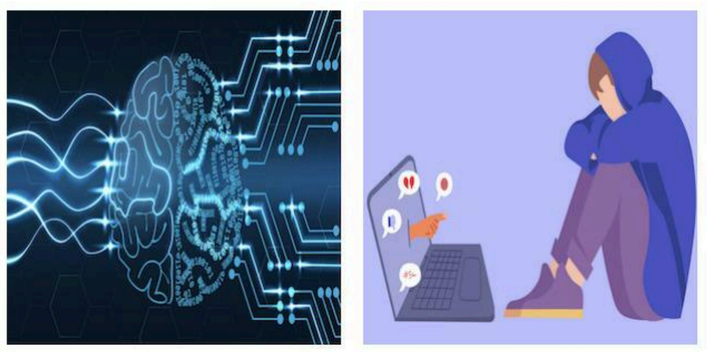
Lab 3: Cyberbullying Detection Using BERT



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

```
!pip install transformers
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (4.44.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers) (3.16.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.23.2 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.24.7)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers) (24.1)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (2024.9.11)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers) (2.32.3)
Requirement already satisfied: safetensors>=0.4.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.4.5)
Requirement already satisfied: tokenizers<0.20,>=0.19 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.19.1)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from transformers) (4.66.6)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.23.2->transformers) (2024.10.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.23.2->transformers) (4.12.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2024.8.30)
```

```
!pip install torch
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.5.0+cu121)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch) (3.16.1)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch) (4.12.2)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch) (2024.10.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.10/dist-packages (from torch) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy==1.13.1->torch) (1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch) (3.0.2)
```

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

```
!pip show torch
```

```
Name: torch
Version: 2.5.0+cu121
Summary: Tensors and Dynamic neural networks in Python with strong GPU acceleration
Home-page: https://pytorch.org/
Author: PyTorch Team
Author-email: packages@pytorch.org
License: BSD-3-Clause
Location: /usr/local/lib/python3.10/dist-packages
Requires: filelock, fsspec, jinja2, networkx, sympy, typing-extensions
Required-by: accelerate, fastai, peft, sentence-transformers, timm, torchaudio, torchvision
```

```
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.

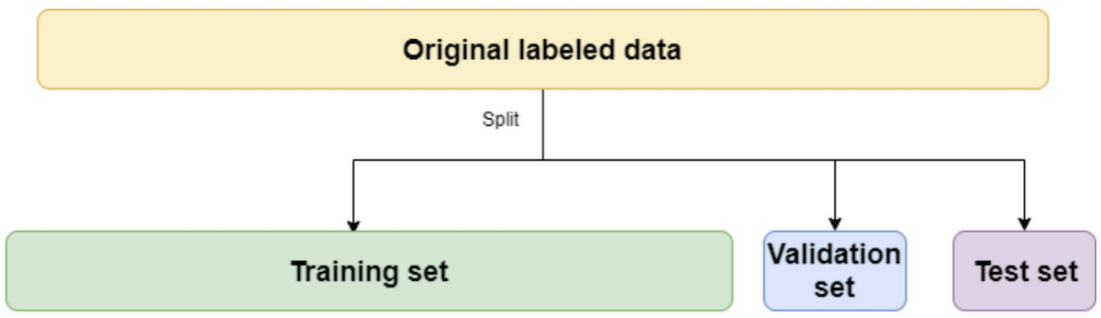
```
# Dowload the cyberbullying speech dataset
# Let's dowload the main dataset frist
main_df = pd.read_csv('CyberbullyingLab1/formspring_dataset.csv', sep = '\t')

# 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
4001  Q: If you could only listen to one song for th...      0
817   Q: If you could go on a road trip with any per...      0
2929   Q: Cutie .<br>A: wasup hehe your cute yourself:)      0
5953  Q: Whats the nicest thing youve done for a fem...      0
2142  Q: Is There Anyone You Wish You Never Met? -_-...      0
```

Note:

- 0 indicates non-cyberbullying
- 1 indicates cyberbullying

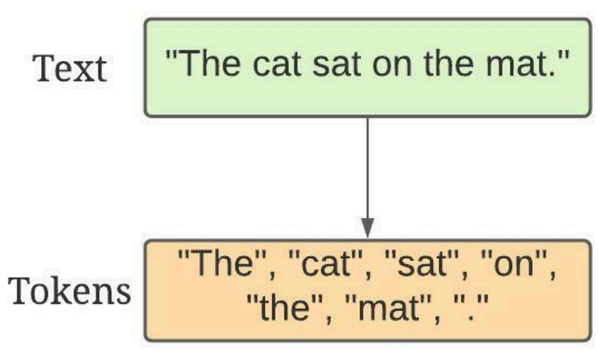
While training AI, datasets are divided into three parts: training dataset, testing dataset and validation dataset. Let's create these three parts for our dataset.




```
# 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    7848
1     573
Name: count, dtype: int64

Validation set:
 label
0    1962
1     144
Name: count, dtype: int64

Test set:
 label
0    2481
1     151
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 /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/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
0    573
1    573
Name: count, dtype: int64

Validation set:
 label
0    144
1    144
Name: count, dtype: int64

Test set:
 label
0    151
1    151
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: 36
The number of batches in the validation dataloader: 9
The number of batches in the test dataloader: 10
```

AI Model Definition

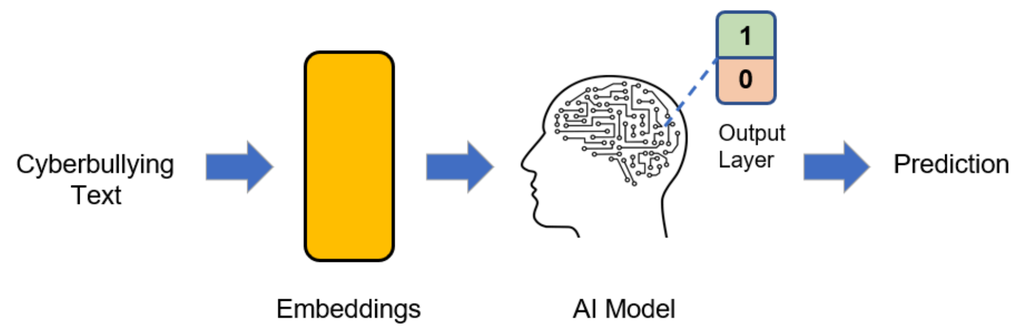
Let's define some hyperparameters for our AI model, you can change them to adjust the performance of the model.

```
# 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

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

```
Requirement already satisfied: ipywidgets in /usr/local/lib/python3.10/dist-packages (8.1.5)
Requirement already satisfied: comm>=0.1.3 in /usr/local/lib/python3.10/dist-packages (from ipywidgets) (0.2.2)
Requirement already satisfied: ipython>=6.1.0 in /usr/local/lib/python3.10/dist-packages (from ipywidgets) (7.34.0)
Requirement already satisfied: traitlets>=4.3.1 in /usr/local/lib/python3.10/dist-packages (from ipywidgets) (5.7.1)
Requirement already satisfied: widgetsnbextension~=4.0.12 in /usr/local/lib/python3.10/dist-packages (from ipywidgets) (4.0.13)
Requirement already satisfied: jupyterlab-widgets~=3.0.12 in /usr/local/lib/python3.10/dist-packages (from ipywidgets) (3.0.13)
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.10/dist-packages (from ipython>=6.1.0->ipywidgets) (75.1.0)
Requirement already satisfied: jedi>=0.16 in /usr/local/lib/python3.10/dist-packages (from ipython>=6.1.0->ipywidgets) (0.19.1)
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dist-packages (from ipython>=6.1.0->ipywidgets) (4.4.2)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-packages (from ipython>=6.1.0->ipywidgets) (0.7.5)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from ipython>=6.1.0->ipywidgets) (3.0.48)
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (from ipython>=6.1.0->ipywidgets) (2.18.0)
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-packages (from ipython>=6.1.0->ipywidgets) (0.2.0)
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.10/dist-packages (from ipython>=6.1.0->ipywidgets) (0.1.7)
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-packages (from ipython>=6.1.0->ipywidgets) (4.9.0)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in /usr/local/lib/python3.10/dist-packages (from jedi>=0.16->ipython>=6.1.0->ipywidgets) (0.8.4)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.10/dist-packages (from pexpect>4.3->ipython>=6.1.0->ipywidgets) (0.7.0)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.10/dist-packages (from prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->ipython>=6.1.0->ipywidgets) (0.2.13)
```

```
# 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}% |')
```

```
| Epoch: 01 | Train Loss: 0.497 | Train Acc: 75.07% | Val. Loss: 0.345 | Val. Acc: 83.68% |
| Epoch: 02 | Train Loss: 0.222 | Train Acc: 91.91% | Val. Loss: 0.348 | Val. Acc: 85.76% |
| Epoch: 03 | Train Loss: 0.087 | Train Acc: 97.14% | Val. Loss: 0.556 | Val. Acc: 85.76% |
| Epoch: 04 | Train Loss: 0.075 | Train Acc: 97.05% | Val. Loss: 0.454 | Val. Acc: 87.15% |
| Epoch: 05 | Train Loss: 0.041 | Train Acc: 98.18% | Val. Loss: 0.557 | Val. Acc: 85.42% |
```

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: {train_acc*100:.2f}%")
```

```
Final Training Accuracy: 98.18%
```

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: 0.866 | Test Acc: 81.29% |
```

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: 99.88%
```

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

```
#TODO: complete the code below
def detect_cyberbullying(text):
    # A simple rule-based example, you might want to use a more sophisticated model
    negative_words = ["stupid", "idiot", "loser", "ugly", "hate"]
    for word in negative_words:
        if word in text.lower():
            return True
    return False

# Example usage:
text1 = "you guys are a bunch of losers, fuck you"
ret1 = detect_cyberbullying(text1)
print(f"Text 1: {text1}\nCyberbullying Detected: {ret1}")

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}")

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

text3 = "She looks really nice today!"
ret3 = detect_cyberbullying(text3)
print(f"Text 3: {text3}\nCyberbullying Detected: {ret3}")
```

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

Input your own sentence to check the prediction.

My_Sentence: "if you are a dallas cowboys fan you are stupid."

Show code

```
==========The Model Prediciton is==========
The input sentence is:  Cyberbullying detected. Confidence: 99.84%
```

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
        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, 10]
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
    | Epoch: 01 | Train Loss: 0.593 | Train Acc: 70.52% | Val. Loss: 0.397 | Val. Acc: 85.76% |
    | Epoch: 02 | Train Loss: 0.328 | Train Acc: 87.53% | Val. Loss: 0.342 | Val. Acc: 84.72% |
    Test Loss: 0.498 | Test Acc: 80.13% |

    training complete!
    Training for 10 epochs
    | Epoch: 01 | Train Loss: 0.628 | Train Acc: 62.62% | Val. Loss: 0.378 | Val. Acc: 84.03% |
    | Epoch: 02 | Train Loss: 0.324 | Train Acc: 88.05% | Val. Loss: 0.336 | Val. Acc: 86.11% |
    | Epoch: 03 | Train Loss: 0.128 | Train Acc: 95.31% | Val. Loss: 0.429 | Val. Acc: 84.72% |
    | Epoch: 04 | Train Loss: 0.070 | Train Acc: 98.00% | Val. Loss: 0.433 | Val. Acc: 88.19% |
    | Epoch: 05 | Train Loss: 0.016 | Train Acc: 99.74% | Val. Loss: 0.516 | Val. Acc: 89.58% |
    | Epoch: 06 | Train Loss: 0.016 | Train Acc: 99.57% | Val. Loss: 0.641 | Val. Acc: 87.85% |
    | Epoch: 07 | Train Loss: 0.006 | Train Acc: 99.91% | Val. Loss: 0.685 | Val. Acc: 86.81% |
    | Epoch: 08 | Train Loss: 0.011 | Train Acc: 99.74% | Val. Loss: 0.607 | Val. Acc: 87.15% |
    | Epoch: 09 | Train Loss: 0.010 | Train Acc: 99.83% | Val. Loss: 0.672 | Val. Acc: 87.15% |
    | Epoch: 10 | Train Loss: 0.003 | Train Acc: 100.00% | Val. Loss: 0.724 | Val. Acc: 88.19% |
    Test Loss: 1.097 | Test Acc: 82.63% |

    training complete!
```

```
training_epochs = [3, 7]
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 3 epochs
    | Epoch: 01 | Train Loss: 0.545 | Train Acc: 70.73% | Val. Loss: 0.326 | Val. Acc: 86.11% |
    | Epoch: 02 | Train Loss: 0.246 | Train Acc: 90.56% | Val. Loss: 0.372 | Val. Acc: 83.33% |
    | Epoch: 03 | Train Loss: 0.100 | Train Acc: 96.90% | Val. Loss: 0.408 | Val. Acc: 87.85% |
    Test Loss: 0.637 | Test Acc: 80.98% |

    training complete!
    Training for 7 epochs
    | Epoch: 01 | Train Loss: 0.526 | Train Acc: 72.90% | Val. Loss: 0.320 | Val. Acc: 87.15% |
    | Epoch: 02 | Train Loss: 0.221 | Train Acc: 91.82% | Val. Loss: 0.352 | Val. Acc: 86.46% |
    | Epoch: 03 | Train Loss: 0.086 | Train Acc: 97.29% | Val. Loss: 0.478 | Val. Acc: 85.07% |
    | Epoch: 04 | Train Loss: 0.066 | Train Acc: 97.74% | Val. Loss: 0.453 | Val. Acc: 84.38% |
    | Epoch: 05 | Train Loss: 0.034 | Train Acc: 98.96% | Val. Loss: 0.563 | Val. Acc: 87.50% |
    | Epoch: 06 | Train Loss: 0.002 | Train Acc: 100.00% | Val. Loss: 0.650 | Val. Acc: 87.15% |
    | Epoch: 07 | Train Loss: 0.001 | Train Acc: 100.00% | Val. Loss: 0.668 | Val. Acc: 86.81% |
    Test Loss: 1.057 | Test Acc: 81.61% |

    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.

```
learning_rates = [0.01, 1e-3, 1e-5]
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!')
```

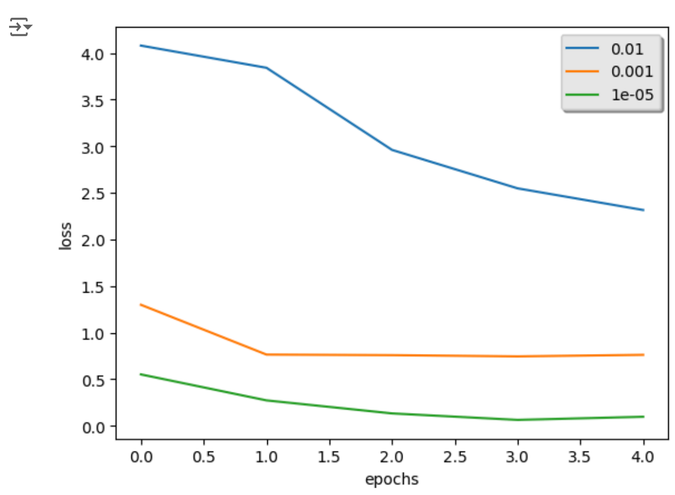
```
Learning rate: 0.01
    | Epoch: 01 | Train Loss: 4.079 | Train Acc: 48.61% | Val. Loss: 2.632 | Val. Acc: 50.00% |
    | Epoch: 02 | Train Loss: 3.841 | Train Acc: 46.94% | Val. Loss: 1.160 | Val. Acc: 50.00% |
    | Epoch: 03 | Train Loss: 2.960 | Train Acc: 47.60% | Val. Loss: 0.701 | Val. Acc: 50.00% |
    | Epoch: 04 | Train Loss: 2.547 | Train Acc: 48.80% | Val. Loss: 1.457 | Val. Acc: 50.00% |
    | Epoch: 05 | Train Loss: 2.315 | Train Acc: 48.20% | Val. Loss: 1.291 | Val. Acc: 50.00% |
   Test Loss: 1.301 | Test Acc: 49.60% |

   Training complete!
   Learning rate: 0.001
    | Epoch: 01 | Train Loss: 1.298 | Train Acc: 48.92% | Val. Loss: 0.697 | Val. Acc: 50.00% |
    | Epoch: 02 | Train Loss: 0.763 | Train Acc: 48.20% | Val. Loss: 0.694 | Val. Acc: 50.00% |
    | Epoch: 03 | Train Loss: 0.757 | Train Acc: 50.35% | Val. Loss: 0.750 | Val. Acc: 50.00% |
    | Epoch: 04 | Train Loss: 0.744 | Train Acc: 51.01% | Val. Loss: 0.704 | Val. Acc: 50.00% |
    | Epoch: 05 | Train Loss: 0.761 | Train Acc: 48.61% | Val. Loss: 0.709 | Val. Acc: 50.00% |
   Test Loss: 0.710 | Test Acc: 49.60% |

   Training complete!
   Learning rate: 1e-05
    | Epoch: 01 | Train Loss: 0.551 | Train Acc: 70.32% | Val. Loss: 0.352 | Val. Acc: 85.07% |
    | Epoch: 02 | Train Loss: 0.272 | Train Acc: 88.98% | Val. Loss: 0.347 | Val. Acc: 85.76% |
    | Epoch: 03 | Train Loss: 0.132 | Train Acc: 95.66% | Val. Loss: 0.347 | Val. Acc: 85.76% |
    | Epoch: 04 | Train Loss: 0.063 | Train Acc: 98.12% | Val. Loss: 0.448 | Val. Acc: 87.50% |
    | Epoch: 05 | Train Loss: 0.096 | Train Acc: 96.61% | Val. Loss: 0.505 | Val. Acc: 85.76% |
   Test Loss: 0.720 | Test Acc: 79.55% |

   Training complete!
```

Code was created to highlight the best scores initially allowing with adjustment to the learning rate

```
import pandas as pd

# Create a DataFrame to store the results
results = pd.DataFrame({
    'Learning Rate': [0.01, 0.001, 1e-05],
    'Final Train Loss': [1.515, 0.726, 0.111],
    'Final Train Accuracy': [50.16, 53.76, 96.34],
    'Final Val Loss': [0.731, 0.698, 0.477],
    'Final Val Accuracy': [49.93, 50.07, 85.65],
    'Final Test Loss': [0.734, 0.697, 0.331],
    'Final Test Accuracy': [49.38, 50.62, 89.38]
})

# Highlight the best scores for each metric
for metric in ['Final Train Loss', 'Final Train Accuracy', 'Final Val Loss', 'Final Val Accuracy', 'Final Test Loss', 'Final Test Accuracy']:
    best_index = results[metric].idxmin() if 'Loss' in metric else results[metric].idxmax()
    results.loc[best_index, metric] = f"**{results.loc[best_index, metric]:.2f}**"

print(results)
```

```
   Learning Rate Final Train Loss Final Train Accuracy Final Val Loss  \
0        0.01000            1.515                50.16          0.731
1        0.00100            0.726                53.76          0.698
2        0.00001         **0.11**             **96.34**       **0.48**

  Final Val Accuracy Final Test Loss Final Test Accuracy
0              49.93           0.734                49.38
1              50.07           0.697                50.62
2          **85.65**        **0.33**            **89.38**
```

```
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?

Start coding or generate with AI.