HOMEWORK 1
GENERATIVE MODELS OF TEXT *

10-423/10-623 GENERATIVE AI
http://423.mlcourse.org

OUT: Jan. 27, 2024
DUE: Feb. 07, 2024
TAs: Advaith, Afreen, Haohui, Tiancheng

Instructions

- **Collaboration Policy**: Please read the collaboration policy in the syllabus.
- **Late Submission Policy:** See the late submission policy in the syllabus.
- **Submitting your work:** You will use Gradescope to submit answers to all questions and code.
  - **Written:** You will submit your completed homework as a PDF to Gradescope. Please use the provided template. Submissions can be handwritten, but must be clearly legible; otherwise, you will not be awarded marks. Alternatively, submissions can be written in LaTeX. Each answer should be within the box provided. If you do not follow the template or your submission is misaligned, your assignment may not be graded correctly by our AI assisted grader.
  - **Programming:** You will submit your code for programming questions to Gradescope. There is no autograder. We will examine your code by hand and may award marks for its submission.
- **Materials:** The data that you will need in order to complete this assignment is posted along with the writeup and template on the course website.

Question	Points
Recurrent Neural Network (RNN) Language Models	7
Transformer Language Models	15
Sliding Window Attention	8
Programming: RoPE and GQA	24
Code Upload	0
Collaboration Questions	2
Total:	56

*Compiled on Sunday 11th February, 2024 at 04:54

1 Recurrent Neural Network (RNN) Language Models (7 points)

1.1. (3 points) **Numerical answer:** Consider an RNN (Elman Network) that takes inputs $\mathbf{x}_t \in \{0,1\}^2$, has hidden vectors $\mathbf{h}_t \in \mathbb{R}^2$, and output units $y_t \in \mathbb{R}$ for all $t \in \{1,\ldots,T\}$. Assume the recurrence is given by:

$$h_t = \text{slide}(W_{hh}h_{t-1} + W_{hx}x_t + b_h)$$
$$y_t = \text{slide}(W_{yh}h_t + b_y)$$

where $\text{slide}(a) = \min(1, \max(0,a))$ is the activation function. Define parameters $W_{hh} \in \mathbb{R}^{2\times 2}, W_{hx} \in \mathbb{R}^{2\times 2}, W_{yh} \in \mathbb{R}^{1\times 2}, b_h \in \mathbb{R}^2, b_y \in \mathbb{R}$ to satisfy the following condition: $y_t = 1$ if $\exists\, r, s \leq t$ such that $x_{r,0} = 1$ and $x_{s,1} = 1$ and $y_t = 0$ otherwise. Assume $h_0 = [0,0]^T$.

Let:

$$W_{hh} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad W_{hx} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad b_h = 0 \qquad W_{hy} = \begin{bmatrix} 1 & 1 \end{bmatrix} \qquad b_y = -1$$

This simplifies the equations to $h_t = \text{slide}(h_{t-1} + x_t)$ and $y_t = \text{slide}(\begin{bmatrix} 1 & 1 \end{bmatrix} h_t + b_y)$.
Given $h_0 = \begin{bmatrix} 0 & 0 \end{bmatrix}$ and $h_t = \text{slide}(h_{t-1} + x_t)$, h_t will contain a 1 in position i if and only if for some x_r where $r \leq t$, $x_{r,i} = 1$ for $i \in \{0,1\}$. Additionally once $h_{t,i}$ becomes 1, all $h_{v,i}$ for $v \geq t$ will also equal 1.
Given $W_{hy} = \begin{bmatrix} 1 & 1 \end{bmatrix}$ and $b_y = -1$, $y_t = \text{slide}(W_{yh}h_t + b_y)$ can be rewritten as $y_t = \text{slide}(\sum_{i=0}^{1}(h_{t,i}) - 1)$. Using the matrices given above, $h_{t,i} \in \{0,1\}$ where $i \in \{0,1\}$. When $h_t = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, or $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$ y_t evaluates to 0. If $h_t = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ then $y_t = 1$ satisfying the constraints given.

1.2. An autoregressive language model defines a probability distribution over sequences $\mathbf{x}_{1:T}$ of the form: $p(\mathbf{x}_{1:T}) = \prod_{t=1}^{T} p(x_t \mid x_1, \ldots, x_{t-1})$.

(a) (2 points) **Short answer:** Suppose we are given an input $\mathbf{x}_{1:T}$ and we define a bidirectional RNN of the following form:

$$f_t = \sigma(W_{ff}f_{t-1} + W_{fx}x_t + b_f), \quad \forall t \in \{1,\ldots,T\}$$
$$g_t = \sigma(W_{gg}g_{t+1} + W_{gx}x_t + b_g), \quad \forall t \in \{1,\ldots,T\}$$
$$h_t = \sigma(W_{hf}f_t + W_{hg}g_t + b_g), \quad \forall t \in \{1,\ldots,T\}$$

(Notice that f_t builds up context from the left, g_t builds up context from the right, and h_t combines the two.) Can we define an autoregressive language model of the form $p(\mathbf{x}_{1:T}) = \prod_{t=1}^{T} p(x_t \mid h_{t-1})$? If so, define the probability distribution. If not, why not?

No because an autoregressive language model defines the probability of the current input as being dependent on all previous. However, h_t depends on all inputs future and past. Hence, $\prod_{t=1}^{T} p(x_t \mid h_{t-1}) \neq \prod_{t=1}^{T} p(x_t \mid x_1, \ldots, x_{t-1})$.

(b) (2 points) **Short answer:** Suppose BiRNN$(\mathbf{x}_{1:t-1})$ computes a bidirectional RNN on the subsequence $\mathbf{x}_{1:t-1}$ and then returns h_{t-1}. Can we define an autoregressive language model of the form $p(\mathbf{x}_{1:T}) = \prod_{t=1}^{T} p(x_t \mid \text{BiRNN}(\mathbf{x}_{1:t-1})$? If so, define the probability distribution. If not, why not?

Yes because BiRNN$(\mathbf{x}_{1:t-1})$ contains info on all the $\mathbf{x}$ from time 1 to time $t-1$. Thus it is possible to represent $\prod_{t=1}^{T} p(x_t \mid x_1, \ldots, x_{t-1})$ as $\prod_{t=1}^{T} p(x_t \mid \text{BiRNN}(\mathbf{x}_{1:t-1}))$.
The probability distribution model is: $p(x_t \mid \text{BiRNN}(\mathbf{x}_{1:t-1})) = p(x_t|h_{t-1})$ where h_i is dependent on $x_0, ..., x_{t-1} \forall i \in 1...(t-1)$.

2 Transformer Language Models (15 points)

2.1. (2 points) Transformers use scaled-dot-product attention:

$$s_{t,j} = \mathbf{k}_j^T \mathbf{q}_t / \sqrt{|\mathbf{k}|}, \forall j, t$$
$$\mathbf{a}_t = \text{softmax}(\mathbf{s}_t), \forall t$$

where the values, queries, and keys are respectively given by: $\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$, $\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j$, and $\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j$ for all j and $\mathbf{v}_j, \mathbf{v}_q, \mathbf{v}_k \in \mathbb{R}^{d_k}$.

(a) (2 points) **Short answer:** Multiplicative attention instead defines the attention weights as:

$$\tilde{s}_{t,j} = \mathbf{k}_j^T \mathbf{W}_s \mathbf{q}_t / \sqrt{|\mathbf{k}|}, \forall j, t$$
$$\tilde{\mathbf{a}}_t = \text{softmax}(\tilde{\mathbf{s}}_t), \forall t$$

where $\mathbf{W}_s \in \mathbb{R}^{d_k \times d_k}$ is a parameter matrix. Could a Transformer with multiplicative attention learn a different class of functions than the simpler scaled-dot-product attention? Briefly justify your answer.

No because the multiplicative attention model can be thought of as a transformed version of the scaled dot-product attention model. In particular, if $\mathbf{W}_s \mathbf{q}_t$ is thought of as a linear transformation of $\mathbf{q}_t$. Then the equation for $\tilde{s}_{t,j}$ becomes almost identical to that of $s_{t,j}$. Instead of using the scalar dot product between $\mathbf{k}_j^T$ and $\mathbf{q}_t$, the multiplicative model uses the scalar dot product between $\mathbf{k}_j^T$ and the transformation of $\mathbf{q}_t$ by the matrix $\mathbf{W}_s$.

(b) (2 points) **Short answer:** Concatenated attention defines the attention weights as:

$$\hat{s}_{t,j} = \mathbf{w}_s^T [\mathbf{k}_j; \mathbf{q}_t], \forall j, t$$
$$\hat{\mathbf{a}}_t = \text{softmax}(\hat{\mathbf{s}}_t), \forall t$$

where $\mathbf{w}_s \in \mathbb{R}^{2d_k}$ is a parameter vector, and $[\mathbf{a}; \mathbf{b}]$ is the concatenation of vectors $\mathbf{a}$ and $\mathbf{b}$. Do there exist parameters $\mathbf{w}_s$ such that $s_{t,j}$ will approximately equal the angle θ between the two vectors $\mathbf{k}_j, \mathbf{q}_t$, or to $\cos(\theta)$? (Briefly justify your answer—a formal proof is not required.)

There does not exist parameters such that $s_{t,j}$ will approximate the angle θ. This is because there is no way for a scalar dot product to approximate the non-linear cosine function.

(c) (2 points) **Short answer:** Additive attention defines the attention weights as:

$$\hat{s}_{t,j} = \mathbf{w}_s^T \tanh(\mathbf{W}_s[\mathbf{k}_j; \mathbf{q}_t]), \forall j, t$$
$$\hat{\mathbf{a}}_t = \text{softmax}(\hat{\mathbf{s}}_t), \forall t$$

where the parameters are $\mathbf{w}_s \in \mathbb{R}^{d_s}$ and $\mathbf{W}_s \in \mathbb{R}^{d_s \times d_s}$, dimensionality d_s is a hyperparameter, and $[\mathbf{a}; \mathbf{b}]$ is the concatenation of vectors $\mathbf{a}$ and $\mathbf{b}$. Do there exist parameters $\mathbf{w}_s, \mathbf{W}_s$ such that $s_{t,j}$ will approximately equal the angle θ between the two vectors $\mathbf{k}_j, \mathbf{q}_t$, or to $\cos(\theta)$? (Briefly justify your answer—a formal proof is not required.)

Yes, $\tanh(\mathbf{W}_s[\mathbf{k}_j; \mathbf{q}_t])$ can be used to approximate $\cos(\theta)$ since $\mathbf{w}_s^T \tanh(\mathbf{W}_s[\mathbf{k}_j; \mathbf{q}_t])$ can be thought of as a layer in a neural network. Since a hidden layer in a neural network can be thought of as a universal function approximator, it can be trained to approximate $cos(\theta)$ where θ is the angle between $\mathbf{k}_j$ and $\mathbf{q}_t$.

2.2. Self-attention is typically computed via matrix multiplication. Here we consider multi-headed attention without a causal attention mask.

$$\begin{aligned}
\mathbf{X} &= [\mathbf{x}_1, \ldots, \mathbf{x}_N]^T \\
\mathbf{V}^{(i)} &= \mathbf{X}\mathbf{W}_v^{(i)} \\
\mathbf{K}^{(i)} &= \mathbf{X}\mathbf{W}_k^{(i)} \\
\mathbf{Q}^{(i)} &= \mathbf{X}\mathbf{W}_q^{(i)} \\
\mathbf{S}^{(i)} &= \mathbf{Q}^{(i)}(\mathbf{K}^{(i)})^T / \sqrt{d_k} \\
\mathbf{A}^{(i)} &= \text{softmax}(\mathbf{S}^{(i)}) \\
\mathbf{X}'^{(i)} &= \mathbf{A}^{(i)}\mathbf{V}^{(i)} \\
\mathbf{X}' &= \text{concat}(\mathbf{X}'^{(1)}, \ldots, \mathbf{X}'^{(h)})
\end{aligned}$$

where N is the sequence length, h is the number of attention heads, and each row involving i is defined $\forall i \in \{1, \ldots, h\}$.

(a) (3 points) **Short answer:** Is the attention matrix $\mathbf{A}^{(i)}$ always symmetric? If yes, show that it is. If not, describe a condition that would ensure it is symmetric.

The attention matrix is not always symmetric. One way to ensure $\mathbf{A}^{(i)}$ is always symmetric is to enforce the condition $\mathbf{W}_k^{(i)} = \mathbf{W}_q^{(i)}$. This condition would cause $\mathbf{K}^{(i)}$ to equal $\mathbf{Q}^{(i)}$. Hence $\mathbf{Q}^{(i)}(\mathbf{K}^{(i)})^T$ would be equivalent to $\mathbf{Q}^{(i)}(\mathbf{Q}^{(i)})^T$ which is symmetric.

(b) (4 points) **Short answer:** Suppose we have two attention heads, $h = 2$, we let $d_k = d_m/h$, and we have a single input $\mathbf{X}$. Let $\mathbf{X}'$ be the output of multi-headed attention on $\mathbf{X}$ with the parameters:

$$\mathbf{W}_v^{(1)}, \mathbf{W}_k^{(1)}, \mathbf{W}_q^{(1)}, \mathbf{W}_v^{(2)}, \mathbf{W}_k^{(2)}, \mathbf{W}_q^{(2)} \in \mathbb{R}^{d_m \times d_k}$$

Now suppose we take those same parameters and concatenate along the rows to yield new parameters:

$$\mathbf{W}_v' = \text{concat}(\mathbf{W}_v^{(1)}, \mathbf{W}_v^{(2)}),\ \mathbf{W}_k' = \text{concat}(\mathbf{W}_k^{(1)}, \mathbf{W}_k^{(2)}),\ \mathbf{W}_q' = \text{concat}(\mathbf{W}_q^{(1)}, \mathbf{W}_q^{(2)}) \in \mathbb{R}^{d_m \times d_m}$$

And let $\mathbf{X}''$ be the output of single-headed attention on $\mathbf{X}$ with the parameters $\mathbf{W}_v', \mathbf{W}_k', \mathbf{W}_q'$.

In this case, does $\mathbf{X}'' = \mathbf{X}'$? Justify your answer.

No, because when calculating $\mathbf{S}^{(i)}$ the matrices $\mathbf{Q}^{(i)}\mathbf{K}^{(i)}$ is scaled by $\sqrt{d_k}$. When there are two heads $d_k = d_m/2$. When there is one head $d_k = d_m$, thus $S \neq S'$ where S is calculated using multi-headed attention and S' is calculated using single-headed attention. This means that the attention matrix between the multi-headed and single-headed attention models won't be the same which means X' and X'' are not the same.

3 Sliding Window Attention (8 points)

3.1. The simplest way to define sliding window attention is by setting the causal mask $\mathbf{M}$ to only include a window of $\frac{1}{2}w + 1$ tokens, with the rightmost window element being the current token (i.e. on the diagonal). Then our attention computation is:

$$\mathbf{X}' = \text{softmax}((\mathbf{Q}\mathbf{K}^T/\sqrt{d_k}) + \mathbf{M})\mathbf{V} \tag{1}$$

For example, if we have a sequence of length $N = 6$, and window size $w = 4$, then our mask matrix is:

$$\mathbf{M} = \begin{bmatrix} 0 & -\infty & -\infty & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty & -\infty & -\infty \\ 0 & 0 & 0 & -\infty & -\infty & -\infty \\ -\infty & 0 & 0 & 0 & -\infty & -\infty \\ -\infty & -\infty & 0 & 0 & 0 & -\infty \\ -\infty & -\infty & -\infty & 0 & 0 & 0 \end{bmatrix}$$

(a) (1 point) **Short answer:** If we implement sliding window using the matrix multiplications described in Equation 1, what is the time complexity in terms of N and w? (For this and subsequent questions, assume that the cost of multiplying two matrices $\mathbf{X} \in \mathbb{R}^{m \times n}$ and $\mathbf{Y} \in \mathbb{R}^{n \times p}$ is $O(mnp)$.)

$O(N^2)$
assuming d_k
constant.

(b) (1 point) **Short answer:** If we implement sliding window using the matrix multiplications described in Equation 1, what is the space complexity in terms of N and w?

N
assuming d_m
constant.

(c) (4 points) **Pseudocode:** Write pseudocode/math for a function that takes in the queries, keys, and values and the window size w and computes the $\mathbf{X}'$:

$$\textsc{ScaledDotProductAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}, d_k, w)$$

Your pseudocode/math must have lower asymptotic computational than the naive matrix multiplication approach described above. Your solution can and should include for loops. Assume access to a function softmax$(\mathbf{x})$ which applies softmax to a vector $\mathbf{x}$ and a function tensor$(\cdot)$ that can be used to contruct vectors, matrices, tensors of arbitrary shape.

Input: $\mathbf{Q}, \mathbf{K}, \mathbf{V}, d_k, w$
Output: softmax$((\mathbf{QK}^T/\sqrt{d_k}) + \mathbf{M})\mathbf{V}$

$QK \leftarrow$ tensor$(N \times (\frac{w}{2} + 1))$
for $i = 1$ to N **do**
  for $j = max(0, i - (\frac{w}{2} + 1))$ to i **do**
    $QK_{\{i,i-j\}} \leftarrow (Q_{\{i,:\}} \cdot K_{\{j,:\}})/\sqrt{d_k}$
$QK \leftarrow$ softmax(QK)
$QKV \leftarrow$ tensor$(N \times d_k)$
for $i = 1$ to N **do**
  for $j = 1$ to d_k **do**
    $idx0 \leftarrow max(0, i - (\frac{w}{2} + 1))$
    $QKV_{i,j} \leftarrow QK \cdot V_{\{idx0:i,j\}}$
return (QKV)

(d) (1 point) **Short answer:** What is the space complexity of your pseudocode in terms of N and w?

$Nd_k \rightarrow N$ assuming constant d_k.

(e) (1 point) **Short answer:** What is the time complexity of your pseudocode in terms of N and w?

$O(Nw)$ assuming constant d_k .

4 Programming: RoPE and GQA (24 points)

Introduction

In this section, you will take a run-of-the-mill GPT model and upgrade it to incorporate two of the key ingredients found in state-of-the-art large language models (LLMs), such as LLAMA-2.

The first ingredient are rotary position embeddings (RoPE). These will replace the existing absolute position embeddings with a relative position embedding that rotates small segments of each key and query vector.

The second ingredient is grouped-query attention (GQA). Although the GQA mechanism is fundamentally still causal attention, it enables the model to use less memory and run faster.

You will experiment with how these two model improvements lead to changes in model performance. And you will even evaluate how they perform in tandem.

Upon completion of this section, you will unfortunately not be able to claim to have trained a *large* language model, for the dataset we provide here (the complete works of Shakespeare) is rather small if not trite. However, you can reasonably claim to have built your own LLAMA-2 model.

Dataset

The dataset for this homework is a collection of the complete works of Shakespeare. The dataset file is `input.txt`, and is around 1.1MB in size.

Starter Code

The starter code was originally authored by Andrej Karpathy, of OpenAI fame, and released as minGPT. It offers a clear glimpse into the inner workings of a GPT model. We have simplified the codebase and provided to you a modified version. Ours contains the following files:

```
hw1/
    requirements.txt
    input.txt
    chargpt.py
    mingpt/
        model.py
        trainer.py
        utils.py
```

Here is what you will find in each file:

1. `requirements.txt`: A list of packages that need to be installed for this homework. This homework only requires 2 packages - `torch` and `einops`.
2. `input.txt`: The dataset—the works of Shakespeare.
3. `chargpt.py`: The main entry point used to train your transformer. It can be run with the command `python chargpt.py`. Append flags to this command to adjust the transformer configuration.
4. `mingpt/model.py`: The only file you need to modify for this homework. This file contains the construction of the GPT model. A vanilla, working transformer implementation

is already provided. You will implement the classes RotaryPositionalEmbeddings and GroupedQueryAttention. You will also need to make changes to the class CausalSelfAttention while implementing RoPE. (Hint: Locations in the code where changes ought to be made are marked with a **TODO**.)

5. mingpt/trainer.py: Code for the training loop of the transformer.
6. mingpt/utils.py: Helper functions for saving logs and configs.

Flags

All the parameters printed in the config can be modified by passing flags to chargpt.py. Table 1 contains a list of flags you may find useful while implementing HW1. You can change other parameters as well in a similar manner. Simply specify the config node (i.e. one of {system,data,model,trainer}), followed by a period '.', followed by the parameter you wish to modify.

Configuration Parameter	Example Flag Usage
Model sequence length	--data.block_size=128 (model.block_size is autoset based on this flag)
Directory where model is stored	--system.work_dir=out/new_chargpt
Number of query heads (hyperparameter for GQA)	--model.n_query_head=6
Number of key-value heads (hyperparameter for GQA)	--model.n_kv_head=3 (n_query_head must be divisible by n_kv_head) (For standard multi-head attention n_query_head = n_kv_head)
Directory from which to load a model trained in a previous run	--model.pretrained_folder=out/chargpt3
Whether to enable RoPE embeddings	--model.rope=True
Number of iterations to train the model	--trainer.max_iters=200
Device type (useful for debugging), one of	--trainer.device=cpu

Table 1: Useful flags for chargpt.py

Model

The default model in chargpt.py is a GPT model with 6 transformer layers. Each attention layer uses $h = 6$ attention heads. The maximum sequence length is $N = 128$. Because the vocabulary is comprised of only characters, the vocabulary size is only 65. The embedding dimension is $d_{model} = 192$ and the key/value/query dimension size is $d_k = d_{model}/h = 32$.

Rotary Position Embeddings (RoPE)

In this section, you will implement Rotary Position Embeddings (RoPE) (Su et al., 2021).

Background: Absolute position embeddings are added to the word embeddings in the first layer of a standard Transformer language model. Subsequent layers propagate position information up from the bottom.

Traditional attention is defined as below.

$$\begin{aligned}
\mathbf{q}_j &= \mathbf{W}_q^T \mathbf{x}_j, \forall j \\
\mathbf{k}_j &= \mathbf{W}_k^T \mathbf{x}_j, \forall j \\
s_{t,j} &= \mathbf{k}_j^T \mathbf{q}_t / \sqrt{d_k}, \forall j, t \\
\mathbf{a}_t &= \text{softmax}(\mathbf{s}_t), \forall t
\end{aligned}$$

where $d_k = |\mathbf{k}_j|$ is the size of the query/key/value vectors.

RoPE: Rotary Position Embeddings (RoPE) (Su et al., 2021) incorporate positional information directly into the attention computation, in every layer. If the input to the next attention layer is $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^T$, then we introduce two functions $f_q(\mathbf{x}_j, j)$ and $f_k(\mathbf{x}_j, j)$, which compute the position-aware queries and keys respectively. Then the attention scores are computed as below:

$$\begin{aligned}
\mathbf{q}_j &= \mathbf{W}_q^T \mathbf{x}_j, \forall j & \mathbf{k}_j &= \mathbf{W}_k^T \mathbf{x}_j, \forall j \\
\widetilde{\mathbf{q}}_j &= \mathbf{R}_{\Theta,j} \mathbf{q}_j & \widetilde{\mathbf{k}}_j &= \mathbf{R}_{\Theta,j} \mathbf{k}_j \\
s_{t,j} &= \widetilde{\mathbf{k}}_j^T \widetilde{\mathbf{q}}_t / \sqrt{d_k}, \forall j, t \\
\mathbf{a}_t &= \text{softmax}(\mathbf{s}_t), \forall t
\end{aligned}$$

where $d = d_k/2$, $\mathbf{W}_k, \mathbf{W}_q \in \mathbb{R}^{d_{model} \times d_k}$. For some fixed absolute position m, the rotary matrix $\mathbf{R}_{\Theta,m} \in \mathbb{R}^{d_k \times d_k}$ is given by:

$$R_{\Theta,m} = \begin{pmatrix}
\cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\
\sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\
0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\
0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d_k/2} & -\sin m\theta_{d_k/2} \\
0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d_k/2} & \cos m\theta_{d_k/2}
\end{pmatrix}$$

The θ_i parameters are fixed ahead of time and defined as below.

$$\Theta = \{\theta_i = 10000^{-2^{i-1}/d}, i \in [1, 2, \ldots, d/2]\}$$

Because of the block sparse pattern in $\mathbf{R}_{\theta,m}$, we can efficiently compute the matrix-vector product of $\mathbf{R}_{\theta,m}$ with some arbitrary vector $\mathbf{y}$ in a more efficient manner:

$$\mathbf{R}_{\Theta,m}\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_{d-1} \\ y_d \end{pmatrix} \otimes \begin{pmatrix} \cos m\theta_1 \\ \cos m\theta_1 \\ \cos m\theta_2 \\ \cos m\theta_2 \\ \vdots \\ \cos m\theta_{d/2} \\ \cos m\theta_{d/2} \end{pmatrix} + \begin{pmatrix} -y_2 \\ y_1 \\ -y_4 \\ y_3 \\ \vdots \\ -y_d \\ y_{d-1} \end{pmatrix} \otimes \begin{pmatrix} \sin m\theta_1 \\ \sin m\theta_1 \\ \sin m\theta_2 \\ \sin m\theta_2 \\ \vdots \\ \sin m\theta_{d/2} \\ \sin m\theta_{d/2} \end{pmatrix}$$

Implementing this efficiently in PyTorch still requires some care. If we have some matrix of embeddings $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_N]^T \in \mathbb{R}^{N \times d_k}$ (in practice this $\mathbf{Y}$ would be either the queries $\mathbf{Q}$ or the keys $\mathbf{K}$), then we want to construct a new matrix $\widetilde{\mathbf{Y}} = g(\mathbf{Y}; \Theta)$ such that $\widetilde{\mathbf{Y}}_{m,\cdot} = \mathbf{R}_{\Theta,m}\mathbf{y}_m$. Without loss of generality, we can permute the indices of the vectors $\mathbf{y}_m$ such that we are working with the indices of the first/second half of the vector instead of the even/odd indices. Below let $d = d_k$ for brevity.

$$\begin{aligned}
\widetilde{\mathbf{Y}} &= g(\mathbf{Y};\Theta) \\
&= \left[\begin{array}{ccc|ccc} Y_{1,1} & \cdots & Y_{1,\frac{d}{2}} & Y_{1,\frac{d}{2}+1} & \cdots & Y_{1,d} \\ \vdots & & \vdots & \vdots & & \vdots \\ Y_{N,1} & \cdots & Y_{N,\frac{d}{2}} & Y_{N,\frac{d}{2}+1} & \cdots & Y_{N,d} \end{array}\right] \otimes \left[\begin{array}{ccc|ccc} \cos 1\theta_1 & \cdots & \cos 1\theta_{\frac{d}{2}} & \cos 1\theta_1 & \cdots & \cos 1\theta_{\frac{d}{2}} \\ \vdots & & \vdots & \vdots & & \vdots \\ \cos N\theta_1 & \cdots & \cos N\theta_{\frac{d}{2}} & \cos N\theta_1 & \cdots & \cos N\theta_{\frac{d}{2}} \end{array}\right] \\
&+ \left[\begin{array}{ccc|ccc} -Y_{1,\frac{d}{2}+1} & \cdots & -Y_{1,d} & Y_{1,1} & \cdots & Y_{1,\frac{d}{2}} \\ \vdots & & \vdots & \vdots & & \vdots \\ -Y_{N,\frac{d}{2}+1} & \cdots & -Y_{N,d} & Y_{N,1} & \cdots & Y_{N,\frac{d}{2}} \end{array}\right] \otimes \left[\begin{array}{ccc|ccc} \sin 1\theta_1 & \cdots & \sin 1\theta_{\frac{d}{2}} & \sin 1\theta_1 & \cdots & \sin 1\theta_{\frac{d}{2}} \\ \vdots & & \vdots & \vdots & & \vdots \\ \sin N\theta_1 & \cdots & \sin N\theta_{\frac{d}{2}} & \sin N\theta_1 & \cdots & \sin N\theta_{\frac{d}{2}} \end{array}\right]
\end{aligned}$$

Or more compactly:

$$\mathbf{C} = \left[\begin{array}{ccc|ccc} 1\theta_1 & \cdots & 1\theta_{\frac{d}{2}} & 1\theta_1 & \cdots & 1\theta_{\frac{d}{2}} \\ \vdots & & \vdots & \vdots & & \vdots \\ N\theta_1 & \cdots & N\theta_{\frac{d}{2}} & N\theta_1 & \cdots & N\theta_{\frac{d}{2}} \end{array}\right]$$

$$\begin{aligned}
\widetilde{\mathbf{Y}} =& g(\mathbf{Y};\Theta) \\
=& \left[\begin{array}{c|c} \mathbf{Y}_{\cdot,1:d/2} & \mathbf{Y}_{\cdot,d/2+1:d} \end{array}\right] \otimes \cos(\mathbf{C}) \\
&+ \left[\begin{array}{c|c} -\mathbf{Y}_{\cdot,d/2+1:d} & \mathbf{Y}_{\cdot,1:d/2} \end{array}\right] \otimes \sin(\mathbf{C})
\end{aligned}$$

Now we can compute RoPE embeddings efficiently as below:

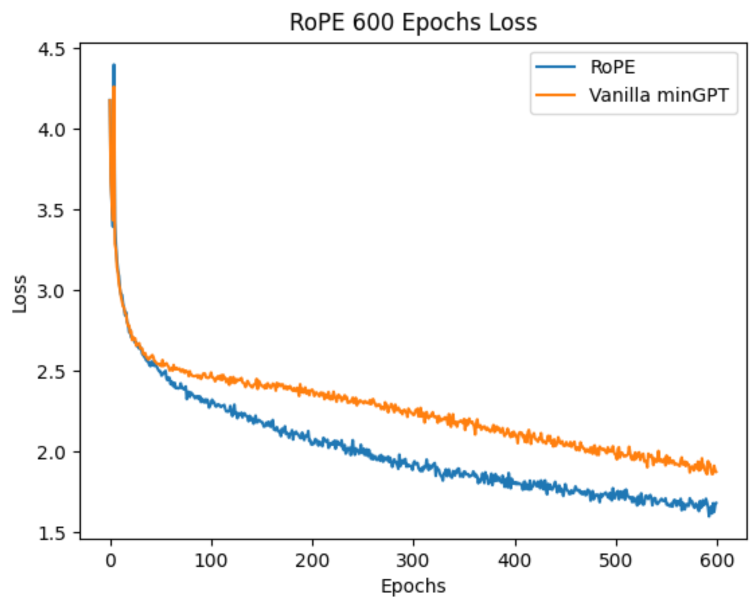
$$\begin{aligned}
\mathbf{Q} &= \mathbf{X}\mathbf{W}_q & \mathbf{K} &= \mathbf{X}\mathbf{W}_k \\
\widetilde{\mathbf{Q}} &= g(\mathbf{Q};\Theta) & \widetilde{\mathbf{K}} &= g(\mathbf{K};\Theta) \\
\mathbf{S} &= \widetilde{\mathbf{Q}}\widetilde{\mathbf{K}}^T/\sqrt{d_k} \\
\mathbf{A} &= \text{softmax}(\mathbf{S})
\end{aligned}$$

You do not have to understand all the math in the paper, but you may go through it to understand the intuition behind RoPE.

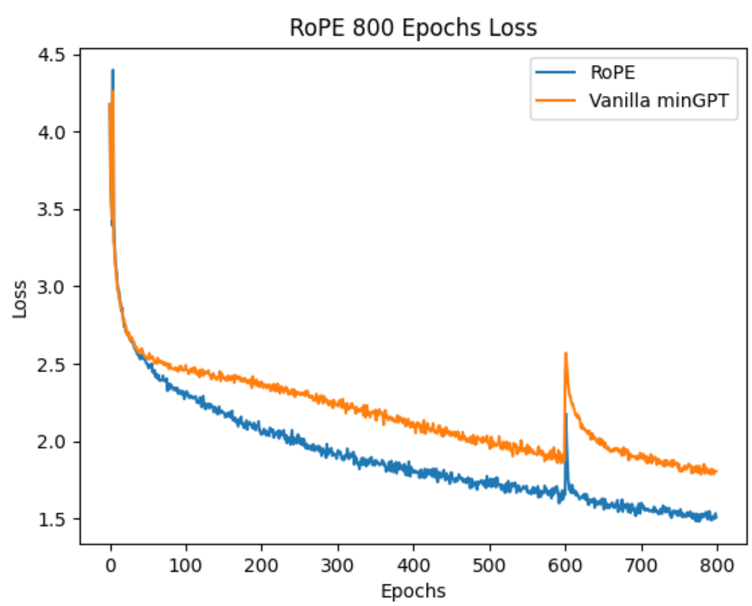
Implementation: You will implement RoPE within `minGPT`. To do so, you should make changes to the `RotaryPositionalEmbeddings` and the `CausalSelfAttention` classes in `mingpt/model.py`.

RoPE Empirical Questions

4.1. (4 points) Plot the training loss of your RoPE implementation and vanilla minGPT over 600 iterations with a sequence length of 128.



4.2. (4 points) Plot the training loss of your RoPE implementation and vanilla minGPT over 800 total training iterations: 600 iterations with a sequence length of 128, followed by 200 iterations with a sequence length of 256.



4.3. (2 points) Provide a sample from your RoPE model after 600 iterations of training with a sequence length of 128. Condition the sample on the first line of your favorite Shakespeare play.

```
When shall we three meet again;
And their agast and bechired the such'd his wook.
I'll me hand him, this dispetion'd had buttain?

DUKE VINCENTIO:
We'll mother hy catcity, and move with must
Thou watuple how foul the swomen and thou wasters,
Which a come, wort is the tome.

CLARENCEN:
Hat but the forsuist, I do the from you are armilled,
O parmant hids wenches of my come so the wise!
Frather the have it both.

LUKE VINCENTIO:
They.

KATHARINA:
A boin, so stay, the seen as they?

COMINIUS:
A heart!
A so herself, well, his hat
```

4.4. (2 points) Provide a sample from your RoPE model after 600 iterations with a sequence length of 128, followed by 200 iterations with a sequence
length of 256. Condition the sample on the first line of your favorite Shakespeare play.

```
When shall we three meet again:
What is the world of heavy duty buishing?

PAULINA:
Wnison, speak you weep you was my brieve?

DUKE VINCENTIO:
Need my fornous, he swood follow of your batter?

CAMILLO:
And itshese from the latter-pass, and will I thark the downs
Who spoke which heavy taken'd.

CHENS OF GONZALO:
And you have must wish yours frotcaling my lord.

MENIAGUS:
I she his hope?
Here to they were would will the must to-lay.

KING EDWARD IV:
Their ghares sorress young of the sense to must
To bethan your fenerate with t
```

Grouped Query Attention (GQA)

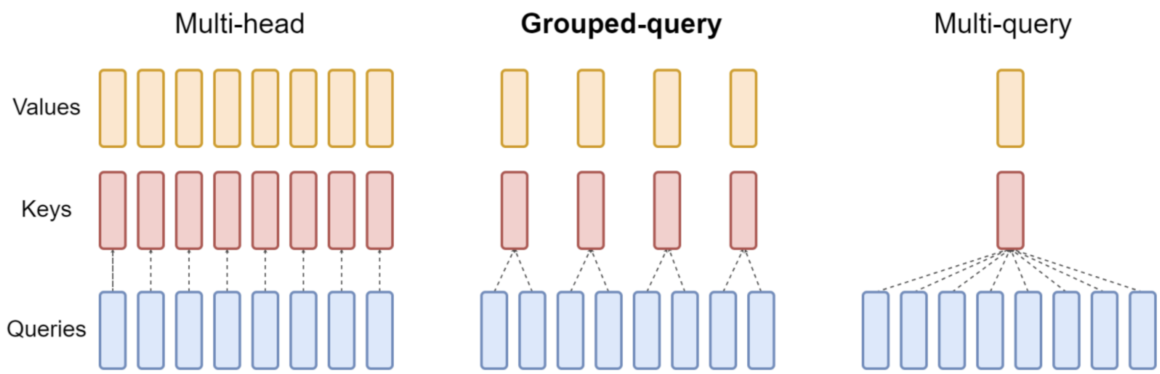


Figure 1: Schematic representation of attention mechanisms, showcasing Multi-head attention with individual keys and values for each head, Grouped-query attention with queries grouped to share common keys and values, and Multi-query attention utilizing a singular key and value for all queries.

In this section, you will implement Grouped Query Attention (GQA) (Ainslie et al., 2023).

GQA: Grouped Query Attention (GQA) is a technique in neural network architectures that modifies the attention mechanism used in models such as transformers. It involves dividing the query heads into groups, each sharing a single key head and value head. This approach can interpolate between Multi-Query Attention (MQA) and Multi-Head Attention (MHA), offering a balance between computational efficiency and model quality [Figure 1].

Let h_q denote the number of query heads and h_{kv} the number of key/value heads. We assume h_q is divisible by h_{kv} and $g = h_q/h_{kv}$ is the size of each group (i.e. the number of query vectors per key/value vector).

Our parameter matrices for GQA are all the same size: $\mathbf{W}_q^{(g,i)}, \mathbf{W}_k^{(g)}, \mathbf{W}_v^{(g)} \in \mathbb{R}^{d_{model} \times d_k}$ where $d_k = d_{model}/h_q$. However, we now have different numbers of query, key, and value heads:

$$\begin{aligned}
\mathbf{X} &= [\mathbf{x}_1, \dots, \mathbf{x}_T]^T \\
\mathbf{V}^{(i)} &= \mathbf{X}\mathbf{W}_v^{(i)}, \forall i \in \{1, \dots, d_{kv}\} \\
\mathbf{K}^{(i)} &= \mathbf{X}\mathbf{W}_k^{(i)}, \forall i \in \{1, \dots, d_{kv}\} \\
\mathbf{Q}^{(i,j)} &= \mathbf{X}\mathbf{W}_q^{(i,j)}, \forall i \in \{1, \dots, d_{kv}\}, \forall j \in \{1, \dots, g\}
\end{aligned}$$

Above, we define g times more query vectors than key/value vectors. Then we compute the scaled dot-product between each query vector (i, j) and its corresponding key (i) and sum over the queries within each group to get the similarity scores. The similarity scores are used to compute an attention matrix, but with only h_{kv} heads:

$$\begin{aligned}
\mathbf{S}^{(i)} &= \sum_{j=1}^{g} \mathbf{Q}^{(i,j)} (\mathbf{K}^{(i)})^T / \sqrt{d_k} \\
\mathbf{A}^{(i)} &= \text{softmax}(\mathbf{S}^{(i)}) \\
\mathbf{X}'^{(i)} &= \mathbf{A}^{(i)} \mathbf{V}^{(i)}
\end{aligned}$$

Implementation Details: You will implement GQA in the `GroupedQueryAttention` class in `mingpt/model.py`. Much of your code will be similar to that in `CausalSelfAttention`.

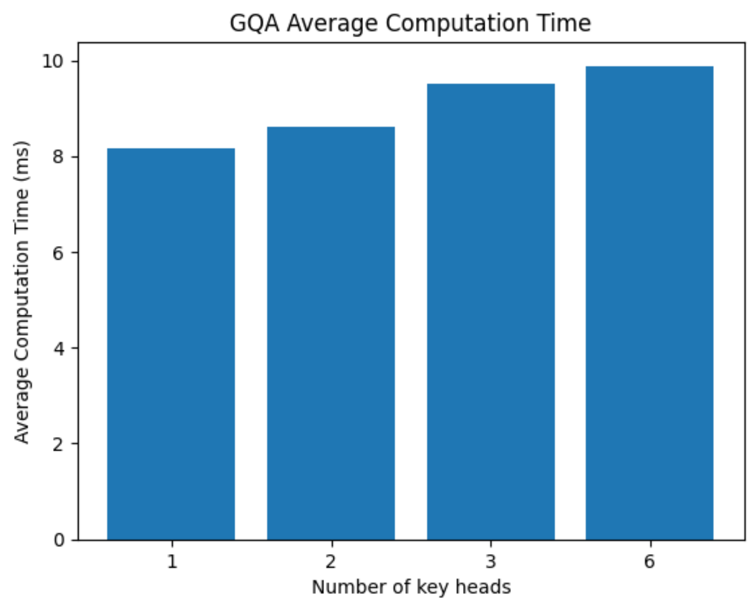
Hint: You may find it easier to first re-implement `CausalSelfAttention` using `einops.rearrange()` in place of `tensor.view()`/`tensor.transpose()` and `einops.einsum()` in place of the `@` operator. If you implement `GroupedQueryAttention` by extending an implementation of this form, it *might* be more straightforward.

- **Initialization:**
  - Familiarize yourself with the configuration settings that initialize the attention mechanism, including the number of query heads, key/value heads, and embedding dimensions.
  - Ensure that the embedding dimension is divisible by the number of query and key/value heads.
- **Regularization:**
  - Incorporate dropout layers for attention and residuals to prevent overfitting.
- **Dimensionality and Projections:**
  - Implement the linear projection layers for queries, keys, and values, considering the dimensionality constraints and the grouped nature of the mechanism.
- **Rotary Positional Embeddings:**
  - If rotary positional embeddings are enabled, integrate RoPE with query and key projections. (Note: Integrating RoPE and GQA is entirely optional, but straightforward.)
- **Forward Pass:**
  - In the forward method, transform the input according to the query, key, and value projections.
  - Apply the attention mechanism by computing grouped scaled dot-product attention
  - Mask the attention to ensure causality (preventing future tokens from being attended to).
  - Aggregate the attention with the values and project the output back to the embedding dimension.
- **Memory Efficiency:**
  - Monitor and record the CUDA memory allocation before and after the attention operation to analyze the memory efficiency of the GQA. A reference code to monitor memory is present in `CausalSelfAttention` class.

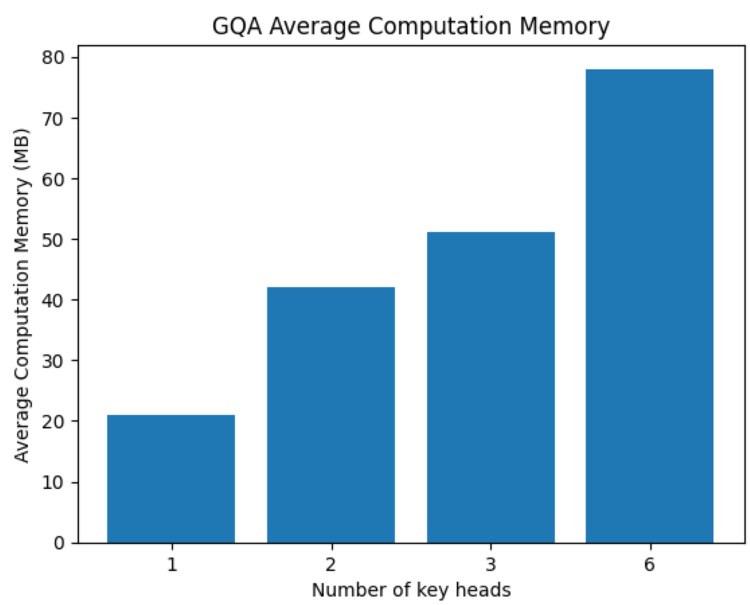
GQA Empirical Questions

The questions below assume you are using absolute position embeddings, not RoPE.

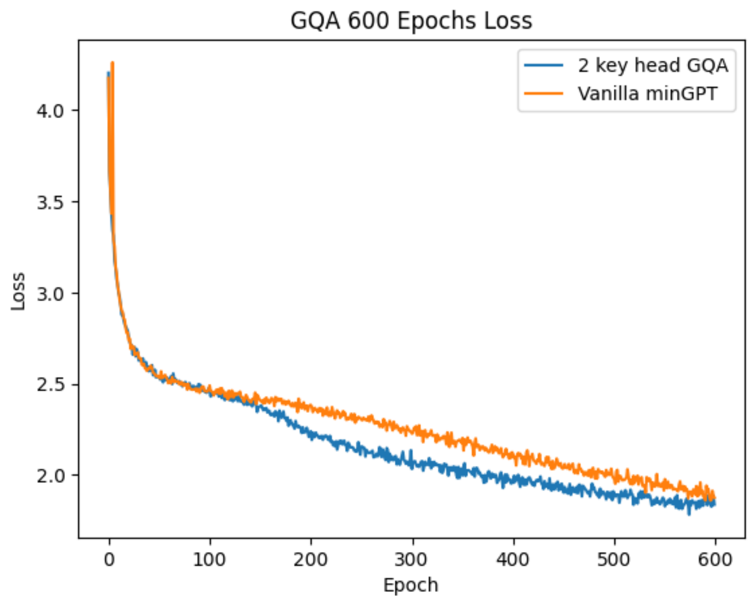
4.5. (4 points) Plot the average time taken to compute attention per iteration in milliseconds across {1, 2, 3, 6} number of key heads.



4.6. (4 points) Plot the memory consumption in MB per iteration across {1, 2, 3, 6} number of key heads.



4.7. (4 points) Plot the training loss of your GQA implementation with 2 key heads vs vanilla (multi head attention) minGPT over 600 iterations with a sequence length of 128.



5 Code Upload (0 points)

5.1. (0 points) Did you upload your code to the appropriate programming slot on Gradescope?
Hint: The correct answer is 'yes'.

- ● Yes
- ○ No

For this homework, you should upload all the code files that contain your new and/or changed code. Files of type `.py` and `.ipynb` are both fine.

6 Collaboration Questions (2 points)

After you have completed all other components of this assignment, report your answers to these questions regarding the collaboration policy. Details of the policy can be found in the syllabus.

6.1. (1 point) Did you collaborate with anyone on this assignment? If so, list their name or Andrew ID and which problems you worked together on.

Gabriel Fonseca (gcfonsec). We worked on questions 1,2,3 and 4 together.

6.2. (1 point) Did you find or come across code that implements any part of this assignment? If so, include full details.

No