Lab 3: Cyberbullying Detection Using BERT


In this lab, we will learn how AI can be developed to detect cyberbullying. We will use a publicly available dataset of cyberbullying texts, and train an AI model on this dataset to automatically detect cyberbullying text. You will learn:

1. AI development process
2. Train and test your own AI for cyberbullying detection
3. Run AI on your own samples
4. Hypterpaprameter tuning to improve model performance

First, we need to download softwares used in the lab. Just hit the 'play' button run the code below.

Preliminaries

```
# change the output font size
from IPython.display import HTML
shell = get_ipython()

def adjust_font_size():
  display(HTML('''<style>
    body {
      font-size: 20px;
    }
  '''))

if adjust_font_size not in shell.events.callbacks['pre_execute']:
  shell.events.register('pre_execute', adjust_font_size)
```

```
# hide warnings
import warnings
warnings.filterwarnings('ignore')
```

```
!git clone https://github.com/nishantvishwamitra/CyberbullyingLab1.git
```

```
fatal: destination path 'CyberbullyingLab1' already exists and is not an empty directory.
```

Next, we install transformers which offer us some tools we can use

The template was creating errors in ARC, which required me to install transformers

```
!pip install transformers
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: transformers in /home/kml614/.local/lib/python3.11/site-packages (4.46.0)
Requirement already satisfied: filelock in /home/kml614/.local/lib/python3.11/site-packages (from transformers) (3.16.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.23.2 in /home/kml614/.local/lib/python3.11/site-packages (from transformers) (0.26.1)
Requirement already satisfied: numpy>=1.17 in /home/kml614/.local/lib/python3.11/site-packages (from transformers) (1.23.0)
Requirement already satisfied: packaging>=20.0 in /home/kml614/.local/lib/python3.11/site-packages (from transformers) (24.1)
Requirement already satisfied: pyyaml>=5.1 in /home/kml614/.local/lib/python3.11/site-packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in /home/kml614/.local/lib/python3.11/site-packages (from transformers) (2024.9.11)
Requirement already satisfied: requests in /home/kml614/.local/lib/python3.11/site-packages (from transformers) (2.32.3)
Requirement already satisfied: safetensors>=0.4.1 in /home/kml614/.local/lib/python3.11/site-packages (from transformers) (0.4.5)
Requirement already satisfied: tokenizers<0.21,>=0.20 in /home/kml614/.local/lib/python3.11/site-packages (from transformers) (0.20.1)
Requirement already satisfied: tqdm>=4.27 in /home/kml614/.local/lib/python3.11/site-packages (from transformers) (4.66.5)
Requirement already satisfied: fsspec>=2023.5.0 in /home/kml614/.local/lib/python3.11/site-packages (from huggingface-hub<1.0,>=0.23.2->transformers) (2024.10.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /home/kml614/.local/lib/python3.11/site-packages (from huggingface-hub<1.0,>=0.23.2->transformers) (4.12.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /home/kml614/.local/lib/python3.11/site-packages (from requests->transformers) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /home/kml614/.local/lib/python3.11/site-packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /home/kml614/.local/lib/python3.11/site-packages (from requests->transformers) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /home/kml614/.local/lib/python3.11/site-packages (from requests->transformers) (2024.8.30)
```

I was also required to install torch

```
!pip install torch
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: torch in /home/kml614/.local/lib/python3.11/site-packages (2.5.0)
Requirement already satisfied: filelock in /home/kml614/.local/lib/python3.11/site-packages (from torch) (3.16.1)
Requirement already satisfied: typing-extensions>=4.8.0 in /home/kml614/.local/lib/python3.11/site-packages (from torch) (4.12.2)
Requirement already satisfied: networkx in /apps/anaconda3/2024.02-1/lib/python3.11/site-packages (from torch) (3.1)
Requirement already satisfied: jinja2 in /apps/anaconda3/2024.02-1/lib/python3.11/site-packages (from torch) (3.1.3)
Requirement already satisfied: fsspec in /home/kml614/.local/lib/python3.11/site-packages (from torch) (2024.10.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /home/kml614/.local/lib/python3.11/site-packages (from torch) (12.4.127)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /home/kml614/.local/lib/python3.11/site-packages (from torch) (12.4.127)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /home/kml614/.local/lib/python3.11/site-packages (from torch) (12.4.127)
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /home/kml614/.local/lib/python3.11/site-packages (from torch) (9.1.0.70)
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /home/kml614/.local/lib/python3.11/site-packages (from torch) (12.4.5.8)
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /home/kml614/.local/lib/python3.11/site-packages (from torch) (11.2.1.3)
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /home/kml614/.local/lib/python3.11/site-packages (from torch) (10.3.5.147)
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /home/kml614/.local/lib/python3.11/site-packages (from torch) (11.6.1.9)
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in /home/kml614/.local/lib/python3.11/site-packages (from torch) (12.3.1.170)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /home/kml614/.local/lib/python3.11/site-packages (from torch) (2.21.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /home/kml614/.local/lib/python3.11/site-packages (from torch) (12.4.127)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /home/kml614/.local/lib/python3.11/site-packages (from torch) (12.4.127)
Requirement already satisfied: triton==3.1.0 in /home/kml614/.local/lib/python3.11/site-packages (from torch) (3.1.0)
Requirement already satisfied: sympy==1.13.1 in /home/kml614/.local/lib/python3.11/site-packages (from torch) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /apps/anaconda3/2024.02-1/lib/python3.11/site-packages (from sympy==1.13.1->torch) (1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in /apps/anaconda3/2024.02-1/lib/python3.11/site-packages (from jinja2->torch) (2.1.3)
```

Let's import all our softwares dependencies in our iPython notebook

```
import pandas as pd
import numpy as np

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from transformers import BertTokenizer, BertModel
from transformers import AdamW

import matplotlib.pyplot as plt

from tqdm.notebook import tqdm
```

```
# change the device to gpu if available
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
device
```

```
device(type='cuda')
```

Data Preprocessing

While training AI, datasets are divided into three parts: training dataset, validation dataset and test dataset.

- Training set: feed the AI, so the AI can keep learning cyberbullying and non-cyberbullying knowledge.
- Validation set: can help us and the AI know whether its predictions are getting better or worse.
- Test set: is to evaluate the AI's performance.

They are different datasets, should have no overlap among them. Let's create these three parts for our dataset.

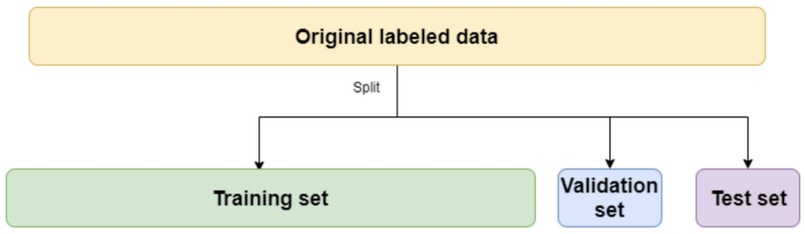
```
# Dowload the cyberbullying speech dataset
# Let's dowload the main dataset frist
main_df = pd.read_csv('CyberbullyingLab1/formspring_dataset.csv', sep = '\t')

# Let's see how many smaples we have
print('Total number of samples:', main_df.shape)
main_df = main_df.sample(n = main_df.shape[0])
main_df = main_df[['text', 'label']]

# Let's take a look at a few samples from our dataset
print(main_df.head())
```

```
Total number of samples: (13159, 2)
                                                   text  label
6455  Q: Do you like pepper? Salt is always used  bu...      0
8078    Q: What&#039;s your favorite color?<br>A: seven      0
4781  Q: Its goin down.!Graduation party At train un...      0
4390  Q: do you think you&#039;re some kind of cutie...      0
1783  Q: wanna come to brazil? I&#039;ll make you ge...      0
```

Note:

- 0 indicates non-cyberbullying
- 1 indicates cyberbullying

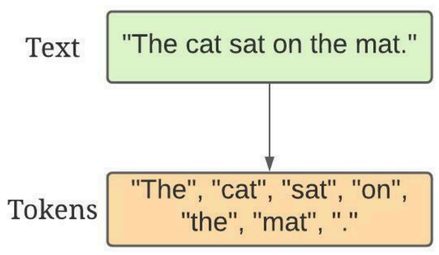
While training AI, datasets are divided into three parts: training dataset, testing dataset and validation dataset. Let's create these three parts for our dataset.




```
# let's divide the dataset into non-cyberbullying and cyberbullying samples
o_class = main_df.loc[main_df.label == 0, :]
l_class = main_df.loc[main_df.label == 1, :]

# let's create train, val and test splits
train_val = main_df.iloc[:int(main_df.shape[0] * .80)]
test_df = main_df.iloc[int(main_df.shape[0] * .80):]
train_df = train_val.iloc[:int(train_val.shape[0] * .80)]
val_df = train_val.iloc[int(train_val.shape[0] * .80):]

#print(train.shape, val.shape, test.shape)
print('\nTraining set:\n', train_df.label.value_counts())
print('\nValidation set:\n', val_df.label.value_counts())
print('\nTest set:\n', test_df.label.value_counts())
```

```
Training set:
 label
0    7877
1     544
Name: count, dtype: int64

Validation set:
 label
0    1968
1     138
Name: count, dtype: int64

Test set:
 label
0    2446
1     186
Name: count, dtype: int64
```

The first step in natural language processing for AI is tokenization. In this process, we split the text into 'tokens', that are then given unique numbers that are understood by a machine. Take a look at the example below.




```
# Let's use a tokenizer. This is the first step in NLP
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

```
#  Let's see how the tokenizer works
sentences = "the cat sat on the mat"

tokens = tokenizer.tokenize(sentences)
for token in tokens:
    print(token)
```

```
the
cat
sat
on
the
mat
```

Task 1: Add code below to preprocess the following cyberbullying text, and include the generated tokens in your report.

"Harlem shake is just an excuse to go full retard for 30 seconds".

```
# TODO: Enter your code here
import nltk
import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

# Ensure nltk resources are downloaded
nltk.download('punkt')
nltk.download('stopwords')

# Initial text
text = "Harlem shake is just an excuse to go full retard for 30 seconds"

# Lowercase the text
text = text.lower()

# Remove punctuation and special characters
text = re.sub(r"[^a-zA-Z0-9\s]", "", text)

# Tokenize text into words
tokens = word_tokenize(text)

# Remove stop words
tokens = [word for word in tokens if word not in stopwords.words('english')]

# Display tokens
print("Generated tokens:", tokens)
```

```
Generated tokens: ['harlem', 'shake', 'excuse', 'go', 'full', 'retard', '30', 'seconds']
[nltk_data] Downloading package punkt to /home/kml614/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /home/kml614/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

With tokenizer, now we can prepare the input data that the AI model needs

```
# Prepare the dataset
class CyberbullyingDataset(Dataset):
    def __init__(self, df, tokenizer, max_len):
        self.df = df
        self.tokenizer = tokenizer
        self.max_len = max_len
        self.text = df.text.to_list()
        self.label = df.label.to_list()

    def __len__(self):
        return len(self.text)

    def __getitem__(self, index):
        text = str(self.text[index])
        text = ' '.join(text.split())

        inputs = self.tokenizer.encode_plus(
            text, # Sentence to encode.
            None, # Add another sequence to the inputs. 'None' means no other sequence is added.
            add_special_tokens = True, # Add '[CLS]' and '[SEP]'
            max_length = self.max_len, # Pad & truncate all sentences.
            pad_to_max_length = True, # Pad all samples to the same length.
            truncation = True, # Truncate all samples to the same length.
            return_token_type_ids = False,
            return_tensors = 'pt' # Return pytorch tensors.
        )
        label = torch.tensor(self.label[index], dtype = torch.long)

        return {
            'text': text,
            'input_ids': inputs['input_ids'].flatten(),
            'attention_mask': inputs['attention_mask'].flatten(),
            'label': label
        }

# The number of unique words in the vocabulary and the number of labels
VOCAB_SIZE = tokenizer.vocab_size
NUM_LABELS = train_df.label.nunique()
print("The number of unique words in the vocabulary:", VOCAB_SIZE)
print("The number of labels:", NUM_LABELS)
```

```
The number of unique words in the vocabulary: 30522
The number of labels: 2
```

In order to make the model understand both cyberbullying and non-cyberbullying data, we typically balance the datasets.

```
# Build a balanced dataset
def balence_data(dataframe):
    o_class = dataframe.loc[dataframe.label == 0, :]
    l_class = dataframe.loc[dataframe.label == 1, :]
    o_class = o_class.sample(n = l_class.shape[0])
    dataframe = pd.concat([o_class, l_class], axis = 0)
    dataframe = dataframe.sample(n = dataframe.shape[0])
    return dataframe

train_df = balence_data(train_df)
val_df = balence_data(val_df)
test_df = balence_data(test_df)

print('\nTraining set:\n', train_df.label.value_counts())
print('\nValidation set:\n', val_df.label.value_counts())
print('\nTest set:\n', test_df.label.value_counts())
```

```
Training set:
 label
1    544
0    544
Name: count, dtype: int64

Validation set:
 label
0    138
1    138
Name: count, dtype: int64

Test set:
 label
1    186
0    186
Name: count, dtype: int64
```

We need iterators to step through our dataset.

```
# Normally, we prepare the dataset with batches, it can help us to train the model faster.
MAX_LEN = 120
BATCH_SIZE = 32

def create_data_loader(df, tokenizer, max_len, batch_size):
    ds = CyberbullyingDataset(df, tokenizer, max_len)
    return DataLoader(ds, batch_size = batch_size)

# Create the dataloaders
train_data_loader = create_data_loader(train_df, tokenizer, MAX_LEN, BATCH_SIZE)
val_data_loader = create_data_loader(val_df, tokenizer, MAX_LEN, BATCH_SIZE)
test_data_loader = create_data_loader(test_df, tokenizer, MAX_LEN, BATCH_SIZE)

print("After we build the dataloaders, we can see the number of batches in each dataloader. It means we can train the model with {} samples in each time.".format(BATCH_SIZE))
print("The number of batches in the training dataloader:", len(train_data_loader))
print("The number of batches in the validation dataloader:", len(val_data_loader))
print("The number of batches in the test dataloader:", len(test_data_loader))
```

```
After we build the dataloaders, we can see the number of batches in each dataloader. It means we can train the model with 32 samples in each time.
The number of batches in the training dataloader: 34
The number of batches in the validation dataloader: 9
The number of batches in the test dataloader: 12
```

AI Model Definition

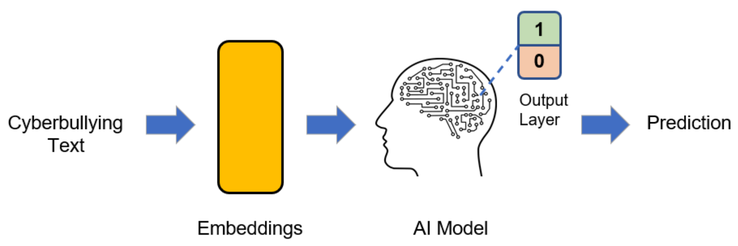
Let's define some hyperparameters for our AI model, you can change them to adjust the performance of the model.

```
# Lets define some hyperparameters
N_EPOCHS = 5 # The number of epochs
LEARNING_RATE = 2e-5 # The learning rate
Num_classes = 2 # The number of classes
```

Let's instantiate our AI model.

```
# Download the tokenizer and model
bert_model = BertModel.from_pretrained('bert-base-uncased', output_hidden_states = True)
bert_model = bert_model.to(device)
```

```
# Define the model
class CyberbullyingDetecter(nn.Module):
    def __init__(self, bert_model, Num_classes):
            super(CyberbullyingDetecter, self).__init__()
            self.bert = bert_model
            self.drop = nn.Dropout(p=0.3)
            self.out = nn.Linear(self.bert.config.hidden_size, Num_classes)

    def forward(self, input_ids, attention_mask):
        pooled_output = self.bert(
            input_ids = input_ids,
            attention_mask = attention_mask,
        )['pooler_output']
        output = self.drop(pooled_output)
        return self.out(output)

# Create the model
model = CyberbullyingDetecter(bert_model, Num_classes)
model = model.to(device)
```



Now, let's define and the loss function.

```
# Define and the loss function and optimizer
criterion = nn.CrossEntropyLoss().to(device)
optimizer = AdamW(model.parameters(), lr=LEARNING_RATE, correct_bias=False)
```

Define some functions for model training

```
# Let's define the training and testing procedures for our AI model
# Lets define our training steps
def accuracy(preds, labels):
    preds = torch.argmax(preds, dim=1).flatten()
    labels = labels.flatten()
    return torch.sum(preds == labels) / len(labels)

def train(model, data_loader, optimizer, criterion):
  epoch_loss = 0
  epoch_acc = 0

  model.train()
  for d in tqdm(data_loader):
    inputs_ids = d['input_ids'].to(device)
    attention_mask = d['attention_mask'].to(device)
    labels = d['label'].to(device)
    # print(inputs_ids.shape, attention_mask.shape, label.shape)
    outputs = model(inputs_ids, attention_mask)

    _, preds = torch.max(outputs, dim=1)
    loss = criterion(outputs, labels)
    acc = accuracy(outputs, labels)

    loss.backward()
    optimizer.step()
    optimizer.zero_grad()

    epoch_loss += loss.item()
    epoch_acc += acc.item()

  return epoch_loss / len(data_loader), epoch_acc / len(data_loader)

# Lets define our testing steps
def evaluate(model, data_loader, criterion):
    epoch_loss = 0
    epoch_acc = 0

    model.eval()
    with torch.no_grad():
      for d in data_loader:
        inputs_ids = d["input_ids"].to(device)
        attention_mask = d["attention_mask"].to(device)
        labels = d["label"].to(device)
        outputs = model(inputs_ids, attention_mask)

        _, preds = torch.max(outputs, dim=1)
        loss = criterion(outputs, labels)
        acc = accuracy(outputs, labels)

        epoch_loss += loss.item()
        epoch_acc += acc.item()

    return epoch_loss / len(data_loader), epoch_acc / len(data_loader)

# define a function for evaluation
def predict_cb(sentence):
    sentence = str(sentence)
    sentence = ' '.join(sentence.split())
    inputs = tokenizer.encode_plus(
        sentence, # Sentence to encode.
        None, # Add another sequence to the inputs. 'None' means no other sequence is added.
        add_special_tokens = True, # Add '[CLS]' and '[SEP]'
        max_length = MAX_LEN, # Pad & truncate all sentences.
        pad_to_max_length = True, # Pad all samples to the same length.
        truncation = True, # Truncate all samples to the same length.
        return_token_type_ids = True # Return token_type_ids
    )
    output = model(torch.tensor(inputs['input_ids']).unsqueeze(0).to(device), torch.tensor(inputs['attention_mask']).unsqueeze(0).to(device))
    # print(output)
    preds, ind= torch.max(F.softmax(output, dim=-1), 1)
    if ind.item() == 1:
        return preds, ind, 'Cyberbullying detected.'
    else:
        return preds, ind, 'Cyberbullying not detected.'
```

Next, Let's begin training our AI model

Training Process

Original code required update hence ipywidgets were applied for update and the code ran successfully

I was also required to install ipywidgets to ensure the template ran smoothly

```
!pip install ipywidgets --upgrade
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: ipywidgets in /home/kml614/.local/lib/python3.11/site-packages (8.1.5)
Requirement already satisfied: comm>=0.1.3 in /home/kml614/.local/lib/python3.11/site-packages (from ipywidgets) (0.2.2)
Requirement already satisfied: ipython>=6.1.0 in /apps/anaconda3/2024.02-1/lib/python3.11/site-packages (from ipywidgets) (8.20.0)
Requirement already satisfied: traitlets>=4.3.1 in /apps/anaconda3/2024.02-1/lib/python3.11/site-packages (from ipywidgets) (5.7.1)
Requirement already satisfied: widgetsnbextension~=4.0.12 in /home/kml614/.local/lib/python3.11/site-packages (from ipywidgets) (4.0.13)
Requirement already satisfied: jupyterlab-widgets~=3.0.12 in /home/kml614/.local/lib/python3.11/site-packages (from ipywidgets) (3.0.13)
Requirement already satisfied: decorator in /apps/anaconda3/2024.02-1/lib/python3.11/site-packages (from ipython>=6.1.0->ipywidgets) (5.1.1)
Requirement already satisfied: jedi>=0.16 in /apps/anaconda3/2024.02-1/lib/python3.11/site-packages (from ipython>=6.1.0->ipywidgets) (0.18.1)
Requirement already satisfied: matplotlib-inline in /apps/anaconda3/2024.02-1/lib/python3.11/site-packages (from ipython>=6.1.0->ipywidgets) (0.1.6)
Requirement already satisfied: prompt-toolkit<3.1.0,>=3.0.41 in /apps/anaconda3/2024.02-1/lib/python3.11/site-packages (from ipython>=6.1.0->ipywidgets) (3.0.43)
Requirement already satisfied: pygments>=2.4.0 in /apps/anaconda3/2024.02-1/lib/python3.11/site-packages (from ipython>=6.1.0->ipywidgets) (2.15.1)
Requirement already satisfied: stack-data in /apps/anaconda3/2024.02-1/lib/python3.11/site-packages (from ipython>=6.1.0->ipywidgets) (0.2.0)
Requirement already satisfied: pexpect>4.3 in /apps/anaconda3/2024.02-1/lib/python3.11/site-packages (from ipython>=6.1.0->ipywidgets) (4.8.0)
Requirement already satisfied: parso<0.9.0,>=0.8.0 in /apps/anaconda3/2024.02-1/lib/python3.11/site-packages (from jedi>=0.16->ipython>=6.1.0->ipywidgets) (0.8.3)
Requirement already satisfied: ptyprocess>=0.5 in /apps/anaconda3/2024.02-1/lib/python3.11/site-packages (from pexpect>4.3->ipython>=6.1.0->ipywidgets) (0.7.0)
Requirement already satisfied: wcwidth in /apps/anaconda3/2024.02-1/lib/python3.11/site-packages (from prompt-toolkit<3.1.0,>=3.0.41->ipython>=6.1.0->ipywidgets) (0.2.5)
Requirement already satisfied: executing in /apps/anaconda3/2024.02-1/lib/python3.11/site-packages (from stack-data->ipython>=6.1.0->ipywidgets) (0.8.3)
Requirement already satisfied: asttokens in /apps/anaconda3/2024.02-1/lib/python3.11/site-packages (from stack-data->ipython>=6.1.0->ipywidgets) (2.0.5)
Requirement already satisfied: pure-eval in /apps/anaconda3/2024.02-1/lib/python3.11/site-packages (from stack-data->ipython>=6.1.0->ipywidgets) (0.2.2)
Requirement already satisfied: six in /apps/anaconda3/2024.02-1/lib/python3.11/site-packages (from asttokens->stack-data->ipython>=6.1.0->ipywidgets) (1.16.0)
```

```
# Let's train our model
for epoch in range(N_EPOCHS):
    train_loss, train_acc = train(model, train_data_loader, optimizer, criterion)
    valid_loss, valid_acc = evaluate(model, val_data_loader, criterion)

    print(f'| Epoch: {epoch+1:02} | Train Loss: {train_loss:.3f} | Train Acc: {train_acc*100:.2f}% | Val. Loss: {valid_loss:.3f} | Val. Acc: {valid_acc*100:.2f}% |')
```

```
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 01 | Train Loss: 0.541 | Train Acc: 71.69% | Val. Loss: 0.418 | Val. Acc: 83.75% |
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 02 | Train Loss: 0.330 | Train Acc: 88.14% | Val. Loss: 0.440 | Val. Acc: 82.29% |
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 03 | Train Loss: 0.191 | Train Acc: 94.21% | Val. Loss: 0.443 | Val. Acc: 84.86% |
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 04 | Train Loss: 0.190 | Train Acc: 93.84% | Val. Loss: 0.473 | Val. Acc: 84.86% |
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 05 | Train Loss: 0.090 | Train Acc: 97.33% | Val. Loss: 0.445 | Val. Acc: 86.88% |
```

Task 2: **After training, what is the training accuracy that your model achieves?**

```
#TODO: add your code below to print the final training accuracy out
print(f"Final Training Accuracy after training model: {train_acc*100:.2f}%")
```

```
Final Training Accuracy after training model: 97.33%
```

Model Evaluation

Task 3: **Let's review the previous code then finish the next code cell**

```
test_loss, test_acc = evaluate(model, test_data_loader, criterion)
print(f'| Test Loss: {test_loss:.3f} | Test Acc: {test_acc*100:.2f}% |')
```

```
| Test Loss: 0.573 | Test Acc: 84.17% |
```

Deployment

We can use the prediction function `predict_cb` to predict whether a sentence is cyberbullying or not.

```
# Example 1: "Hello World!"
text = 'hello world!'
ret = predict_cb(text)
print("Sample prediction: ", ret[2], f'Confidence: {ret[0].item() * 100:.2f}%')
```

```
Sample prediction:  Cyberbullying not detected. Confidence: 99.08%
```

Task 4: **Use the samples in this file and your model to detect the cyberbullying samples**

```
text1 = "you guys are a bunch of losers, fuck you"
ret1 = detect_cyberbullying(text1)
print("==========================================")
print(f"Text 1: {text1}\nCyberbullying Detected: {ret1}")
ret = predict_cb(text1)
print(f"Confidence: {ret[0].item() * 100:.2f}%")

print("==========================================")

text2 = "I'm never going to see your little pathetic self again"
ret2 = detect_cyberbullying(text2)
print(f"Text 2: {text2}\nCyberbullying Detected: {ret2}")
ret = predict_cb(text2)
print(f"Confidence: {ret[0].item() * 100:.2f}%")

print("==========================================")

text3 = "She looks really nice today!"
ret3 = detect_cyberbullying(text3)
print(f"Text 3: {text3}\nCyberbullying Detected: {ret3}")
ret = predict_cb(text3)
print(f"Confidence: {ret[0].item() * 100:.2f}%")
```

```
==========================================
Text 1: you guys are a bunch of losers, fuck you
Cyberbullying Detected: True
Confidence: 99.52%
==========================================
Text 2: I'm never going to see your little pathetic self again
Cyberbullying Detected: False
Confidence: 98.73%
==========================================
Text 3: She looks really nice today!
Cyberbullying Detected: False
Confidence: 99.74%
```

Input your own sentence to check the prediction.

My_Sentence: "you are very ugly"

Show code

```
==========The Model Prediciton is==========
The input sentence is:  Cyberbullying detected. Confidence: 99.31%
```

Hyperparameter Tuning

A fast training function

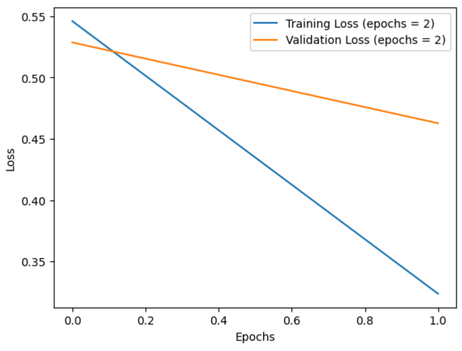
```
# @title A fast training function
def train_model(model, train_data_loader, val_data_loader, number_of_epochs, learning_rate, verbose=True):
    """
    Trains our AI model and plots the learning curve
    Arguments:
        model: model to be trained
        train_iterator: an iterator over the training %set_env
        validation_iterator: an iterator over the validation set
        number_of_epochs: The number of times to go through our entire dataset
        optimizer: the optimization function, defaults to None
        criterion: the loss function, defaults to None
        learning_rate: the learning rate, defaults to 0.001
        verbose: Boolean - whether to print accuracy and loss
    Returns:
        learning_curve: Dictionary - contains variables for plotting the learning curve
    """

    # initialize variables for plotting
    epochs = [i for i in range(number_of_epochs)]
    train_losses = []
    validation_losses = []
    validation_accs = []

    # define the optimizer and loss function
    optimizer = AdamW(model.parameters(), lr=learning_rate, correct_bias=False)
    criterion = nn.CrossEntropyLoss().to(device)

    model = model.to(device)

    # train the model
    for epoch in range(number_of_epochs):
        train_loss, train_acc = train(model, train_data_loader, optimizer, criterion)
        valid_loss, valid_acc = evaluate(model, val_data_loader, criterion)
        train_losses.append(train_loss)
        validation_losses.append(valid_loss)
        validation_accs.append(valid_acc)
        if verbose:
            print(f'| Epoch: {epoch+1:02} | Train Loss: {train_loss:.3f} | Train Acc: {train_acc*100:.2f}% | Val. Loss: {valid_loss:.3f} | Val. Acc: {valid_acc*100:.2f}% |')

    test_loss, test_acc = evaluate(model, test_data_loader, criterion)
    if verbose:
        print(f'Test Loss: {test_loss:.3f} | Test Acc: {test_acc*100:.2f}% |')
    print()

    epochs = np.asarray(epochs)
    train_losses = np.asarray(train_losses)
    validation_losses = np.asarray(validation_losses)
    validation_accs = np.asarray(validation_accs)

    learning_curve = {
        'epochs': epochs,
        'train_losses': train_losses,
        'validation_losses': validation_losses,
        'validation_accs': validation_accs,
        'learning_rate': learning_rate,
    }

    return learning_curve
```

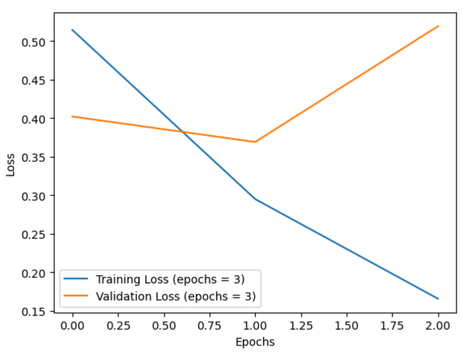
Task 5: **Compare different traning epochs with 2, 10. You can try more different settings and find a suitable epoch number.**

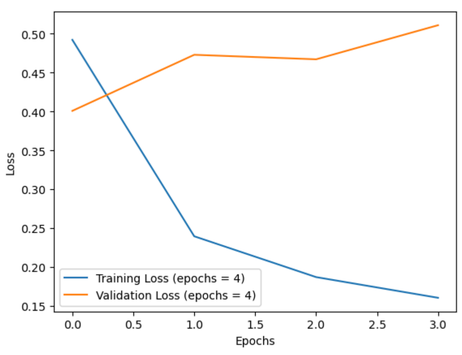
```
training_epochs = [2,3,4]
learning_curve = {}

for i, epoch in enumerate(training_epochs,1):
    print(f'Training for {epoch} epochs')
    # Initialize the model
    bert_model = BertModel.from_pretrained('bert-base-uncased', output_hidden_states = True).to(device)
    model = CyberbullyingDetecter(bert_model, Num_classes).to(device)
    # Train the model
    learning_curve[i] = train_model(model, train_data_loader, val_data_loader, epoch, 2e-5, verbose=True)
    print('training complete!')
```

```
Training for 2 epochs
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 01 | Train Loss: 0.546 | Train Acc: 72.52% | Val. Loss: 0.529 | Val. Acc: 73.89% |
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 02 | Train Loss: 0.324 | Train Acc: 87.22% | Val. Loss: 0.463 | Val. Acc: 82.29% |
Test Loss: 0.504 | Test Acc: 81.67% |

training complete!
Training for 3 epochs
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 01 | Train Loss: 0.515 | Train Acc: 73.25% | Val. Loss: 0.402 | Val. Acc: 85.76% |
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 02 | Train Loss: 0.295 | Train Acc: 88.05% | Val. Loss: 0.369 | Val. Acc: 86.88% |
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 03 | Train Loss: 0.166 | Train Acc: 95.04% | Val. Loss: 0.519 | Val. Acc: 86.46% |
Test Loss: 0.493 | Test Acc: 84.58% |

training complete!
Training for 4 epochs
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 01 | Train Loss: 0.492 | Train Acc: 75.09% | Val. Loss: 0.401 | Val. Acc: 84.93% |
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 02 | Train Loss: 0.240 | Train Acc: 91.36% | Val. Loss: 0.473 | Val. Acc: 84.65% |
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 03 | Train Loss: 0.187 | Train Acc: 94.85% | Val. Loss: 0.467 | Val. Acc: 85.76% |
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 04 | Train Loss: 0.160 | Train Acc: 94.12% | Val. Loss: 0.511 | Val. Acc: 83.82% |
Test Loss: 0.543 | Test Acc: 84.69% |

training complete!
```

```
for i, epoch in enumerate(training_epochs,1):
    plt.plot(learning_curve[i]['epochs'], learning_curve[i]['train_losses'], label=f'Training Loss (epochs = {epoch})')
    plt.plot(learning_curve[i]['epochs'], learning_curve[i]['validation_losses'], label=f'Validation Loss (epochs = {epoch})')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()
```







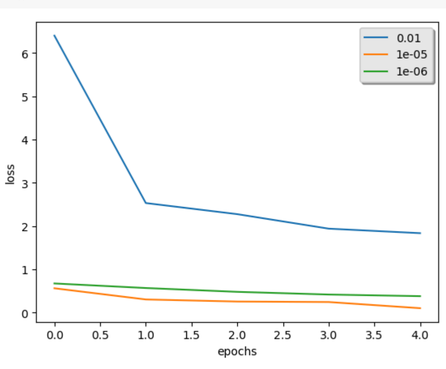
Task 6: Compare different learning rates with 0.1, 1e-3 and 1e-5. You can try with your own settings and find the best learning rate.

Learning rate needs to be chosen carefully in order for gradient descent to work properly. How quickly we update the parameters of our models is determined by the learning rate. If we choose the learning rate to be too small, we may need a lot more iteration to converge to the optimal values. If we choose the learning rate to be too big, we may go past our optimal values. So, it is important to choose the learning rate carefully.

```
#Duplicate Code for ensuring no mistakes
learning_rates = [0.01, 1e-5, 1e-6]
learning_curve = {}  # No semicolon needed here

for i, lr in enumerate(learning_rates,1):
    print('Learning rate:', lr)
    # Initialize the model
    bert_model = BertModel.from_pretrained('bert-base-uncased', output_hidden_states = True).to(device)
    model = CyberbullyingDetecter(bert_model, Num_classes).to(device)
    # Train the model
    learning_curve[i] = train_model(model, train_data_loader, val_data_loader, 5, lr, verbose=True)
    print('Training complete!')
```

```
Learning rate: 0.01
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 01 | Train Loss: 6.403 | Train Acc: 49.54% | Val. Loss: 1.221 | Val. Acc: 49.58% |
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 02 | Train Loss: 2.530 | Train Acc: 50.55% | Val. Loss: 0.728 | Val. Acc: 49.58% |
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 03 | Train Loss: 2.273 | Train Acc: 52.02% | Val. Loss: 1.247 | Val. Acc: 49.58% |
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 04 | Train Loss: 1.938 | Train Acc: 51.38% | Val. Loss: 0.696 | Val. Acc: 50.42% |
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 05 | Train Loss: 1.834 | Train Acc: 47.79% | Val. Loss: 1.920 | Val. Acc: 50.42% |
Test Loss: 1.948 | Test Acc: 49.69% |

Training complete!
Learning rate: 1e-05
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 01 | Train Loss: 0.559 | Train Acc: 70.31% | Val. Loss: 0.442 | Val. Acc: 76.94% |
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 02 | Train Loss: 0.300 | Train Acc: 87.96% | Val. Loss: 0.393 | Val. Acc: 83.19% |
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 03 | Train Loss: 0.252 | Train Acc: 90.99% | Val. Loss: 0.447 | Val. Acc: 86.32% |
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 04 | Train Loss: 0.241 | Train Acc: 90.99% | Val. Loss: 0.381 | Val. Acc: 86.18% |
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 05 | Train Loss: 0.098 | Train Acc: 97.70% | Val. Loss: 0.472 | Val. Acc: 84.38% |
Test Loss: 0.437 | Test Acc: 85.31% |

Training complete!
Learning rate: 1e-06
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 01 | Train Loss: 0.670 | Train Acc: 57.35% | Val. Loss: 0.597 | Val. Acc: 74.38% |
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 02 | Train Loss: 0.565 | Train Acc: 71.69% | Val. Loss: 0.490 | Val. Acc: 78.33% |
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 03 | Train Loss: 0.476 | Train Acc: 81.25% | Val. Loss: 0.483 | Val. Acc: 79.38% |
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 04 | Train Loss: 0.414 | Train Acc: 85.66% | Val. Loss: 0.446 | Val. Acc: 80.76% |
  0%|          | 0/34 [00:00<?, ?it/s]
| Epoch: 05 | Train Loss: 0.376 | Train Acc: 86.49% | Val. Loss: 0.434 | Val. Acc: 80.76% |
Test Loss: 0.442 | Test Acc: 81.46% |

Training complete!
```

```
for i, lr in enumerate(learning_rates,1):
    plt.plot(learning_curve[i]["epochs"], np.squeeze(learning_curve[i]["train_losses"]), label=learning_curve[i]["learning_rate"])

plt.ylabel('loss')
plt.xlabel('epochs')

legend = plt.legend(loc='best', shadow=True)
frame = legend.get_frame()
frame.set_facecolor('0.90')
plt.show()
```



Task 7: Discussion

We experimented with different hyperparameters in this lab, what can you conclude about training AI models? Specifically, what are your observations about the model before Vs. after hyperparameter tuning?

Initial Training: The model shows strong training accuracy improvement from 71.69% to 97.33%, but the validation accuracy improves more modestly from 83.75% to 86.88%. The increasing gap between training and validation performance, particularly after epoch 3, suggests the model begins to overfit to the training data. Task 5: In this experiment comparing different training epochs (2, 3, and 4), the results demonstrate that 3 epochs provide the optimal balance between model performance and training efficiency. With 2 epochs, the model achieves 81.67% test accuracy but shows signs of underfitting with relatively high loss values. The 3-epoch training yields the best results with 84.58% test accuracy

Task 7: Discussion

We experimented with different hyperparameters in this lab, what can you conclude about training AI models? Specifically, what are your observations about the model before Vs. after hyperparameter tuning?

Initial Training: The model shows strong training accuracy improvement from 71.69% to 97.33%, but the validation accuracy improves more modestly from 83.75% to 86.88%. The increasing gap between training and validation performance, particularly after epoch 3, suggests the model begins to overfit to the training data.
Task 5: In this experiment comparing different training epochs (2, 3, and 4), the results demonstrate that 3 epochs provide the optimal balance between model performance and training efficiency. With 2 epochs, the model achieves 81.67% test accuracy but shows signs of underfitting with relatively high loss values. The 3-epoch training yields the best results with 84.58% test accuracy and shows good progression in both training and validation metrics. While the 4-epoch training achieves a similar test accuracy of 84.69%, it shows signs of overfitting with declining validation accuracy in the final epoch and higher test loss (0.543). This indicates that extending training beyond 3 epochs doesn't provide significant benefits and may harm the model's generalization ability. Therefore, 3 epochs appear to be the most suitable choice for this particular model and dataset.
Task 6: In comparing three different learning rates (0.01, 1e-5, and 1e-6), the experiments reveal that 1e-5 provides the optimal learning rate for this model. With a learning rate of 0.01, the model performed poorly, achieving only 49.69% test accuracy and showing unstable learning with high loss values, indicating the learning rate was too aggressive. The learning rate of 1e-5 demonstrated the best performance, reaching 85.31% test accuracy with stable learning progression and good generalization. The smallest learning rate of 1e-6, while showing steady improvement, resulted in slower convergence and lower final performance with 81.46% test accuracy. These results confirm that 1e-5 strikes the right balance between learning speed and stability, avoiding both the overshooting of higher learning rates and the slow convergence of lower ones.
Through a series of experiments in hyperparameter tuning, we identified optimal settings for model performance. Initial training revealed overfitting tendencies with a significant gap between training (97.33%) and validation accuracy (86.88%). Testing different epoch counts showed that 3 epochs provided the best balance, achieving 84.58% test accuracy while avoiding both underfitting (seen at 2 epochs) and overfitting (at 4 epochs). Learning rate optimization demonstrated that 1e-5 was optimal, achieving 85.31% test accuracy, while higher rates (0.01) led to unstable learning and lower rates (1e-6) resulted in slow convergence. Overall, systematic hyperparameter tuning improved not just accuracy but also model stability and generalization, highlighting the importance of careful parameter selection in AI model development.