Applied Machine Learning

Homework 4

In [7]:
```
import pandas as pd
import torch
from torch.utils.data import TensorDataset, DataLoader, RandomSampler, SequentialSampler
from transformers import BertTokenizer, BertForSequenceClassification, AdamW
from sklearn.metrics import accuracy_score
import os
```

Question 2

a. Create your own dataset for text classification. It should contain at least 1000 words in total and at least two categories with at least 100 examples per category. You can create it by scraping the web or using some of the documents you have on your computer (do not use anything confidential).

Our dataset is a behavioural type dataset where there are certain habits written, with each habit being labeled good habit or bad habit. Our dataset has a total of 400 samples, evenly divided among the 2 labels.

In [11]:
```
df = pd.read_csv('text_classification.csv')
df.head()
```

Out[11]:

	text	label
0	Biting your lip or cheek when stressed	Bad habit
1	Finding ways to reduce plastic waste by using ...	Good Habit
2	Not being mindful of your environmental impact	Bad habit
3	Learning a new skill or hobby to promote perso...	Good Habit
4	Being open to constructive feedback and criticism	Good Habit

In [12]:
```
len(df['text'])
```

Out[12]:
```
400
```

In [13]:
```
good = 0
bad = 0
for i in df['label']:
    if i == 'Good Habit':
        good+=1
    else:
        bad+=1

print("No. of Good habits: ", good)
print("No. of Bad habits: ", bad)
```

```
No. of Good habits:  200
No. of Bad habits:  200
```

We are utilizing the BERT model available in the transformers package to tokenize our dataset and subsequently save the encoded input ids. After this, we transform the input data into PyTorch tensors and assign the labels with binary values of 0 and 1, where "Good Habit" is labeled as 1 and "Bad Habit" is labeled as 0.

In [14]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_case=True)

input_ids = []
attention_masks = []

for text in df['text']:
    encoded_dict = tokenizer.encode_plus(
                        text,
                        add_special_tokens = True,
                        max_length = 64,
                        pad_to_max_length = True,
                        return_attention_mask = True,
                        return_tensors = 'pt'
                   )

    input_ids.append(encoded_dict['input_ids'])
    attention_masks.append(encoded_dict['attention_mask'])

input_ids = torch.cat(input_ids, dim=0)
attention_masks = torch.cat(attention_masks, dim=0)
labels = []
for i in df['label']:
  if i == 'Good Habit':
    labels.append(1)
  else:
    labels.append(0)
labels = torch.tensor(labels)
```

```
Downloading (…)solve/main/vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]
```

```
C:\Users\jeets\AppData\Roaming\Python\Python39\site-packages\huggingface_hub\file_download.py:133: UserWarni
ng: `huggingface_hub` cache-system uses symlinks by default to efficiently store duplicated files but your m
achine does not support them in C:\Users\jeets\.cache\huggingface\hub. Caching files will still work but in
a degraded version that might require more space on your disk. This warning can be disabled by setting the `
HF_HUB_DISABLE_SYMLINKS_WARNING` environment variable. For more details, see https://huggingface.co/docs/hug
gingface_hub/how-to-cache#limitations. (https://huggingface.co/docs/huggingface_hub/how-to-cache#limitation
s.)
To support symlinks on Windows, you either need to activate Developer Mode or to run Python as an administra
tor. In order to see activate developer mode, see this article: https://docs.microsoft.com/en-us/windows/app
s/get-started/enable-your-device-for-development (https://docs.microsoft.com/en-us/windows/apps/get-started/
enable-your-device-for-development)
  warnings.warn(message)

Downloading (…)okenizer_config.json:   0%|          | 0.00/28.0 [00:00<?, ?B/s]

Downloading (…)lve/main/config.json:   0%|          | 0.00/570 [00:00<?, ?B/s]

Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncatio
n=True` to explicitly truncate examples to max length. Defaulting to 'longest_first' truncation strategy. If
you encode pairs of sequences (GLUE-style) with the tokenizer you can select this strategy more precisely by
providing a specific strategy to `truncation`.
C:\Users\jeets\AppData\Roaming\Python\Python39\site-packages\transformers\tokenization_utils_base.py:2354: F
utureWarning: The `pad_to_max_length` argument is deprecated and will be removed in a future version, use `p
adding=True` or `padding='longest'` to pad to the longest sequence in the batch, or use `padding='max_lengt
h'` to pad to a max length. In this case, you can give a specific length with `max_length` (e.g. `max_length
=45`) or leave max_length to None to pad to the maximal input size of the model (e.g. 512 for Bert).
  warnings.warn(
```

b. Split the dataset into training (at least 160examples) and test (at least 40 examples) sets.

We split the dataset in 80:20 ratio for train and test.

In [1]:
```
dataset = TensorDataset(input_ids, attention_masks, labels)
train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size
train_dataset, test_dataset = torch.utils.data.random_split(dataset, [train_size, test_size])
```

c. Fine tune a pretrained language model capable of generating text (e.g., GPT) that you can take from the Hugging Face Transformers library with the dataset your created (I suggest using this tutorial:

https://huggingface.co/docs/transformers/training (https://huggingface.co/docs/transformers/training)). Report the test

We will be using the BERT (Bidirectional Encoder Representations from Transformers) pre-trained model for our text classification. We first load the data using a data loader. For optimization, we use the AdamW optimizer. We train the model for 4 epochs and track the training loss.

In [17]:
```
batch_size = 32
train_dataloader = DataLoader(train_dataset,
                              sampler = RandomSampler(train_dataset),
                              batch_size = batch_size)

test_dataloader = DataLoader(test_dataset,
                             sampler = SequentialSampler(test_dataset),
                             batch_size = batch_size)

model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels = 2)
model.to(device)

optimizer = AdamW(model.parameters(),
                  lr = 2e-5,
                  eps = 1e-8)

epochs = 4
total_steps = len(train_dataloader) * epochs

for epoch in range(epochs):
    model.train()
    total_loss = 0

    for step, batch in enumerate(train_dataloader):
        batch_input_ids = batch[0].to(device)
        batch_attention_masks = batch[1].to(device)
        batch_labels = batch[2].to(device)

        model.zero_grad()

        outputs = model(batch_input_ids,
                        token_type_ids=None,
                        attention_mask=batch_attention_masks,
                        labels=batch_labels)

        loss = outputs.loss
        total_loss += loss.item()

        loss.backward()

        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)

        optimizer.step()
```

```
    avg_train_loss = total_loss / len(train_dataloader)
    print("Average training loss: {}".format(avg_train_loss))
```

```
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncatio
n=True` to explicitly truncate examples to max length. Defaulting to 'longest_first' truncation strategy. If
you encode pairs of sequences (GLUE-style) with the tokenizer you can select this strategy more precisely by
providing a specific strategy to `truncation`.
/usr/local/lib/python3.9/dist-packages/transformers/tokenization_utils_base.py:2354: FutureWarning: The `pad
_to_max_length` argument is deprecated and will be removed in a future version, use `padding=True` or `paddi
ng='longest'` to pad to the longest sequence in the batch, or use `padding='max_length'` to pad to a max len
gth. In this case, you can give a specific length with `max_length` (e.g. `max_length=45`) or leave max_leng
th to None to pad to the maximal input size of the model (e.g. 512 for Bert).
  warnings.warn(
Some weights of the model checkpoint at bert-base-uncased were not used when initializing BertForSequenceCla
ssification: ['cls.seq_relationship.bias', 'cls.predictions.transform.dense.weight', 'cls.predictions.transf
orm.LayerNorm.weight', 'cls.predictions.bias', 'cls.predictions.transform.dense.bias', 'cls.seq_relationshi
p.weight', 'cls.predictions.decoder.weight', 'cls.predictions.transform.LayerNorm.bias']
- This IS expected if you are initializing BertForSequenceClassification from the checkpoint of a model trai
ned on another task or with another architecture (e.g. initializing a BertForSequenceClassification model fr
om a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForSequenceClassification from the checkpoint of a model
that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSe
quenceClassification model).
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-un
cased and are newly initialized: ['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inferenc
e.
/usr/local/lib/python3.9/dist-packages/transformers/optimization.py:391: FutureWarning: This implementation
of AdamW is deprecated and will be removed in a future version. Use the PyTorch implementation torch.optim.A
damW instead, or set `no_deprecation_warning=True` to disable this warning
  warnings.warn(

Average training loss: 0.5902607768774033
Average training loss: 0.29664637595415116
Average training loss: 0.15551427006721497
Average training loss: 0.08083114549517631
```

The training loss after 4 epochs reduces down to 0.0808, which is pretty low. This shows the model has been able to converge succesfully. Now we will evaluate our model on the test set.

In [18]:
```
model.eval()

predictions = []
true_labels = []

for batch in test_dataloader:
    batch_input_ids = batch[0].to(device)
    batch_attention_masks = batch[1].to(device)
    batch_labels = batch[2].to(device)

    with torch.no_grad():
        outputs = model(batch_input_ids,
                        token_type_ids=None,
                        attention_mask=batch_attention_masks)

    logits = outputs.logits
    logits = logits.detach().cpu().numpy()
    label_ids = batch_labels.to('cpu').numpy()

    batch_predictions = logits.argmax(axis=1).flatten()
    batch_labels = label_ids.flatten()

    predictions.extend(batch_predictions)
    true_labels.extend(batch_labels)

accuracy = accuracy_score(true_labels, predictions)
print("Test accuracy: {}".format(accuracy))
```

```
Test accuracy: 1.0
```

As we can see, it gives a perfect 100% accuracy on out test data. It is important to note that achieving 100% accuracy on test data does not necessarily mean that the BERT model is perfect for text classification on the dataset. While the model may perform well on the specific data it was trained and tested on, its performance may vary when applied to new and unseen data.