; grade independently of earlier counting errors.
Sources: Transformer slides 62–65 and Handout 12 §2.