Assignment #4: Transformer and LLMs

Total score: 150
This assignment can be completed individually or by a team of a maximum of three members.
Due date: See the course page

Objectives

This assignment has four major objectives:

(1) Sequence classification using a transformer encoder

(2) Extractive and abstractive text summarization

(3) Parameter-efficient fine-tuning of a pretrained LLM

(4) Retrieval-Augmented Generation (RAG) for LLM

Read the requirements carefully and follow all instructions.

Datasets, Organization of Program Files, and Directories

You will use the following datasets and resources:

- "all_news.csv" that contains (date, symbol, headline, URL, article, publisher)
- "aggregated_news.csv" that contains (date, symbol, news)
- Skip-gram word embeddings created in Assignment #3
- The 2022 CS undergraduate handbook: "cpsc-handbook-2022.pdf"

For this assignment, no template files will be given. All I ask of you is that you follow the names of the files assigned to each task and provide a main.py that runs all the sections of this assignment.

Required Tasks

1 Sequence Classification for Sentiment Analysis

You will build a Transformer encoder from scratch without using pretrained models in PyTorch and use it for sentiment classification.

Complete the program "**transformer_classifier.py**" to implement a Transformer encoder-based classifier.

1.1 Transformer design, training, and evaluation

(a) Model design and training

Design a Transformer encoder-only architecture with PyTorch (not pretrained models).

In the report file, clearly describe your design decision for the following choices:

- Positional embedding choice: one of sinusoidal, learnable, rotary, or relative
- Transformer encoder architecture: number of attention heads, feedforward dimension, number of encoder layers, layer norm strategy (pre-LN or post-LN)
- Classification head choice: CLS token or pooling
- Trainable hyperparameters: batch size, number of epochs

You have precomputed embeddings of news articles, using a Skip-gram model in the previous assignment: vectorized_news_skip_gram_embeddings. Build a classification model by training it on the vectorized sequences.

(b) Evaluate your model and compare the performance of the Transformer classifier and the classifier built in Assignment #3 using Skip-gram embeddings and discuss the followings in your report file:

- Differences in accuracy (% or other metrics)
- Whether the results are comparable
- Whether the Transformer improves performance
- Possible reasons for improvement or lack of improvement (e.g., sequence modeling capability vs. bag-of-words limitations)

2 Text summarization

This part covers extractive summarization and abstractive summarization. You will use one selected article from "all_news.csv".

2.1 Extractive summarization (no training required)

Complete the program "**extractive_summary.py**" to implement an extractive summarization method based on sentence similarity:

- Represent each sentence using Skip-gram embeddings
- Compute pairwise similarity (e.g., cosine similarity)
- Select the most representative sentences to form a short summary

In the report file, briefly describe your method (sentence embeddings strategy, similarity metric, selection method.

Note: Extractive summarization does not require any training, simply computing the sentence similarities.

2.2 Abstractive summarization using a Transformer encoder-decoder

Complete the program "**abstractive_summary.py**" to implement an abstractive summarization method using PyTorch's built-in **nn.Transformer** to create an encoder-decoder architecture.

Treat the summarization task as an article-to-headline generation problem. Each article/headline pair in all_news.csv forms one training example. Create a small training dataset with at least 50 article/headline pairs. A larger dataset would lead to better performance, but we will train the model with this for demonstration purposes.

(a) Transformer model design and training

In the report file, clearly describe the key architectural choices, including:

- Positional embedding choices: one of sinusoidal, learnable, rotary, or relative
- Encoder block: number of heads, feedforward dimension, number of layers, and layer norm strategy
- Decoder block: number of heads, feedforward dimension, number of layers, layer norm strategy, autoregressive masking explanation, generation strategy (greedy, beam search, etc.)

Train the Transformer encoder-decoder to generate the headline from the article (an article-headline pair).

Note: The output is sequence of words, not concatenated embeddings.

(b) Evaluation and discussion

In your report file, pick one article and headline, compare the extractive summary and the abstractive summary of the actual article and headline from all_news.csv, and discuss:

- How similar each summary is to the headline
- Which method produced the most accurate or useful content
- Possible reasons for mismatch (e.g., limited training data, summarization model complexity, etc.)

3 Fine-tuning LLMs

You will fine-tune a small pretrained model: "TinyLlama/TinyLlama-1.1B-Chat-v1.0".
Use the CS undergraduate handbook, "cpsc-handbook-2022.pdf".

Complete the program "**params_finetune.py**" to fine-tune the Tiny Llama LLM using various parameter-efficient tuning methods.

(a) Fine-tune the model using following methods:

- Adapter
- Prefix-tuning
- LoRA
- Full fine-tuning (if memory issues arise, consider using small batch sizes, e.g., 1-4)

For each method, clearly describe in your report file:

- Architecture modification (what changes are made to the model architecture?)
- Number of trainable parameters (how many parameters are updated during training?)
- Expected training efficiency (memory requirements, training time)

(b) Inference-time output control

In your report file, describe the decoding (inference) parameters used to control model output:

- Temperature (randomness: 0.0-0.3 factual, 0.5-0.8 balanced, 0.9-1.2 creative or risky)
- Top-k (only the top-k highest probability tokens at each step)
- Top-p (limit to the top *p* probability)
- Repetition penalty

(c) In your report file, describe the dataset design used for CS undergraduate handbook

Chunk size, chunk overlap, number of instruction-response pairs

The target task and model goal

An example instruction-response pair

(d) In your report file, include the evaluation of fine-tuning models

Create at least three non-trivial questions from the handbook, for example:

- What are the core courses required for a computer science undergraduate degree?
- Describe the rules for completing a senior project, including prerequisites
- What are the degree requirements for graduation?

For each fine-tuned model:

- Generate answers using the model
- Evaluate answers on three criteria: correctness, completeness, and clarity and score or annotate each answer qualitatively (e.g., High/Medium/Low)

Discuss:

- Best-performing method
- Worst-performing method
- Trade-offs between compute cost and correctness

4 Retrieval-Augmented Generation (RAG) with LLMs

You will implement a RAG pipeline using the same undergraduate handbook. Complete the program "**rag_tune.py**" to build a RAG system using the Tiny Llama LLM.

(a) In your report file, briefly describe your RAG system, including:

- Chunk size and chunk overlap
- Vector storage: vector database name
- Retrieval strategy: K value, scoring

(b) In your report file, include your prompt template of RAG

Provide one example of a prompt template used for the RAG system, showing how retrieved context and user queries are combined for the generator.

(c) In your report file, include your evaluation of the RAG-tuned system using the same three questions from section 3:

- Compare answers between the fine-tuned model and the RAG-tuned
- Discuss the similarity, difference, and accuracy
- Trade-offs
- Use cases for RAG

Deliverables

- **One report file** (PDF or Word) that includes:
  - All team member names and % contribution of each member. If every member contributed equally, simply state "equal contribution." If there is disagreement on contributions, provide a brief task description for each member. In this case, different grades may be assigned individually.
  - All answers and results from the required tasks
  - When including example data in your report, show only a few representative samples instead of the entire dataset.

- **All Python program files** you created, **CSV files generated**, and **your .pth files**. DO not upload the original news datasets or Docker images. Please contact Tenshi (the grader) if the total size exceeds 1 GB. In that case, include only a portion of your datasets.

Grading Criteria

- Quality of work shown on the report (e.g., modeling process and results, methods used, and correctness of implementation and analysis)
- Level of understanding of related subjects as reflected in the report
- Effort (10%)