CSC 7700/4700 Foundational AI Project 2

Learning Objective: Students will develop a deeper understanding of sequential deep learning models by implementing Recurrent Neural Networks (RNNs), Long Short-Term Memory networks (LSTMs), and Transformer models using PyTorch. They will train these models on a small language dataset for a generative AI application and evaluate their performance using standard metrics.

Tasks: Students will write a program in **Python 3.10+** using **PyTorch 2.x** to develop, train, and evaluate RNN, LSTM, and Transformer models for a text-generation task. Students are allowed to use PyTorch's high-level functions (e.g., torch.nn.LSTM, torch.nn.Transformer) to implement these models.
The main tasks of this project are:

- **Implement** three models (four for graduate students):
  - A **vanilla RNN-based** language model
  - An **LSTM-based** language model
  - A **Transformer-based** language model
- **Train each model** on a small text dataset for generative AI.
- **Evaluate each model** using:
  - **Perplexity (PPL)** as a measure of how well the model predicts the next word.
  - **BLEU score** to compare generated text against ground truth sentences.
- **Generate text** using each trained model by providing a prompt.
- **Compare the performance** of the models, discussing trade-offs in performance and computational requirements.

Instructions:

Dataset:
The dataset consists of short text sequences extracted from **public domain literature**, specifically, Project Gutenberg. These data were used to build a training and testing dataset, which are available in the course Github.

1. You must train a BPE tokenizer with subword tokenization, using the SentencePiece library, which you will then use to convert the dataset into **tokenized sequences**. The **vocabulary** size should be set to 10000.

Model Implementations:
Your code must define the following models using PyTorch. You are allowed to organize the code as you see fit, unless otherwise specified below.

2. Each model's architecture will include an embedding layer that maps token IDs to token embeddings.
3. Each model 's architecture will include a fully connected output layer that predicts token probabilities.
4. Each model's architecture will have one or more hidden layers, according to the model type:
   a. Recurrent Neural Network (RNN): One or more RNN layers (torch.nn.RNN) between embedding and output layer.
   b. Long Short-Term Memory (LSTM): One or more LSTM layers (torch.nn.LSTM) between embedding and output layer.
   c. Transformer: One or more transformer encoders with 2 or more attention heads between embedding and output layer. The maximum input sequence length should be 512.
5. Each model class (code) must have a ***forward*** method that predicts the vocabulary token probabilities and samples the next token in the sequence, returning that token ID.

   a. **Undergraduate Students**: The generated token should be the one with the highest probability.
   b. **Grad Students**: Your forward method must allow one to specify temperature for sampling.
6. Each model class (code) must have a ***prompt*** method that takes a textual prompt as input, tokenizes it, processes it through the model, and returns the model's reponse:
   a. The response should be autoregressively generated and stop when the model output an end of sequence token OR the sequence hits a maximum length (optional argument to the function, *max_seq_length*).

Training Recommendation:
- Loss Function: CrossEntropyLoss.
- Optimizer: AdamW
- Batch Size: Start at 128.
- Epochs: 30 epochs with early stopping and learning rate scheduler

Evaluation Metrics:
- Compute and report perplexity (PPL) on the test dataset.
- Compute and report BLEU score to measure how well the model generates text similar to the dataset.

Deliverables:

You will prepare a short report in 12 point Calibri font (1/2" margins), composed of the following sections:

Abstract: In one paragraph, summarize your approach and key results.

Methodology: 2 to 5 pages describing your approach to designing, training, and evaluating each model. You must include diagrams for the architecture of each model you design, I recommend using draw.io, but you are welcome to use other tools.

Results: For both applications, use matplotlib to generate plots of the loss curves (both training and validation).

The following should be presented:
- Training/Validation Loss Curve plots for each of the models
- Table containing evaluation metrics for each model on the test dataset
- For each model, show the response for the following prompt: "Which do you prefer? Dogs or cats?"
- Select a prompt of your choosing and show each model's response to the prompt.

Code Repo Link: Provide a link to the Github repository containing all of the code for the project. There should be a README.md file with instructions on how to run the code (this should not be complicated).

Discussion & Conclusion: One half page or less discussing the results and what you learned from the project.