- Keras fine-tuning, transfer learning (https://keras.io/guides/transfer_learning/)
    - Transfer Learning vs Fine tuning
    - Recursive setting of the trainable attribute
    - Workflow
- Documentation: DistilBert (https://huggingface.co/docs/transformers/model_doc/d
    - DistilBert code (https://github.com/huggingface/transformers/blob/main/src/transformer
- DistilBert model card (https://huggingface.co/distilbert-base-uncased)
- BERT model card (https://huggingface.co/bert-base-uncased#limitations-and-bias)
- Documentation: Models (https://huggingface.co/docs/transformers/main_classes/m
- Fine tuning lesson (https://huggingface.co/docs/transformers/training)
    - Fine tune for downstream tasks (https://huggingface.co/docs/transformers

---

- Behind the pipeline (https://huggingface.co/course/chapter2/2?fw=tf)
- Course Chapt 2, putting it all together: Using a pretrained model for sequence classification (https://huggingface.co/course/chapter2/6?fw=tf)
- Using a Model: Chapt 2 (https://huggingface.co/course/chapter2/3?fw=tf)

My notebook: Finetuning FinancialPhraseBank, based on the Fine_tune_Hugg.. (HuggingFace/Course/Financial_PhraseBank.ipynb)

- My Fine_tune_Hugg.. from course, can't find in course anymore (HuggingFace/Course/Fine_tune_HuggingFace_model_in_Keras_with_plain_dataset
- seems to have changed
    - using Yelp rather than IMdB
        - imbdb is in transformers/doc, but now uses Trainer (https://huggingface.co/docs/transformers/tasks/sequence_classi
    - using Trainer

Hugging Face

Hugging Face (huggingface.co) is a Data Science platform that (among many other functions) hosts

- Pre-trained models
- Datasets

They are particularly (but not exclusively) known from Transformers and NLP tasks.

We will use them as our paradigm for Transfer Learning.

Our overview will necessarily be brief.

We recommend the free, on-line course (https://huggingface.co/course)

- We will not explain the datasets API
    - similar to Tensorflow Dataset API

NLP: more than just a model

You will find many models for NLP on HuggingFace.

One of the most important aspects of a Model Hub for NLP is

- providing a consistent interface to multiple models
- with different authors

Pre and post processing

To ensure universality of models obtained from many sources

- all NLP models take sequences of *tokens* as inputs
- **not** sequences of words (the raw input format)

Thus, there needs to be some *pre-processing* performed before using a pre-trained model

Similarly, there may need to be some *post-processing* performed on the model output.

For example: creating a task-specific "head"

- e.g., classifier over *our task's* classes

In addition to models, HuggingFace

- supplies common pre-processors and post-processors
- wraps them all up into a simple to use, powerful organization

One model, many flavors

As we shall see, HuggingFace models come in many variations

- `Keras` or `Pytorch`
- with a "head" specific to a common task

This reduces the need for user customization.

HuggingFace also makes it easy to match the correct pre-processor (Tokenizer) to a model

- a model is identified by a "check-point" referring to its training dataset

```
checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"
```

- one instantiates a model from a checkpoint

```
model = TFAutoModelForSequenceClassification.from_pretrained(checkpoint)
```

  - this model has a "head" specialized for the task of Sequence Classification (e.g., Sentiment Analysis)
    - the `ForSequenceClassification` suffix
  - this model is implemented in TensorFlow
    - the TF prefix

- it's easy to choose the "correct" Tokenizer by using the same one as was used in training the model

```
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
```

Preparing input examples

A unique aspect to sequence inputs is that the sequence length varies across examples.

Hence, "padding" is usually necessary to make example lengths uniform.

It is tempting equalize the length of each example to the length of the longest example in the *training dataset*.

This is not the best strategy

- the model works in much smaller *mini-batches*
- only necessary to equalize the length of each example *within* a batch
    - not *across* all batches

Equalizing sequence length of examples within a batch is just one example of preparing examples in a batch.

Another case: masking words (usually randomly) in a example for a Masked Language Modeling task.

HuggingFace has an abstract class `DataCollator` (https://huggingface.co/docs/transformers/main_classes/data_collator#data-collator) to prepare examples within a batch

- specializations for common tasks

Putting it all together: running an NLP task

Easy ! So is actually running all the steps needed to generate output

```
sequences = ["I've been waiting for a HuggingFace course my whole life.", "So h
ave I!"]

tokens = tokenizer(sequences, padding=True, truncation=True, return_tensors="t
f")

output = model(**tokens)
```

And one "not so easy" part

- Like any classifier: the output is a *vector* of probabilities over the universe of tokens
    - Technically: they are "logits" rather than probabilities
- So the *user* needs to sample from this vector in order to choose a single output token
    - e.g., the one with highest probability
- This maximizes the flexibility for the user
    - to sample rather than choose the token with highest probability
    - to add further logic
        - sample "top 5" tokens

Using a pre-trained model for NLP (w/o Transfer Learning)

Our first demo will be to show how to use a pre-trained model.

- We will not (yet) specialize it to a new Task

The task is Text Sequence Classification (sentiment)

The model is based on the Transformer architecture.

We give a quick review of how Transformers are typically used for NLP.

Pre processing

Text is a sequence of words, but the words must be parsed into *tokens*

- a word may turn into multiple tokens: Byte Piece Encoding

The tokens must be converted into integer indices in a finite vocabulary

- It is the sequence of *token ids* that is consumed by the Transformer

Post processsing

Hugging Face Classification models are slightly different in that they return

- logits (scores)
- **not** probabilities

We can use `argmax` on the score to decide the class, but it's nice to see the associated probability too.

Let's go to the notebook (HF_quick_intro_to_models.ipynb)

A model has many ”flavors”

In our discussion of Modern NLP, we highlighted the "Text to Text Universal API"

- Can translate many NLP tasks into variations of translating Text to Text
- A post-processing step can map back from the output text to the actual Task outputs

Hugging Face

- implements a "base model" (universal API)
- and the derives specializations with task-specific "heads"

Moreover, Hugging Face models usually have implementations in both TensorFlow and PyTorch.

The name of a Hugging Face model is a compound string indicating

- the base
- the specialized head
- the implementation language

Illustrating this with the DistilBERT model implemented in TensorFlow (github (https://github.com/huggingface/transformers/blob/main/src/transformers/models/distilb here is the class hierarchy:

- `class TFDistilBertPreTrainedModel(TFPreTrainedModel):`
- `class TFDistilBertModel(TFDistilBertPreTrainedModel)`
- `class TFDistilBertForMaskedLM(TFDistilBertPreTrainedModel, TFMaskedLanguageModelingLoss):`
- `class TFDistilBertForSequenceClassification(TFDistilBertPreTrained TFSequenceClassificationLoss):`
- `class TFDistilBertForTokenClassification(TFDistilBertPreTrainedMod TFTokenClassificationLoss):`
- ⋮

The `TF` prefix indicates that the implementation language is TensorFlow.

- There is an abstract class for Pretrained TensorFlow models: `TFPreTrainedModel)`
- `TFDistilBertPreTrainedModel` is a pretrained model
- There are specialization of `TFDistilBertPreTrainedModel` for each NLP task
    - `TFDistilBertForMaskedLM` for the Masked Language Modelin task
    - `TFDistilBertForSequenceClassification` for the Sequence Classification task
    - `TFDistilBertForTokenClassification` for the Token Classification task

You get to choose !

- If you choose a model instance with the `*For*` pattern in the name
    - you get a pre-trained model with a Classification head (post-processing) for a specific task
- If you choose a model instances *without* the `*For*` pattern
    - you get a model without a head (or post-processing)
    - you can add your own

You can also instantiate a model *without* pre-trained weights

- just the architecture

Case study:`TFDistilBertForSequenceClassification`

Let us examine the implemenation of the `TFDistilBert` specialization for Sequence Classification.

github link to class `TFDistilBertForSequenceClassification` (https://github.com/huggingface/transformers/blob/main/src/transformers/models/distilb

```
class TFDistilBertForSequenceClassification
```

- Derives from `TFDistilBertPreTrainedModel`
- creates a Classifier head
- modifies the `call` method
    - pass the input sequence to the base `DistilBert` model
    - the `Distilbert` model output is
        - sequence of latent representations of length 768 (one latent per element of the input sequence)
    - choose the single sequence element at position 0, corresponding to the the special `[CLS]` input token
    - pass the single element to a two layer Classifier head
        - `preclassifier`, 'classifier`
        - `Dense` layers separated by `Dropout`

Part of a good model hub is a *model card* for each model that describes

- The model
- The data on which it was trained
- Gives reference to the original work (paper, code)
- Discusses potential *bias*

Bias of models (really, the data on which they were trained) is important and is being increasingly recognized.

Models are often trained from publicly available Web content (e.g., Wikipedia).

Although the datasets were not designed to be biased

- the frequency of biased concepts that they contain
- causes the biases to be transfered to models on which they are trained.

In the Intro course, we discussed how the GloVe word embeddings may reveal bias:

$$\begin{aligned} &\text{doctor - man} + \text{woman} && \approx_{n',d} && \text{nurse} \\ &\text{mechanic - man} + \text{woman} && \approx_{n',d} && \text{teacher} \end{aligned}$$

Discovering and documenting potential bias is non-trivial and potentially time-consuming.

Here are the biases (https://huggingface.co/distilbert-base-uncased#limitations-and-bias) discussed on the model card.

Here (https://huggingface.co/distilbert-base-uncased) is the model card for `DistilBert`.

For the purposes of Transfer Learning:

- note that the implemention of `DistilBert` on Hugging Face consumed 90 hours on 8 GPUs (https://huggingface.co/distilbert-base-uncased#pretraining)
- we will leverage this work without any training cost to us

Transfer Learning + Fine-Tuning: rolling our own specialization of `DistilBert`

We will now demonstrate

- Transfer Learning
    - grafting a Classification head onto a pre-trained `DistilBert` model
- Fine-tuning
    - once our grafted model Classifier head is trained
        - train *all* the weights, including the contained `DistilBert` model

This module is a nice summary of the course

- We use a `Functional` architecture
- Override the `call` method
- Transfer Learning + Fine-Tuning

We are going to adapt `DistilBert` to a new "Target" task: Text Sequence Classification

- `DistilBert` was trained on the Masked Language Modelling task
- Our task is different: Text Sequence Classification
- We will therefore invoke a "headless" version of the model and graft on our own head
    - which will need to be trained

Highlights

- Create a new model that *contains* a pre-trained `DistilBert`
- Graft on a Classifier head
    - override `call` method
        - invoke the contained `DistilBertModel`
        - post-process
        - pass result to Classifier head
- Transfer Learning
    - freeze weights of the contained `DistilBert`
    - train weights of Classifier head
- Fine Tuning
    - re-train *all* the weights

Let's visit the notebook (Transfer_Learning_example_HF.ipynb)