Thie notebook explores using BERT for text classification. Before starting, change the runtime to GPU: Runtime > Change runtime type > Hardware accelerator: GPU.

First, create a folder named ANLP21 in your Google drive account, and copy this notebook to that folder, along with data/lmrd from the Github repo. Double click on this notebook from your drive account, which will open it in the Google Colab environment. Begin executing the cells from that environment.

Let's give this notebook access to the data in your ANLP21 folder so we can train and evaluate BERT on the `lmrd` data. (Note you are only providing this access to yourself as you execute this notebook.) You can give Colab notebooks access to other data in the same way (by uploading it first to your Drive account, and then providing access here).

In [ ]:

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
Mounted at /content/drive
```

In [ ]:

```
#!pip install transformers
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.7/dist-p
ackages (4.11.3)
Requirement already satisfied: sacremoses in /usr/local/lib/python3.7/dist-pac
kages (from transformers) (0.0.46)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.7/dist-pa
ckages (from transformers) (6.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-pa
ckages (from transformers) (1.19.5)
Requirement already satisfied: huggingface-hub>=0.0.17 in /usr/local/lib/pytho
n3.7/dist-packages (from transformers) (0.0.19)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packa
ges (from transformers) (2.23.0)
Requirement already satisfied: tokenizers<0.11,>=0.10.1 in /usr/local/lib/pyth
on3.7/dist-packages (from transformers) (0.10.3)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.7/dist-pac
kages (from transformers) (4.62.3)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/
dist-packages (from transformers) (4.8.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.7/d
ist-packages (from transformers) (2019.12.20)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.7/dis
t-packages (from transformers) (21.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-packa
ges (from transformers) (3.3.0)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/d
ist-packages (from huggingface-hub>=0.0.17->transformers) (3.7.4.3)
Requirement already satisfied: pyparsing>=2.0.2 in /usr/local/lib/python3.7/di
st-packages (from packaging>=20.0->transformers) (2.4.7)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-pack
ages (from importlib-metadata->transformers) (3.6.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/
dist-packages (from requests->transformers) (2021.5.30)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/d
ist-packages (from requests->transformers) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-p
ackages (from requests->transformers) (2.10)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr
/local/lib/python3.7/dist-packages (from requests->transformers) (1.24.3)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (
from sacremoses->transformers) (1.15.0)
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-package
s (from sacremoses->transformers) (1.0.1)
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages
(from sacremoses->transformers) (7.1.2)
```

In [ ]:

```
from transformers import BertModel, BertTokenizer
import torch
from tqdm import tqdm
import torch.nn as nn
import numpy as np
import random
import time
```

Double-check that this notebook is running on the GPU (this should "Running on cuda").

In [ ]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Running on {}".format(device))
```

```
Running on cuda
```

In [ ]:
```
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Select the Runtime > "Change runtime type" menu to enable a GPU acce
  print('and then re-execute this cell.')
else:
  print(gpu_info)
```

```
Sun Oct 17 19:47:14 2021
+-----------------------------------------------------------------------------
+
| NVIDIA-SMI 470.74       Driver Version: 460.32.03    CUDA Version: 11.2
|
|-------------------------------+----------------------+----------------------
+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC
|
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M.
|
|                               |                      |               MIG M.
|
|===============================+======================+======================
|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0
|
| N/A   51C    P0    41W / 250W |  12477MiB / 16280MiB |      0%      Default
|
|                               |                      |                  N/A
|
+-------------------------------+----------------------+----------------------
+

+-----------------------------------------------------------------------------
+
| Processes:
|
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory
|
|        ID   ID                                                   Usage
|
|=============================================================================
|
|  No running processes found
|
+-----------------------------------------------------------------------------
+
```

In [ ]:

```
from psutil import virtual_memory
ram_gb = virtual_memory().total / 1e9
print('Your runtime has {:.1f} gigabytes of available RAM\n'.format(ram_gb))

if ram_gb < 20:
  print('To enable a high-RAM runtime, select the Runtime > "Change runtime t
  print('menu, and then select High-RAM in the Runtime shape dropdown. Then,
  print('re-execute this cell.')
else:
  print('You are using a high-RAM runtime!')
```

```
Your runtime has 27.3 gigabytes of available RAM

You are using a high-RAM runtime!
```

In [ ]:

```python
def read_labels(filename):
    labels={}
    with open(filename) as file:
        for line in file:
            cols = line.split("\t")
            label = cols[0]
            if label not in labels:
                labels[label]=len(labels)
    return labels
```

In [ ]:

```python
def read_data(filename, labels, max_data_points):
    """
    :param filename: the name of the file
    :return: list of tuple ([word index list], label)
    as input for the forward and backward function
    """
    data = []
    data_labels = []
    with open(filename) as file:
        for line in file:
            cols = line.split("\t")
            label = cols[0]
            text = cols[1]

            data.append(text)
            data_labels.append(labels[label])


    # shuffle the data
    tmp = list(zip(data, data_labels))
    random.shuffle(tmp)
    data, data_labels = zip(*tmp)

    if max_data_points is None:
        return data, data_labels

    return data[:max_data_points], data_labels[:max_data_points]
```

In [ ]:

```python
labels=read_labels("/content/drive/MyDrive/Colab Files/anlp-data/spooky/train
print(labels)
assert len(labels) == 2
```

```
{'EAP': 0, 'HPL': 1}
```

We'll limit the training and dev data to 10,000 data points for this exercise.

In [ ]:
```
train_x, train_y=read_data("/content/drive/MyDrive/Colab Files/anlp-data/spoo
```

In [ ]:
```
dev_x, dev_y=read_data("/content/drive/MyDrive/Colab Files/anlp-data/spooky/d
```

In [ ]:
```
len(train_x)
```

Out[ ]:
```
8681
```

In [ ]:
```
def evaluate(model, x, y):
    model.eval()
    corr = 0.
    total = 0.
    with torch.no_grad():
        for x, y in zip(x, y):
            y_preds=model.forward(x)
            for idx, y_pred in enumerate(y_preds):
                prediction=torch.argmax(y_pred)
                if prediction == y[idx]:
                    corr += 1.
                total+=1
    return corr/total
```

In [ ]:
```
class BERTClassifier(nn.Module):


    def __init__(self, params):
        super().__init__()

        self.model_name=params["model_name"]
        self.tokenizer = BertTokenizer.from_pretrained(self.model_name, do_lo
        self.bert = BertModel.from_pretrained(self.model_name)

        self.num_labels = params["label_length"]

        self.fc = nn.Linear(params["embedding_size"], self.num_labels)

    def get_batches(self, all_x, all_y, batch_size=32, max_toks=256):

        """ Get batches for input x, y data, with data tokenized according to
      (and limited to a maximum number of WordPiece tokens """

        batches_x=[]
        batches_y=[]

        for i in range(0, len(all_x), batch_size):
```

```
            current_batch=[]

            x=all_x[i:i+batch_size]

            batch_x = self.tokenizer(x, padding=True, truncation=True, return_
            batch_y=all_y[i:i+batch_size]

            batches_x.append(batch_x.to(device))
            batches_y.append(torch.LongTensor(batch_y).to(device))

        return batches_x, batches_y


    def forward(self, batch_x):

        bert_output = self.bert(input_ids=batch_x["input_ids"],
                         attention_mask=batch_x["attention_mask"],
                         token_type_ids=batch_x["token_type_ids"],
                         output_hidden_states=True)

        bert_hidden_states = bert_output['hidden_states']

        # We're going to represent an entire document just by its [CLS] embed
        out = bert_hidden_states[-1][:,0,:]

        out = self.fc(out)

        return out.squeeze()
```

Now let's train BERT on this data. A few practicalities of this environment: if you encounter an out of memory error:

- Reset the notebook (Runtime > Factory reset runtime) and execute all cells from the beginning.
- If your `max_length` is high, try reducing the `batch_size` in `get_batches` above.

Even on a GPU, BERT can take a long time to train, so you might try experimenting first with smaller `max_data_points` above. before running it on the full training data.

In [ ]:

```
def train_and_evaluate(bert_model_name, model_filename, train_x, train_y, dev_

  start_time=time.time()
  bert_model = BERTClassifier(params={"doLowerCase": doLowerCase, "model_name
  bert_model.to(device)

  batch_x, batch_y = bert_model.get_batches(train_x, train_y)
  dev_batch_x, dev_batch_y = bert_model.get_batches(dev_x, dev_y)

  optimizer = torch.optim.Adam(bert_model.parameters(), lr=1e-5)
  cross_entropy=nn.CrossEntropyLoss()

  num_epochs=5
  best_dev_acc = 0.

  for epoch in range(num_epochs):
      bert_model.train()

      # Train
      for x, y in tqdm(list(zip(batch_x, batch_y))):
          y_pred = bert_model.forward(x)
          loss = cross_entropy(y_pred.view(-1, bert_model.num_labels), y.view
          optimizer.zero_grad()
          loss.backward()
          optimizer.step()

      # Evaluate
      dev_accuracy=evaluate(bert_model, dev_batch_x, dev_batch_y)
      if epoch % 1 == 0:
          print("Epoch %s, dev accuracy: %.3f" % (epoch, dev_accuracy))
          if dev_accuracy > best_dev_acc:
              torch.save(bert_model.state_dict(), model_filename)
              best_dev_acc = dev_accuracy

  bert_model.load_state_dict(torch.load(model_filename))
  print("\nBest Performing Model achieves dev accuracy of : %.3f" % (best_dev_
  print("Time: %.3f seconds ---" % (time.time() - start_time))
```

In [ ]:

```
train_and_evaluate("bert-base-cased", "lmrd-bert-base-cased", train_x, train_
```

```
Some weights of the model checkpoint at bert-base-cased were not used when ini
tializing BertModel: ['cls.predictions.transform.LayerNorm.weight', 'cls.predi
ctions.bias', 'cls.predictions.transform.dense.weight', 'cls.seq_relationship.
weight', 'cls.seq_relationship.bias', 'cls.predictions.decoder.weight', 'cls.p
redictions.transform.dense.bias', 'cls.predictions.transform.LayerNorm.bias']
- This IS expected if you are initializing BertModel from the checkpoint of a
model trained on another task or with another architecture (e.g. initializing
a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint o
f a model that you expect to be exactly identical (initializing a BertForSeque
nceClassification model from a BertForSequenceClassification model).
100%|██████████| 272/272 [01:29<00:00,  3.04it/s]
Epoch 0, dev accuracy: 0.876
100%|██████████| 272/272 [01:29<00:00,  3.04it/s]
Epoch 1, dev accuracy: 0.867
100%|██████████| 272/272 [01:29<00:00,  3.04it/s]
Epoch 2, dev accuracy: 0.844
100%|██████████| 272/272 [01:29<00:00,  3.04it/s]
Epoch 3, dev accuracy: 0.894
100%|██████████| 272/272 [01:29<00:00,  3.04it/s]
Epoch 4, dev accuracy: 0.886

Best Performing Model achieves dev accuracy of : 0.894
Time: 475.467 seconds ---
```

Q1. Train BERT on your classification data (from the CheckData_TODO.ipynb exercise) by following the steps outlined above. At this point you have evaluated a number of different classification methods for that data. Report those development accuracies below. Be sure to enable a fair comparison by training and evaluating each method on **the same amount of data**. Provide a one-sentence short description that details the essentials of each method (e.g., major feature classes, number of hidden layers, hidden layer size, convolutional window, dropout rate, etc.)

Amount of Data: N = 8681

Method	Short description	Accuracy
Logistic regression (6.classification/FeatureExploration.ipynb)	Logistic regression (C=1.0, solver='lbfgs', penalty='l2', max_iter=10000) trained with unigram features, tfidf, and two custom dictionary vocab features.	0.875
FFNN (9.neural/FFNN.ipynb)	Feed Forward Neural Network trained with unigrams from the 1000 most frequent words in the vocab (1 hidden layer with dim = 100, no dropout, 5 epochs).	0.827
CNN (9.neural/CNN.ipynb)	CNN over window sizes each 96 filters of unigrams and bigrams (1 hidden layer proceeded by convolution and pooling, dropout = 0.2, 5 epochs).	0.845
BERT-Base	BERT model with attention layers extracting context based features (12 layers, H = 768, 5 epochs)	0.894
BERT-Base	BERT model with attention layers extracting context based features (12 layers, H = 768, 20 epochs)	0.911
BERT-Medium	BERT model with attention layers extracting context based features (8 layers, H = 512, 5 epochs)	0.869
BERT-Medium	BERT model with attention layers extracting context based features (8 layers, H = 512, 20 epochs)	0.893

Q2. As you can see, training `bert-base` can be expensive. Google has released a number of smaller BERT models with fewer layers (2, 4, 6, 8, 10) and smaller dimensions (128, 256, 512) that effectively trade off accuracy for speed. Select three of these models and train them; report both their accuracy and training time.

To use these models in the huggingface library that we have been using, the huggingface name of the model can be derived from the URL linking to it:

https://storage.googleapis.com/bert_models/2020_02_20/uncased_L-2_H-128_A-2.zip -> `google/bert_uncased_L-2_H-128_A-2`

All of these smaller models are uncased (so all text is lowercase), so be sure to set `doLowerCase` to be true. You'll also need to change the `embedding_size` parameter to this function based on the H value from the model (listed both on the BERT Github page and in the model's URL). One sample model is provided below.

In [ ]:

```
train_and_evaluate("google/bert_uncased_L-8_H-512_A-8", "lmrd-uncased_L-8_H-5
```

```
Some weights of the model checkpoint at google/bert_uncased_L-8_H-512_A-8 were
not used when initializing BertModel: ['cls.predictions.transform.LayerNorm.we
ight', 'cls.predictions.bias', 'cls.predictions.transform.dense.weight', 'cls.
seq_relationship.weight', 'cls.seq_relationship.bias', 'cls.predictions.decode
r.weight', 'cls.predictions.decoder.bias', 'cls.predictions.transform.dense.bi
as', 'cls.predictions.transform.LayerNorm.bias']
- This IS expected if you are initializing BertModel from the checkpoint of a
model trained on another task or with another architecture (e.g. initializing
a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint o
f a model that you expect to be exactly identical (initializing a BertForSeque
nceClassification model from a BertForSequenceClassification model).
100%|██████████| 272/272 [00:28<00:00,  9.60it/s]
Epoch 0, dev accuracy: 0.863
100%|██████████| 272/272 [00:28<00:00,  9.60it/s]
Epoch 1, dev accuracy: 0.869
100%|██████████| 272/272 [00:28<00:00,  9.60it/s]
Epoch 2, dev accuracy: 0.844
100%|██████████| 272/272 [00:28<00:00,  9.61it/s]
Epoch 3, dev accuracy: 0.855
100%|██████████| 272/272 [00:28<00:00,  9.61it/s]
Epoch 4, dev accuracy: 0.849

Best Performing Model achieves dev accuracy of : 0.869
Time: 154.334 seconds ---
```