Text Classification

Introduction

The IMDb Movie Reviews dataset is a collection of 50,000 reviews from IMDb that are labeled as either positive or negative. The dataset is used for binary sentiment analysis, and it contains an equal number of positive and negative reviews.

In this assignment, we will walk through three text classification pipelines on classifying IMDB moview review sentiment:

1. Training a simple model from scratch
2. Fine-tuning embeddings from a pre-trained BERT model
3. Using an out-of-the-box zero-shot LLM

Problem 1:

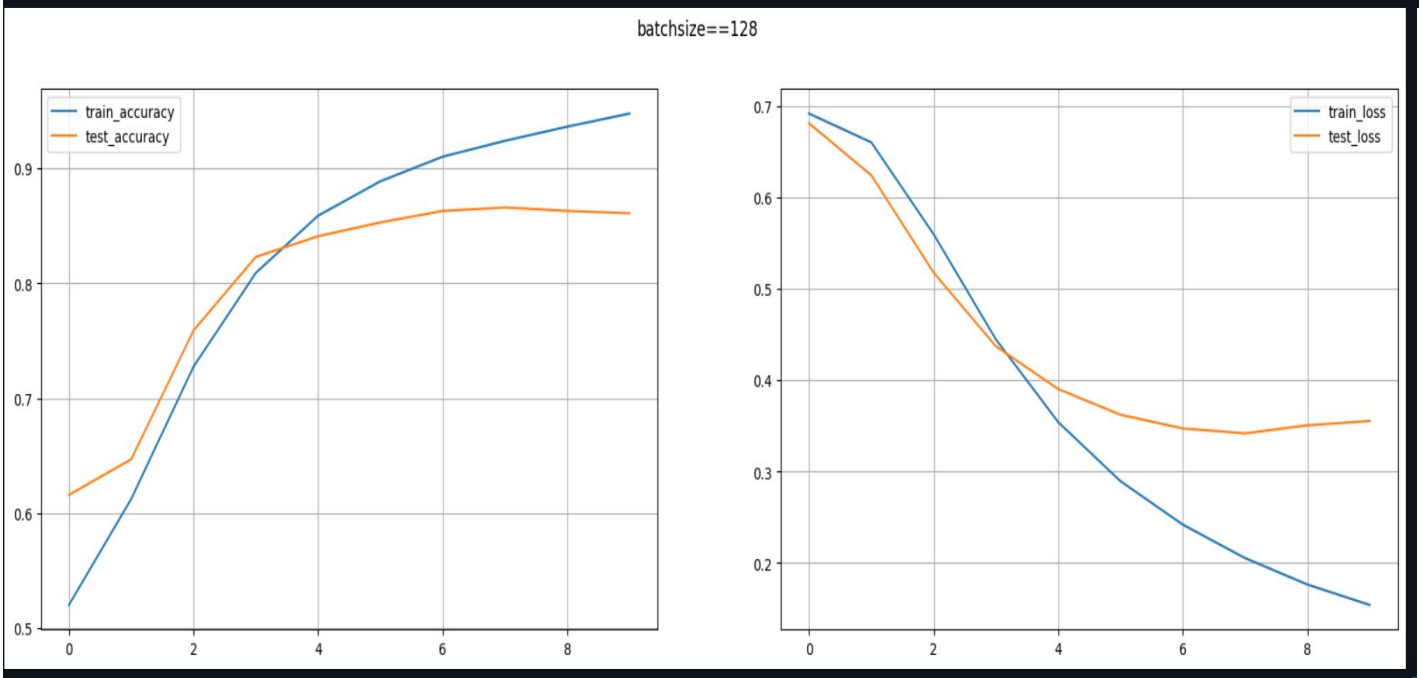
Using the simple model below, which is the same model discussed in class, modify the code by adding a hidden layer between the averaged embedding and the final logits. Use a ReLU activation function for this hidden layer, and set its dimension to 128.

1. [2 pts] Calculate the number of trainable parameters in your modified model.

```
>>> 3_923_586
```

2. [1 pt] Run the classification pipeline with your modified model and report the **accuracy** and **loss** after 10 epochs.

```
BatchSize==128
============ FINAL RESULTS =============
|            |   Trian    |    Test    |
| ---------- | ---------- | ---------- |
|    Accu    |   0.9477   |   0.8610   |
|    Loss    |   0.1539   |   0.3554   |
```

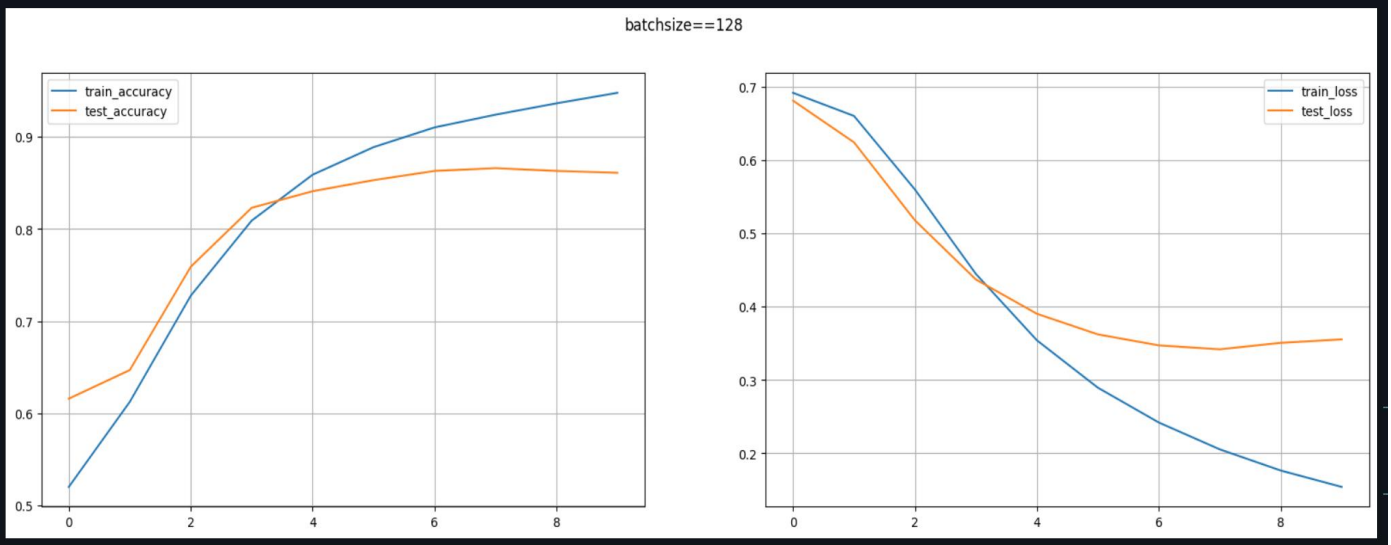


[2 pts] Increase the batch size from 128 to 1024 and run the classification pipeline again (ensure to reinitialize your model). What differences do you observe in the results, and how would you explain these changes?

```
BatchSize==1024
============ FINAL RESULTS ============
|            |   Trian    |    Test    |
| ---------- | ---------- | ---------- |
|    Accu    |   0.7364   |   0.7420   |
|    Loss    |   0.5660   |   0.5650   |
```

3. [2 pts] Increase the batch size from 128 to 1024 and run the classification pipeline again (ensure to reinitialize your model). What differences do you observe in the results, and how would you explain these changes?

```
BatchSize==1024
============ FINAL RESULTS =============
|            |   Trian    |    Test    |
| ---------- | ---------- | ---------- |
|    Accu    |   0.7364   |   0.7420   |
|    Loss    |   0.5660   |   0.5650   |
```



Why larger batch_size -> (lower accuracy, greater loss)?

- Fewer param updates -> #updates == #samples / batch_size * #epoch
- Larger sample set per update(batch) --> each iteration would find params that over-smoothen the model. (In extreme case, when batch_size == #sample -> +inf, model updates the exact average gradient over the dataset, no variance, no randomness.)

Generate | + Code | + Markdown | Run All | Restart | Clear All Outputs | Jupyter Variables | Outline | ... | py312 (Python 3.12.9)

Problem 2

Modify the code for fine-tuning using BERT.

1. [2 pts] Freeze the base model (making its parameters non-trainable) and add a hidden layer (with a dimension size of 128) with a ReLU activation function, similar to what was done in Problem 1. Report the accuracy and loss after 2 epochs.

- BatchSize==64

	Trian	Test
Accu	0.8522	0.8730
Loss	0.3443	0.3110

```
## Make BatchSize comparable to problem 1
```

- BatchSize==128

	Trian	Test
Accu	0.8505	0.8610
Loss	0.3462	0.3284

2. [1 pt] Provide an explanation for any difference observed in Problem 2 compared to Problem 1.

P1-simpleModel (Epoch 2/10)

	Trian	Test
Accu	0.5951	0.6290
Loss	0.6676	0.6396

Difference:

The simple baseline model trained from scratch seems underfitting. After 2 epochs, the pre-trained bert model gives nearly the same accuracy as the simple model does after 10 epochs.

Explanation (why P2-model is better?):

- Pretrained: bert model starts with pretrained token knowledge, even with most params frozen and only updated for the last 2 layers.
- BERT(attention) is better than AvePooling: the ***same*** parts for 2 models are the hidden layer + ReLU + Logits layer. So the ***delta*** lies in the BERT(attention) vs Bagging(average), where BERT gives a better sentence/paragraph-level CLS input to the same later layers.

Problem 3

Modify the code for Zero-shot LLM text classification. You may check the code in the notebook from Lecture 2 as well.

1. [2 pts] Optimize the prompt as much as possible (e.g., by applying the chain of thought method discussed in the lecture). Run your optimized prompt with 1000 test samples. Report both the optimized prompt and the resulting accuracy.

Baseline Prompt

91.9%

NewPrompt

```
You are given a movie review text and you need to classify it as 'positive' or negative.
Applying chain-of-thoughts, first extract keywords in the reviews,
then calculate the positive-to-negative keyword ratio. Based on this ratio,
give the final decision of positive(True) or not(False).
```

93%

```python

class KeyWord(BaseModel):
    pos: List[str]
    neg: List[str]

class Tag(BaseModel):
    is_pos: bool
    kws: KeyWord
    pos2neg: float

def fetch_openai(
        prompt,
        user_txt_input,
        model_name="gpt-4o-2024-08-06",
    ) -> Tag:
    client = OpenAI()
    res = client.responses.parse(
            model=model,
            input=[
                {
                    "role": "system",
                    "content": prompt,
                },
                # {"role": "user", "content": tt}
                {"role": "user", "content": user_txt_input}
            ],
            text_format=Tag,
        ).output_parsed
    return res
```