Deep Learning - Spring 2025

Assignment 4

Domain-Specific Transformer for Question Answering

Due Date: 11:59 PM on Wednesday, 18-May -2025 **Total Marks:** 100

Submission:
Submit all of your codes and results in a single zip file with the name FirstName_RollNumber_04.zip

- Submit a single zip file containing
  (a) codes (b) report (c) Saved Models (d) Readme.txt
- There should be **Report.pdf** detailing your experience and highlighting any interesting results. Kindly don't explain your code in the report, just explain the results. Your report should include your comments on the results of all the steps, with images, for example, what happened when you changed the learning rate, etc.
- Readme.txt should explain how to run your code, preferably it should accept the command line arguments e.g dataset path used for training the model.
- In the root directory, there should be 2 python files, a report, and a folder containing saved models.
- Root directory should be named as **FirstName_RollNumber_04**
- Your code script files should be named as **'rollNumber_04_task1.py'.**
- You have to submit a .py code file**.** Follow all the naming conventions.
- For each convention, there is a 3% penalty if you don't follow it.
- **Email the instructor or TA if there are any questions**. You cannot look at others' code or use others' code, however, you can discuss it with each other. **Plagiarism will lead to a straight zero with additional consequences as well.**
- 10% (of obtained marks) deduction per day for late submission.
- **DON'T RESUBMIT THE DATASETS PROVIDED IN YOUR SUBMISSION.**
- Use the dataset provided for this assignment, do not use any other dataset.
- Marks will be allocated based on viva and written code. Prepare yourself accordingly.

Note: For this assignment (and for others in general) **you are not allowed to search online for any kind of implementation.** Do not share code or look at the other's code. You should not be in possession of any implementation related to the assignment, other than your own. In case of any confusion please contact the TAs (email them or visit them during their office hours).

Objectives: In this assignment, you will implement and train a **small Transformer-based language model** similar specifically tuned for **question answering in programming**.
The goals of this assignment are:

- Understand how transformer architectures work.
- Learn preprocessing techniques for NLP data.
- Train a domain-specific language model from scratch.
- Evaluate model performance both quantitatively and qualitatively.

⚠ **Important:** With your submission, you also need to submit one .py file named **'rollNumber_04_allCode.py'** in your zip folder. This file should contain the code of all 'code files' in your project. NO SUBMISSION WILL BE ACCEPTED WITHOUT IT.

Task-01 Image Captioning Generation

(100 Marks)

1. **Dataset**:

   You are provided with a subset of the **Stack Overflow Python-tagged questions and answers**. The dataset is related to the stackoverflow questions and answers. In the dataset there are three csv files (**encoding = "latin1")** named as Questions.csv, Answers.csv and Tags.csv. There were 3,23,929 questions in the Questions.csv and have 4,46,326 answers of these questions in the Answers.csv file.

   - Each data point contains:
     - A **question title**
     - A **question body** (may contain code)
     - The **accepted answer**

   Format: CSV file with columns: title, body, answer.
   Source: Kaggle / Pre-downloaded subset provided.

   You may combine the title and body into a single input field during preprocessing.

2. **Data Preprocessing:**
   a. **Text Preparation:**
      - Convert all text to lowercase.
      - Remove HTML tags and code formatting.
      - Remove unnecessary special characters.
      - Replace code blocks with [code] token (optional).
      - Use <yourname_Rollnumber> and </ yourname_Rollnumber > tokens to mark the start/end of sequences.

   b. **Split Data**:
      i. Split the data into training, testing and validation.
      ii. Make sure there is no data leakage in the test set while splitting it.
      iii. Note: According to your computational resources select limited samples. DO NOT RUN ON WHOLE DATASET.

3. **Tokenization**:
   a. To input sentence information into the model, convert string questions/answers into numbers.
   b. For that you need a tokenizer. Word2vec is already implemented in the previous assignment, and in this you have to use tokenizer.
   c. You can use any tokenizer to make tokens.
   d. Make sure the tokens generated all are with same sequence.
      i. Take a look into "truncate" and "padding" parameter.
   e. Tokenizer code:
      i. tokenizer = AutoTokenizer.from_pretrained**("openai-community/openai-gpt**")
      ii. You can use any tokenizer, but make sure at the time of training and inference you have to use the same one.
      iii. Read more about the usage from the "Hugging-Face"

4. **Data Loader**
   Create a standard data loader to return the desired input/inputs and labels according to your designed model.

5. **Model Architecture:**
   Implement a custom architecture of a small transformer model as per your understanding. Implement the code attention network as mentioned in the "Attention is all you need".

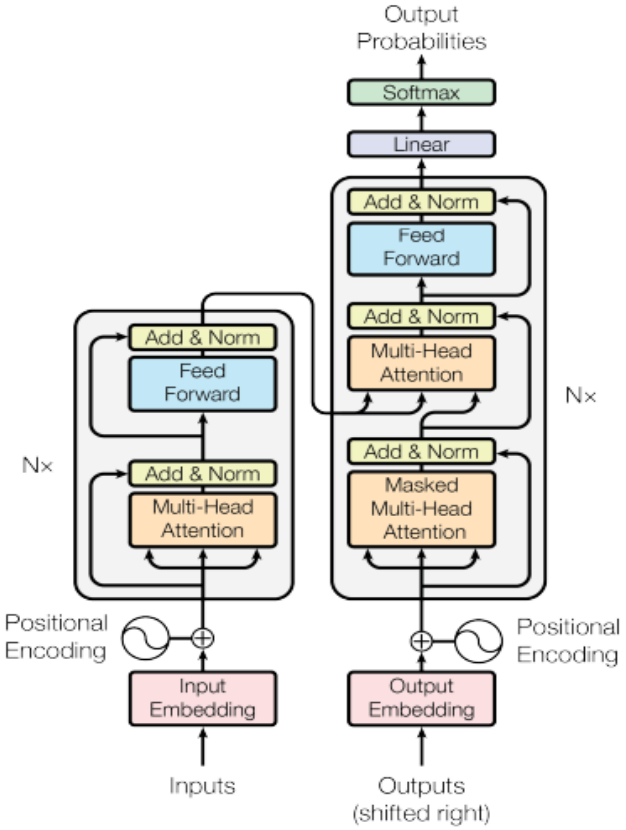

Figure 1: The Transformer Model Architecture.

Read out the concepts of scaled Dot-Product attention and Multi-Head Attention from the paper and implement it withing the architecture. Use a suitable size of the model in which it will return a suitable answer.

You can use built-in layers in Pytorch and other libraries but have to implement the architecture from scratch.

- Components Required:
  - Embedding Layer for input tokens.
  - Positional Encoding to retain order information.
  - Masked Multi-head Self-Attention layers.
  - Feedforward layers, normalization, and residuals.
  - Output projection layer to predict next tokens.

6. **Model Setting**:
   a. Loss Function: CrossEntropyLoss
   b. Optimizer: Adam or AdamW or any suitable if you found any.
   c. Metrics: **Perplexity**, **BLEU**
   d. **Callbacks**:
      i. EarlyStopping when validation loss does not improve.

ii. Checkpointing to save best model weights (on minimum loss and maximum accuracy).
iii. Learning Rate Scheduler (CosineAnnealingLR).

7. **Train & Test Methods**
   Write separate methods for **training** and **testing**. Ensure that your code is modular by dividing it into functions. If any operation is used multiple times, create a reusable function and call it where necessary.

   Evaluation (Answer Generation**)**
   Create a separate file named test_eval.py, where:
   - The best model is loaded.
   - Write an interactive script to:
     - Accept a question as input (title + description).
     - Generate a natural language answer using your trained model.
     - Use sampling or top-k/top-p decoding for generation.

8. **Visualization**
   a. **Graphs**:
      i. Make sure to add loss training and validation graphs for the best experiment.
      ii. Make sure to add metrics training and validation graphs for the best experiment.

Deliverables

```
project/
|-- rollNumber_04_task1.py'      # All code from data-loader to test & visualization.
|-- train.py                     # Training script
|-- test.py                      # Testing Script
|-- model.py                     # Captioning Model class
|-- data_utils.py                # Dataset/dataloaders
|-- weights                      # If weights are too large then upload it on your drive
                                 and share link in submission.
|-- Report.pdf                   # Detailed report including analysis.
|-- graphs                       # contain all the graphs as mentioned.
|-- requirements.txt             # Dependencies
'-- README.md                    # Execution instructions
```

Note

A file with named as "rollNumber_04_allCode.py" contained the code of task-1 must be included in the main directory. Otherwise, mentioned penalty will be applied.

Good Luck