Capstone Project – Question Answering Model

By: Kathy Tong

Problem Statement

Scanning for information through large PDFs can be an issue, this project focuses on fine-tuning pretrained NLP models to build a custom question answering bot using given question and context.

Data

Stanford Question Answering Dataset, or SQuAD, is a dataset designed for training and evaluating question answering systems. It consists of 87,000 training data points of real questions humans asked where the answer can be found within the context given.

Text Pre-Processing

Text pre-processing involves extracting the question from the example and removing the leading/trailing whitespace. Then it uses tokenized to convert the questions and associated context into a tokenized format. The tokenizer also returns offset mapping which is the mapping between tokens and original characters in the text. Then, answers are retrieved and calculates a start and end character position of the answer in the original context. Next, it identifies the start and end positions of the context within the tokenized sequence, and iterates through the sequence_ids to find the start and end positions of the context. If answer is not in the context, then it returns (0, 0) instead.

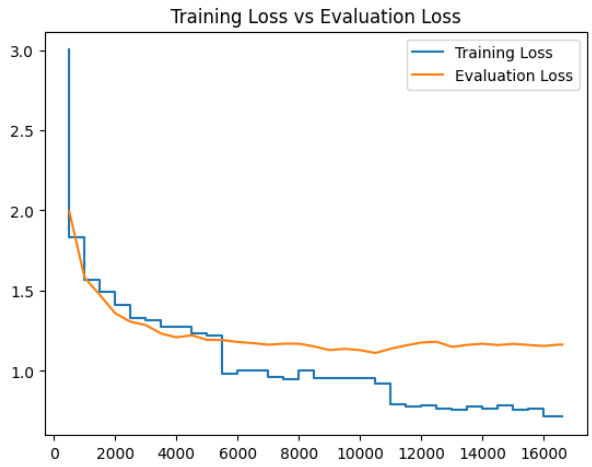
Model Experimentation

The first model I tried is using bert-base-uncased because it is a powerful NLP model and works bidirectionally. However, the model training was too intensive for my GPU to handle, and the validation error looks off.

Next, I tried to use deberta-v3-small, which is the decoding-enhancing BERT model, however the model training validation error increased while the training loss decreased. This tells me that my model is overfitting and captures too much noise.

Some solutions might be reducing model complexity or gradually decrease learning rate. After trial and error, I went back to my first model of using distilbert-base-uncased with the following parameters. The model uses epoch to evaluate, meaning the evaluation on the validation set will be performed at the end of each training epoch. The learning rate is set to 2e-5, learning rate controls the step size during optimization. Batch size is the batch size in GPU used during training and evaluation are set to 16. Number of epoch is 3, meaning model will go through the entire training dataset 3 times. Weight decay of 0.01, which adds L2 regularization to the optimizer with specified weight decay, and helps prevent overfitting.

Below are the model and validation loss function during training:



Model Evaluation

First, the validation data is tokenized in the same way as training data, only excluding adding answers. Then, raw_predictions are taken using trainer.predict() on the tokenized validation dataset which gives starting and ending logits that are the raw score of probability assigned to each position in the input sequence. These scores are used to identify the most likely starting and ending positions of the answer within the given context. In post-processing, the final predicted answer are generated using the raw predicted answers. Here, the mapping from each example are looped through to initialize min_null_score and valid_answers to store potential answers. Then for each feature, it selects the top-N-start and end logits indices and iterate to find the possible answer spans. Then the answer filters out out-of-scope answers and selects the best answer based on sum of start and end logit scores.

The metric chosen was rouge-2 score because it measures the quality of sentence by looking at the bigram accuracy of the predicted answer vs the given answer. Recall indicates how well the generated summary captures the context of the given answer. Precision measures how many of the generated bigram are relevant. F1 score measures the mean of both recall and precision. Below is the full rouge score for model validation,

	Precision	Recall	F1
Rouge1 Mid Score	0.7810348870637847	0.3014158916002328	0.42869974737972544
Rouge2 Mid Score	.5193410768347696	0.18069728846290992	0.26432133347291376

Frontend App

Frontend is built using Streamlit in python that allows users to either enter a paragraph of context or upload a PDF document as context. If a pdf is uploaded, then the text is extracted and broken down into chunks using a text splitter. Then embeddings are computed using huggingface BERT model and FAISS index is built from the full text. User is then prompted to ask a question and the pretrained question answering model is loaded using pipeline, and context is picked using rag in similarity_search. The model will then use the context given, question, and display the answer on the app.

References

https://huggingface.co/docs/transformers/tasks/question_answering
https://towardsdatascience.com/a-practitioners-guide-to-natural-language-processing-part-i-processing-understanding-text-9f4abfd13e72
https://towardsdatascience.com/understanding-feature-engineering-part-3-traditional-methods-for-text-data-f6f7d70acd41
https://towardsdatascience.com/understanding-feature-engineering-part-4-deep-learning-methods-for-text-data-96c44370bbfa
https://medium.com/@b.terryjack/nlp-everything-about-word-embeddings-9ea21f51ccfe
https://huggingface.co/learn/nlp-course/chapter2/6?fw=pt