Assignment 3
Transformer Mechanics, Application, and Pre-training/Fine-tuning Analysis

STAT8021: Big Data Analytics (Spring 2025)
STAT8307: Natural Language Processing and Text Analysis (Spring 2025)

due: **April 27, 2025, Sunday, 11:59 PM**

Goal

This assignment aims to provide a comprehensive understanding of Transformer models. You will:

- Implement key architectural components to grasp their fundamental mechanics (Part 1).
- Gain practical experience by fine-tuning a pre-trained model for text classification (Part 2).
- Analyze concepts related to model architecture, pre-training, and fine-tuning techniques (Part 3).

Note

You should use Python 3.8+ and PyTorch to finish this assignment. If you need GPUs, we suggest you to use Google Colab (`https://colab.research.google.com/`).

Submission

Please submit the following **two files** to Moodle for grading:

- A PDF report of your answers to all the questions.
- Your Python file: `Transformer_practice.ipynb`.

Total score of this assignment is **100 points**. Please write your procedures in the PDF report if you do not completely finish the code. Note that Part 1 and Part 2 are independent; you do not need to complete Part 1 to start Part 2, or vice versa.

Part 1: Understanding Transformer

The innovation of the Transformer lies in replacing recurrent connections with self-attention mechanisms, allowing for parallel computation and effective capture of long-range dependencies. This part helps you understand the specific architecture of the Transformer by implementing its core components.

Q1 Complete codes of two key building blocks in `Transformer_practice.ipynb`.: Multi-Head Attention and Positional Encoding. **[TOTAL: 30 points]**

(a) Follow the instructions in the Python file, and complete the function `get_multihead_qkv`. Output the shape of the resulting query tensor and its L_2 norm. **[6 points]**

(b) Complete the fuction `calculate_multihead_attention`. Output the shape of the attention weights and the L_2 norm of `self_attn_output` and `masked_self_attn_output`. **[6 points]**

(c) Complete class `MultiHeadAttention`. **Important: Implement the logic directly within the class; do not call the standalone functions defined in Q1(a) and Q1(b).** Output the error of each attention output, which should be no more than `1e-3`. **[12 points]**

(d) Complete class `PositionalEncoding` and print out the `pe_output` error. A correct implementation should output error no more than `1e-3`. **[6 points]**

Part 2: Applying Transformer

You will finetune a pre-trained Transformer model from hugging face[1]. The dataset in this part is part of AG News[2]. You will do a text classification task to predict the news type (0 for World news, 1 for Sports news, 2 for Business news, 3 for Sci/Tech news). Please write your code of this part in `Transformer_practice.ipynb`.

Q2 Finetune the `DistilBertForSequenceClassification`[3] model provided by the hugging face community to predict the type of a given news. **[TOTAL: 40 points]**

(a) The inputs to the transformer model should be tensors. You can use `DistilBertTokenizerFast`[4] to preprocess the `small_ag_news_dataset` we provided in `hf_practice.ipynb`, and then set the dataset as `torch` format. Print the processed first 3 samples in the train set. **[12 points]**

(b) To finetune the model, first you should load the train and test data into `Dataloader`, then define a transformer model with pre-trained weights form `DistilBertForSequenceClassification` and set the number of prediction classes as 4, finally finetune the model and evaluate the performance on the test set. **You are free to tune hyperparameters, including learning rate, batch size, epochs, etc.**. Print the training and testing accuracy for each epoch. **[12 points]**

(c) Test your finetuned model on a small external dataset `chatgpt_generated_new` we provided. Print the predictions and see if the predictions match your human judgement. **[8 points]**

(d) You can choose one other pre-trained transformer model from hugging face community to finetinue. Print the training and testing accuracy for each epoch and compare the performance and efficiency of the models you have finetuned in your report. You may need to restart the Colab runtime and execute only the code cells necessary for this question. **[8 points]**

Hint: If you cannot find an appropriate model from hugging face community, you can try `RobertaTokenizer` and `RobertaForSequenceClassification`[5].

To receive full credit of Q2(b) and Q2(d), you need to get at least 85% **accuracy on the validation set.** Even if you are not able to get this part fully working, write up and document as much as you can so we can give appropriate partial credit.

Part 3: Transformer Pretraining and Finetuning Analysis

Q3 A research consortium is launching "Project Aether" to pre-train a large language model specialized in understanding and generating complex scientific text across multiple domains (physics, biology, chemistry, computer science). Here are more details about the model: **[30 points]**

- **Architecture:** Standard Transformer Decoder-only (GPT-style).
- **Layers (L):** 48
- **Hidden Dimension (H):** 6144
- **Attention Heads (h):** 48 (Note: $\texttt{H/h} = 6144/48 = 128$)
- **Context Length (S_{max}):** $16,384$ tokens

[1] https://huggingface.co/
[2] https://huggingface.co/datasets/ag_news
[3] https://huggingface.co/docs/transformers/v4.27.2/en/model_doc/distilbert
[4] https://huggingface.co/docs/transformers/v4.27.2/en/model_doc/distilbert
[5] https://huggingface.co/docs/transformers/model_doc/roberta

(a) This research consortium decided to pre-train the model using the language modeling task. **[6 points]**

- (I) Please briefly explain the objective of this task. **[2 points]**
- (II) Imagine a simplified scenario for the pre-training. The input is"the cat sat on the [MASK]". The vocabulary is {cat, dog, mat, ran, sat}. For the [MASK] position, the model outputs logits: [0.5, 0.1, 3.0, 0.2, 1.0]. The original word was"mat" (index 2). Calculate the probability the model assigns to the correct word "mat" and cross-entropy loss. **[4 points]**

(b) Please comment on the chosen model architecture by comparing it with the other two architectures (Encoder-only and Encoder-Decoder). What are the potential advantages and disadvantages? **[4 points]**

(c) During the forward pass of a single Transformer layer, calculate the total memory required *just to store the attention score matrix (QK^T)* for a single sequence at the maximum context length ($\mathtt{S}_{\mathtt{max}} = 16,384$), considering all $\mathtt{h} = 48$ attention heads simultaneously. Assume the scores are stored in 16-bit floating-point format (2 bytes per score). Express your answer in Gigabytes (GB). **[4 points]**

(d) Calculate the total number of parameters contributed ***only*** by the Feed-Forward Network (FFN) sub-layers across all L=48 layers. Assume the standard FFN structure includes two linear layers (bias terms included) that transform dimensions $H \to 4H \to H$. Express the result in billions of parameters. **[4 points]**

(e) Consider using LoRA for fine-tuning, specifically adapting only the Query (W_Q) and Value (W_V) projection matrices in the multi-head self-attention blocks of all $\mathtt{L} = 48$ layers, using a rank $\mathtt{r} = 8$. **[12 points]**

- (I) Estimate the total number of *trainable* parameters introduced by LoRA. (Recall LoRA adds two matrices, A and B, per adapted matrix. For a matrix mapping $\mathtt{D}_{\mathtt{in}} \to \mathtt{D}_{\mathtt{out}}$, LoRA adds $\mathtt{r} \times \mathtt{D}_{\mathtt{in}} + \mathtt{D}_{\mathtt{out}} \times \mathtt{r}$ parameters. Here, $\mathtt{D}_{\mathtt{in}} = \mathtt{D}_{\mathtt{out}} = \mathtt{H}$). **[4 points]**
- (II) Calculate the memory required just to store the **Adam optimizer states** (assuming 8 bytes per trainable parameter) for *only these LoRA parameters*. Specifically, we assume Adam stores two state values (momentum and variance) per trainable parameter, and each value is stored as a 32-bit float (4 bytes). **[4 points]**
- (III) How does this LoRA optimizer state memory compare (as a rough percentage or factor) to the optimizer state memory required for just the FFN parameters calculated in Question (d)? **[4 points]**