COMP4650/6490 Document Analysis – Semester 2 / 2023

Assignment 3

Due 17:00 on Thursday 12 October 2023 AEDT (UTC +11)

Last updated September 28, 2023

Overview

In this assignment you will:

1. Develop a better understanding of how machine learning models are used in text processing, including text classification and generation.
2. Become familiar with the `PyTorch`[1] framework for implementing transformer based machine learning models.
3. Explore the potential of using pre-trained large language models for information retrieval with the `CTransformers`[2] library.

Throughout this assignment you will make changes to the provided code to improve or complete existing functions. In some cases, you will write your own code from scratch after reviewing an example.

Submission

- You will produce an answers file with your responses to each question. Your answers file must be a PDF file named `u1234567.pdf` where `u1234567` should be replaced with your Uni ID.
- The answers to this assignment (including your code files) have to be submitted online in Wattle.
- You should submit a ZIP file containing all of the code files and your answers PDF file, **BUT NO DATA**.
- **No late submission will be permitted without a pre-arranged extension**. A mark of 0 will be awarded if not submitted by the due date.

Marking

This assignment will be marked out of 100, and it will contribute 10% of your final course mark.

Your answers to coding questions (or coding parts of each question) will be marked based on the quality of your code (is it efficient, is it readable, is it extendable, is it correct) and the solution in general (is it appropriate, is it reliable, does it demonstrate a suitable level of understanding).

Your answers to discussion questions (or discussion parts of each question) will be marked based on how convincing your explanations are (are they sufficiently detailed, are they well-reasoned, are they backed by appropriate evidence, are they clear, do they use appropriate visual aids such as tables, charts, or diagrams where necessary).

This is an individual assignment. Group work is not permitted. Assignments will be checked for similarities.

[1] `https://pytorch.org/`

[2] `https://github.com/marella/ctransformers`

Question 1: Movie Review Sentiment Classification (30%)

For this question you have been provided with a labelled movie review dataset – the same dataset you explored in Lab 3 and Assignment 2. The dataset consists of 50,000 review articles written for movies on IMDb, each labelled with the sentiment of the review – either positive or negative. The overall distribution of labels is balanced (25,000 `pos` and 25,000 `neg`). Your task is to apply a transformer based classifier to predict the sentiment label from the review text, and compare the classification performance with the results you got for Question 1 in Assignment 2.

Specifically, you will train a transformer encoder using `PyTorch`. An input sequence is first tokenised and a special `[CLS]` token prepended to the list of tokens. Similar to BERT[3], the final hidden state (from the transformer encoder) corresponding to the `[CLS]` token is used as a representation of the input sequence in this sentiment classification task.

First, implement the `get_positional_encoding` function in `sentiment_classifier.py`. This function computes the following positional encoding:

$$\mathrm{PE}_{t,2i} = \sin\left(\frac{t}{10000^{\frac{2i}{d}}}\right), \quad \mathrm{PE}_{t,2i+1} = \cos\left(\frac{t}{10000^{\frac{2i}{d}}}\right), \quad t \in \{0, \ldots, T-1\},$$

where d is the dimension, T the length of the sequence and $i \in \{0, \ldots, \frac{d}{2} - 1\}$. You can assume d is an even number for this question.

Next, complete the `__init__` method of class `SentimentClassifier` by creating a `TransformerEncoder`[4] consists of a few (determined by parameter `num_tfm_layer`) `TransformerEncoderLayer`[5] with the specified input dimension (`emb_size`), number of heads (`num_head`), hidden dimension of the feedforward sub-network (`ffn_size`) and dropout probability (`p_dropout`). Then implement the `forward` method of class `SentimentClassifier`. You should implement the following steps sequentially in the function:

(a) add the positional encoding to the embeddings of the input sequence;

(b) apply dropout (i.e. call the dropout layer of `SentimentClassifier`);

(c) call the transformer encoder you created in the `__init__` method; (Hint: make sure you set the parameter `src_key_padding_mask` to an appropriate value.)

(d) extract the final hidden state of the `[CLS]` token from the transformer encoder for each sequence in the input (mini-batch);

(e) use the linear layer to compute the *logits* (i.e. unnormalised probabilities) for binary classification, and return the *logits*.

You should read the documentation before implementing these methods.

Lastly, complete the `train_model` function to learn the classifier using the training dataset. You should optimise the model parameters through mini-batch stochastic gradient descent with the `Adam`[6] optimiser. Make sure you evaluate the model on the validation set after each epoch of training, and save the parameters of the model that achieves the best accuracy on the validation set.

You are now able to run `sentiment_classifier.py` which prepares the training, validation and test datasets, trains a sentiment classifier and evaluates its accuracy on the test set. Note that this approach does not directly use the combined training and validation datasets to re-train the model, and we adopt it here for simplicity[7]. Tuning hyper-parameters systematically is not required (but encouraged if you have

[3] `https://en.wikipedia.org/wiki/BERT_(language_model)`

[4] `https://pytorch.org/docs/stable/generated/torch.nn.TransformerEncoder.html`

[5] `https://pytorch.org/docs/stable/generated/torch.nn.TransformerEncoderLayer.html`

[6] `https://pytorch.org/docs/stable/generated/torch.optim.Adam.html`

[7] Approaches that use the combined dataset to re-train the model can be found in Section 7.8 of (Goodfellow et al., 2016) or at `https://www.deeplearningbook.org/contents/regularization.html`

access to sufficient computational resources) for this question, and setting the hyper-parameters to some reasonable values (from your experience) is acceptable. **In your answers PDF, you should record the values of hyper-parameters used in your sentiment classifier**.

Answer the following questions in your answers PDF:

1. What is the architecture of your sentiment classifier? (Hint: you may print your classifier in Python and record the output in your answers PDF.)
2. What was your accuracy on the test set?
3. Compare your classification performance with those you got for Question 1 (Part A and Part B) in Assignment 2, discuss the advantages and limitations of the three approaches to movie review sentiment classification.

Also make sure you submit your code.

Question 2: Genre Classification (Kaggle competition: 30%, Write-up: 20%)

For this task you will design and implement a classification algorithm that identifies the genre of a piece of text. This task will be run as a competition on Kaggle. Your marks for this question will be partially based on your results in this competition, but your mark will not be affected by other students' scores, instead you will be graded against several benchmark solutions. The other part of your mark will come from your code and write-up.

The dataset consists of text sequences from English language books in the genres:

- `horror` (class id `0`),
- `science fiction` (class id `1`),
- `humour` (class id `2`), and
- `crime fiction` (class id `3`).

Each text sequence is 10 contiguous sentences. Your task is to build the best classifier when evaluated with **macro averaged F1 score**.

Note: the training data and the test data come from different sets of books. You have been provided with `docid`s (examples with the same `docid` come from the same book) for the training data but not the test data.

You have been provided with an example solution in `genre_classifier_0pc.py` which shows you how to read in the training data (`genre_train.json`), test data (`genre_test.json`) and output a CSV file that the judging system can read. This solution is provided only as an example (it is the 0% benchmark for this problem), you will want to build your own solution from scratch.

Setup

Please register for Kaggle[8] using any email account you like. You do not have to use your full or real name if you would prefer not to.

You should submit your solutions to the competition on Kaggle. The competition URL can be found on Wattle.

To ensure student submissions are anonymous to other students but not the examiner, a unique Kaggle ID has been provided on Wattle as text feedback to a dummy assignment called "Kaggle Unique ID". This dummy assignment does not accept submissions, and its only purpose is to distribute unique ids. Set

[8] `https://www.kaggle.com/`

your "team name" to your unique id (e.g. `pid123456789`). This will allow us to match your submission to your assignment for marking purposes. Note that "team name" is a Kaggle term, **this is an individual assignment, and so you should not work with others to complete it**.

The Kaggle contest deadline is the same as the assignment deadline. See "**What to submit**" below if you have been granted an extension.

Rules

These rules are designed to ensure a degree of fairness between students with different access to computational resources and to ensure that the task is not trivial. Breaching the contest rules will likely result in 0 marks for the competition part of this assignment.

- Do not use additional supervised training data. That is, you are not allowed to collect a new genre classification dataset to use for training. Pre-training on other tasks, such as language modelling, is permitted.
- Pre-trained word vectors (such as `word2vec`, `GloVe`, `fasttext`) may be used even if they require an additional download (e.g. you may use word vectors from `spacy`, `genism`, or `fasttext`).
- You may use the pre-trained transformers (e.g. `bert-base-cased` or `bert-base-uncased`) from the `transformers`[9] library.
- This is an individual task, do not collude with other individuals. **Copying code from other people's models or models available on the internet is not permitted**.

Judging system

- You will upload your predictions for the test set as a CSV file for judging.
- You are allowed to **submit up to 10 times per UTC Day**. Since you get immediate feedback from every submission it is best to start submitting early and plan ahead.
- The results shown on the public scoreboard prior to the conclusion of the contest only include 50% of the test data. Your solution will be judged on the other 50% of the test data when computing final rankings and marks.
- The judging system allows you to choose which of all your submissions to be your final one.

Competition marking

Marks will be assigned based on which judge baselines you beat on the hidden 50% of the test data. If, for example, you beat the 80% baseline but do not beat the 90% baseline you will be awarded a mark on a linear scale between these two based on your macro averaged F1 score. Exceeding the score of the 100% baseline will give you a mark of 100% for the competition component of this question. Using Google Colab[10] is recommended if you want to do GPU training but do not have access to a dedicated GPU.

Write-up

A fraction of your marks will be based on a write-up that a minimum describes:

- How your final solution works.
- How you trained and tested your model (e.g. validation split(s), hyper-parameter search etc.).
- What models you tested, which worked, and which didn't.
- Why you think these other models didn't work.

[9] `https://github.com/huggingface/transformers`

[10] `https://wattlecourses.anu.edu.au/mod/resource/view.php?id=2985670`

Aim for 1 page or slightly less. Only the first 2 pages will be marked. Bullet points are acceptable if they are understandable.

What to submit

- Submit the code of your best solution (including training pipeline) to Wattle. **Also make sure your "team name" is set to your unique Kaggle ID**. **Do not submit stored parameters or data**.
- Submit your write-up in your answer PDF file.
- In one of the two cases below, you should also submit a CSV file with your model's output in the correct judging format. The CSV file should be named with your Uni ID, e.g. `your_uid.csv`.
  (a) You have been granted an extension, OR
  (b) You could not use Kaggle and only if you could not use Kaggle.

Question 3: Pre-trained Language Model for Information Retrieval (20%)

A pre-trained large language model (LLM) can be used to answer queries like a standard information retrieval (IR) system by incorporating a query into the prompt. However, the response provided by an LLM might be inaccurate due to, for example, the data used to train the LLM are outdated. One approach that may mitigate this issue is to provide up-to-date data as contextual information when querying an LLM. An example of this approach is the Retrieval-Augmented Generation (RAG)[11] which retrieves documents relevant to a query and pass them to the LLM when generating an answer to the query.

Your task is to implement a basic RAG system and compare it with a standard IR system as well as the approach that uses an LLM to answer queries directly. You will reuse code (namely `indexer.py`, `query.py`, `string_processing.py`) and data provided in Assignment 1 for this question.

First, you should install the `CTransformers`[12] library which will help you load and run pre-trained LLMs on a local machine such as your laptop computer[13]. Once the library is successfully installed, you can run the starter code `llm_for_IR.py` which uses a 4-bit quantised, pre-trained (and fine-tuned) Llama 2 language model with 7 billion parameters[14]. (You may use a different pre-trained LLM that suits your computing platform, e.g. another model from the Hugging Face repository `https://huggingface.co/TheBloke`.) Make sure you first uncomment the line with the `query_llm` function in the top-level code block (i.e. the block after the line "`if __name__ == '__main__':`") and then run `llm_for_IR.py` to directly query the pre-trained LLM. **You should record in your answers PDF the text generated by the LLM** for the below queries you explored in Assignment 1:

- Query 1: `Is nuclear power plant eco-friendly?`
- Query 2: `How to stay safe during severe weather?`

Next, you shall implement retrieval-augmented generation. Given a query, the RAG approach can use a standard IR system to rank a set of documents by relevance, and a few top ranked documents will be used by the LLM to generate a response to the query. However, simply concatenating the text in top ranked documents will likely make the length of the prompt exceed the maximum context length of the LLM[15]. To deal with this limitation, one may summarise the content of the documents and use a short summary as additional context for the query in the prompt. Interestingly, one can also use a pre-trained LLM for the summarisation task.

[11] `https://arxiv.org/abs/2005.11401`

[12] `https://github.com/marella/ctransformers`

[13] When developing a real-world application powered by language models you would be better off using a framework such as LangChain `https://www.langchain.com/`

[14] `https://arxiv.org/abs/2307.09288`

[15] The maximum context length is 512 for the default LLM used in `llm_for_IR.py`.

You should implement three functions in `llm_for_IR.py`:

(i) the `summarise_text` function to summarise the input text using a pre-trained LLM;

(ii) the `get_prompt` function to create a prompt for the given query and its context; and

(iii) the `retrieval_augmented_generation` function to retrieve relevant documents for the query, create a short context string using summarises produced by `summarise_text`, and call the LLM with a prompt output by `get_prompt` to generate a response.

You should read the description provided in each of the three functions.

Lastly, you should uncomment the line with the `retrieval_augmented_generation` function (and comment out the line with the `query_llm` function) in the top-level code block of `llm_for_IR.py`, and then run the script to generate a response to both Query 1 and Query 2 using retrieval-augmented generation. **Make sure you record the responses in your answers PDF**.

Answer the following questions in your answers PDF:

1. Compare the answers to Query 1 and Query 2 produced by the two approaches – querying LLM directly and RAG – and discuss the main differences.
2. You have now implemented three approaches to information retrieval throughout the course, i.e. using a standard IR system, directly querying a pre-trained LLM, or retrieval-augmented generation with a pre-trained LLM. Compare and discuss the advantages and limitations of these three approaches.

Also make sure you submit your code.