Deep Learning for Natural Language Processing (NLP) tasks with Python and Google Colab: an introduction

Don't panic! *You are going to use Large Language Models (LLMs); you are not going to create new ones*

1. Preliminaries

- Preliminaries: Python libraries
  - NumPy is a Python library for scientific computing
  - scikit-learn is a Python library for machine learning, built on NumPy, SciPy, and Matplotlib
    - SciPy is a Python library for scientific computing
    - Matplotlib is a Python library for static, animated, and interactive visualisations
  - PyTorch and TensorFlow are Python-based frameworks for machine learning
  - Flair and Sentence Transformers are Python-based frameworks for state-of-the-art NLP
  - Transformers is a Python library supporting Deep Learning for NLP
    - Transformers library can work with both PyTorch and TensorFlow

- Preliminaries: Google Colaboratory (Google Colab for short)
  - Is a product from Google Research
  - Is well suited to machine learning, supporting Deep Learning for NLP using GPUs
  - Allows anybody to write and execute arbitrary python code through a Web browser

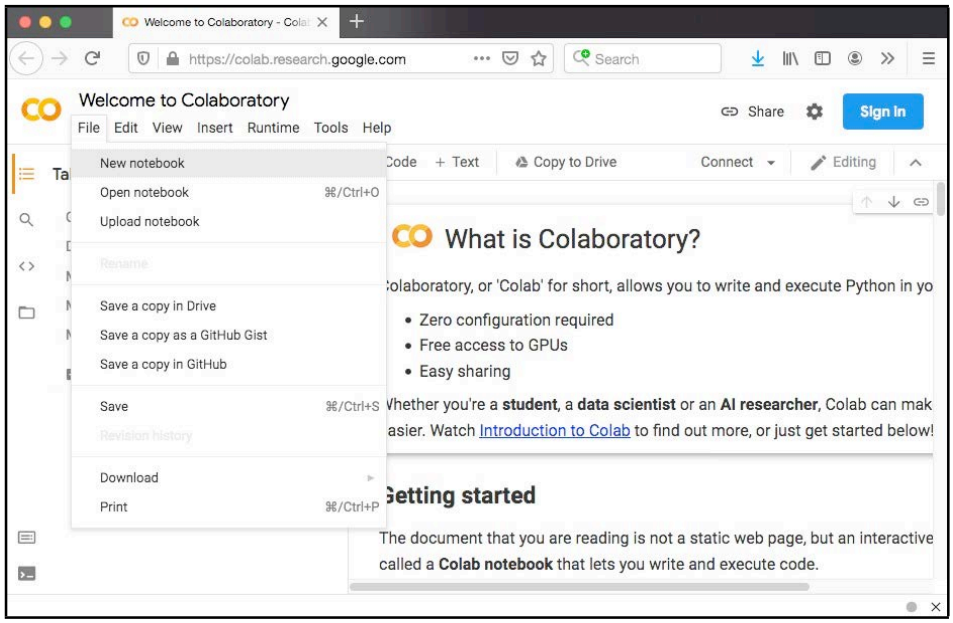
2. Using Google Colab for NLP tasks

Masked word: the model is given a sequence of words with the goal of predicting a "masked" word

- Let's exemplify the "masked" word task with Google Colab
  - Open a Web browser (e.g. Google Chrome or Mozilla Firefox)
  - Paste the following Uniform Resource Locator (URL) in the navigator bar of the Web browser

https://colab.research.google.com

  - You can create a new Colab notebook by going to the File menu as in the figure below

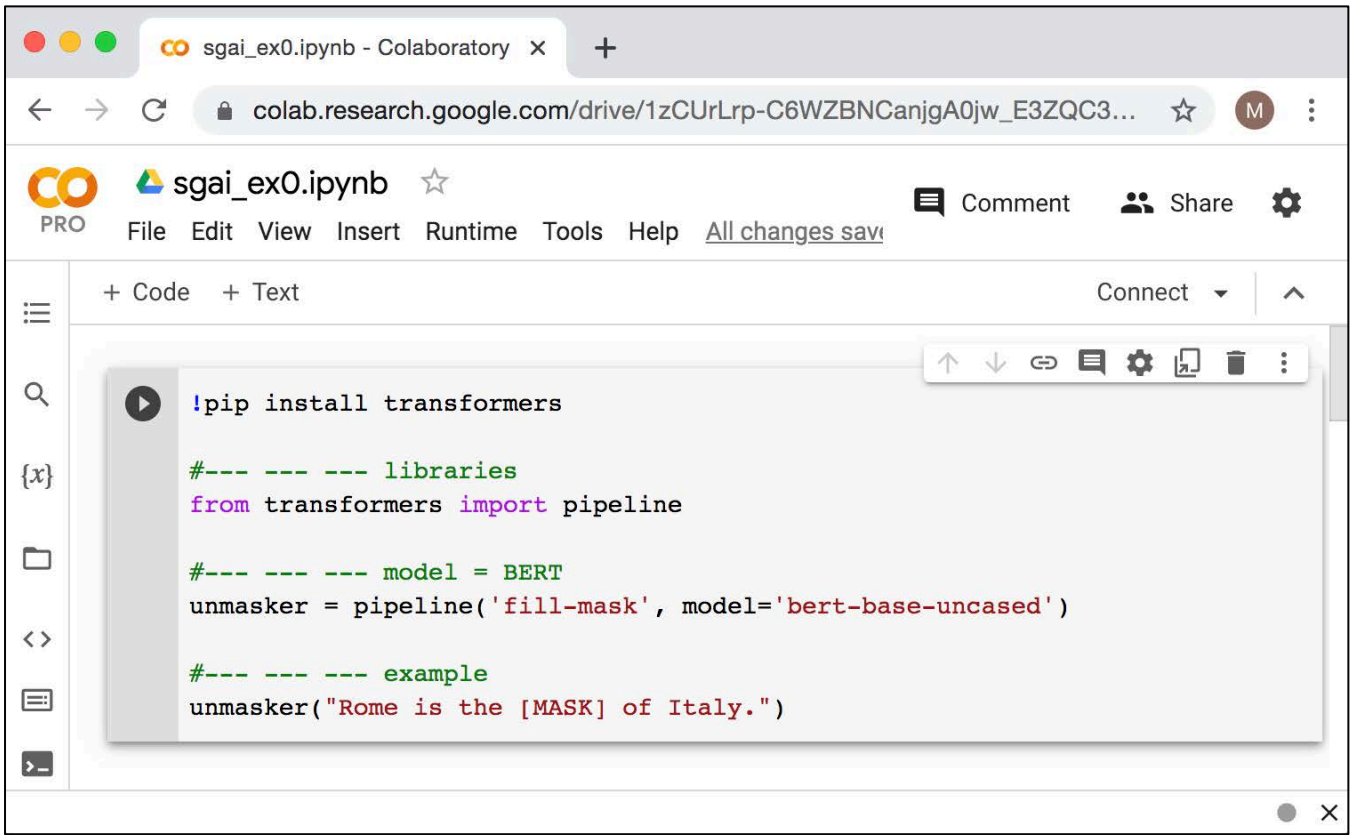


  - Create a new Colab notebook with the name "ex0.ipynb". Careful! Google sign-in is required

- The LLM is bert-base-uncased available at https://huggingface.co/bert-base-uncased
- Paste the following Python code in the "new" text area (also called cell)

```
#--- --- --- to install transformers
!pip install transformers

#--- --- --- to load libraries
from transformers import pipeline

#--- --- --- model = BERT
unmasker = pipeline('fill-mask', model='bert-base-uncased')

#--- --- --- example
unmasker("Rome is the [MASK] of Italy.")
```

- Being all fine, you will see something like the following



- Run cell: to execute click on circle-triangle (top left-hand side)

- Being all fine, you will see something like the following after clicking on circle-triangle

- At the end of the cell executed (image above) appear the predictions for a "masked" word

```
[{'score': 0.9928465485572815,
  'token': 3007,
  'token_str': 'capital',
  'sequence': 'rome is the capital of italy.'},
 {'score': 0.0012917355634272099,
  'token': 2415,
  'token_str': 'center',
  'sequence': 'rome is the center of italy.'},
 {'score': 0.0009973242413252592,
  'token': 14508,
  'token_str': 'birthplace',
  'sequence': 'rome is the birthplace of italy.'},
 {'score': 0.0008744171936996281,
  'token': 2540,
  'token_str': 'heart',
  'sequence': 'rome is the heart of italy.'},
 {'score': 0.0006965672364458442,
  'token': 2803,
  'token_str': 'centre',
  'sequence': 'rome is the centre of italy.'}]
```

3. Large Language Models (LLMs): examples from SGAI 2023 series

Examples from http://www.bcs-sgai.org/seminars/2023-05/?section=programme

- **Predicting a "masked" word**
  - The LLM is given a sequence of words. The goal is to predict a "masked" word
  - The LLM is bert-base-uncased available at https://huggingface.co/bert-base-uncased
  - Example: The man worked as a [MASK].

Python code ready to be executed in Google Colab

```
#--- --- --- to install transformers
!pip install transformers
#--- --- --- to load libraries
from transformers import pipeline
#--- --- --- model = BERT
unmasker = pipeline('fill-mask', model='bert-base-uncased')
#--- --- --- example
unmasker("The man worked as a [MASK].")
```

Output excerpt from Google Colab after the python code in the cell is executed

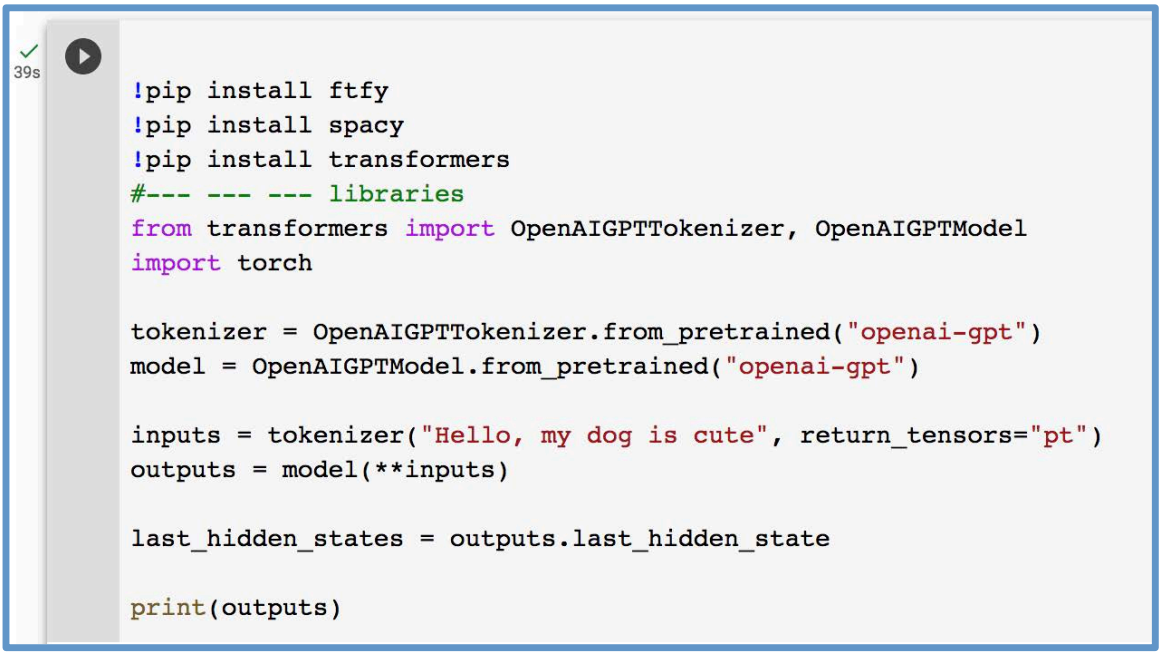
```
[{'score': 0.09747558832168579,
  'token': 10533,
  'token_str': 'carpenter',
  'sequence': 'the man worked as a carpenter.'},
 {'score': 0.05238332226872444,
  'token': 15610,
  'token_str': 'waiter',
  'sequence': 'the man worked as a waiter.'},
 {'score': 0.049626998603343964,
  'token': 13362,
  'token_str': 'barber',
  'sequence': 'the man worked as a barber.'},
 {'score': 0.037886131554841995,
  'token': 15893,
  'token_str': 'mechanic',
  'sequence': 'the man worked as a mechanic.'},
 {'score': 0.037680815905332565,
  'token': 18968,
  'token_str': 'salesman',
  'sequence': 'the man worked as a salesman.'}]
```

  - Another example: The woman worked as a [MASK].

Python code ready to be executed in Google Colab

```
#--- --- --- to install transformers
!pip install transformers
#--- --- --- to load libraries
from transformers import pipeline
#--- --- --- model = BERT
unmasker = pipeline('fill-mask', model='bert-base-uncased')
#--- --- --- example
unmasker("The woman worked as a [MASK].")
```

- **Getting some insights into the inner-workings**
  - The goal is to print the model output for the prompt "Hello, my dog is cute"
  - The LLM is openai-gpt that is available at https://huggingface.co/openai-gpt
  - The following python code was executed in 39 seconds after clicking on the circle with triangle



  - "*All models have outputs that are instances of subclasses of ModelOutput. Those are data structures containing all the information returned by the model*". More information at: https://huggingface.co/docs/transformers/main_classes/output

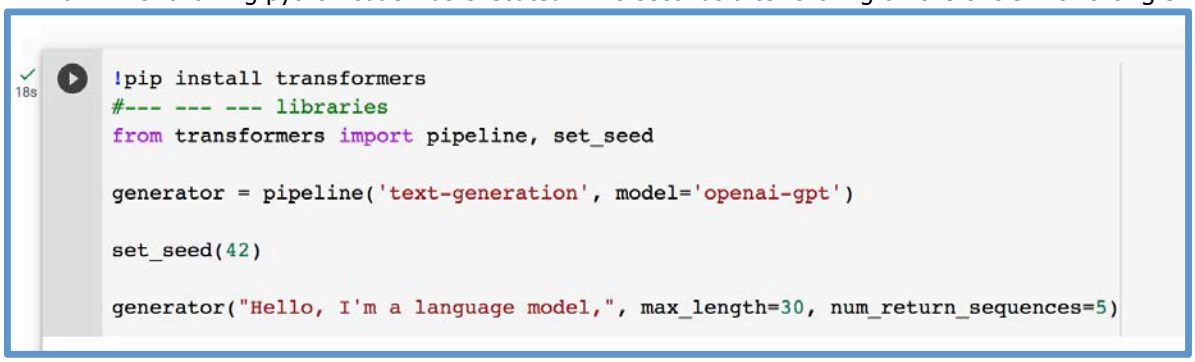
Python code ready to be executed in Google Colab

```
!pip install ftfy
!pip install spacy
!pip install transformers
#--- --- --- libraries
from transformers import OpenAIGPTTokenizer, OpenAIGPTModel
import torch

tokenizer = OpenAIGPTTokenizer.from_pretrained("openai-gpt")
model = OpenAIGPTModel.from_pretrained("openai-gpt")

inputs = tokenizer("Hello, my dog is cute", return_tensors="pt")
outputs = model(**inputs)

last_hidden_states = outputs.last_hidden_state

print(outputs)
```

Output excerpt from Google Colab after the python code in the cell is executed

```
... ... ...
BaseModelOutput(last_hidden_state=tensor([[[ 0.4653,  0.0642,  0.5910,  ...,  0.1177, -
0.0021, -1.2262],
         [-0.3697, -0.0957,  0.6613,  ..., -0.0344, -0.2164,  0.1205],
         [ 0.1700, -0.3252,  0.0407,  ...,  0.1589, -0.8057, -0.2830],
         [-0.3669, -0.0448,  0.8061,  ..., -0.0090, -0.0872, -0.5224],
         [-0.5047,  0.6522,  0.6932,  ...,  0.0811,  0.6475,  0.3190],
         [-0.2972,  0.0591,  1.2333,  ..., -0.7394, -0.2600,  0.0863]]],
       grad_fn=<ViewBackward0>), hidden_states=None, attentions=None)
```

- **Response to the prompt**
  - The goal is to obtain 5 replies for the prompt "Hello, I'm a language model,"
  - The LLM is openai-gpt that is available at https://huggingface.co/openai-gpt
  - The following python code was executed in 18 seconds after clicking on the circle with triangle



Python code ready to be executed in Google Colab

```
!pip install transformers
#--- --- --- libraries
from transformers import pipeline, set_seed

generator = pipeline('text-generation', model='openai-gpt')
set_seed(42)
generator("Hello, I'm a language model,", max_length=30, num_return_sequences=5)
```

Output excerpt from Google Colab after the python code in the cell is executed

```
[{'generated_text': "Hello, I'm a language model, for'e's a very big one. they call me'e'n e'n'"'},
 {'generated_text': 'Hello, I\'m a language model, " he said and leaned his left shoulder against the corner of the door. \n i couldn\'t take my'},
 {'generated_text': "Hello, I'm a language model,'came their reply. \n'it's a man thing! if you have three hundred thousand words,"},
 {'generated_text': 'Hello, I\'m a language model, " he said, the sound of his voice sending a shiver down her spine. \n " a language model'},
 {'generated_text': "Hello, I'm a language model,'he said.'you call me a'man out'in... well, er, in the"}]
```

- Response to the prompt "I am unhappy.": let's compare the text generated by two LLMs
  - The LLM is openai-gpt that is available at https://huggingface.co/openai-gpt

Python code ready to be executed in Google Colab

```
!pip install transformers
#--- --- --- libraries
from transformers import pipeline, set_seed

generator = pipeline('text-generation', model='openai-gpt')
set_seed(42)
generator("I am unhappy.", max_length=30, num_return_sequences=5)
```

Output excerpt from Google Colab after the python code in the cell is executed

```
[{'generated_text': 'I am unhappy. for the first time since i began this mission, it\'s all over. i\'m ready to go home. " \n " my'},
 {'generated_text': 'I am unhappy. \n if this is indeed a sign of the prophecy... \n then there has to be a way to avert the coming apocalypse. \n'},
 {'generated_text': 'I am unhappy. i\'ve tried to understand why you are like this, but i\'ve no idea why you\'re being like this. " \n he'},
 {'generated_text': 'I am unhappy. " even though he made her miserable, despite everything he \'d done, she needed him. \n his hand stilled on her back,'},
 {'generated_text': 'I am unhappy. " \n " you can be unhappy all you want, but let it lay dormant until you finally get the answer. " \n "'}]
```

- The LLM is gpt2 that is available at https://huggingface.co/gpt2

Python code ready to be executed in Google Colab

```
!pip install transformers
#--- --- --- libraries
from transformers import pipeline, set_seed

generator = pipeline('text-generation', model='gpt2')
set_seed(42)
generator("I am unhappy.", max_length=30, num_return_sequences=5)
```

Output excerpt from Google Colab after the python code in the cell is executed

```
[{'generated_text': 'I am unhappy. It means that some people want to go to a hospital for minor problems such as burns or diarrhoea. I do not like'},
 {'generated_text': 'I am unhappy.\n\nI would love to see more people like me before I die. I am still so much in my young years. For'},
 {'generated_text': "I am unhappy.\n\nYou should get in touch with your representative and tell us as to why you didn't get into it with this person."},
 {'generated_text': 'I am unhappy.\n\nAdvertisement Continue reading the main story\n\nThe United States has spent more than $700 billion over the last few years to'},
 {'generated_text': 'I am unhappy. I think all of us in our present circumstances have tried a good deal to make sense of them."\n\nThe president, though'}]
```

- **OpenAI examples**
  - The GPT-2 model is available from Hugging Face. However, we don't have direct access to the GPT-3 model. Therefore, you first need to get an API key from OpenAI
  - The LLM text-davinci-003 belongs to the GPT-3 model family. This model builds on top of OpenAI previous InstructGPT models.
  - *The details of how to fine-tune GPT-3 are mentioned in the OpenAI paper with the title "Training language models to follow instructions with human feedback", available at* https://arxiv.org/pdf/2203.02155.pdf
  - You can create a new Colab notebook to get a response for a prompt (see image below)
  - Prompt: do you have consciousness?

```
import openai

openai.api_key = "[redacted]"

response = openai.Completion.create(
  model="text-davinci-003",
  prompt="do you have consciousness? ->\n",
  temperature=0.7,
  max_tokens=256,
  top_p=1,
  frequency_penalty=0,
  presence_penalty=0,
  stop=["END"]
)

print(response)
```

- Prompt: do you have consciousness?

Python code ready to be executed in Google Colab

```
import openai

openai.api_key = "<insert-here-your_API key from OpenAI>"

response = openai.Completion.create(
  model="text-davinci-003",
  prompt="do you have consciousness? ->\n",
  temperature=0.7,
  max_tokens=256,
  top_p=1,
  frequency_penalty=0,
  presence_penalty=0,
  stop=["END"]
)

print(response)
```

Output excerpt from Google Colab after the python code in the cell is executed


```
{
  "choices": [
    {
      "finish_reason": "stop",
      "index": 0,
      "logprobs": null,
      "text": "\nNo, I do not have consciousness."
    }
  ],
  "created": 1683646955,
  "id": "cmpl-7EJlzxVG5CYuBWbzbnYz6ArOte5RV",
  "model": "text-davinci-003",
  "object": "text_completion",
  "usage": {
    "completion_tokens": 9,
    "prompt_tokens": 7,
    "total_tokens": 16
  }
}
```

- **Question answering (QA)**
  - The goal is using BERT for QA by passing to BERT model a question and a passage
  - Before the advent of LLMs like ChatGPT, BERT models were the state-of-the-art for QA

  - You can create a new Colab notebook to try QA
  - Example of question: How many people live in London?

```
!pip install sentence-transformers

# load library
from sentence_transformers import SentenceTransformer, util
# load model
model = SentenceTransformer('bert-base-nli-mean-tokens')

# question
query_embedding = model.encode('How many people live in London?')
# passage
# a passage is encoded as [title, text]
passage_embedding = model.encode([['London', 'London has 9,787,426 inhabitants at the 2011 census.']])

# compute and display cosine similarity between question and passage
print(util.pytorch_cos_sim(query_embedding, passage_embedding))
```

  - Example of question: How many people live in London?

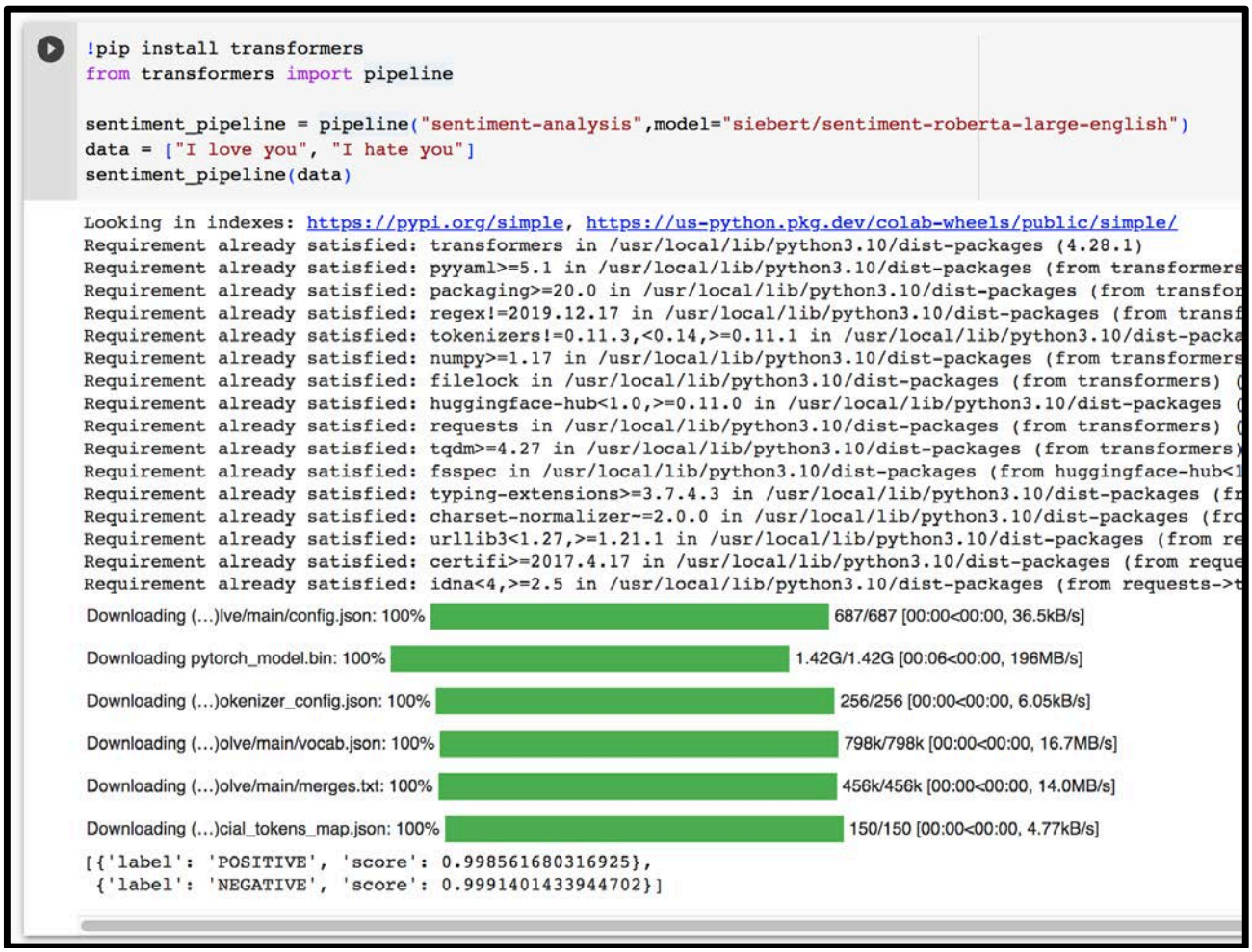
Python code ready to be executed in Google Colab

```
!pip install sentence-transformers

# load library
from sentence_transformers import SentenceTransformer, util
# load model
model = SentenceTransformer('bert-base-nli-mean-tokens')

# question
query_embedding = model.encode('How many people live in London?')
# passage
# a passage is encoded as [title, text]
passage_embedding = model.encode([['London', 'London has 9,787,426 inhabitants at the 2011 census.']])

# compute and display cosine similarity between question and passage
print(util.pytorch_cos_sim(query_embedding, passage_embedding))
```

Output excerpt from Google Colab after the python code in the cell is executed

```
... ... ...
tensor([[0.6778]])
```

- **LLM already fine-tuned for Sentiment Analysis (binary classification)**
  - RoBERTa is a general-domain language model
  - The LLM roberta-large is available at https://huggingface.co/roberta-large
  - *You can read more about how to fine-tune a pre-trained model at* https://huggingface.co/docs/transformers/training
  - SiEBERT is a fine-tuned model of RoBERTa-large for binary sentiment analysis
  - The LLM SiEBERT is available at https://huggingface.co/siebert/sentiment-roberta-large-english
  - You can create a new Colab notebook to try sentiment analysis with SiEBERT
  - Predict the sentiment of two sentences: data = ["I love you", "I hate you"]



  - Predict the sentiment of two sentences: data = ["I love you", "I hate you"]

Python code ready to be executed in Google Colab

```
!pip install transformers
from transformers import pipeline

sentiment_pipeline = pipeline("sentiment-analysis",model="siebert/sentiment-roberta-large-english")
data = ["I love you", "I hate you"]
sentiment_pipeline(data)
```

Output excerpt from Google Colab after the python code in the cell is executed

```
... ... ...
[{'label': 'POSITIVE', 'score': 0.998561680316925},
 {'label': 'NEGATIVE', 'score': 0.9991401433944702}]
```

- **Fine-tune BERT for Sentiment Analysis (binary classification)**
  - The LLM is bert-base-cased available at https://huggingface.co/bert-base-cased
  - The dataset is available at https://raw.githubusercontent.com/clairett/pytorch-sentiment-classification/master/data/SST2/train.tsv
  - *You can read more about how to fine-tune a pre-trained model at* https://huggingface.co/docs/transformers/training
  - The fine-tuning of BERT implies a sequence of steps. The python code shown can be improved!

Step 1: install and load libraries

Python code ready to be executed in Google Colab

```
!pip install transformers
!pip install datasets
#--- --- --- libraries
import numpy as np
import pandas as pd

from tqdm.auto import tqdm

import torch
from torch.utils.data import DataLoader

import transformers
from transformers import AutoTokenizer, AutoModelForSequenceClassification,
get_scheduler, AdamW, TrainingArguments, Trainer

from datasets import Dataset, load_metric

print("libraries loaded")
```

Output excerpt from Google Colab after the python code in the cell is executed

```
... ... ...
libraries loaded
```

Step 2: load dataset

Python code ready to be executed in Google Colab

```
#--- --- --- Google Colab can access your Google Drive
from google.colab import drive
drive.mount('/content/gdrive')

#--- --- --- https://raw.githubusercontent.com/clairett/pytorch-sentiment-
classification/master/data/SST2/train.tsv

#--- --- --- dataset

df=pd.read_csv("gdrive/My Drive/2023temp2/train.tsv",delimiter='\t', header=None)
df.columns = ["text", "labels"]

print(df.head())
print('--- --- ---')

batch_1 = df[:2000]

dataset = Dataset.from_pandas(batch_1)
print(dataset)

ds2 = dataset.train_test_split(test_size=0.1)
print(ds2)
```

- **Step 2**: load dataset [below a screenshot with the output from Google Colab]

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/co
                                                text  labels
0  a stirring , funny and finally transporting re...       1
1  apparently reassembled from the cutting room f...       0
2  they presume their audience wo n't sit still f...       0
3  this is a visually stunning rumination on love...       1
4  jonathan parker 's bartleby should have been t...       1
--- --- ---
Dataset({
    features: ['text', 'labels'],
    num_rows: 2000
})
DatasetDict({
    train: Dataset({
        features: ['text', 'labels'],
        num_rows: 1800
    })
    test: Dataset({
        features: ['text', 'labels'],
        num_rows: 200
    })
})
```

- **Step 3**: prepare dataset

Python code ready to be executed in Google Colab

```
#--- --- --- prepare dataset for BERT
tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")

def tokenize_function(examples):
    return tokenizer(examples["text"], padding="max_length", truncation=True)

tokenized_datasets = ds2.map(tokenize_function, batched=True)

#--- --- --- divide dataset into training and evaluation

tokenized_datasets = tokenized_datasets.remove_columns(["text"])
tokenized_datasets.set_format("torch")

small_train_dataset = tokenized_datasets["train"]
small_eval_dataset = tokenized_datasets["test"]

train_dataloader = DataLoader(small_train_dataset, shuffle=True, batch_size=8)
eval_dataloader = DataLoader(small_eval_dataset, batch_size=8)
```

- **Step 4**: select AutoModel and create scheduler

Python code ready to be executed in Google Colab

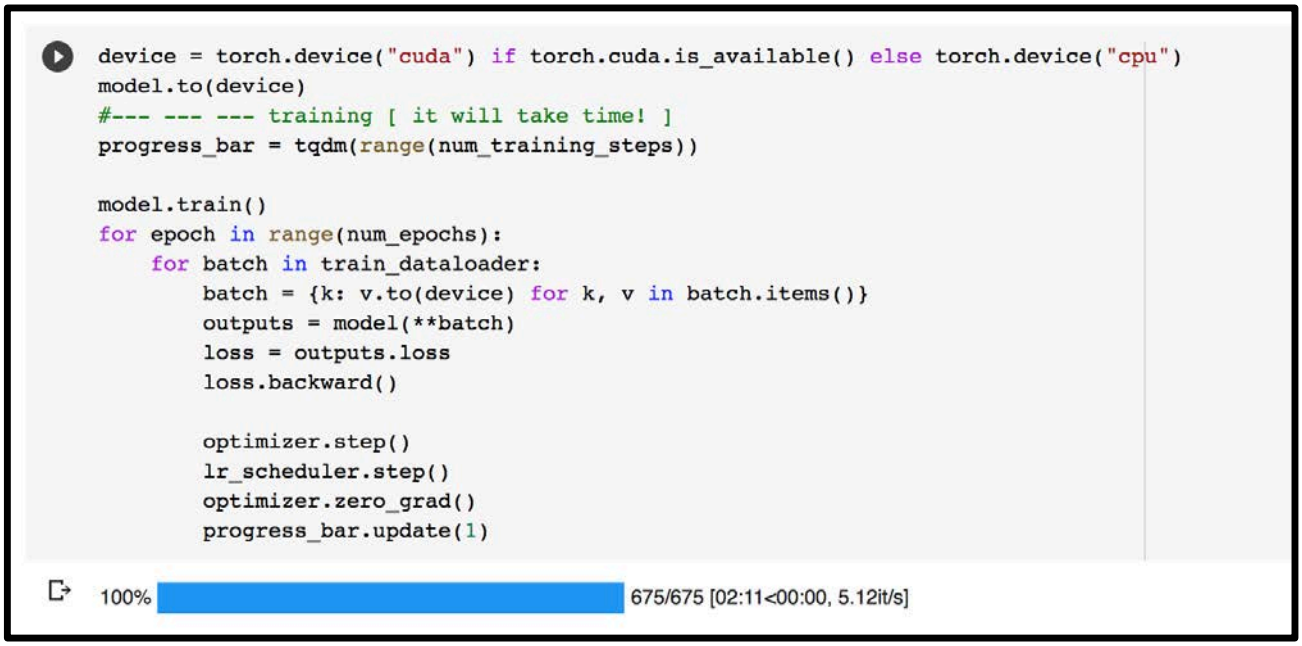
```
#--- --- --- select BERT model
model = AutoModelForSequenceClassification.from_pretrained("bert-base-cased", num_labels=2)

optimizer = AdamW(model.parameters(), lr=5e-5)

#--- --- --- create scheduler
num_epochs = 3
num_training_steps = num_epochs * len(train_dataloader)
lr_scheduler = get_scheduler(
    "linear",
    optimizer=optimizer,
    num_warmup_steps=0,
    num_training_steps=num_training_steps
)
```

- **Step 5**: fine-tune (train) BERT model

Python code ready to be executed in Google Colab

```
device = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")
model.to(device)

#--- --- --- training [ it will take time! ]

progress_bar = tqdm(range(num_training_steps))

model.train()
for epoch in range(num_epochs):
    for batch in train_dataloader:
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        loss = outputs.loss
        loss.backward()

        optimizer.step()
        lr_scheduler.step()
        optimizer.zero_grad()
        progress_bar.update(1)
```

- **Step 5**: fine-tune BERT model [below a screenshot of this step in Google Colab]

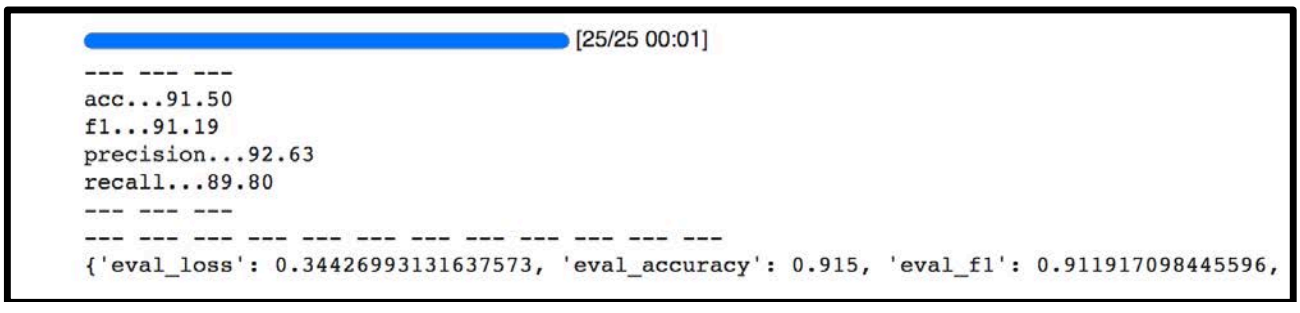


- **Step 6**: evaluation [what is the performance?]

Python code ready to be executed in Google Colab

```
#--- --- --- evaluation
from sklearn.metrics import accuracy_score, precision_recall_fscore_support

def compute_metrics(pred):
    labels = pred.label_ids
    preds = pred.predictions.argmax(-1)
    precision, recall, f1, _ = precision_recall_fscore_support(labels, preds, average='binary')
    acc = accuracy_score(labels, preds)

    print('--- --- ---')
    print('acc...' + "{0:.2f}".format(round(100*acc,2)) )
    print ('f1...' + "{0:.2f}".format(round(100*f1,2)) )
    print ('precision...' + "{0:.2f}".format(round(100*precision,2)) )
    print ('recall...' + "{0:.2f}".format(round(100*recall,2)) )
    print('--- --- ---')
    print('--- --- --- --- --- --- --- --- --- --- --- ---')

    return {
        'accuracy': acc,
        'f1': f1,
        'precision': precision,
        'recall': recall
    }

#--- --- ---

training_args = TrainingArguments("test_trainer")

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=small_train_dataset,
    eval_dataset=small_eval_dataset,
    compute_metrics=compute_metrics,
)
print(trainer.evaluate())
```

- **Step 6**: evaluation [below a screenshot with the output of this step in Google Colab]



- **Step 7**: save the fine-tuned BERT model

Python code ready to be executed in Google Colab

```
model_dir = 'gdrive/My Drive/2023temp2/'
trainer2.save_model(model_dir + 'SAexpBERT/model')
```

The End