Due Date: April 4th, 23:59 2025

Instructions

- For all questions that are not graded only on the answer, show your work! Any problem without work shown will get no marks regardless of the correctness of the final answer.
- Please try to use a document preparation system such as LaTeX. If you write your answers by hand, note that you risk losing marks if your writing is illegible without any possibility of regrade, at the discretion of the grader.
- Submit your answers electronically via the course GradeScope. Incorrectly assigned answers can be given 0 automatically at the discretion of the grader. To assign answers properly, please:
  - Make sure that the top of the first assigned page is the question being graded.
  - Do not include any part of answer to any other questions within the assigned pages.
  - Assigned pages need to be placed in order.
  - For questions with multiple parts, the answers should be written in order of the parts within the question.
- Questions requiring written responses should be short and concise when necessary.Unnecessary wordiness will be penalized at the grader's discretion.
- Please sign the agreement below.
- It is your responsibility to follow updates to the assignment after release. All changes will be visible on Overleaf and Piazza.
- Any questions should be directed towards the TAs for this assignment (theoretical part): *Alireza Dizaji, Pascal Tikeng.*

I acknowledge I have read the above instructions and will abide by them throughout this assignment. I further acknowledge that any assignment submitted without the following form completed will result in no marks being given for this portion of the assignment.

Name: **Félix Wilhelmy**

UdeM Student ID: **20333575**

Signature: 

Question 1

In this question, we compute the mean and variance of a $4 \times 2 \times 3$ tensor using three normalization techniques. The dimensions of the tensor are defined as follows:

- B (Batch size): Number of samples $\rightarrow 4$
- C (Channels): Number of channels $\rightarrow 2$
- F (Features): Number of features per channel $\rightarrow 3$

Below, we first state the normalization equations and then present the computed results.

Batch Normalization

For Batch Normalization, the statistics are computed per channel (aggregating over all samples and features in the batch)

$$\mu_c = \frac{1}{B \cdot F} \sum_{i=1}^{B} \sum_{j=1}^{F} x_{i,c,j}, \quad \sigma_c^2 = \frac{1}{B \cdot F} \sum_{i=1}^{B} \sum_{j=1}^{F} (x_{i,c,j} - \mu_c)^2 .$$

The table below shows the computed mean and variance for each channel

Channel	Mean	Variance
1	2.58333	1.57639
2	2.91667	1.24306

Table 1: Batch Norm Results

Layer Normalization

Layer Normalization computes the statistics per sample (across all channels and features)

$$\mu_i = \frac{1}{C \cdot F} \sum_{c=1}^{C} \sum_{j=1}^{F} x_{i,c,j}, \quad \sigma_i^2 = \frac{1}{C \cdot F} \sum_{c=1}^{C} \sum_{j=1}^{F} (x_{i,c,j} - \mu_i)^2 .$$

The computed results for each sample are presented in the following table

Sample	Mean	Variance
1	3.16667	1.80556
2	2.00000	1.00000
3	2.50000	0.91667
4	3.33333	0.88889

Table 2: Layer Norm Results

Instance Normalization

Instance Normalization computes the statistics per sample and per channel (over the features)

$$\mu_{i,c} = \frac{1}{F}\sum_{j=1}^{F} x_{i,c,j}, \quad \sigma^2_{i,c} = \frac{1}{F}\sum_{j=1}^{F} (x_{i,c,j} - \mu_{i,c})^2 .$$

The results for each sample and channel are summarized in the table below

Sample	Channel	Mean	Variance
1	1	2.66667	2.88889
1	2	3.66667	0.22222
2	1	2.33333	1.55556
2	2	1.66667	0.22222
3	1	2.00000	0.66667
3	2	3.00000	0.66667
4	1	3.33333	0.22222
4	2	3.33333	1.55556

Table 3: Instance Norm Results

Question 2

Question 2.1

The vocabulary consists of N words (including the special token <BOS>). Since every generated sequence of length $T+1$ must start with <BOS>, the first token is fixed. This leaves T positions, each of which can be filled with any of the N words.

Therefore, the total number of possible sequences is : N^T

Question 2.2

Performing an exhaustive enumeration of all possible sequences requires evaluating N^T candidates, where N is the vocabulary size. If with a fixed sequence length T, the number of candidates grows as $O(N^T)$, which is a polynomial growth in N. However, even with polynomial growth, T is often non-negligible in practical scenarios, and vocabularies can be very large. Therefore, in real-world applications, exhaustive enumeration is not scalable.

Question 2.3

In the beam search algorithm, at each time step t the following operations are performed:

- For each of the B beams, the algorithm computes scores for all N possible next words, which requires $B \times N$ score computations.
- It then selects the top B candidates from the $B \times N$ scores.

Time Complexity At each time step, computing scores and selecting the top B takes $O(B \cdot N)$ time. Since these steps are repeated for T time steps, the overall time complexity is

$$O(T \cdot B \cdot N)$$

This complexity assumes that the selection of the top beams from the $B \times N$ candidates is performed in constant time (or using an efficient algorithm that does not add significant overhead).

Space Complexity The algorithm maintains B partial sequences of maximum length T along with their corresponding log-likelihoods, which requires $O(B \cdot T)$ space. In addition, temporary storage

for the $B \times N$ scores is needed at each time step, contributing an extra $O(B \cdot N)$. Thus, the total space complexity is

$$O(B \cdot T + B \cdot N)$$

While beam search significantly reduces the search space compared to exhaustive enumeration, its linear dependence on N means that the decoding time can still be considerable when the vocabulary size is very large.

Question 2.4

Here we must apply two decoding algorithms to find the best sequences in t

The final log-likelihood of a generated sequence $w_{1:T}$ is computed as the sum of the conditional log-probabilities at each time step:

$$\log p(w_{1:T}) = \sum_{t=1}^{T} \log p(w_t \mid w_{1:t-1}).$$

a) Greedy Decoding

Using greedy decoding, we select at each time step the word with the highest conditional probability. That is, for each time step t we compute

$$\overline{w}_t = \arg\max_{w_t} \log p\left(w_t \mid w_{0:t-1}\right)$$

Step 1 ($t = 1$): From the initial token $\langle \texttt{BOS} \rangle$, suppose the candidates are

$$\log p(w_1 = \texttt{the} \mid \langle \texttt{BOS} \rangle) = -1.7 \quad \log p(w_1 = \texttt{a} \mid \langle \texttt{BOS} \rangle) = -2.0$$

Thus, we select

$$\overline{w}_1 = \texttt{the} \quad \text{with } \log p(w_1 \mid \langle \texttt{BOS} \rangle) = -1.7$$

Step 2 ($t = 2$): Given the partial sequence ($\langle \texttt{BOS} \rangle$, `the`), we choose

$$\overline{w}_2 = \texttt{red} \quad \text{with } \log p(w_2 \mid \langle \texttt{BOS} \rangle, \texttt{the}) = -3.3$$

Step 3 ($t = 3$): For the sequence ($\langle \texttt{BOS} \rangle$, `the`, `red`), we select

$$\overline{w}_3 = \texttt{rectangle} \quad \text{with } \log p(w_3 \mid \langle \texttt{BOS} \rangle, \texttt{the}, \texttt{red}) = -4.7$$

The greedy decoded sequence is then

$$\overline{w}_{1:3} = (\texttt{the}, \texttt{red}, \texttt{rectangle}), \text{ with } \log p(\overline{w}_{1:3}) = -9.7.$$

b) Beam Search Decoding

Using beam search with a beam width $B = 2$, we maintain the top 2 partial sequences $w^{(b)}$ at each time step.

Step 1 ($t = 1$): Initially, we simply take the two first nodes after <BOS>

$$w_1^{(2)} = \texttt{the} \quad \text{with } \log p(w_1^{(2)} = \texttt{the} \mid \langle \texttt{BOS} \rangle) = -1.7$$
$$w_1^{(1)} = \texttt{a} \quad \text{with } \log p(w_1^{(1)} = \texttt{a} \mid \langle \texttt{BOS} \rangle) = -2.0$$

Step 2 ($t = 2$): We explore all possible partial sequences based on our two beams from $t = 1$

For the first beam $w^{(1)}$ (the sequence starting with `the`), we find the partial sequences

$$\log p(w_2^{(1)} = \texttt{green} \mid \langle \texttt{BOS} \rangle, \texttt{the}) = -3.6 \quad \log p(w_2^{(1)} = \texttt{red} \mid \langle \texttt{BOS} \rangle, \texttt{the}) = -3.3$$

For the second beam $w^{(2)}$ (the sequence starting with `a`), we find

$$\log p(w_2^{(2)} = \texttt{blue} \mid \langle \texttt{BOS} \rangle, \texttt{a}) = -2.7, \quad \log p(w_2^{(2)} = \texttt{red} \mid \langle \texttt{BOS} \rangle, \texttt{a}) = -2.9.$$

Thus, after $t = 2$ the top two beams are

$$w_{1:2}^{(1)} = (\texttt{a}, \texttt{blue}), \text{ with } \log p(w_{1:2}^{(1)}) = -4.7$$
$$w_{1:2}^{(2)} = (\texttt{a}, \texttt{red}), \text{ with } \log p(w_{1:2}^{(2)}) = -4.9$$

Step 3 ($t = 3$): To avoid unnecessary wordiness, we can assume that the same exploration is done at step 3. Giving us the final top 2 beams

$$w_{1:3}^{(1)} = (\texttt{a},\ \texttt{blue},\ \texttt{square}), \text{ with } \log p(w_{1:3}^{(1)}) = -9.9.$$
$$w_{1:3}^{(2)} = (\texttt{a},\ \texttt{red},\ \texttt{circle}), \text{ with } \log p(w_{1:3}^{(2)}) = -10,$$

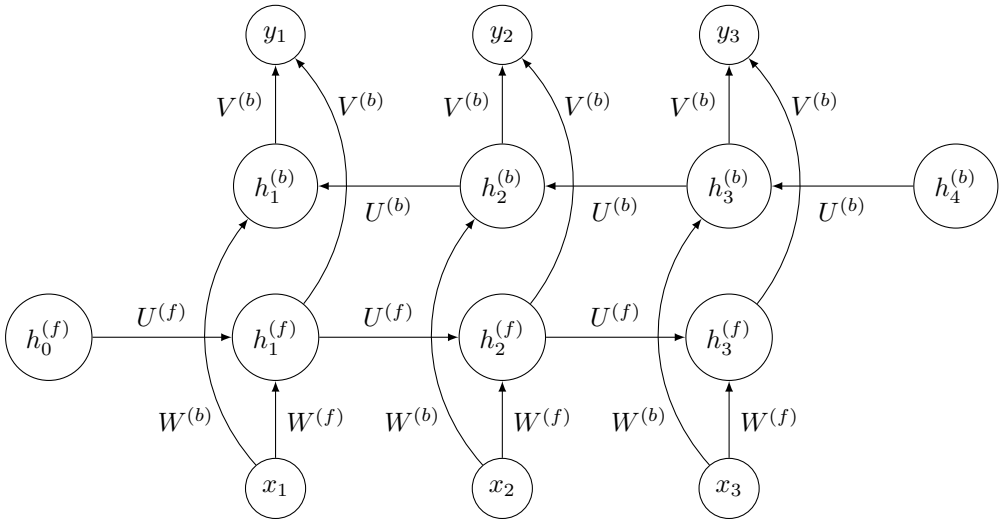
Question 2.5

Greedy Decoding Greedy decoding picks at each time step the token with the highest conditional probability given the previously decoded tokens. While this makes greedy decoding computationally inexpensive and straightforward to implement, it can easily miss the globally optimal sequence, especially as the sequence length T grows.

Beam Search Beam search keeps track of multiple candidate sequences (up to a beam width B) at each decoding step. This typically yields better sequences than greedy decoding. However, it can still prune out the true best sequence if that sequence does not appear promising in its early steps or if B is too small. Furthermore, the computational cost of beam search grows linearly with B and N (the vocabulary size) for each time step T, making it more expensive than greedy decoding (though still far cheaper than exhaustive search).

Question 3

Question 3.1



Question 3.2

a) Deriving the Local Gradients and Jacobian Elements

Local Gradients Recall that we aim to derive the expressions for $\nabla_{h_t^{(f)}} L_t$ and $\nabla_{h_t^{(b)}} L_t$. For simplicity, we drop the forward/backward superscripts and derive a general expression

$$\frac{dL_t}{dh_t} = \frac{dL_t}{dy_t} \cdot \frac{dy_t}{dh_t}$$

Then we compute the derivative of the loss with respect to y_t and the derivative of y_t with respect to the hidden states

$$\frac{dL_t}{dy_t} = (2(y_t - z_t))^T \quad \text{and} \quad \frac{dy_t}{dh_t} = V$$

In this derivation we adopt the numerator layout where gradients are represented as row vectors. That is, we define the derivative $\frac{dL}{dh}$ as a $1 \times d$ row vector. With this convention the chain rule is written as

$$\nabla_{h_t} L_t = (2(y_t - z_t))^T \cdot V$$

This translates to the forward and backward RNNs as

$$\nabla_{h_t^{(f)}} L_t = (2(y_t - z_t))^T \cdot V^{(f)} \quad \text{and} \quad \nabla_{h_t^{(b)}} L_t = (2(y_t - z_t))^T \cdot V^{(b)}$$

Jacobian Elements For the forward RNN, we define the pre-activation as

$$a_{t+1}^{(f)} = W^{(f)} x_{t+1} + U^{(f)} h_t^{(f)} \quad \text{and then} \quad h_{t+1}^{(f)} = \tanh(a_{t+1}^{(f)})$$

Using the chain rule, the elementwise Jacobian, in the numerator layout, is defined as

$$\left(\frac{\partial h_{t+1}^{(f)}}{\partial h_t^{(f)}}\right)_{ij} = \left[1 - \tanh^2\big((a_{t+1}^{(f)})_i\big)\right] \left(\frac{\partial (a_{t+1}^{(f)})_i}{\partial (h_t^{(f)})_j}\right) = \left[1 - \big((h_{t+1}^{(f)})_i\big)^2\right] U_{ij}^{(f)}$$

Similarly, for the backward RNN we have

$$\left(\frac{\partial h_{t-1}^{(b)}}{\partial h_t^{(b)}}\right)_{ij} = \left[1 - \big((h_{t-1}^{(b)})_i\big)^2\right] U_{ij}^{(b)}$$

(b) Deriving the Jacobian Matrices

Since the activation function tanh is applied elementwise its derivative is zero for elements where $i \neq j$. Therefore the Jacobian matrices are diagonal with respect to the pre-activation vectors. In compact form we have

$$\frac{\partial h_{t+1}^{(f)}}{\partial h_t^{(f)}} = \operatorname{diag}\Big(1 - \big(h_{t+1}^{(f)}\big)^2\Big) \cdot U^{(f)} \quad \text{and} \quad \frac{\partial h_{t-1}^{(b)}}{\partial h_t^{(b)}} = \operatorname{diag}\Big(1 - \big(h_{t-1}^{(b)}\big)^2\Big) \cdot U^{(b)}$$

Here each Jacobian is a square matrix of size $d \times d$ and the diagonal structure reflects the elementwise derivative of tanh

Question 4

Question 4.1

LoRA inject low-rank trainable matrices $A \in \mathbb{R}^{r \times d}$ and $B \in \mathbb{R}^{d \times r}$ into the model such that

$$h = W_0 x + BAx$$

The base weights W_0 are frozen during training, and only A, B are updated.

The key difference is that QLoRA adds on top of this a quantization step: W_0 is stored in 4-bit precision (NF4), significantly reducing memory usage. W_0 is then dequantized on-the-fly during forward and backward passes. This enables QLoRA to fine-tune large models within limited hardware, while preserving the benefits of LoRA's efficiency.

Question 4.2

The choice of the rank r directly affects memory usage and the number of trainable parameters. More precisely, each adapter layer introduces $2r(d + k)$ trainable parameters, where d and k are the input and output dimensions of the transformed layer, respectively.

As r increases, the model capacity and performance also improve (often approaching full fine-tuning performance when r is sufficiently high), but at the cost of increased memory and computation. In LoRA, where the base model is stored in FP16/FP32, this impact is even more pronounced. In QLoRA, the use of a 4-bit base mitigates memory usage, yet increasing r still increases the adapter size.

Thus, r is a critical hyperparameter that balances adapter expressiveness and memory efficiency.

Question 4.3

LoRA allows the low-rank weights to be merged into the pre-trained weights at the end of training, resulting in an inference process that is identical to a fully fine-tuned model with zero extra latency.

QLoRA, on the other hand, requires on-the-fly dequantization of the 4-bit base weights at inference time. This introduces overhead, but because the model moves significantly less data, it can still lead to *faster end-to-end inference*, especially in large models where memory bandwidth is the bottleneck.

Question 4.4

Note: The O Complexity is shown at the end in the summary table.

a) Time and Space Complexities

We consider a Transformer with embedding dimension d, low-rank dimension r (with $r \ll d$), h heads, and a context length of n. In both LoRA and QLoRA, only the low-rank adapter parameters are trainable while the frozen base weights are either stored in full precision (LoRA) or quantized (QLoRA).

FFN Module

The FFN layer is given by

$$\text{FFN}(X) = XW^F + XA^F(B^F)^T$$

where $W^F \in \mathbb{R}^{d\times d}$ is frozen and $A^F, B^F \in \mathbb{R}^{d\times r}$ are the trainable low-rank matrices.

Time Complexity If we decompose this equation, we have three matrix multiplications and one matrix addition. We can represent these as follows

$$\text{FFN}(X) = \underbrace{\underbrace{XW^F}_{n\,d^2} + \underbrace{XA^F(B^F)^T}_{2\,n\,d\,r}}_{n\,d}$$

This brings the overall operation count to

$$n\,d^2 + 2\,n\,d\,r + n\,d$$

In the case of LoRA, we could reduce this by merging the low-rank matrices into the pre-trained weights to reduce the time complexity to

$$n\,d^2$$

This would be possible only after the training of the low-rank matrices. In the case of QLoRA, we need to add the computational overhead of dequantising the pre-trained weights W^F. Which is d^2 operations. This would save $28\,d^2$ bits.

Space Complexity The FFN has two 32-bit (FP) trainable matrices A^F and B^F each with $d\,r$ parameters giving a total of

$$2\,d\,r$$

MHA Module

The multi-head attention can be computed by

$$\text{MHA}(H) = [\text{head}^1(H), \text{head}^2(H), \ldots, \text{head}^h(H)]W^O$$

Each head $i \in \{1, \ldots, h\}$ computes its attention as

$$\text{head}^i(H) = softmax\Big(\frac{Q^i(K^i)^T}{\sqrt{d}}\Big)V^i$$

The projections for queries Q^i, keys K^i, and values V^i are given by

$$Q^i = HW^{Q,i} + HA^{Q,i}(B^{Q,i})^T, \quad K^i = HW^{K,i} + HA^{K,i}(B^{K,i})^T, \quad V^i = HW^{V,i} + HA^{V,i}(B^{V,i})^T$$

Here, the frozen matrices $W^{*,i} \in \mathbb{R}^{d\times d}$ are fixed while the trainable matrices $A^{*,i}$ and $B^{*,i}$ are in $\mathbb{R}^{d\times r}$

Time Complexity The first operation in the MHA is the projections (e.g. Q^i). Each projection can be computed in the same number of operations as the FFN

$$n\,d^2 + 2\,n\,d\,r + n\,d$$

Since there are 3 projections per head (query, key, value) we get

$$3\Big[n\,d^2 + 2\,n\,d\,r + n\,d\Big]$$

The computation of the attention with these projections is given by

$$\text{head}^i(H) = softmax\Big(\frac{Q^i(K^i)^T}{\sqrt{d}}\Big)V^i$$

The resulting matrix $Q^i(K^i)^T$ is in $\mathbb{R}^{n\times n}$ so that if we include the elementwise softmax and the division by $\sqrt{d}$ this would add n^2 FLOPs per head. Thus in total per head we end up with

$$2\,n^2\,d + 3\Big[n\,d^2 + 2\,n\,d\,r + n\,d\Big] + n^2$$

Since there are h heads the number of operations for all heads becomes

$$h\Big[2\,n^2\,d + 3\,(n\,d^2 + 2\,n\,d\,r + n\,d) + n^2\Big]$$

Finally we multiply the full context of our MHA by $W^O \in \mathbb{R}^{d\times d}$ which gives

$$h\Big[2\,n^2\,d + 3\,(n\,d^2 + 2\,n\,d\,r + n\,d) + n^2\Big] + n\,d^2$$

In LoRA there would again be the possibility of merging the low-rank matrices into the pre-trained weights reducing the inference cost (after training) to

$$h\Big[2\,n^2\,d + 3\,(n\,d^2) + n^2\Big] + n\,d^2$$

In QLoRA we would need to dequantize the four W^* weight matrices using the quantisation scheme. Giving us $4\,d^2$ additional operations for the quantisation/dequantisation. Saving $112\,d^2$ bits.

Space Complexity The trainable matrices in the MHA are only the A^* and B^* matrices. This gives $2\,d\,r$ parameters per head so that for the three projections we have

$$3 \times \big(d \cdot r + d \cdot r\big) = 6dr$$

Thus, for h heads the overall number of trainable parameters for MHA is

$$6\,h\,d\,r$$

Summary

It's worth noting that we might have wanted to omit r when computing the Big-O as it was noted that $r \ll d$. But, for clarity purpose, I decided to leave it here.

Module	**Time Complexity**	**Space Complexity**
FFN	$O(n\,d^2)$	$O(d\,r)$
MHA	$O(h\,(n^2\,d + n\,d^2))$	$O(h\,d\,r)$

Table 4: Big-O time and space complexities for the FFN and MHA modules

b) Choosing between LoRA and QLoRA

When choosing between LoRA and QLoRA for fine-tuning, the following trade-offs apply:

	MHA	**LFFN**
Memory Limit	QLoRA	QLoRA
Faster Inference	LoRA	LoRA
Both	QLoRA	LoRA

Table 5: Preferred low-rank adaptation method for MHA and LFFN under different constraints

Explanation

- For both modules, under tight memory constraints, QLoRA is preferred due to its significant memory savings from 4-bit quantization.
- For faster inference, LoRA is preferred for both modules because merging the adapters with the pre-trained weights incurs no additional latency.
- When balancing both memory and inference speed, QLoRA is favored for the MHA module because the multi-head attention module typically involves large frozen weight matrices, and quantizing these weights leads to substantial memory savings. Furthermore, although dequantization incurs an additional computational overhead, this overhead is relatively moderate compared to the benefits of drastically reduced memory requirements for the MHA module.