


**Due Date: March 28th, 23:00 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.**

Signature: 

Name: Daniel Lofeodo

UdeM Student ID: 20226991

---

**Question 1 (6pts). (Normalization)**

Batch normalization, layer normalization and instance normalization all involve calculating the mean  $\mu$  and variance  $\sigma^2$  with respect to different subsets of the tensor dimensions. Given the following 3D tensor, calculate the corresponding mean and variance tensors for each normalization technique:  $\mu_{batch}$ ,  $\mu_{layer}$ ,  $\mu_{instance}$ ,  $\sigma_{batch}^2$ ,  $\sigma_{layer}^2$ , and  $\sigma_{instance}^2$ .

$$\left[ \begin{bmatrix} 2, 5, 1 \\ 4, 3, 4 \end{bmatrix}, \begin{bmatrix} 1, 2, 4 \\ 1, 2, 2 \end{bmatrix}, \begin{bmatrix} 3, 1, 2 \\ 3, 2, 4 \end{bmatrix}, \begin{bmatrix} 3, 4, 3 \\ 5, 2, 3 \end{bmatrix} \right]$$

The size of this tensor is 4 x 2 x 3, which corresponds to the batch size, number of input channels, and number of features, respectively.

**Batch norm:**

Let  $B$  be the batch size and  $F$  be the number of features.

$$\begin{aligned} \mu_1 &= \frac{1}{BF} \sum_{i=1}^B \sum_{j=1}^F x_{i,1,j} = \frac{1}{4 \cdot 3} \sum_{i=1}^4 \sum_{j=1}^3 x_{i,1,j} \\ &= \frac{1}{12} (2 + 5 + 1 + 1 + 2 + 4 + 3 + 1 + 2 + 3 + 4 + 3) \\ &= 2.583 \\ \mu_2 &= \frac{1}{12} (4 + 3 + 4 + 1 + 2 + 2 + 3 + 2 + 4 + 5 + 2 + 3) \\ &= 2.916 \\ \sigma_1^2 &= \frac{1}{BF} \sum_{i=1}^B \sum_{j=1}^F (x_{i,1,j} - \mu_1)^2 = \frac{1}{4 \cdot 3} \sum_{i=1}^4 \sum_{j=1}^3 (x_{i,1,j} - \mu_1)^2 \\ &= \frac{1}{12} ((2 - 2.583)^2 + (5 - 2.583)^2 + \dots + (3 - 2.583)^2) \\ &= 1.576 \\ \sigma_2^2 &= \frac{1}{12} ((4 - 2.916)^2 + (3 - 2.916)^2 + \dots + (3 - 2.916)^2) \\ &= 1.243 \end{aligned}$$

$$\begin{aligned} \mu_{batch} &= \begin{bmatrix} 2.583 \\ 2.916 \end{bmatrix} \\ \sigma_{batch}^2 &= \begin{bmatrix} 1.576 \\ 1.243 \end{bmatrix} \end{aligned}$$

**Layer norm:**

Let  $C$  be the number of channels and  $F$  be the number of features.

$$\begin{aligned}
 \mu_1 &= \frac{1}{CF} \sum_{i=1}^C \sum_{j=1}^F x_{1,i,j} = \frac{1}{2 \cdot 3} \sum_{i=1}^2 \sum_{j=1}^3 x_{1,i,j} \\
 &= \frac{1}{6} (2 + 5 + 1 + 4 + 3 + 4) \\
 &= 3.167 \\
 \mu_2 &= \frac{1}{6} (1 + 2 + 4 + 1 + 2 + 2) \\
 &= 2.000 \\
 &\dots \\
 \sigma_1^2 &= \frac{1}{CF} \sum_{i=1}^C \sum_{j=1}^F (x_{1,i,j} - \mu_1)^2 = \frac{1}{2 \cdot 3} \sum_{i=1}^2 \sum_{j=1}^3 (x_{1,i,j} - \mu_1)^2 \\
 &= \frac{1}{6} ((2 - 3.167)^2 + (5 - 3.167)^2 + \dots + (4 - 3.167)^2) \\
 &= 1.806 \\
 \sigma_2^2 &= \frac{1}{6} ((1 - 2)^2 + (2 - 2)^2 + \dots + (2 - 2)^2) \\
 &= 1.000 \\
 &\dots
 \end{aligned}$$

$$\begin{aligned}
 \mu_{layer} &= \begin{bmatrix} 3.167 \\ 2.000 \\ 2.500 \\ 3.333 \end{bmatrix} \\
 \sigma_{layer}^2 &= \begin{bmatrix} 1.806 \\ 1.000 \\ 0.917 \\ 0.889 \end{bmatrix}
 \end{aligned}$$

**Instance norm:**

Let  $b$  be the batch and  $c$  be the channel.

$$\begin{aligned}\mu_{b=1,c=1} &= \frac{1}{F} \sum_{i=1}^F x_{1,1,i} = \frac{1}{3} \sum_{i=1}^3 x_{1,1,i} \\ &= \frac{1}{3}(2 + 5 + 1) \\ &= 2.667\end{aligned}$$

$$\begin{aligned}\mu_{b=2,c=1} &= \frac{1}{F} \sum_{i=1}^F x_{2,1,i} = \frac{1}{3} \sum_{i=1}^3 x_{2,1,i} \\ &= \frac{1}{3}(1 + 2 + 4) \\ &= 2.333\end{aligned}$$

...

$$\begin{aligned}\sigma_{b=1,c=1}^2 &= \frac{1}{3} \sum_{i=1}^3 (x_{1,1,i} - \mu_{b=1,c=1})^2 \\ &= \frac{1}{3} ((2 - 2.667)^2 + (5 - 2.667)^2 + (1 - 2.667)^2) \\ &= 2.889\end{aligned}$$

$$\begin{aligned}\sigma_{b=2,c=1}^2 &= \frac{1}{3} \sum_{i=1}^3 (x_{2,1,i} - \mu_{b=2,c=1})^2 \\ &= \frac{1}{3} ((1 - 2.667)^2 + (2 - 2.667)^2 + (4 - 2.667)^2) \\ &= 1.556\end{aligned}$$

...

$$\begin{aligned}\mu_{instance} &= \begin{bmatrix} 2.667 & 3.667 \\ 2.333 & 1.667 \\ 2.000 & 3.000 \\ 3.333 & 3.333 \end{bmatrix} \\ \sigma_{instance}^2 &= \begin{bmatrix} 2.889 & 0.222 \\ 1.556 & 0.222 \\ 0.667 & 0.667 \\ 0.222 & 1.556 \end{bmatrix}\end{aligned}$$

**Question 2** (20pts). (**Decoding**)

Suppose that we have a vocabulary containing  $N$  possible words, including a special token  $\langle \text{BOS} \rangle$  to indicate the beginning of a sentence. Recall that in general, a language model with a full context can be written as

$$p(w_1, w_2, \dots, w_T \mid w_0) = \prod_{t=1}^T p(w_t \mid w_0, \dots, w_{t-1}).$$

We will use the notation  $\mathbf{w}_{0:t-1}$  to denote the (partial) sequence  $(w_0, \dots, w_{t-1})$ . Once we have a fully trained language model, we would like to generate realistic sequences of words from our language model, starting with our special token  $\langle \text{BOS} \rangle$ . In particular, we might be interested in generating the most likely sequence  $\mathbf{w}_{1:T}^*$  under this model, defined as

$$\mathbf{w}_{1:T}^* = \arg \max_{\mathbf{w}_{1:T}} p(\mathbf{w}_{1:T} \mid w_0 = \langle \text{BOS} \rangle). \quad (1)$$

For clarity we will drop the explicit conditioning on  $w_0$ , assuming from now on that the sequences always start with the  $\langle \text{BOS} \rangle$  token.

1. (2 pts) How many possible sequences of length  $T + 1$  starting with the token  $\langle \text{BOS} \rangle$  can be generated in total? Give an exact expression, without the  $O$  notation. Note that the length  $T + 1$  here includes the  $\langle \text{BOS} \rangle$  token.

There are  $N^T$  possible sequences of words.

2. (2 pts) To determine the most likely sequence  $\mathbf{w}_{1:T}^*$ , one could perform exhaustive enumeration of all possible sequences and select the one with the highest joint probability (as defined in equation 1). Comment on the feasibility of this approach. Is it scalable with vocabulary size?

This method is not feasible due to memory constraints and time complexity.

(a) **Memory Constraints:**

Assuming a reasonably sized vocabulary of 20 000 words and a small sequence length of 10, we would need to compute  $20000^{10} \approx 10^{43}$  joint probabilities. Assuming we are using python `float` of size 8 bytes to store the values, it would require approximately  $8^{21}$  terabytes to store all of the probabilities.

(b) **Time complexity:**

Because there are  $N^T$  possible sequences of words, and we need to compute the conditional probability of every token in the sequence  $T$ , the time complexity required to compute all joint probabilities would be  $O(T \cdot N^T)$ , which makes this method unfeasible.

3. (3 pts) In light of the difficulties associated with exhaustive enumeration, it becomes essential to employ practical search strategies to obtain a reasonable approximation of the most likely sequence.

In order to generate  $B$  sequences having high likelihood, one can use a heuristic algorithm called Beam search decoding, whose pseudo-code is given in Algorithm 1 below

---

**Algorithm 1:** Beam search decoding

---

**Input:** A language model  $p(\mathbf{w}_{1:T} | w_0)$ , the beam width  $B$   
**Output:**  $B$  sequences  $\mathbf{w}_{1:T}^{(b)}$  for  $b \in \{1, \dots, B\}$   
Initialization:  $w_0^{(b)} \leftarrow \langle \text{BOS} \rangle$  for all  $b \in \{1, \dots, B\}$   
Initial log-likelihoods:  $l_0^{(b)} \leftarrow 0$  for all  $b \in \{1, \dots, B\}$   
**for**  $t = 1$  **to**  $T$  **do**  
    **for**  $b = 1$  **to**  $B$  **do**  
        **for**  $j = 1$  **to**  $N$  **do**  
             $s_b(j) \leftarrow l_{t-1}^{(b)} + \log p(w_t = j | \mathbf{w}_{0:t-1}^{(b)})$   
        **for**  $b = 1$  **to**  $B$  **do**  
            Find  $(b', j)$  such that  $s_{b'}(j)$  is the  $b$ -th largest score  
            Save the partial sequence  $b'$ :  $\tilde{\mathbf{w}}_{0:t-1}^{(b)} \leftarrow \mathbf{w}_{0:t-1}^{(b')}$   
            Add the word  $j$  to the sequence  $b$ :  $w_t^{(b)} \leftarrow j$   
            Update the log-likelihood:  $l_t^{(b)} \leftarrow s_{b'}(j)$   
        Assign the partial sequences:  $\mathbf{w}_{0:t-1}^{(b)} \leftarrow \tilde{\mathbf{w}}_{0:t-1}^{(b)}$  for all  $b \in \{1, \dots, B\}$

---

What is the time complexity of Algorithm 1? Its space complexity? Write the algorithmic complexities using the  $O$  notation, as a function of  $T$ ,  $B$ , and  $N$ . Is this a practical decoding algorithm when the size of the vocabulary is large?

• **Time complexity:**

The time complexity of individual operations are the following:

- Initialization of  $w_0 \rightarrow O(B)$
- Initialization of  $l_0 \rightarrow O(B)$
- Inferring  $p(w_t | \mathbf{w}_{0:t-1}^{(b)})$ , assuming we are using a transformer for inference  $\rightarrow O(T^2)$ 
  - \* Repeated  $T \cdot B$  times for a total complexity  $O(B \cdot T^3)$ . **Repeating inference for every iteration of N would be redundant so I assume inference would only be performed for every iteration of B and accessed for every iteration of N.**
- Calculating  $s_b(j) \rightarrow O(1)$ 
  - \* Repeated  $T \cdot B \cdot N$  times for a total complexity  $O(T \cdot B \cdot N)$
- Sorting cumulative beam scores  $s_b(j)$  to find  $(b', j) \rightarrow O(B \cdot N \log(B \cdot N))$  (this requires sorting  $B \cdot N$  scores).
  - \* Repeated  $T$  times for a total complexity  $O(T \cdot B \cdot N \cdot \log(B \cdot N))$
- Saving the partial sequence  $\tilde{\mathbf{w}}_{0:t-1} \rightarrow O(T)$ 
  - \* Repeated  $T \cdot B$  times for a total complexity  $O(B \cdot T^2)$
- Adding the word  $j \rightarrow O(1)$ 
  - \* Repeated  $T \cdot B$  times for a total complexity  $O(B \cdot T)$
- Update the log-likelihood  $\rightarrow O(1)$ 
  - \* Repeated  $T \cdot B$  times for a total complexity  $O(B \cdot T)$
- Assigning  $\mathbf{w}_{0:t-1}^{(b)} \rightarrow O(T)$ 
  - \* Repeated  $T$  times for a total complexity  $O(T^2)$

The total time complexity is  $O(B \cdot T^3 + T \cdot B \cdot N \cdot \log(B \cdot N))$ . Thus, the time complexity increases linearly times logarithmically with vocabulary size. This is a significant improvement with respect to exhaustive search, but may still become prohibitive with large vocabulary sizes.

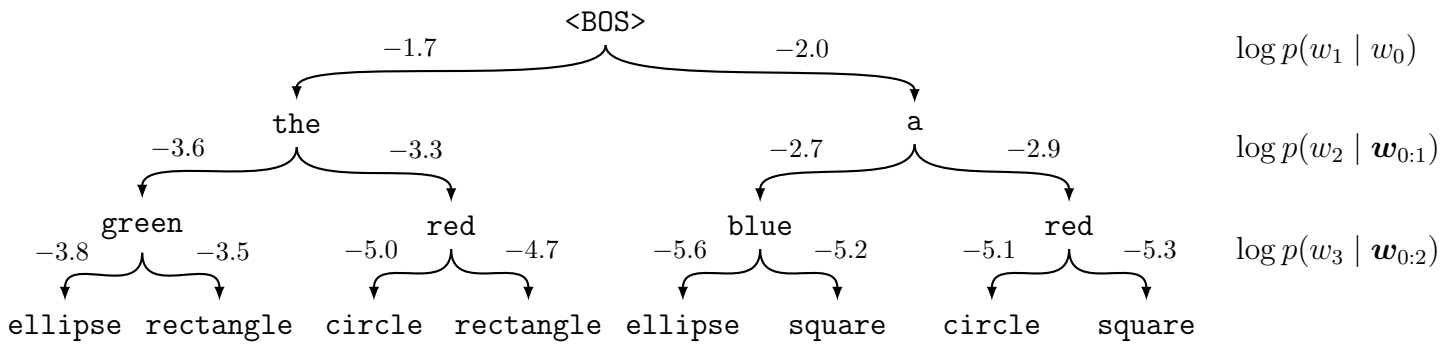
• **Space complexity:**

- $w_{1:T}^{(b)} \rightarrow O(B \cdot T)$
- $l_{1:T}^{(b)} \rightarrow O(B \cdot T)$
- $s_b(j) \rightarrow O(N \cdot B)$

The total space complexity is  $O(B \cdot (T + N))$ . The space complexity increases linearly with respect to vocabulary size, so as the vocabulary grows, the space required also grows.

But even for a large vocabulary of size 500,000 and a large beam width of 100, the required space is  $50,000,000 \times 4 \text{ bytes} = 200\text{MB}$ , so space complexity should never be an issue.

4. (10 pts) The different sequences that can be generated with a language model can be represented as a tree, where the nodes correspond to words and edges are labeled with the log-probability  $\log p(w_t | \mathbf{w}_{0:t-1})$ , depending on the path  $\mathbf{w}_{0:t-1}$ . In this question, consider the following language model (where the low probability paths have been removed for clarity)



- (a) (2 pts) Greedy decoding is a simple algorithm where the next word  $\bar{w}_t$  is selected by maximizing the conditional probability  $p(w_t | \bar{\mathbf{w}}_{0:t-1})$  (with  $\bar{w}_0 = \text{<BOS>}$ )

$$\bar{w}_t = \arg \max_{w_t} \log p(w_t | \bar{\mathbf{w}}_{0:t-1}).$$

Find  $\bar{\mathbf{w}}_{1:3}$  using greedy decoding on this language model, and its log-likelihood  $\log p(\bar{\mathbf{w}}_{1:3})$ .

"the red rectangle"

$$\begin{aligned} \log p(\bar{\mathbf{w}}_{1:3}) &= \log \prod_{i=1}^3 p(\bar{w}_{0:i}) = \sum_{i=1}^3 \log p(\bar{w}_{0:i}) \\ &= -1.7 + -3.3 + -4.7 \\ &= -9.7 \end{aligned}$$

(b) (8 pts) Apply beam search decoding with a beam width  $B = 2$  to this language model, and find  $\mathbf{w}_{1:3}^{(1)}$  and  $\mathbf{w}_{1:3}^{(2)}$ , together with their respective log-likelihoods.

- Time step 1:
  - "the" with joint log probability -1.7
  - "a" with joint log probability -2.0
- Time step 2:
  - Candidates from "the":
    - \* "the green" with joint log probability -5.3
    - \* "the red" with joint log probability -5.0
  - Candidates from "a":
    - \* "a blue" with joint log probability -4.7
    - \* "a red" with joint log probability -4.9

Select the top two: "a blue" and "a red"
- Time step 3:
  - Candidates from "a blue"
    - \* "a blue ellipse" with joint log probability -10.3
    - \* "a blue square" with joint log probability -9.9
  - Candidates from "a red"
    - \* "a red circle" with joint log probability -10.0
    - \* "a red square" with joint log probability -10.2

$$\begin{aligned}w_{1:3}^1 &= \text{"a blue square"} & \log p(w_{1:3}^1) &= -9.9 \\w_{1:3}^2 &= \text{"a red circle"} & \log p(w_{1:3}^2) &= -10.0\end{aligned}$$

5. (3 pts) Please highlight the primary limitation that stands out to you for each of the discussed methods (greedy decoding and beam search).

- **Beam search:**

Beam search with a beam size of 1 is equivalent to a greedy search. But as the beam size increases, the computational cost increases as well, making this a more exhaustive but also more computationally demanding algorithm.

- **Greedy search:**

Greedy search is less computationally demanding, but is also less exhaustive. If the first word it selects leads to a very low probability sequence, for instance, greedy search can lead to subpar results. In other words, greedy search is prone to stalling in local minima.



**Question 3** (17pts). (RNNs)

Consider the following Bidirectional RNN:

$$\begin{aligned} \mathbf{h}_t^{(f)} &= \tanh(\mathbf{W}^{(f)} \mathbf{x}_t + \mathbf{U}^{(f)} \mathbf{h}_{t-1}^{(f)}) \\ \mathbf{h}_t^{(b)} &= \tanh(\mathbf{W}^{(b)} \mathbf{x}_t + \mathbf{U}^{(b)} \mathbf{h}_{t+1}^{(b)}) \\ \mathbf{y}_t &= \mathbf{V}^{(f)} \mathbf{h}_t^{(f)} + \mathbf{V}^{(b)} \mathbf{h}_t^{(b)} \end{aligned}$$

where  $\mathbf{W}^{(f)}, \mathbf{W}^{(b)}, \mathbf{U}^{(f)}, \mathbf{U}^{(b)}, \mathbf{V}^{(f)}, \mathbf{V}^{(b)} \in \mathbb{R}^{d \times d}$ , and  $\mathbf{x}_i, \mathbf{h}_i^{(f)}, \mathbf{h}_i^{(b)} \in \mathbb{R}^d, \forall i \in [T]$  where the superscripts  $f$  and  $b$  correspond to the forward and backward RNNs respectively and  $\tanh$  denotes the hyperbolic tangent function. Let  $\mathbf{z}_t$  be the true target of the prediction  $\mathbf{y}_t$  and consider the sum of squared loss  $L = \sum_t L_t$  where  $L_t = \|\mathbf{z}_t - \mathbf{y}_t\|_2^2$ .

In this question, our goal is to obtain an expression for the gradients  $\nabla_{\mathbf{W}^{(b)}} L$  and  $\nabla_{\mathbf{U}^{(f)}} L$ .

- (2 pts) First, complete the following computational graph for this RNN, unrolled for 3 time steps (from  $t = 1$  to  $t = 3$ ). Label each node with the corresponding hidden unit and each edge with the corresponding weight. Note that it includes the initial hidden states for both the forward and backward RNNs.

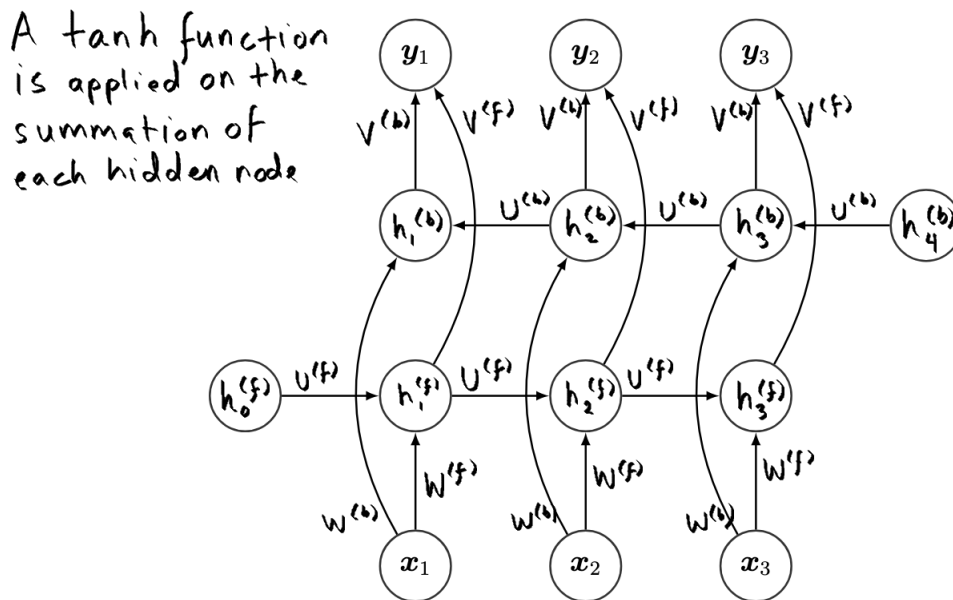


FIGURE 1 – Computational graph of the bidirectional RNN unrolled for three timesteps.

2. (10 pts) Using total derivatives we can express the gradients  $\nabla_{\mathbf{h}_t^{(f)}} L$  and  $\nabla_{\mathbf{h}_t^{(b)}} L$  recursively in terms of  $\nabla_{\mathbf{h}_{t+1}^{(f)}} L$  and  $\nabla_{\mathbf{h}_{t-1}^{(b)}} L$  as follows:

$$\begin{aligned}\nabla_{\mathbf{h}_t^{(f)}} L &= \nabla_{\mathbf{h}_t^{(f)}} L_t + \left( \frac{\partial \mathbf{h}_{t+1}^{(f)}}{\partial \mathbf{h}_t^{(f)}} \right)^\top \nabla_{\mathbf{h}_{t+1}^{(f)}} L \\ \nabla_{\mathbf{h}_t^{(b)}} L &= \nabla_{\mathbf{h}_t^{(b)}} L_t + \left( \frac{\partial \mathbf{h}_{t-1}^{(b)}}{\partial \mathbf{h}_t^{(b)}} \right)^\top \nabla_{\mathbf{h}_{t-1}^{(b)}} L\end{aligned}$$

- (a) (8 pts) Derive the expression for  $\nabla_{\mathbf{h}_t^{(f)}} L_t$ ,  $\nabla_{\mathbf{h}_t^{(b)}} L_t$ ,  $\left( \frac{\partial \mathbf{h}_{t+1}^{(f)}}{\partial \mathbf{h}_t^{(f)}} \right)_{ij}$  and  $\left( \frac{\partial \mathbf{h}_{t-1}^{(b)}}{\partial \mathbf{h}_t^{(b)}} \right)_{ij}$ .

- i.  $\nabla_{\mathbf{h}_t^{(f)}} L_t$ :

$$\begin{aligned}\nabla_{\mathbf{h}_t^{(f)}} L_t &= \frac{\partial L_t}{\partial \mathbf{h}_t^{(f)}} \\ &= \frac{\partial}{\partial \mathbf{h}_t^{(f)}} \|\mathbf{z}_t - \mathbf{y}_t\|_2^2 \\ &= -2(\mathbf{z}_t - \mathbf{y}_t)^T \frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t^{(f)}} \\ \frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t^{(f)}} &= \frac{\partial}{\partial \mathbf{h}_t^{(f)}} \left( \mathbf{V}^{(f)} \mathbf{h}_t^{(f)} + \mathbf{V}^{(b)} \mathbf{h}_t^{(b)} \right) \\ &= \mathbf{V}^{(f)} \\ \implies \nabla_{\mathbf{h}_t^{(f)}} L_t &= -2(\mathbf{z}_t - \mathbf{y}_t)^T \mathbf{V}^{(f)}\end{aligned}$$

- ii.  $\nabla_{\mathbf{h}_t^{(b)}} L_t$ :

$$\begin{aligned}\nabla_{\mathbf{h}_t^{(b)}} L_t &= -2(\mathbf{z}_t - \mathbf{y}_t)^T \frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t^{(b)}} \\ \frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t^{(b)}} &= \frac{\partial}{\partial \mathbf{h}_t^{(b)}} \left( \mathbf{V}^{(f)} \mathbf{h}_t^{(f)} + \mathbf{V}^{(b)} \mathbf{h}_t^{(b)} \right) \\ &= \mathbf{V}^{(b)} \\ \implies \nabla_{\mathbf{h}_t^{(b)}} L_t &= -2(\mathbf{z}_t - \mathbf{y}_t)^T \mathbf{V}^{(b)}\end{aligned}$$

iii.  $\left(\frac{\partial \mathbf{h}_{t+1}^{(f)}}{\partial \mathbf{h}_t^{(f)}}\right)_{ij} :$

Let  $\left(\mathbf{a}_{t+1}^{(f)}\right)_i = \sum_k \mathbf{W}_{ik}^{(f)} \mathbf{x}_{t+1,k} + \sum_k \mathbf{U}_{ik}^{(f)} \mathbf{h}_{t,k}^{(f)} :$

$$\left(\frac{\partial \mathbf{h}_{t+1}^{(f)}}{\partial \mathbf{h}_t^{(f)}}\right)_{ij} = \frac{\partial}{\partial \left(\mathbf{h}_t^{(f)}\right)_j} \tanh \left( \sum_k \mathbf{W}_{ik}^{(f)} \mathbf{x}_{t+1,k} + \sum_k \mathbf{U}_{ik}^{(f)} \mathbf{h}_{t,k}^{(f)} \right)$$

$$\left(\frac{\partial \mathbf{h}_{t+1}^{(f)}}{\partial \mathbf{h}_t^{(f)}}\right)_{ij} = \frac{\partial}{\partial \left(\mathbf{h}_t^{(f)}\right)_j} \tanh \left( \left(\mathbf{a}_{t+1}^{(f)}\right)_i \right)$$

$$= \frac{\partial}{\partial \left(\mathbf{a}_{t+1}^{(f)}\right)_i} \tanh \left( \left(\mathbf{a}_{t+1}^{(f)}\right)_i \right) \frac{\partial \left(\mathbf{a}_{t+1}^{(f)}\right)_i}{\partial \left(\mathbf{h}_t^{(f)}\right)_j}$$

$$= \left(1 - \tanh^2 \left( \left(\mathbf{a}_{t+1}^{(f)}\right)_i \right)\right) \mathbf{U}_{ij}^{(f)}$$

$$= \left(1 - \left(\mathbf{h}_{t+1}^{(f)}\right)_i^2\right) \mathbf{U}_{ij}^{(f)}$$

iv.  $\left(\frac{\partial \mathbf{h}_{t+1}^{(b)}}{\partial \mathbf{h}_t^{(b)}}\right)_{ij} :$

Similarly,

$$\left(\frac{\partial \mathbf{h}_{t+1}^{(b)}}{\partial \mathbf{h}_t^{(b)}}\right)_{ij} = \left(1 - \left(\mathbf{h}_{t+1}^{(b)}\right)_i^2\right) \mathbf{U}_{ij}^{(b)}$$

(b) (2 pts) Now using prev section, drive the expression for Jacobian matrices  $\frac{\partial \mathbf{h}_{t+1}^{(f)}}{\partial \mathbf{h}_t^{(f)}}$  and  $\frac{\partial \mathbf{h}_{t-1}^{(b)}}{\partial \mathbf{h}_t^{(b)}}$ .

i.  $\frac{\partial \mathbf{h}_{t+1}^{(f)}}{\partial \mathbf{h}_t^{(f)}} :$

$$\begin{aligned} \frac{\partial \mathbf{h}_{t+1}^{(f)}}{\partial \mathbf{h}_t^{(f)}} &= \frac{\partial}{\partial \mathbf{h}_t^{(f)}} \tanh(\mathbf{W}^{(f)} \mathbf{x}_{t+1} + \mathbf{U}^{(f)} \mathbf{h}_t^{(f)}) \\ &= \frac{\partial}{\partial \mathbf{h}_t^{(f)}} \tanh(\mathbf{a}_{t+1}^{(f)}) \quad \text{where } \mathbf{a}_{t+1}^{(f)} = \mathbf{W}^{(f)} \mathbf{x}_{t+1} + \mathbf{U}^{(f)} \mathbf{h}_t^{(f)} \\ &= \frac{\partial}{\partial \mathbf{a}_{t+1}^{(f)}} \tanh(\mathbf{a}_{t+1}^{(f)}) \frac{\partial \mathbf{a}_{t+1}^{(f)}}{\partial \mathbf{h}_t^{(f)}} \\ &= \text{diag}(1 - \tanh^2(\mathbf{a}_{t+1}^{(f)})) \cdot \mathbf{U}^{(f)} \\ &= \text{diag}(1 - (\mathbf{h}_{t+1}^{(f)})^2) \mathbf{U}^{(f)} \end{aligned}$$

ii.  $\frac{\partial \mathbf{h}_{t-1}^{(b)}}{\partial \mathbf{h}_t^{(b)}} :$

Similarly,

$$\frac{\partial \mathbf{h}_{t-1}^{(b)}}{\partial \mathbf{h}_t^{(b)}} = \text{diag}(1 - (\mathbf{h}_{t-1}^{(b)})^2) \mathbf{U}^{(b)}$$

3. (5 pts) Now derive  $\nabla_{\mathbf{W}^{(b)}} L$  and  $\nabla_{\mathbf{U}^{(f)}} L$  as functions of  $\nabla_{\mathbf{h}_t^{(b)}} L$  and  $\nabla_{\mathbf{h}_t^{(f)}} L$ , respectively.  
Hint: It might be useful to consider the contribution of the weight matrices when computing the recurrent hidden unit at a particular time  $t$  and how those contributions might be aggregated.

(a)  $\nabla_{\mathbf{W}^{(b)}} L$ :

$$\begin{aligned}\nabla_{\mathbf{W}^{(b)}} L &= \sum_t \frac{\partial L}{\partial \mathbf{W}^{(b)}} \\ &= \sum_t \frac{\partial L}{\partial \mathbf{h}_t^{(b)}} \frac{\partial \mathbf{h}_t^{(b)}}{\partial \mathbf{W}^{(b)}} \\ &= \sum_t \nabla_{\mathbf{h}_t^{(b)}} L \cdot \frac{\partial}{\partial \mathbf{W}^{(b)}} \tanh \left( \mathbf{W}^{(b)} \mathbf{x}_t + \mathbf{U}^{(b)} \mathbf{h}_{t+1}^{(b)} \right) \\ &= \sum_t \nabla_{\mathbf{h}_t^{(b)}} L \cdot \text{diag} \left( 1 - \left( \mathbf{h}_t^{(b)} \right)^2 \right) \mathbf{x}_t^T\end{aligned}$$

(b)  $\nabla_{\mathbf{U}^{(b)}} L$ :

$$\begin{aligned}\nabla_{\mathbf{U}^{(b)}} L &= \sum_t \frac{\partial L}{\partial \mathbf{U}^{(b)}} \\ &= \sum_t \frac{\partial L}{\partial \mathbf{h}_t^{(b)}} \frac{\partial \mathbf{h}_t^{(b)}}{\partial \mathbf{U}^{(b)}} \\ &= \sum_t \nabla_{\mathbf{h}_t^{(b)}} L \cdot \frac{\partial}{\partial \mathbf{U}^{(b)}} \tanh \left( \mathbf{W}^{(b)} \mathbf{x}_t + \mathbf{U}^{(b)} \mathbf{h}_{t+1}^{(b)} \right) \\ &= \sum_t \nabla_{\mathbf{h}_t^{(b)}} L \cdot \text{diag} \left( 1 - \left( \mathbf{h}_t^{(b)} \right)^2 \right) \left( \mathbf{h}_{t+1}^{(b)} \right)^T\end{aligned}$$

(c)  $\nabla_{\mathbf{W}^{(f)}} L$ :

$$\begin{aligned}\nabla_{\mathbf{W}^{(f)}} L &= \sum_t \frac{\partial L}{\partial \mathbf{W}^{(f)}} \\ &= \sum_t \frac{\partial L}{\partial \mathbf{h}_t^{(f)}} \frac{\partial \mathbf{h}_t^{(f)}}{\partial \mathbf{W}^{(f)}} \\ &= \sum_t \nabla_{\mathbf{h}_t^{(f)}} L \cdot \frac{\partial}{\partial \mathbf{W}^{(f)}} \tanh \left( \mathbf{W}^{(f)} \mathbf{x}_t + \mathbf{U}^{(f)} \mathbf{h}_{t-1}^{(f)} \right) \\ &= \sum_t \nabla_{\mathbf{h}_t^{(f)}} L \cdot \text{diag} \left( 1 - \left( \mathbf{h}_t^{(f)} \right)^2 \right) \mathbf{x}_t^T\end{aligned}$$

(d)  $\nabla_{\mathbf{U}^{(f)}} L$ :

$$\begin{aligned}\nabla_{\mathbf{U}^{(f)}} L &= \sum_t \frac{\partial L}{\partial \mathbf{U}^{(f)}} \\ &= \sum_t \frac{\partial L}{\partial \mathbf{h}_t^{(f)}} \frac{\partial \mathbf{h}_t^{(f)}}{\partial \mathbf{U}^{(f)}} \\ &= \sum_t \nabla_{\mathbf{h}_t^{(f)}} L \cdot \frac{\partial}{\partial \mathbf{U}^{(f)}} \tanh \left( \mathbf{W}^{(f)} \mathbf{x}_t + \mathbf{U}^{(f)} \mathbf{h}_{t-1}^{(f)} \right) \\ &= \sum_t \nabla_{\mathbf{h}_t^{(f)}} L \cdot \text{diag} \left( 1 - \left( \mathbf{h}_t^{(f)} \right)^2 \right) \left( \mathbf{h}_{t-1}^{(f)} \right)^T\end{aligned}$$

**Question 4 (18pts). (Low-rank adaptation)**

In this question, the goal is to study the low-rank adaptation techniques. Low-rank adaptation techniques are parameter-efficient fine-tuning methods that adapt large pre-trained models to specific tasks by adding small, trainable low-rank matrices to the frozen base weights, which is a common practice to reduce the memory usage with slight to zero changes on the performance. Here, you

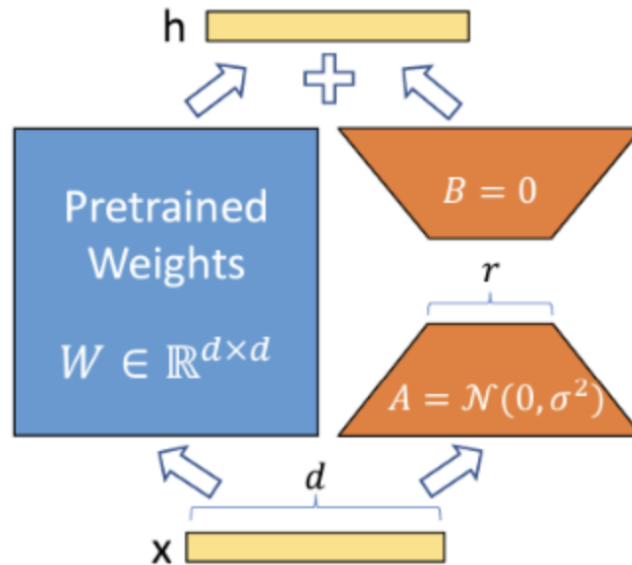


Figure 1: LoRA architecture

will study two well-known low-rank adaptation techniques, LoRA and QLoRA; You can access their papers by clicking [here](#) and [here](#). Provide your answers for the following questions:

1. (2 pts) How do QLoRA and LoRA differ in their implementation of low-rank updates?

**LoRA** freezes the learned weight matrices and adds low-rank weight matrices that are less computationally demanding to update for fine-tuning models efficiently.

**QLoRA** does the same, but quantizes the frozen weight matrices down to a lower precision. For instance, it might lower the frozen weight matrices' precision from FP16 to 4-bit . This allows large models/modules to fit on modest GPUs for fine-tuning but adds a computational overhead at inference time when the frozen weight matrices must be de-quantized back to native precision.

2. (2 pts) How does the choice of rank affect the performance and memory usage in both QLoRA and LoRA?

As shown in the figure above, the low-rank adapted matrices added to the model for fine-tuning in LoRA and QLoRA define the following weights  $W'$ :

$$W' = AB, \quad A \in \mathbb{R}^{d \times r}, \quad B \in \mathbb{R}^{r \times d}$$

We can see that as  $r$  increases, the number of parameters increases. This increases the capacity of the model, allowing for better adaptation during fine-tuning and reducing the risk of under-fitting. However, the resulting improvements are asymptotic: after reaching a certain value of  $r$ , the benefits plateau.

Increasing  $r$  also comes with increased memory usage. From the equation, we see that the total amount of parameters added to the model using LoRA is  $2 \times r \times d$ . Therefore, the memory usage scales linearly with  $r$ .

3. (2 pts) What are the trade-offs in speed and latency between QLoRA and LoRA during inference?

**QLoRA** reduces memory required by quantizing frozen parameters to a lower precision, which adds computational overhead due to the de-quantization required during inference.

In addition, although the model is de-quantized back to native precision, some mixed-precision matrix operations may still occur at inference time, which tends to be slower to compute as most GPUs are optimized specifically for same-precision operations, typically FP16 or FP32.

**LoRA** maintains the model in its native precision, which requires more memory but does not require de-quantization at inference time, contributing to improved latency.

For larger models, **QLoRA** could be faster at inference if parameter storage in memory is more efficient allowing for quicker access (e.g. the model is otherwise too large to fit on a single GPU).

Nevertheless, overall, **LoRA** tends to be quicker than QLoRA at inference, especially for larger batch sizes.

4. (12 pts) In this section, you will examine the differences in time and space complexities when applying LoRA and QLoRA for fine-tuning a Transformer architecture that includes Linear Feed-Forward network (LFFN) and multi-head attention (MHA) layers:

$$\begin{aligned} \mathbf{H}^l &= \text{LFFN}^{(l)}(\text{MHA}^{(l)}(\mathbf{H}^{l-1})) \\ \text{LFFN}^{(l)}(\mathbf{X}) &= \mathbf{X}\mathbf{W}^{F,l} + \mathbf{X}\mathbf{A}^{F,l}(\mathbf{B}^{F,l})^T \\ \text{MHA}^{(l)}(\mathbf{H}) &= [\text{head}^{l,1}(\mathbf{H}), \text{head}^{l,2}(\mathbf{H}), \dots, \text{head}^{l,h}(\mathbf{H})]\mathbf{W}^{O,l} \\ \text{head}^{l,i}(\mathbf{H}) &= \text{softmax}\left(\frac{\mathbf{Q}^{l,i}(\mathbf{K}^{l,i})^T}{\sqrt{d}}\right)\mathbf{V}^{l,i} \\ \mathbf{Q}^{l,i} &= \mathbf{H}\mathbf{W}^{Q,l,i} + \mathbf{H}\mathbf{A}^{Q,l,i}(\mathbf{B}^{Q,l,i})^T \\ \mathbf{K}^{l,i} &= \mathbf{H}\mathbf{W}^{K,l,i} + \mathbf{H}\mathbf{A}^{K,l,i}(\mathbf{B}^{K,l,i})^T \\ \mathbf{V}^{l,i} &= \mathbf{H}\mathbf{W}^{V,l,i} + \mathbf{H}\mathbf{A}^{V,l,i}(\mathbf{B}^{V,l,i})^T \end{aligned}$$

All pretrained parameters ( $\mathbf{W}^*$ ) are frozen, where  $\mathbf{W}^{Q,*}, \mathbf{W}^{K,*}, \mathbf{W}^{V,*}, \mathbf{W}^{F,*} \in \mathbb{R}^{d \times d}$ , while  $\mathbf{W}^{O,*} \in \mathbb{R}^{dh \times d}$ , and  $\mathbf{A}^*, \mathbf{B}^* \in \mathbb{R}^{d \times r}$ . By default, the precision used is 32-bit floating point (FP), while QLoRA first quantizes the pretrained parameters to  $m$ -bit integer precision for fine-tuning, where the quantization of a single matrix  $\mathbf{W}$  is formulated as following:

$$\tilde{\mathbf{W}} = \text{quant}(\mathbf{W}) = \left\lfloor \frac{\mathbf{W}}{\Delta} \right\rfloor, \Delta = \frac{\max(|\mathbf{W}|)}{2^{(m-1)} - 1}$$

and the dequantization of the matrix  $\tilde{\mathbf{W}}$  to the original precision is computed by:

$$\mathbf{W} = \text{dequant}(\Delta, \tilde{\mathbf{W}}) = \Delta \tilde{\mathbf{W}}$$

Given the embedding dimension  $d$ , low-rank dimension  $r \ll d$ ,  $h$  heads, and the context length  $n$ , in a single forward pass:

- (a) (9 pts) Compute the time and space complexities of applying QLoRA and LoRA to FFN and MHA separately. In this problem, space complexity includes only the trainable parameters.  
*Hint: Ensure that operands at each stage are in the same precision.*

**LoRA:**

• **FFN:**

– **Time complexity**

- \* Term 1:  $\mathbf{XW}^{F,l} \implies O(nd^2)$
- \* Term 2:  $\mathbf{XA}^{F,l}(\mathbf{B}^{F,l})^T \implies O(n dr) + O(nrd) = O(n dr)$
- \* **Total time complexity:**  $O(nd^2 + n dr)$

– **Space complexity**

- \* Trainable terms are  $\mathbf{A}^{F,l}, \mathbf{B}^{F,l} \in \mathbb{R}^{d \times r} \implies O(2dr)$

• **MHA:**

– **Time complexity**

- \* Term 1 of  $Q, K, V$ :  $\mathbf{HW}^{Q/K/V,l,i} \implies 3 \cdot O(nd^2) = O(nd^2)$
- \* Term 2 of  $Q, K, V$ :  $\mathbf{HA}^{Q/K/V,l,i}(\mathbf{B}^{Q/K/V,l,i})^T \implies 3 \cdot (O(n dr) + O(nrd)) = O(n dr)$
- \*  $\frac{\mathbf{Q}^{l,i}(\mathbf{K}^{l,i})^T}{\sqrt{d}} \implies O(n^2 d)$  for  $h$  heads  $\implies O(hn^2 d)$
- \*  $\text{softmax}(\dots)\mathbf{V}^{l,i} \implies O(n^2) + O(n^2 d) = O(n^2 d)$  for  $h$  heads  $\implies O(hn^2 d)$ 
  - i.  $O(n^2)$  for softmax
  - ii.  $O(n^2 d)$  for the multiplication with  $\mathbf{V}^{l,i}$
- \* Final output projection:  $O(nd^2)$  for  $h$  heads  $\implies O(hnd^2)$ .
- \* **Total time complexity:**  $O(hnd^2 + n dr + hn^2 d)$

– **Space complexity**

- \* Trainable parameters per head:  $\mathbf{A}^Q, \mathbf{B}^Q, \mathbf{A}^K, \mathbf{B}^K, \mathbf{A}^V, \mathbf{B}^V \in \mathbb{R}^{d \times r} \implies O(6dr)$
- \* For  $h$  heads, total space complexity:  $O(6hdr)$

**QLoRA:**

• **FFN:**

– **Time complexity**

- \* De-quantizing  $\mathbf{W} \implies O(d^2)$
- \* Other terms are same as in LoRA:  $O(nd^2) + O(n dr)$
- \* **Total time complexity:**  $O(d^2 + nd^2 + n dr) = O((n+1)d^2 + n dr) = O(nd^2 + n dr)$

– **Space complexity**

- \* Quantization does not add trainable parameters, so same as LoRA:  $O(2dr)$

• **MHA:**

– **Time complexity**

- \* De-quantizing  $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V, \mathbf{W}^O \implies O(4d^2) = O(d^2)$
- \* Other operations are the same as in LoRA, so  $O(hnd^2 + n dr + hn^2 d)$
- \* **Total time complexity:**  $O(hnd^2 + n dr + hn^2 d + d^2) = O(h(n+1)d^2 + n dr + hn^2 d) = O(hnd^2 + n dr + hn^2 d)$

– **Space complexity**

- \* Quantization does not add trainable parameters, so same as LoRA:  $O(6hdr)$



- (b) (3 pts) If you were assigned to fine-tune using low-rank adapters and had the choice of applying either LoRA or QLoRA to each of these two modules, under the following conditions, which would you select and why? Fill in the table below based on your answer.

**LFFN:**

i. **Memory limit:**

In this particular problem, the LFFN's frozen weight matrix  $\mathbf{W}^{f,l}$  is of size  $d \times d$ , which is smaller than the MHA's weight matrix. Thus, assuming memory is not a bottleneck, we can select **LoRA** as it maintains precision (otherwise select QLoRA).

ii. **Faster inference:**

Assuming no memory bottleneck, **LoRA** is the better choice for inference speed because of QLoRA's de-quantization computational overhead.

iii. **Overall:**

If memory is not a bottleneck, which it shouldn't be because the LFFN in this particular architecture is less memory demanding than the MHA module, **LoRA** is the best option due to faster inference.

**MHA:**

i. **Memory limit:**

In this particular architecture, the MHA's frozen weight matrix  $\mathbf{W}^{o,l}$  is of size  $dh \times d > d \times d$ . This larger size may cause a memory bottleneck, in which case it would be best to use **QLoRA**. Otherwise, hardware permitting, LoRA is a better choice as it maintains precision.

ii. **Faster inference:**

For faster inference, **LoRA** is typically a better choice, although it depends on hardware. For instance, if model parallelization is required for fine-tuning with LoRA but not QLoRA, then QLoRA may be a better option if the de-quantization overhead is offset by faster parameter access from being able to store the entire module on a single GPU.

iii. **Overall:**

**QLoRA** is the better choice overall as it allows users to fine-tune large models on consumer-grade GPUs. If larger GPUs are available and LoRA does not cause a memory bottleneck, then LoRA would be the better choice for faster inference time.

	MHA	LFFN
Memory limit	QLoRA	LoRA
Faster inference	LoRA	LoRA
Both	QLoRA	LoRA