Assignment

In this assignment you will use the Unsupervised Pre-Training + Supervised Fine-Tuning paradigm.

The objective is to wind up with a model that can perform Sentiment Analysis on sentences from a Financial domain.

Specifically

- you will choose a Pre-Trained Language Model from the HuggingFace platform
- Fine-Tune the model on the Financial PhraseBank dataset
    - a collection of hand-annotated sentences with sentiment classified a Negative, Neutral, Positive

The assignment will consist of

- a "basic" part
    - This is the minimum that you must submit (and succeed on) in order to earn a passing grade
- "extras"
    - more challenging objectives
    - earn points beyond the basic

Financial PhraseBank dataset

The dataset was created as part of a research paper in Finance.

There were several humans who labeled the sentiment of sentences. Naturally humans don't always agree. Thus, the dataset has several "flavors" (called "subsets") based on the fraction of annotators who agree.

As part of the "basic" assignment, you should use the "all agree" flavor.

There are multiple ways to access this dataset.

- The preferred way: as a HuggingFace dataset (https://huggingface.co/datasets/financial_phrasebank)
- Discouraged: as a TFDS (https://www.tensorflow.org/datasets/community_catalog/huggingface/financial_ph
    - this is discouraged *only* because there will be an "extra" part asking you to c the HuggingFace dataset to a TFDS
    - you won't get credit for this "extra" if you obtain it directly as a TFDS !
- Discouraged: From the authors (https://www.researchgate.net/publication/251231364_FinancialPhraseBank-v10)
    - the "Download the PDF" link contains an archive file with the data
    - this is discouraged only because it involves extra work
    - but you might find it handy in creating your own TFDS in the "extra" part

You don't need to understand the paper but here is a reference (https://arxiv.org/pdf/1307.5336.pdf#page=9) to the paper that introduced it.

Choose a Pre-Trained Language Model from the HuggingFace platform

There are lots of Language Models on HuggingFace. They vary by, among other things

- model size
- training objective
- language of training data

An important consideration is computational

- the model fits in the memory of the machine you are using
- the execution time is rapid enough for you to experiment
    - it will help to have a machine with a GPU (e.g., Google Colab)

You are free to choose *subject to limitations*

- the model should *not* have already been trained on a Financial dataset
    - e.g., you may not use FinBERT

I can verify that the choice of `distilbert-base-uncased` is feasible when using the free version of Google Colab

One of the options for extra credit is to experiment with different models and report and explain the differences.

Submission requirements

You will submit a Jupyter notebook that follows the steps in the Recipe for Machine Learning (https://github.com/kenperry-public/ML_Fall_2022/blob/master/Recipe_Overview.ipynb)

That is, rather than just submitting a final model

- you should follow the steps (as appropriate) in the Recipe
- treat the notebook as a movie showing
    - the Recipe steps that you took
        - Exploratory Data analysis and Error Analysis are steps that are ignored at your own peril
    - including steps that led to failure (e.g., bad choices)
        - what you learned from the failure
        - how you overcame it or learned to make a different choice

Use Markup in Jupyter rather than code comments to convey your thoughts.

Use Section Headings to clearly indicate different parts of the assignment and different experiments.

Your code should be TensorFlow/Keras.

Discussion

In each part, or each experiment, I would like to see a Discussion of results

- Can you conclude that your choices led to better of worse out of sample performance ?
- Do you have a theory as to why the performance changed/did not change ?

The objective is to get you to think like a scientist rather than just completing a task.

Originality

There are no doubt published notebooks that use this dataset for a similar purpose.

The work that you submit should be your own.

Getting started

The HuggingFace course is a great way to get up to speed on various aspects of this project.

Basic parts of assignment

Part 1: Fine-tuning a pre-trained model, conventional approach

There are several sub-parts.

- Fine-tune *only the Classifier head*
    - This will help determine how well your Pre-Trained model performs on financial data
- *Keeping the trained Classifier head of the first part*: Fine-tune *all* the weights
    - This will help determine how much fine-tuning improved your Pre-Trained model
    - *Do not* start this step with an uninitialized Classifier head
- Fine-tune all the weights (Pre-Trained + Classifier Head) simultaneously
    - start with an uninitialized Classifier head

You will present a discussion of how your out of sample performance was affected by any of the sub-parts.

The Basic part allows you several simplifications

- your model fitting can take features/labels that are Python data structures
    - these are most familiar to you
    - obtained from the dataset
- you can using a HuggingFace model that already comes with a Classifier head

I suggest that one section of your notebook clearly demonstrates success on the Basic Part.

Part 2: Fine-tuning a pre-trained model using LoRA

You will use LoRA to fine-tune a pre-trained model.

Our objective is identical to Part 1

- compare the performance of a model that has *not* been fine-tuned with Financial data
- to a model that *has* been fine-tuned with Financial data

The difference:

- you will use LoRA to perform the fine-tuning
- we will fine-tune only a subset of the model's weights

We now describe Part 2 in more detail.

In Part 1, you were able to choose your own pre-trained model.

In Part 2 (to simplify grading), the model we require you to fine tune is `distilbert-base-uncased`

Your first step: repeat the first sub-part of Part 1 using the Part 2 model

- Fine-tune *only the Classifier head*
- If you also used the Part 2 model in Part 1
    - you can save a copy of the model from Part 1 (after training only the Classifier head)
    - and re-use it rather than performing this step

Your second step: perform something *similar* to the second sub-part of Part 1 (using the Part 2 model)

- *Keeping the trained Classifier head of the first part*: Fine-tune *a specified subset* of the weights
    - In Part 1: you fine-tuned **all** the weights
    - In Part 2: you will fine-tune **only the weights** of the FFN blocks in every layer
        - this is where we hypothesize "world knowledge" acquired during training is stored
        - each FFN block has 2 `Dense` layers: `lin1`, `lin2`
            - apply LoRA **to each** of the 2 `Dense` layers

The way you will fine-tune a subset of the weights is

- using the LoRA technique
    - adding LoRA adapters to the blocks to be fine-tuned
        - **use rank 8** for LoRA adapters

Required questions to answer

- How does the performance of the model fine-tuned with LoRA compare to that of the pre-trained (not fine-tuned) model
- How does the performance of the model fine-tuned with LoRA compare to that of the model fine-tuned using the conventional approach

Extras for Part 1

You can demonstrate greater skill (and earn more points) by completing some extra parts.

After each extra, please complete a Discussion section as you did for the Basic part.

My suggestions follow, but I'm open to your ideas.

I will determine the amount of extra points by my perception of the difficulty of each extra.

Create (and fit the model with) a TensorFlow Dataset (TFDS)

Relative difficulty: low to medium

Starting off with the HuggingFace dataset, your objective

- is to create a TFDS such that you could train a model without knowing anything about how the TFDS was prepared beforehand.
- similar to how you would use any other externally supplied dataset

We want to *pretend* that our machine's memory is too small to load the entire dataset into memory *for training*, and hence need to use a TFDS.

- it is perfectly acceptable to load the entire dataset into memory for the sole purpose of creating the TFDS

Given an arbitrary dataset, you might need to perform some operations for training, such as

- shuffling
- splitting into train and validation and/or test datasets
- batching

All operations for training on the dataset you create should be performed by TFDS operations *after* you have created the TFDS.

For training, you *may not assume* that any necessary operation has been performed before-hand , for example

- you may not shuffle or split the source dataset before turning it into a TFDS

You may create the TFDS in any way that you like but, once created, anyone should be able to use it without knowing the steps involved in its creation

Create your own Classification head

Relative difficulty: medium to hard

Rather than using a Classification head automatically provided with the HuggingFace model

- you will obtain a head-less model from HugginFace
- you will add your own Classification head
    - with as many layers as you like
- you *must* explain
    - the logic of how you knew where to graft the head on the headless model
    - how you adapted (if necessary) the output of the headless model to the shape requirements of the Classifier head

The objective is for you to demonstrate some skills with the Keras Functional Model API.

Use different ”flavors” of the dataset

Relative difficulty: low, but time-consuming

The basic part used examples on which all annotators agreed.

Try different flavors and discuss the results.

Address any Imbalanced Data issues

Relative difficulty: low

- If the distribution between labels in the dataset is not uniform, you may want to address the imbalance

Superior Error Analysis

Relative difficulty: medium

Rather than just reporting a single summary statistic for out of sample performance, analyze the results in detail

- Is one class harder to correctly classify than others
- Is there some systematic pattern of errors
    - e.g., characteristics of input sentences that are more difficult to correctly classify

Try to use the results of this analysis to improve the model

Experiment with different Pre-Trained models

Relative difficulty: low, but time-consuming

Try several different pre-trained Language Models.

Discuss the results. For example

- does a bigger pre-trained model lead to better results
    - before and after fine-tuning all the weights
- does the type of data on which the model was trained make a difference

Experiment with Fine-Tuning

Relative difficulty: low, but time-consuming

Does out of sample performance vary with changing

- the number of examples in Fine-Tuning
    - what is the smallest number that you think is sufficient
- there are many choices of proper subsets of a given size
    - does it matter which one you choose ?

In-context learning

Relative difficulty: low but fun !

Can you use few-shot learning successfully (i.e., no further training) ?

It would be great to do this for Financial PhraseBank but the sentences may be too long

- pre-trained models have maximum sequence lengths that may be too small

Propose some interesting task related to Finance and try to achieve Few Shot Learning on the task.

Extras for Part 2

Experiment with the rank used in LoRA

In the basic part, we asked that your LoRA adapters using rank 8

- this facilitated our comparison of the number of parameters you added with what we expect

Conduct experiments in varying the rank

- how does the Performance Metric change with rank ?

Some helpful tips

Making sure your model has trained

The assignment asks you to train a Classifier over 3 possible classes.

If your model performs no better than random chance, you will find that it has an Accuracy of around 33%.

So if, after training, your model's accuracy is not *much* higher than this: your model has a problem -- It's very likely that some model weight's have not been trained.

Freezing the pre-trained model

HuggingFace allows you the option of instantiating a model with a task-specific head.

In class I have always used a simple one-layer head, e.g., a Dense layer to implement a Classifier.

But you *should not assume* that the head supplied by someone else's model has a head designed exactly that way. Always look !

Using the `summary` method on the model returned by HuggingFace will help you determine what to freeze.

Unfreezing

Before you can fit a model in Keras, you must execute a `compile` statement.

On the surface, the purpose is to associate a loss function (and optionally: an optimizer and metrics) with the model.

Below the surface, it causes the `trainable` attribute of the weights of a layer or model to become static.

The practical implication: if you "freeze" the weights of some part of the model for an initial round of training, and then wish to unfreeze those parts of the model so that they will change in subsequent training, you must re-issue a `compile` statement after unfreezing.

See the Keras guide to Transfer Learning (<https://urldefense.proofpoint.com/v2/url?u=https-3A__keras.io_guides_transfer-5Flearning_-23finetuning&d=DwMBaQ&c=slrrB7dE8n7gBJbeO0g-IQ&r=58cSHneR9qnbk2_epkhw3g&m=BemEXtkIoZA-tri)

Extras for Part 2

Experiment with the rank of the LoRA adapters

We required you to use a specific value for the rank in the basic part

- this facilitated our comparing the number of parameters you added with what we would expect

Vary the rank and examine the impact on performance.

In [2]:
```
print("Done")
```

```
Done
```