Google Classroom Code: mhxgl24

NLP & Hugging Face

Deep Learning (DS-5006)

Dr. Adeel Mumtaz

Lecture 10

Fall, 2022

National University
Of Computer and Emerging Sciences

Contents

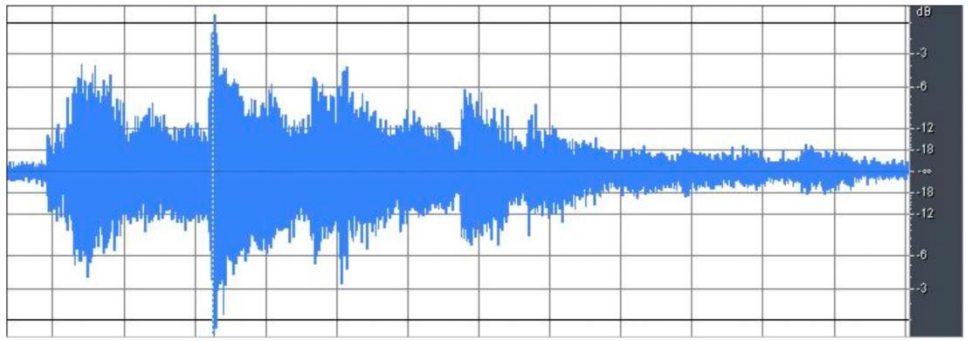
Sequence Problems

- In **Sequence Problems**, an **ordered** sequence of data points shares a single label
- Problems Involving Sequence Data
  - Time Series
  - Natural Language Processing
- Deep Learning Models
  - RNNs
  - Transformers

Text

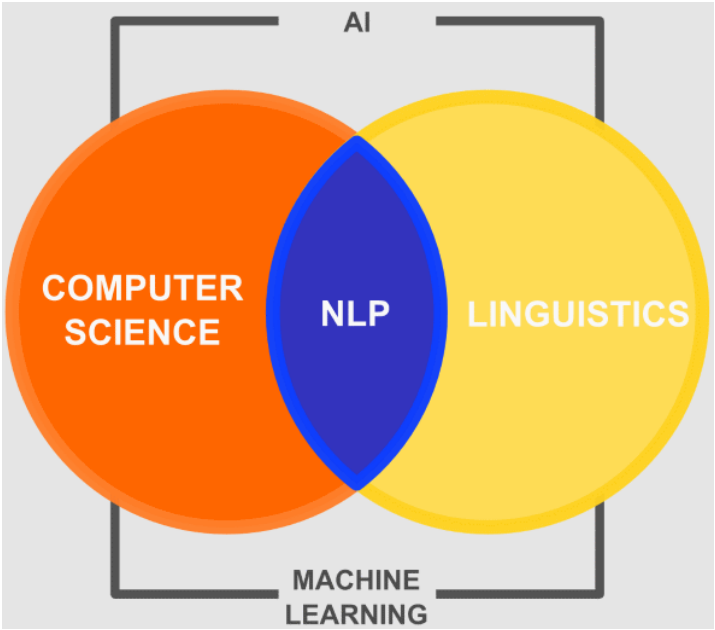
- "The quick brown fox jumps over the lazy dog"

Audio



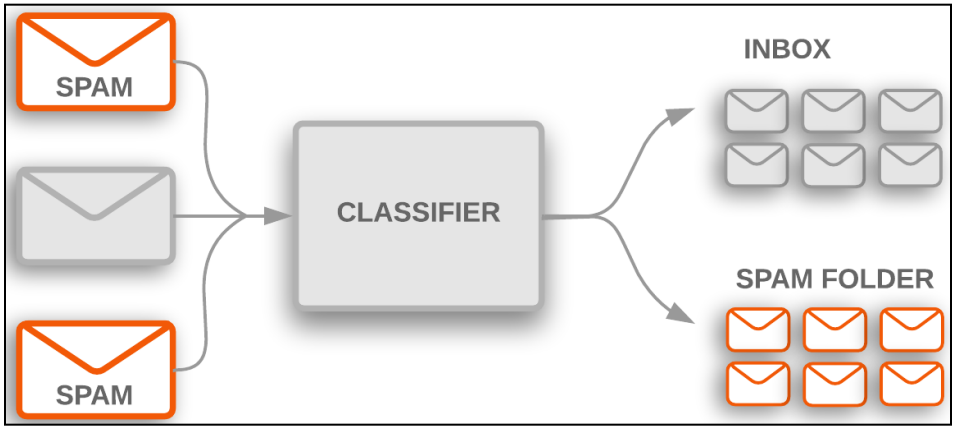
World of NLP

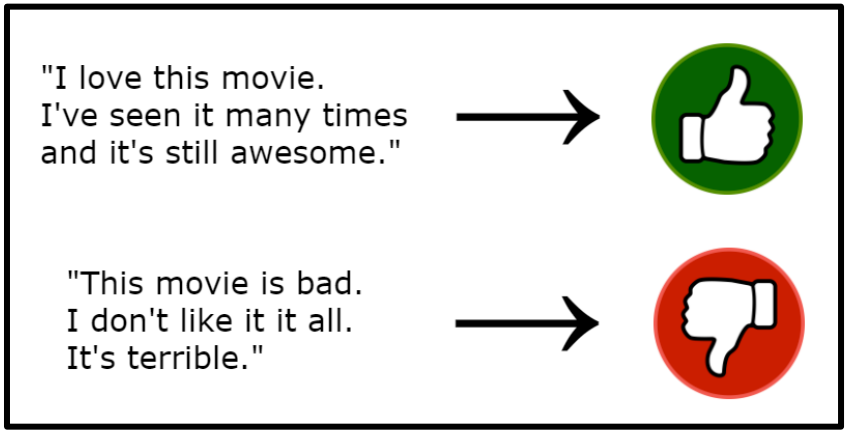
- NLP is a field of **linguistics** and machine learning focused on:
  - Understanding everything related to **human language**
- The aim of NLP tasks is:
  - Not only to understand single words individually
  - But to be able to understand the **context** of those words.



Main Tasks in NLP

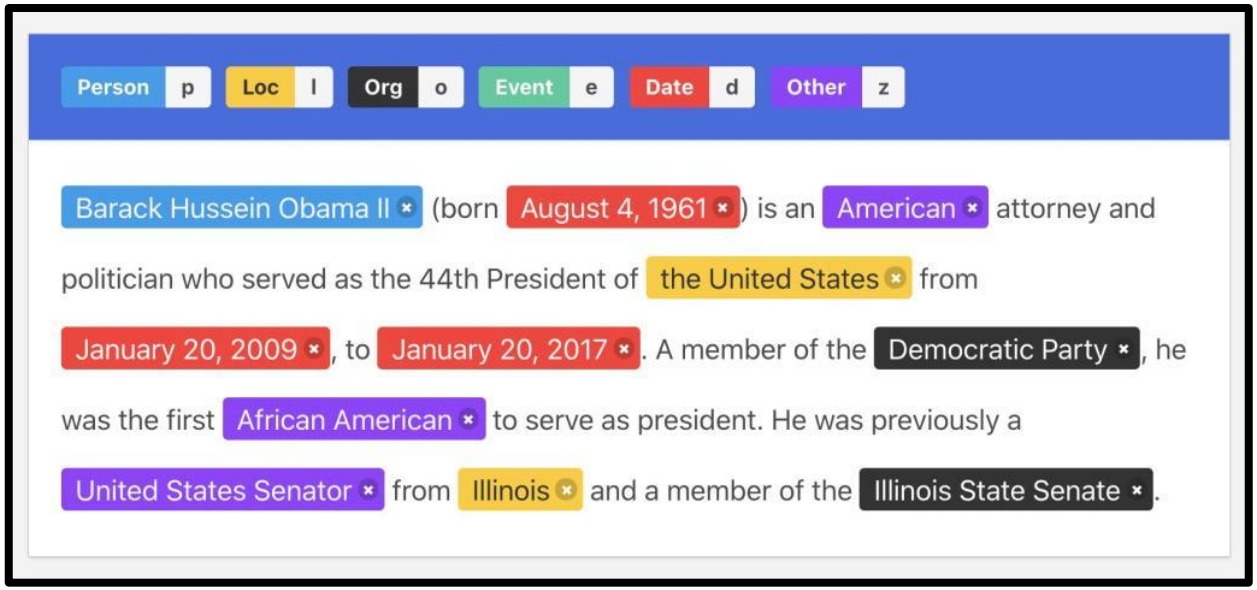
- Classifying Whole Sentences
  - Sentiment of a review
  - Detecting if an email is spam
  - Determining if a sentence is grammatically correct





Main Tasks in NLP

- Classifying each word in a sentence
  - Identifying the grammatical components of a sentence (noun, verb, adjective)
  - Identifying named entities (person, location, organization)



Main Tasks in NLP

- Generating text content
  - Completing a prompt with auto-generated text
  - Filling in the blanks in a text with masked words

Tom has fully ___ ___ ___ illness.

Best Guess : recovered from his

Main Tasks in NLP

- Extracting an answer from a text
  - Given a question and a context, extracting the answer to the question based on the information provided in the context

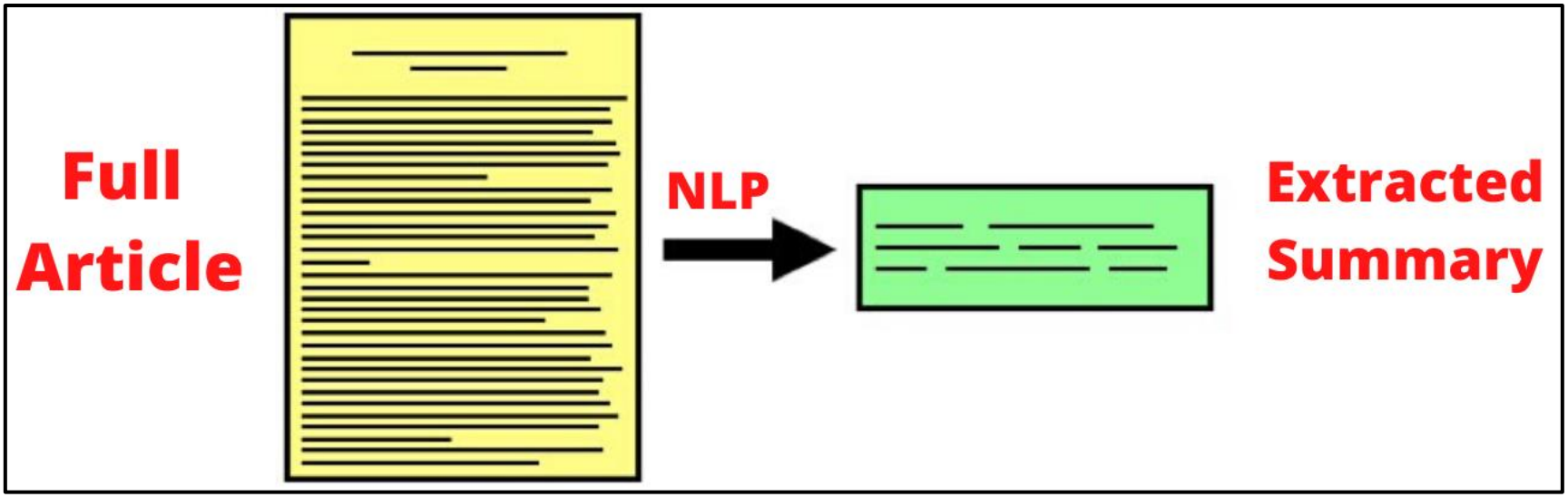
```
question="Where do I work?",
context="My name is Sylvain and I work at Hugging Face in Brooklyn",
```

```
'answer': 'Hugging Face'
```

Main Tasks in NLP

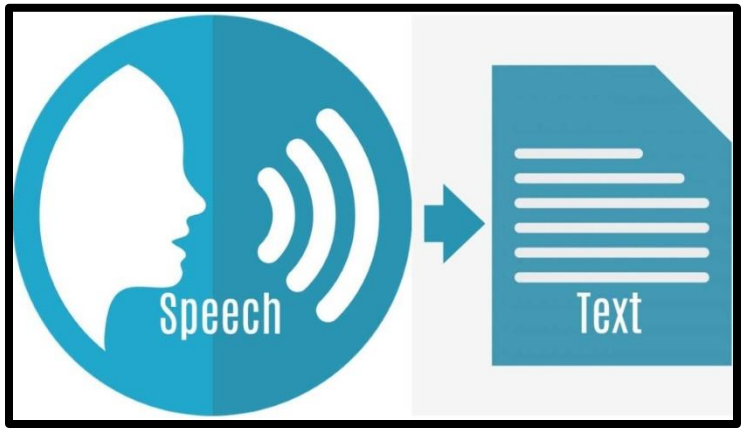
- Generating a new sentence from an input text
  - Translating a text into another language
  - Summarizing a text

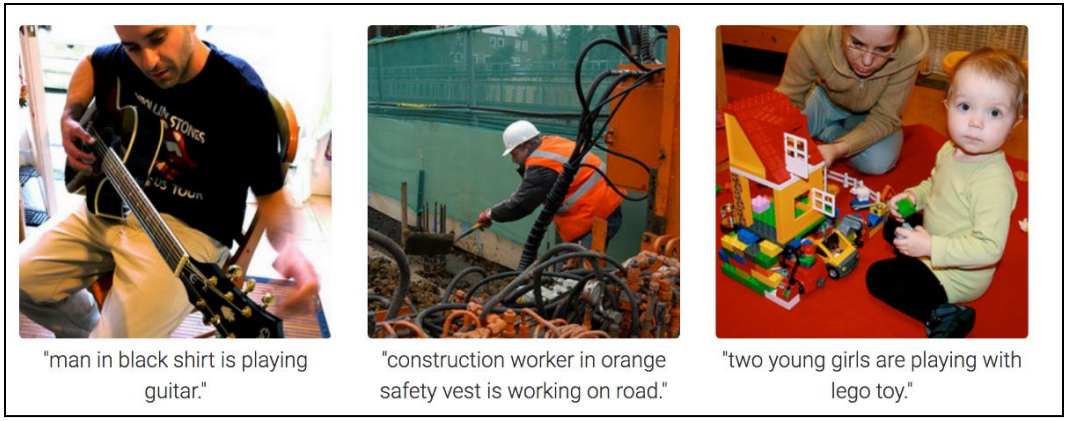
(Es regnet draußen)$_{\text{German}}$ ➡ (It's raining outside)$_{\text{English}}$



Main Tasks in NLP

- NLP in other fields
  - NLP isn't limited to written text
  - It also tackles complex challenges in
    - Speech Recognition
      - Generating a **transcript** of an audio sample
    - Computer Vision
      - Description of an **image**.





Transformer Models

NIPS 2017

ATTENTION IS ALL YOU NEED

Google Brain

Ashish Vaswani[*] Google Brain avaswani@google.com

Noam Shazeer[*] Google Brain noam@google.com

Niki Parmar[*] Google Research nikip@google.com

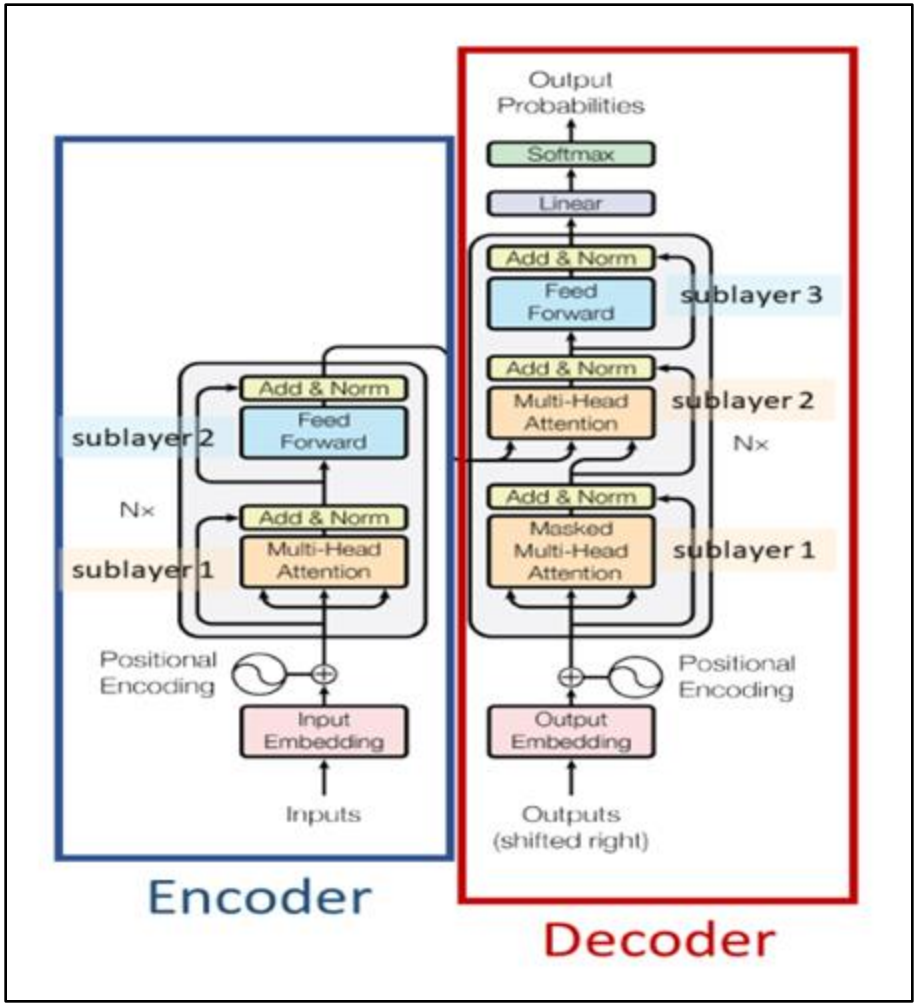
Jakob Uszkoreit[*] Google Research usz@google.com

Llion Jones[*] Google Research llion@google.com

Aidan N. Gomez[*] [†] University of Toronto aidan@cs.toronto.edu

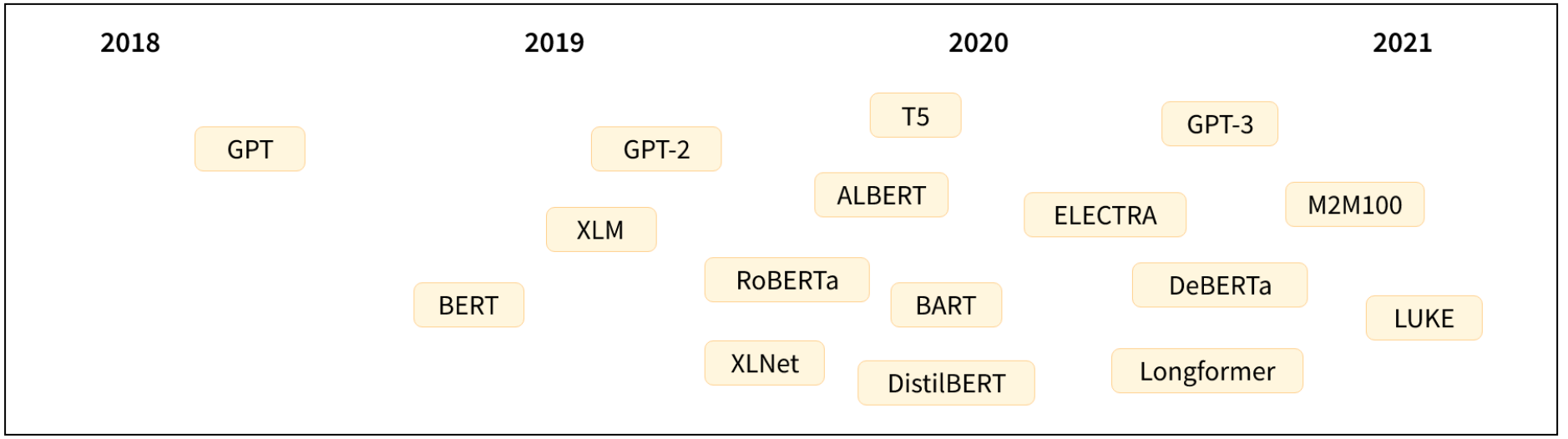
Łukasz Kaiser[*] Google Brain lukaszkaiser@google.com

Illia Polosukhin[*] [‡] illia.polosukhin@gmail.com



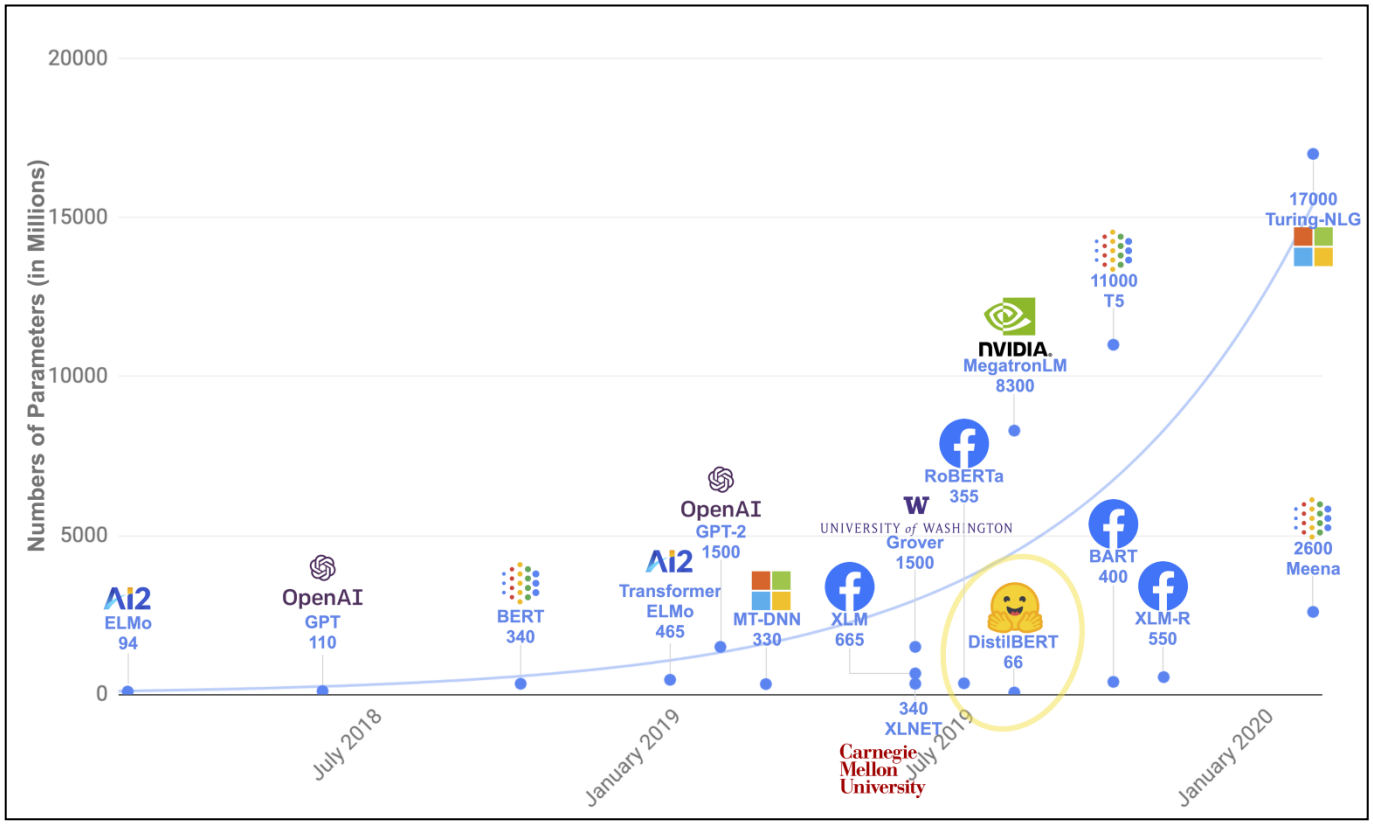
History of Transformers

- The Transformer architecture was introduced in **June 2017**.
  - The focus of the original research was on translation tasks.



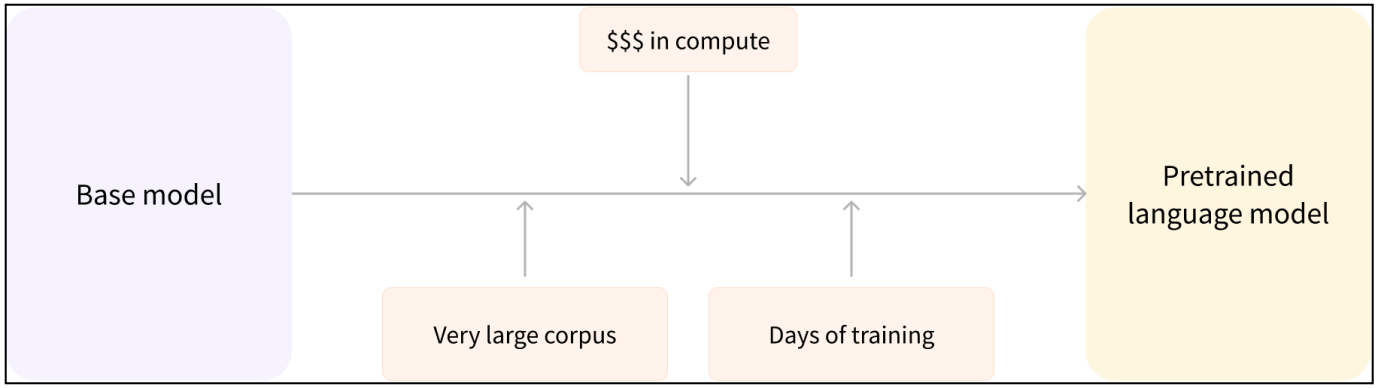
Transformers Size Problem

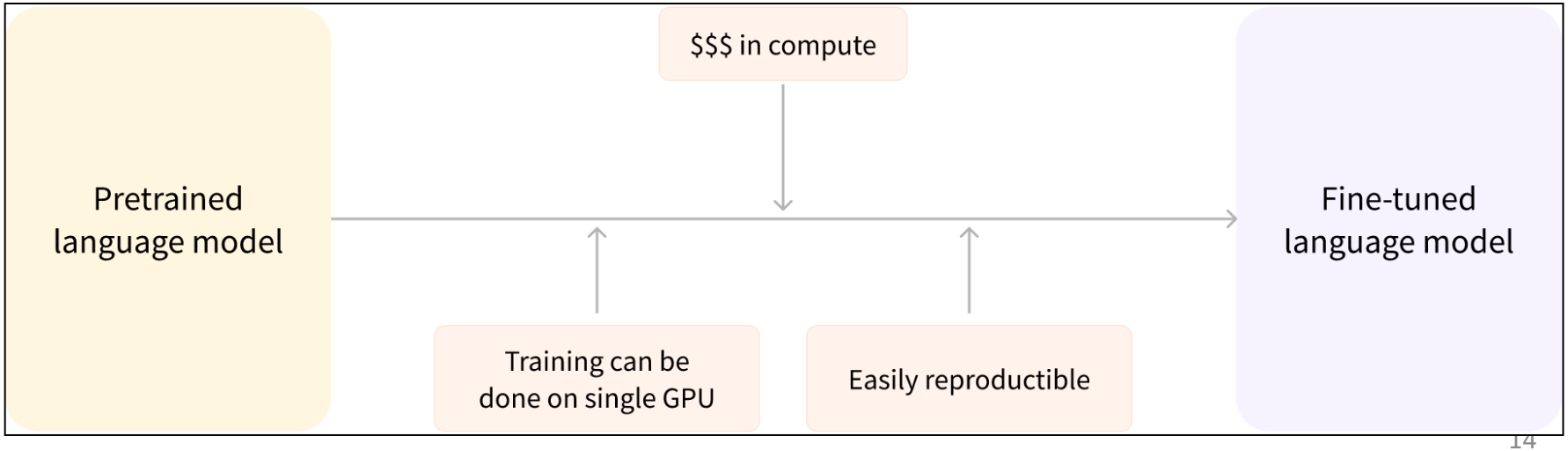
- Transformers are big models
  - Training requires a large amount of data



Solution

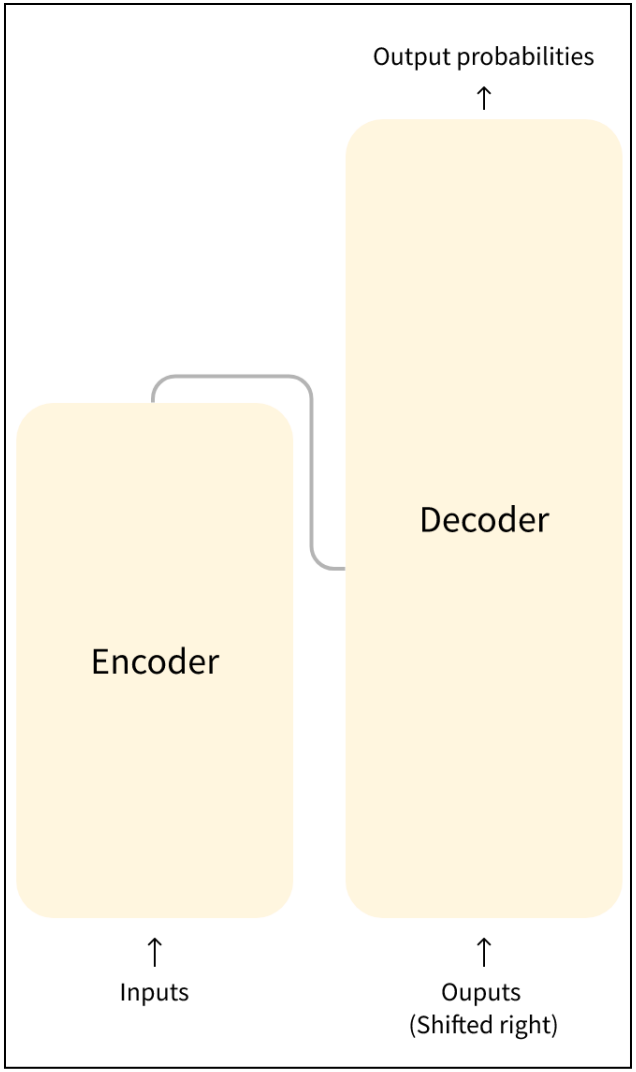
- Pretarining & Finetuning with transformers





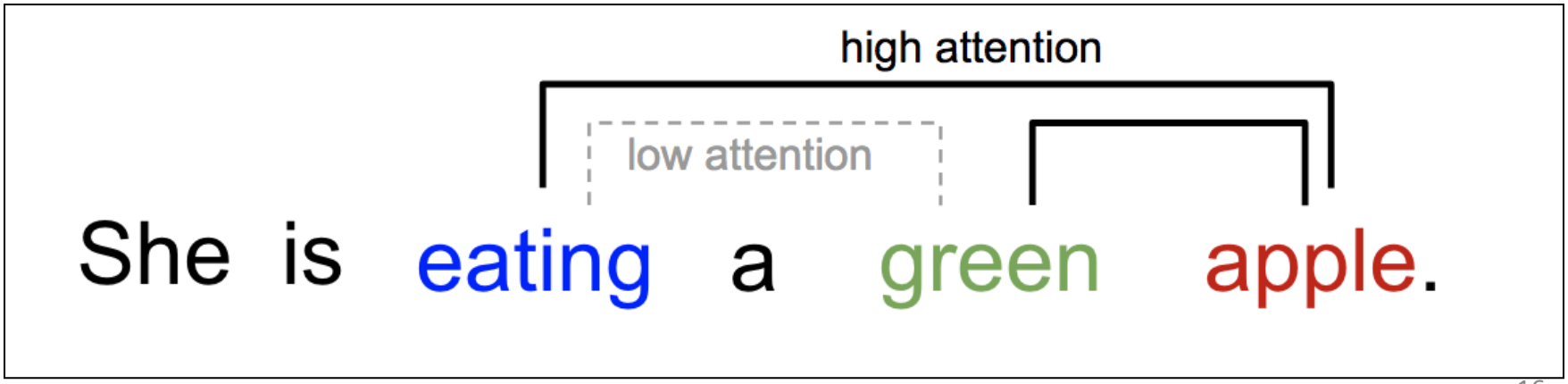
Transformer Architecture

- Two blocks:
  - **Encoder (left):** The encoder receives an input and builds a representation of it (its **features**).
    - Optimized to acquire understanding from the input.
  - **Decoder (right):** The decoder uses the encoder's representation (features) along with other inputs to generate a **target sequence**.
    - Optimized for generating outputs.



Attention Layers

- Main feature of Transformer models is that they are built with special layers called **attention layers**
- A word by itself has a meaning, but that meaning is deeply affected by the **context**, which can be any other word (or words) before or after the word being studied.



TYPES OF TRANSFORMER MODELS

Encoder-only models

- Encoder models use only the encoder of a Transformer model
- Attention layers can access all the words in the sentence called "**bi-directional**" attention
- Also called **auto-encoding models**.
- Used for tasks that require understanding of the input
  - Sentence **classification**
  - Named entity recognition
- Family of encoder models
  - ALBERT, **BERT**, DistilBERT, ELECTRA, RoBERTa

Decoder-only models

- Decoder models use **only the decoder** of a Transformer model.
- For each given word the attention layers can only access the **words positioned before** it in the sentence.
- These models are often called auto-regressive models.
- Also called **Language Models**
- Used for tasks involving text generation.
  - **Predicting the next word in the sentence**.
- Representatives of this family of models include:
  - CTRL, GPT, GPT-2, Transformer XL

Sequence-to-Sequence Models

- **Encoder-decoder** models (also called sequence-to-sequence models) use both parts of the Transformer architecture.
- Sequence-to-sequence models are best suited for tasks revolving around **generating new sentences depending on a given input**
  - Summarization
  - Translation
  - Generative question answering.
- Representatives of this family of models include:
  - BART
  - mBART
  - Marian
  - T5

Transformer Types Summary

Model	Examples	Tasks
Encoder	ALBERT, BERT, DistilBERT, ELECTRA, RoBERTa	Sentence classification, named entity recognition, extractive question answering
Decoder	CTRL, GPT, GPT-2, Transformer XL	Text generation
Encoder-decoder	BART, T5, Marian, mBART	Summarization, translation, generative question answering

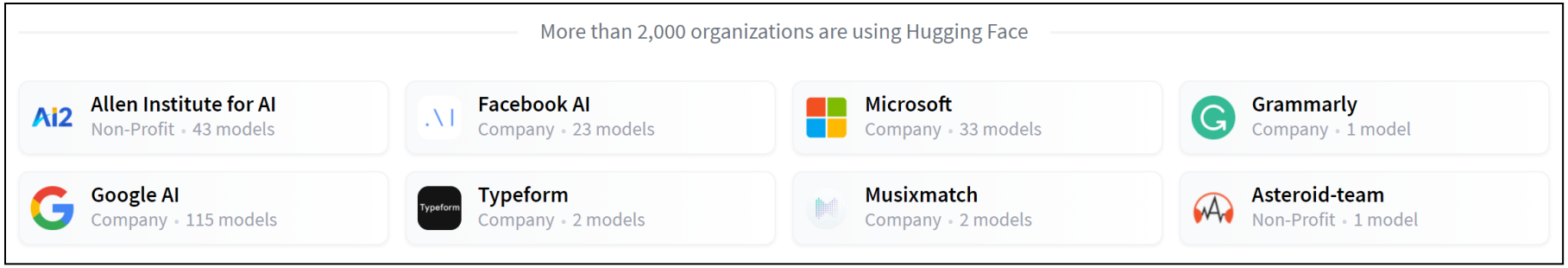
NLP Useful Libraries

- Gensim
- NLTK

HUGGING FACE TRANSFORMERS LIBRARY

Transformers Library

- The **Transformers library** provides the functionality to create and use of shared models.
- The **Model Hub** contains thousands of pretrained models & datasets that anyone can download and use.



Why Transformers Library

- Ease of use:
  - Downloading, loading, and using a state-of-the-art NLP model for inference can be done in just **two lines of code**.
- Flexibility:
  - At their core, all models are simple **PyTorch nn.Module** or TensorFlow tf.keras.Model classes and can be handled like any other models in their respective machine learning (ML) frameworks.
- Simplicity:
  - Hardly any abstractions are made across the library. The **"All in one file"** is a core concept: a model's forward pass is entirely defined in a single file, so that the code itself is understandable and **hackable**.

The Pipeline Tool

- The most basic object in the Transformers library is the **pipeline() function**.
- It connects a model with its necessary **preprocessing** and **postprocessing** steps, allowing us to directly input any text and get an intelligible answer:

```
from transformers import pipeline

classifier = pipeline("sentiment-analysis")
classifier("I've been waiting for a HuggingFace course my whole life.")
```

```
[{'label': 'POSITIVE', 'score': 0.9598047137260437}]
```

Available Pipelines

- `feature-extraction` (get the vector representation of a text)
- `fill-mask`
- `ner` (named entity recognition)
- `question-answering`
- `sentiment-analysis`
- `summarization`
- `text-generation`
- `translation`
- `zero-shot-classification`

Zero-shot classification

```
from transformers import pipeline

classifier = pipeline("zero-shot-classification")
classifier(
    "This is a course about the Transformers library",
    candidate_labels=["education", "politics", "business"],
)
```

```
{'sequence': 'This is a course about the Transformers library',
 'labels': ['education', 'business', 'politics'],
 'scores': [0.8445963859558105, 0.111976258456707, 0.043427448719739914]}
```

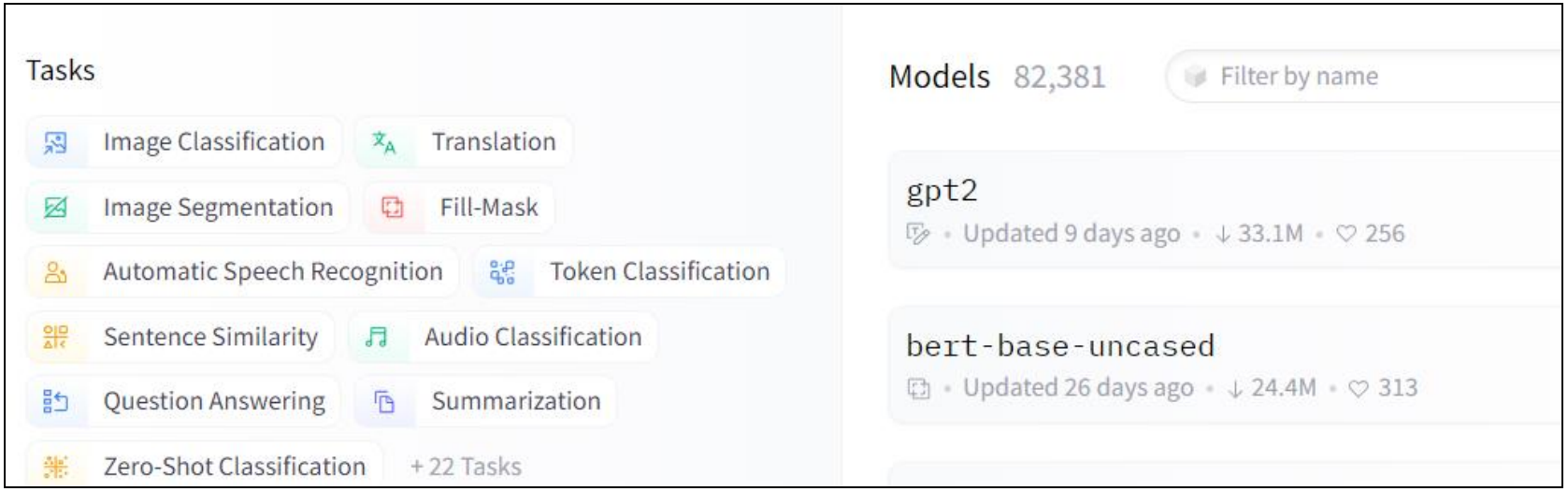
Text Generation

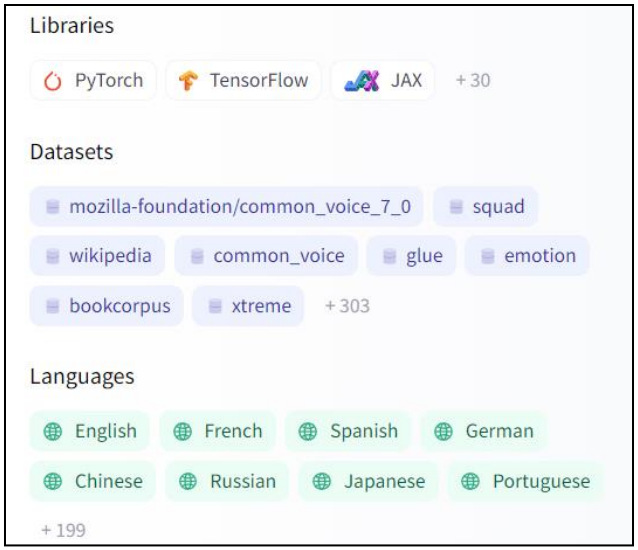
```
from transformers import pipeline

generator = pipeline("text-generation")
generator("In this course, we will teach you how to")
```

```
[{'generated_text': 'In this course, we will teach you how to understand and use '
                    'data flow and data interchange when handling user data. We '
                    'will be working with one or more of the most commonly used '
                    'data flows — data flows of various types, as seen by the '
                    'HTTP'}]
```

You can control how many different sequences are generated with the argument `num_return_sequences` and the total length of the output text with the argument `max_length`.

Model-Hub


Tasks
Image Classification
Translation
Image Segmentation
Fill-Mask
Automatic Speech Recognition
Token Classification
Sentence Similarity
Audio Classification
Question Answering
Summarization
Zero-Shot Classification
+ 22 Tasks
Models 82,381
Filter by name
gpt2
Updated 9 days ago
↓ 33.1M
♡ 256
bert-base-uncased
Updated 26 days ago
↓ 24.4M
♡ 313


Libraries
PyTorch
TensorFlow
JAX
+ 30
Datasets
mozilla-foundation/common_voice_7_0
squad
wikipedia
common_voice
glue
emotion
bookcorpus
xtreme
+ 303
Languages
English
French
Spanish
German
Chinese
Russian
Japanese
Portuguese
+ 199

Pipeline with Selected Model

```
from transformers import pipeline

generator = pipeline("text-generation", model="distilgpt2")
generator(
    "In this course, we will teach you how to",
    max_length=30,
    num_return_sequences=2,
)
```

```
[{'generated_text': 'In this course, we will teach you how to manipulate the world and
                     'move your mental and physical capabilities to your advantage.'},
 {'generated_text': 'In this course, we will teach you how to become an expert and '
                     'practice realtime, and with a hands on experience on both real '
                     'time and real'}]
```

Mask Filling Pipeline

```
from transformers import pipeline

unmasker = pipeline("fill-mask")
unmasker("This course will teach you all about <mask> models.", top_k=2)
```

```
[{'sequence': 'This course will teach you all about mathematical models.',
  'score': 0.19619831442832947,
  'token': 30412,
  'token_str': ' mathematical'},
 {'sequence': 'This course will teach you all about computational models.',
  'score': 0.04052725434303284,
  'token': 38163,
  'token_str': ' computational'}]
```

Named Entity Recognition

```
from transformers import pipeline

ner = pipeline("ner", grouped_entities=True)
ner("My name is Sylvain and I work at Hugging Face in Brooklyn.")
```

```
[{'entity_group': 'PER', 'score': 0.99816, 'word': 'Sylvain', 'start': 11, 'end': 18},
 {'entity_group': 'ORG', 'score': 0.97960, 'word': 'Hugging Face', 'start': 33, 'end': 45},
 {'entity_group': 'LOC', 'score': 0.99321, 'word': 'Brooklyn', 'start': 49, 'end': 57}
]
```

Question Answering

```
from transformers import pipeline

question_answerer = pipeline("question-answering")
question_answerer(
    question="Where do I work?",
    context="My name is Sylvain and I work at Hugging Face in Brooklyn",
)
```

```
{'score': 0.6385916471481323, 'start': 33, 'end': 45, 'answer': 'Hugging Face'}
```

Translation

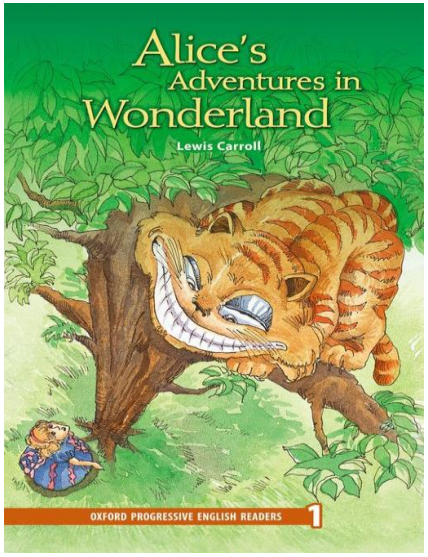
```
from transformers import pipeline

translator = pipeline("translation", model="Helsinki-NLP/opus-mt-fr-en")
translator("Ce cours est produit par Hugging Face.")
```

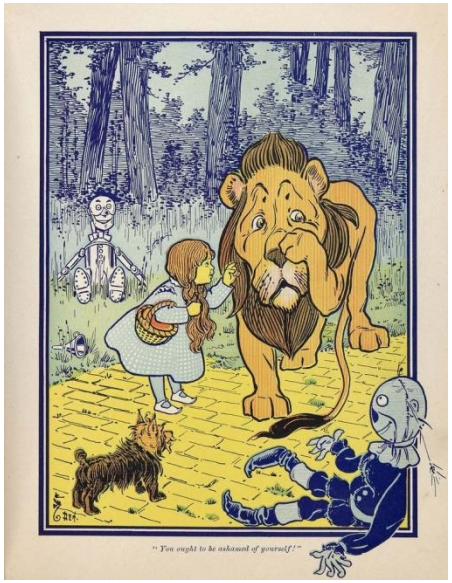
```
[{'translation_text': 'This course is produced by Hugging Face.'}]
```

NLP BASICS

Corpus

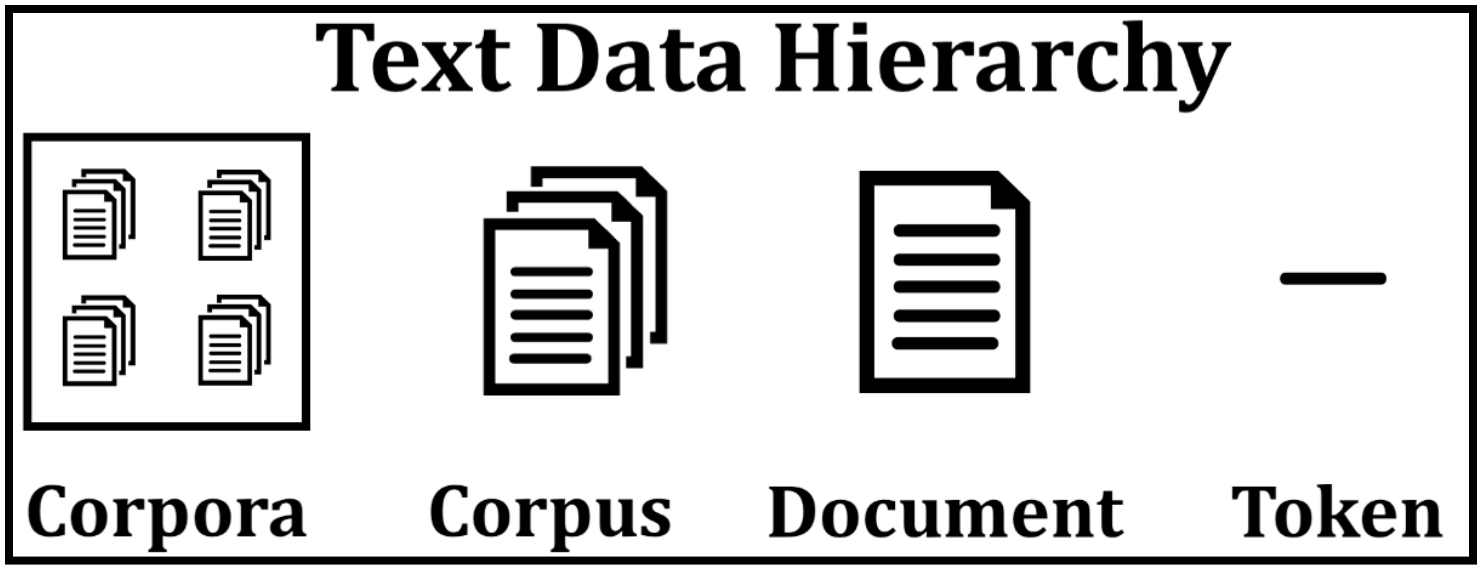
- A corpus is a structured set of documents organized into a dataset
  - Document can be
    - Sentence
    - Paragraph
    - Whole book
- Example Two books
  - "Alice's Adventures in Wonderland"
  - "The wonderful wizard of Oz"





Corpus Example

- Two books (Corpora)
  - "Alice's Adventures in Wonderland" (Corpus)
  - "The wonderful wizard of Oz" (Corpus)
- We apply NLP models at **document level**
  - **Usually Sentences or group of Sentences**



Tokenization

- Tokenization is the task of chopping it up into **pieces**, called tokens , perhaps at the same time throwing away certain characters, such as punctuation
- Translate text into data that can be processed by the model
- Creating **Vocabulary** is the ultimate goal of Tokenization
- Models can only process numbers, so tokenizers need to convert our **text inputs** to numerical data.

```
sentence = "I'm following the white rabbit"
tokens = sentence.split(' ')
tokens
```

Output

```
["I'm", 'following', 'the', 'white', 'rabbit']
```

Tokenization Types

- Word-based
  - huge amount of tokens
- Character-based
  - it's less meaningful: each character doesn't mean a lot on its own
- **Subword** tokenization
  - rare words should be decomposed into meaningful subwords
- Other methods:
  - Byte-level BPE, as used in GPT-2
  - WordPiece, as used in BERT
  - SentencePiece or Unigram, as used in several multilingual models

```
['Using', 'a', 'transform', '##er', 'network', 'is', 'simple']
```

Tokenization (Filters)

```
from gensim.parsing.preprocessing import *
preprocess_string(sentence)
```

Outpu

```
['follow', 'white', 'rabbit']
```

Tokenization (Filters)

- `strip_tags` (for removing HTML-like tags between brackets)
- `strip_punctuation`
- `strip_multiple_whitespaces`
- `strip_numeric`

- `strip_short`: it **discards** any word **less than three characters** long
- `remove_stopwords`: it **discards** any word that is considered a **stopword** (like "*the*", "*but*", "*then*", and so on)
- `stem_text`: it **modifies** words by **stemming** them, that is, reducing them to a **common base form** (from "*following*" to its base "*follow*", for example)

Tokenization (Filters)

```
filters = [lambda x: x.lower(),
           strip_tags,
           strip_punctuation,
           strip_multiple_whitespaces, strip_numeric]
preprocess_string(sentence, filters=filters)
```

Output

```
['i', 'm', 'following', 'the', 'white', 'rabbit']
```

Tokenization (Vocabulary)

- After Tokenization we can build vocabulary
- The **vocabulary** is a list of unique words that appear in text corpora (after tokenization!)

```
sentences = train_dataset['sentence']
tokens = [simple_preprocess(sent) for sent in sentences]
tokens[0]
```

```
from gensim import corpora
dictionary = corpora.Dictionary(tokens)
print(dictionary)
```

```
Dictionary(3704 unique tokens: ['and', 'as', 'far', 'knew', 'quite'
]...)
```

Tokenization (Vocabulary)

```
sentence = 'follow the white rabbit'
new_tokens = simple_preprocess(sentence)
ids = dictionary.doc2idx(new_tokens)
print(new_tokens)
print(ids)
```

```
['follow', 'the', 'white', 'rabbit']
[1482, 20, 497, 333]
```

Converted text to numbers!

Tokenization (Special Tokens)

- What if a word does not exist in the dictionary
  - We add **[UNK]** token (exist in dictionary at some index)
- To make length of two sentences same. **Why?**
  - We add **[PAD]** token
- What if document contain multiple sentences. **Example?**
  - We add **[SEP]** tokens
- Classification token (special token whose features will be used for classification)
  - We add **[CLS]** token (exist in dictionary at some index)

Hugging Face Tokenizers

- In NLP mostly we will use pre-trained models like BERT
  - So we have to tokenize our documents according to the tokenizers on which model was trained
  - Thanks to HF each model comes with its own tokenizer (***we don't need to build our own*** ☺)

Hugging Face Tokenizers (Encoding)

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")

sequence = "Using a Transformer network is simple"
tokens = tokenizer.tokenize(sequence)

print(tokens)
```

The output of this method is a list of strings, or tokens:

```
['Using', 'a', 'transform', '##er', 'network', 'is', 'simple']
```

Hugging Face Tokenizers (Ids)

```
ids = tokenizer.convert_tokens_to_ids(tokens)

print(ids)

```

```
[7993, 170, 11303, 1200, 2443, 1110, 3014]
```

Hugging Face Tokenizers (Decoding)

```
decoded_string = tokenizer.decode([7993, 170, 11303, 1200, 2443, 1110, 3014])
print(decoded_string)
```

```
'Using a Transformer network is simple'
```

Hugging Face Tokenizers (All)

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")
```

```
tokenizer("Using a Transformer network is simple")
```

```
{'input_ids': [101, 7993, 170, 11303, 1200, 2443, 1110, 3014, 102],
 'token_type_ids': [0, 0, 0, 0, 0, 0, 0, 0, 0],
 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1]}
```

Hugging Face Tokenizers (with/without special tokens)

```
sequence = "I've been waiting for a HuggingFace course my whole life."

model_inputs = tokenizer(sequence)
print(model_inputs["input_ids"])

tokens = tokenizer.tokenize(sequence)
ids = tokenizer.convert_tokens_to_ids(tokens)
print(ids)
```

```
[101, 1045, 1005, 2310, 2042, 3403, 2005, 1037, 17662, 12172, 2607, 2026, 2878, 2166, 1012, 102]
[1045, 1005, 2310, 2042, 3403, 2005, 1037, 17662, 12172, 2607, 2026, 2878, 2166, 1012]
```

```
print(tokenizer.decode(model_inputs["input_ids"]))
print(tokenizer.decode(ids))
```

```
"[CLS] i've been waiting for a huggingface course my whole life. [SEP]"
"i've been waiting for a huggingface course my whole life."
```

Hugging Face Tokenizers (Batch of Sequences. Smaller Sequence)

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

model_inputs = tokenizer(sequences)
```

```
# Will pad the sequences up to the maximum sequence length
model_inputs = tokenizer(sequences, padding="longest")

# Will pad the sequences up to the model max length
# (512 for BERT or DistilBERT)
model_inputs = tokenizer(sequences, padding="max_length")

# Will pad the sequences up to the specified max length
model_inputs = tokenizer(sequences, padding="max_length", max_length=8)
```

Hugging Face Tokenizers (longer Sentences)

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

# Will truncate the sequences that are longer than the model max length
# (512 for BERT or DistilBERT)
model_inputs = tokenizer(sequences, truncation=True)

# Will truncate the sequences that are longer than the specified max length
model_inputs = tokenizer(sequences, max_length=8, truncation=True)
```

Hugging Face Tokenizers (Framework)

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

# Returns PyTorch tensors
model_inputs = tokenizer(sequences, padding=True, return_tensors="pt")

# Returns TensorFlow tensors
model_inputs = tokenizer(sequences, padding=True, return_tensors="tf")

# Returns NumPy arrays
model_inputs = tokenizer(sequences, padding=True, return_tensors="np")
```

DATA AUGMENTATION IN NLP

Data Augmentation in NLP

- **Word dropout**
  - Simply randomly replaces words by some other random word or a special [UNK] token (word)
- Replace words with their **synonyms**, so the meaning of the text is preserved
  - **WordNet** (database for synonyms)

```
# !pip install textattack
from textattack.augmentation import EmbeddingAugmenter
augmenter = EmbeddingAugmenter()
feynman = 'What I cannot create, I do not understand.'
```

```
for i in range(5):
    print(augmenter.augment(feynman))
```

```
['What I cannot create, I do not fathom.']
['What I cannot create, I do not understood.']
['What I notable create, I do not understand.']
['What I cannot creating, I do not understand.']
['What I significant create, I do not understand.']
```

HUGGING FACE DATASETS LIBRARY

Hugging Face Datasets Library

- Load a dataset from the Hugging Face **Hub**.
- **Preprocess?** the data with Dataset.map()
- Provides loading scripts to handle the loading of **local and remote** datasets.
- **Slicing and dicing** our data
- Frees you from **memory** management
- Datasets in lots of different **languages**
- Support **conversion** to Pytorch Dataset

Loading a dataset from the Hub

```
from datasets import load_dataset

raw_datasets = load_dataset("glue", "mrpc")
raw_datasets
```

```
DatasetDict({
    train: Dataset({
        features: ['sentence1', 'sentence2', 'label', 'idx'],
        num_rows: 3668
    })
    validation: Dataset({
        features: ['sentence1', 'sentence2', 'label', 'idx'],
        num_rows: 408
    })
    test: Dataset({
        features: ['sentence1', 'sentence2', 'label', 'idx'],
        num_rows: 1725
    })
})
```

```
raw_train_dataset = raw_datasets["train"]
raw_train_dataset[0]
```

```
{'idx': 0,
 'label': 1,
 'sentence1': 'Amrozi accused his brother , whom he ca
 'sentence2': 'Referring to him as only " the witness
```

Preprocessing a dataset

```
from transformers import AutoTokenizer

checkpoint = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
tokenized_sentences_1 = tokenizer(raw_datasets["train"]["sentence1"])
tokenized_sentences_2 = tokenizer(raw_datasets["train"]["sentence2"])
```

What's wrong here?

Preprocessing a dataset

```
inputs = tokenizer("This is the first sentence.", "This is the second one.")
inputs
```

```
{
  'input_ids': [101, 2023, 2003, 1996, 2034, 6251, 1012, 102, 2023, 2003, 1996, 2117, 2028, 1012
  'token_type_ids': [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1],
  'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
}
```

```
tokenizer.convert_ids_to_tokens(inputs["input_ids"])
```

```
['[CLS]', 'this', 'is', 'the', 'first', 'sentence', '.', '[SEP]', 'this', 'is', 'the', 'second
```

Preprocessing a dataset

```
tokenized_dataset = tokenizer(
    raw_datasets["train"]["sentence1"],
    raw_datasets["train"]["sentence2"],
    padding=True,
    truncation=True,
)
```

Only work if you have enough RAM to store your whole dataset during the tokenization

Preprocessing a dataset (with map function)

```
def tokenize_function(example):
    return tokenizer(example["sentence1"], example["sentence2"], truncation=True)
```

```
tokenized_datasets = raw_datasets.map(tokenize_function, batched=True)
tokenized_datasets
```

- Keeps the dataset as dataset
- Memory efficient
- Batch wise to speed up

Hugging Face Models

```python
from transformers import BertConfig, BertModel

config = BertConfig()
model = BertModel(config)

# Model is randomly initialized!
```

```
BertConfig {
  [...]
  "hidden_size": 768,
  "intermediate_size": 3072,
  "max_position_embeddings": 512,
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  [...]
}
```

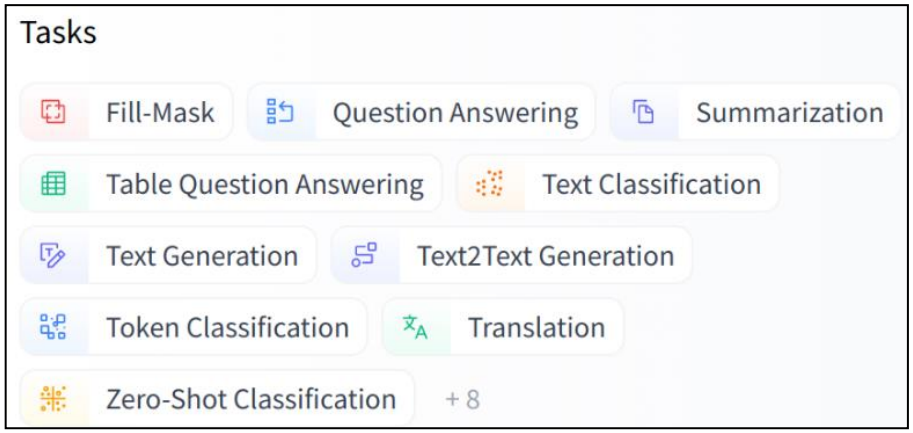
```python
from transformers import BertModel

model = BertModel.from_pretrained("bert-base-cased")
```

```python
model.save_pretrained("directory_on_my_computer")
```

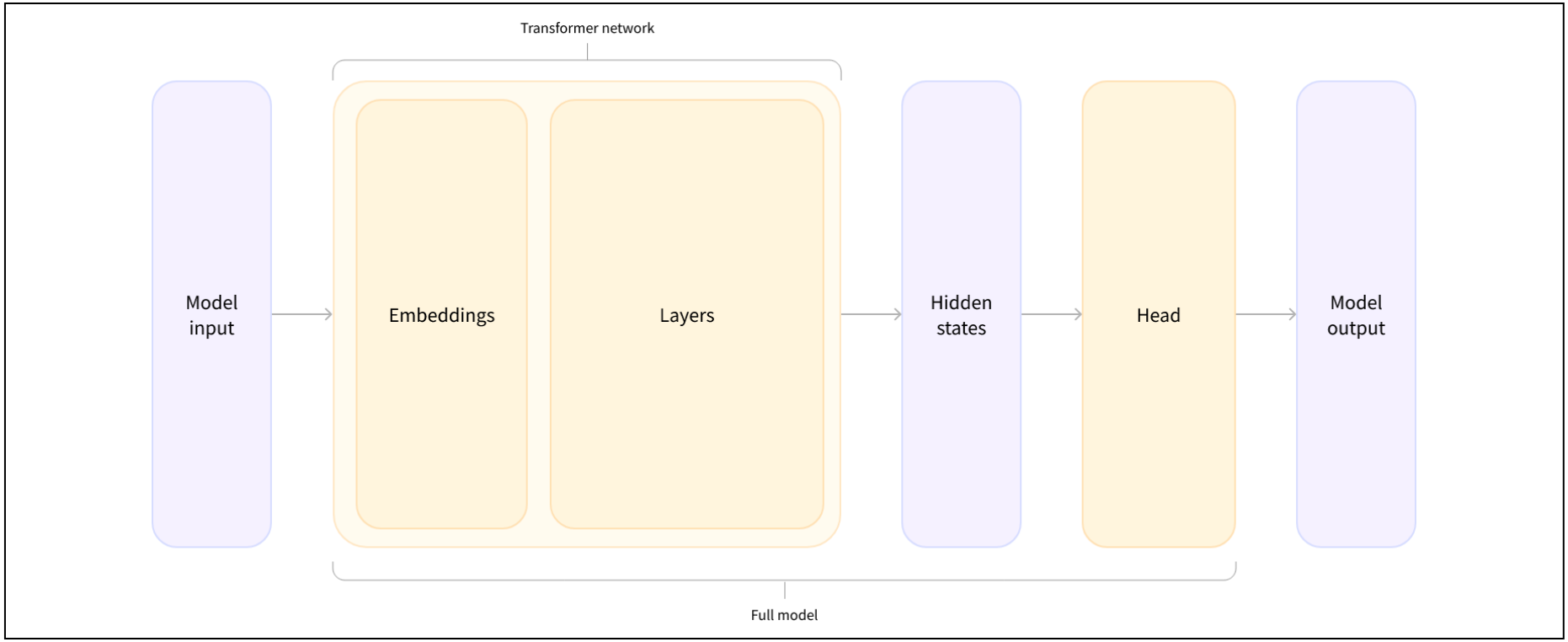
```python
#loading a pretrained model
model = AutoModelForSequenceClassification.from_pretrained("bert-base-cased", num_labels=5,cache_dir="HuggingFace")
```

Using a Transformer model for Inference

```python
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification

checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
model = AutoModelForSequenceClassification.from_pretrained(checkpoint)
sequences = ["I've been waiting for a HuggingFace course my whole life.", "So have I!"]

tokens = tokenizer(sequences, padding=True, truncation=True, return_tensors="pt")
output = model(**tokens)
```



Model Head + Embedding



We will dig it next lecture for deeper understanding of transformers & attention

5 Class Text Classification

FINE TUNING BERT ON YELP REVIEW DATASET

Imports

```
from datasets import load_dataset
from transformers import AutoTokenizer
from datasets import load_dataset
from transformers import AutoTokenizer
from transformers import AutoModelForSequenceClassification
from transformers import TrainingArguments
import evaluate
from transformers import TrainingArguments, Trainer
from torch.optim import AdamW
```

Loading dataset

```
#loading/dowloading dataset
dataset = load_dataset("yelp_review_full",cache_dir="HuggingFace")

#Tokenzing the text in dataset
tokenizer = AutoTokenizer.from_pretrained("bert-base-cased",cache_dir="HuggingFace")
def tokenize_function(examples):
    return tokenizer(examples["text"], padding="max_length", truncation=True)
tokenized_datasets = dataset.map(tokenize_function, batched=True)
```

```
tokenized_datasets = tokenized_datasets.remove_columns(["text"])
tokenized_datasets = tokenized_datasets.rename_column("label", "labels")
tokenized_datasets.set_format("torch")
small_train_dataset = tokenized_datasets["train"].shuffle(seed=42).select(range(1000))
small_eval_dataset = tokenized_datasets["test"].shuffle(seed=42).select(range(1000))
```

Loading Model

```python
#loading a pretrained model
model = AutoModelForSequenceClassification.from_pretrained("bert-base-cased", num_labels=5,cache_dir="HuggingFace")
print(model)
summary(model,input_size=(768,),depth=1,batch_dim=1, dtypes=['torch.IntTensor'])
for name, param in model.named_parameters():
    if param.requires_grad == True:
        print(name)

model.to(device)
num_params = sum(param.numel() for param in model.parameters())
num_trainable_params = sum(param.numel() for param in model.parameters() if param.requires_grad)
```

```
==========================================================================
Layer (type:depth-idx)                   Param #
==========================================================================
├─BertModel: 1-1                        108,310,272
├─Dropout: 1-2                          --
├─Linear: 1-3                           3,845
==========================================================================
Total params: 108,314,117
Trainable params: 108,314,117
```

Training Loop

```
lr=5e-5
optimizer = optim.AdamW(model.parameters(), lr=lr)
#tensorboard
tboardWriter=SummaryWriter('runs/TextClassification-BERT')
#batch wise training loop
epochs = 2
train_losses = []
val_losses = []
best_accuracy=0
for epoch in range(epochs):  #epochs loop
    all_Y_train_epoch=np.array([]).reshape(0,1)
    all_Yhat_train_epoch=np.array([]).reshape(0,1)
    all_train_losses_epoch=np.array([])

    for batch in train_loader:        #batch wise  training on train set
        model.train()
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        loss = outputs.loss
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
        logits = outputs.logits
```

Any Changes?

COLAB version

```
!pip install datasets
!pip install transformers
!pip install evaluate
```

```
dataset = load_dataset("yelp_review_full",cache_dir="/content/drive/MyDrive/HuggingFace/")
```

```
#Traiing PYtorch way
tokenizer = AutoTokenizer.from_pretrained("bert-base-cased",cache_dir="/content/drive/MyDrive/HuggingFace/")
```

Home Task 6

- Fine tune BERT for 3 class classification using **tweet_eval** dataset

For `sentiment` config:

- `text`: a `string` feature containing the tweet.
- `label`: an `int` classification label with the following mapping:

  `0`: negative

  `1`: neutral

  `2`: positive