Fall 2023 Applied NLP Homework 4

Instructors: Dr. Mahdi Roozbahani, Wafa Louhichi, Dr. Nimisha Roy

Deadline: December 1st, 11:59PM AoE

Honor Code and Assignment Deadline

- No unapproved extension of the deadline is allowed. Late submission will lead to 0 credit.
- Discussion is encouraged on Ed as part of the Q/A. However, all assignments should be done individually.
- Plagiarism is a **serious offense**. You are responsible for completing your own work. You are not allowed to copy and paste, or paraphrase, or submit materials created or published by others, as if you created the materials. All materials submitted must be your own.
- All incidents of suspected dishonesty, plagiarism, or violations of the Georgia Tech Honor Code will be subject to the institute's Academic Integrity procedures. If we observe any (even small) similarities/plagiarisms detected by Gradescope or our TAs, **WE WILL DIRECTLY REPORT ALL CASES TO OSI**, which may, unfortunately, lead to a very harsh outcome. **Consequences can be severe, e.g., academic probation or dismissal, grade penalties, a 0 grade for assignments concerned, and prohibition from withdrawing from the class.**

Instructions for the assignment

- This entire assignment will be autograded through Gradescope. There is only one Gradescope submissions for this assignment:
  - **Homework 4**: Submit all files to this section
- We provided you different .py files and we added libraries in those files please DO NOT remove those lines and add your code after those lines. Note that these are the only allowed libraries that you can use for the homework.
- You will submit your implemented .py files to the corresponding homework section on Gradescope.
- You are allowed to make as many submissions until the deadline as you like. Additionally, note that the autograder tests each function separately, therefore it can serve as a useful tool to help you debug your code if you are not sure of what part of your implementation might have an issue.

Using the local tests

- For some of the programming questions we have included a local test using a small toy dataset to aid in debugging. The local test sample data and outputs are stored in .py files in the **local_tests** folder
- There are no points associated with passing or failing the local tests, you must still pass the autograder to get points.
- **It is possible to fail the local test and pass the autograder** since the autograder has a certain allowed error tolerance while the local test allowed error may be smaller. Likewise, passing the local tests does not guarantee passing the autograder.
- **You do not need to pass both local and autograder tests to get points, passing the Gradescope autograder is what is required for credit.**
- It might be helpful to comment out the tests for functions that have not been completed yet.
- It is recommended to test the functions as it gets completed instead of completing the whole class and then testing. This may help in isolating errors. Do not solely rely on the local tests, continue to test on the autograder regularly as well.

Google Colab Setup (Optional for running on Colab)

If you choose to work on the assignment on Google Colab, the following cell may help get you set up. You may need to right click on the Applied NLP folder and `Add shortcut to Drive`. You do not have to run this cell if you are working on the notebook locally.

```
# # Mount google drive
# from google.colab import drive
# drive.mount('/content/drive/')
# # You may need to create an Applied_NLP/HW#/hw#_code/ folder
# %cd '/content/drive/MyDrive/Applied_NLP/HW#/hw#_code/'

# ## If no GPU selected it will ask for GPU to be selected
# 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 accelerator, ')
#   print('and then re-execute this cell.')
# else:
#   print(gpu_info)


# ## This wraps output text according to the window size
# from IPython.display import HTML, display

# def set_css():
#   display(HTML('''
#   <style>
#     pre {
#         white-space: pre-wrap;
#     }
#   </style>
#   '''))
# get_ipython().events.register('pre_run_cell', set_css)
```

Assignment Overview

In this homework we will explore non-linear text classification algorithms using deep neural networks.

We will reuse the datasets from HW3 and a new dataset for this exploration:

- The first dataset is a subset of a Clickbait Dataset that has article headlines and a binary label on whether the headline is considered clickbait.
- The second dataset is a subset of Web of Science Dataset that has articles and a corresponding label on the domain of the articles.

- The third dataset is the CoNLL-2003 that has text corresponding with Part of Speech and Name Entity Recognition labels.

We will first explore LSTM and LSTM with Attention. Then we will be looking into attention based architectures : Transformers and we will apply it to sequence labeling task : Part of Speech Tagging (POS) and Named Entity Recognition (NER). We will also look into Topic Modeling.

Deliverables and Points Distribution

Q1: LSTM [15pts]

- **1.1 Implementing the LSTM Model** [10pts] Deliverables: lstm.py
  - [4pts] __init__
  - [6pts] forward
- **1.2 Classifying Clickbait Dataset using LSTM** [5pts] Deliverables: lstm.py, best_lstm.model.pt

Q2: LSTM with Attention [15pts]

- **2.1 Implementing the LSTM + Attention Model** [10pts] Deliverables: attention.py
  - [2pts] __init__
  - [2pts] forward_lstm
  - [2pts] forward_attention
  - [2pts] forward_context
  - [2pts] forward
- **2.2 Classifying Web of Science Dataset using LSTM + Attention** [5pts] Deliverables: attention.py, best_attention.model.pt

Q3: Transformers [15pts]

- **3.1 Implementing the BERT Classifier** [5pts] Deliverables: bert.py
  - [2pts] __init__
  - [3pts] forward
- **3.4 Classifying Web of Science Dataset using BERT Classifier** [5pts] Deliverables: bert_clickbait.pkl
- **3.5 Classifying Web of Science Dataset using BERT Classifier** [5pts] Deliverables: bert_wos.pkl

Q4: Sequence Labeling [15pts]

- **4.1 Implementing Sequence Labeling** [5pts] Deliverables: sequenceLabeling.py
  - [2pts] __init__
  - [3pts] forward
- **4.5 POS Tagging and NER using CoNLL-2003 Dataset** [10pts] Deliverables: bert_pos.pkl, bert_ner.pkl

Q5: Topic Modeling [20pts]

- **Topic modeling with Latent Dirichlet Allocation (LDA)** [20pts] Deliverables: lda.py
  - [5pts] tokenize_words
  - [5pts] remove_stopwords
  - [5pts] create_dictionary
  - [5pts] build_LDAModel

Setup

Please checkout the environment_setup.md file to create the environment for this homework. This notebook is tested under the package versions noted in the Library Imports cell output below, and the corresponding packages can be downloaded from miniconda. You may also want to get yourself familiar with several packages:

- jupyter notebook
- numpy
- sklearn
- pytorch

In the .py files please implement the functions that have `raise NotImplementedError`, and after you finish the coding, please delete or comment out `raise NotImplementedError`.

Library imports

```
## This cell only needs to be run once after the environment is created and then it can be commented out
!pip install transformers
!pip install datasets
!pip install pyLDAvis
# !pip install numpy==1.24 # may be necessary if running on colab only
```

```
#Import the necessary libraries
import pandas as pd
import pickle
import numpy as np
import scipy as sp
import sys
import re
from copy import deepcopy
import random
from sklearn.metrics import accuracy_score
import torch
import torch.nn as nn
from torch import optim
torch.manual_seed(10)
from torch.autograd import Variable
```

```
import torch.nn.functional as F
from torch.utils.data import DataLoader
import transformers
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
import gensim
import sklearn


import pyLDAvis.gensim_models
import pickle
import pyLDAvis

import torchtext
import datasets

import warnings
warnings.filterwarnings("ignore")

%load_ext autoreload
%autoreload 2
%reload_ext autoreload

# If set to True then all N_EPOCHS will be set to run for one epoch only
# Some compute intensive training loop will break after the first batch
SHORT = False

print('Version information')

print('python: {}'.format(sys.version))
print('numpy: {}'.format(np.__version__))
print('pd: {}'.format(pd.__version__))
print('torch: {}'.format(torch.__version__))
print('transformers: {}'.format(transformers.__version__))
print('gensim: {}'.format(gensim.__version__))
print('pyLDAvis: {}'.format(pyLDAvis.__version__))
print('torchtext: {}'.format(torchtext.__version__))
print('datasets: {}'.format(datasets.__version__))
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
Version information
python: 3.10.12 (main, Jun 11 2023, 05:26:28) [GCC 11.4.0]
numpy: 1.24.0
pd: 2.1.2
torch: 2.1.0+cu118
transformers: 4.35.0
gensim: 4.3.2
pyLDAvis: 3.4.0
torchtext: 0.16.0+cpu
datasets: 2.14.6
```

Load Dataset

We start by loading both data sets already split into an 80/20 train and test set.

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

df_train = pd.read_csv('./data/train.csv')
df_test = pd.read_csv('./data/test.csv')

# Separate dataframes into train and test lists
x_train, y_train = list(df_train['headline']), list(df_train['label'])
x_test, y_test = list(df_test['headline']), list(df_test['label'])
```

Below is the number of headlines in the train and test set as well as a sample of the article headlines and its binary label, where 0 is considered not clickbait and 1 is clickbait.

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

print(f'Number of Train Headlines: {len(x_train)}')
print(f'Number of Test Headlines: {len(x_test)}')

print('\n\nSample Label and Headlines:')
x = 105
for label, line in zip(y_train[x:x+5], x_train[x:x+5]):
    print(f'{label}: {line}')

print('\nOutput of Sample Headlines without Print Statement:')
x_train[x:x+5]
```

```
Number of Train Headlines: 19200
Number of Test Headlines: 4800
```

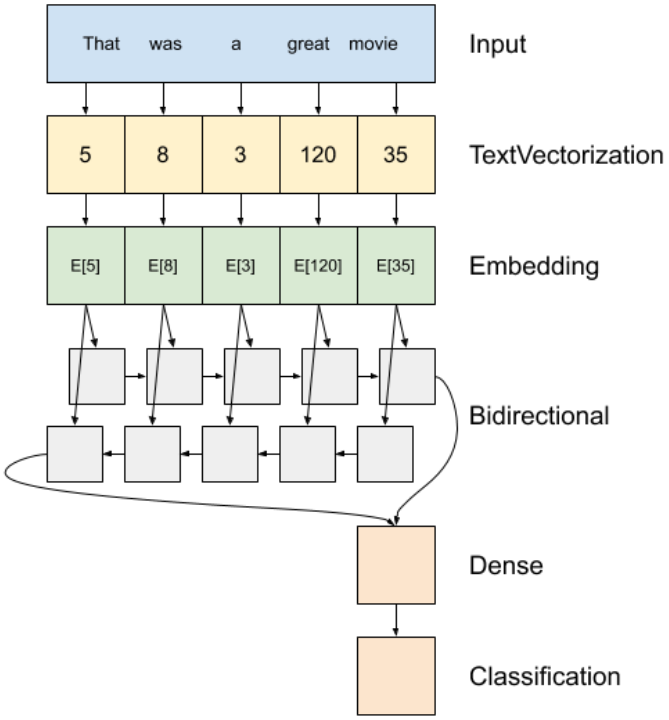
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

# Save test and train as csv
df_train_wos = pd.read_csv('./data/train_wos.csv')
df_test_wos = pd.read_csv('./data/test_wos.csv')

# Separate dataframes into train and test lists
x_train_wos, y_train_wos = list(df_train_wos['article']), list(df_train_wos['label'])
x_test_wos, y_test_wos = list(df_test_wos['article']), list(df_test_wos['label'])

# Numerical label to domain mapping
wos_label = {0:'CS', 1:'ECE', 2:'Civil', 3:'Medical'}
# Numerical label to Numerical mapping
label_mapping = {0:0, 1:1, 4:2, 5:3}

for i, label in enumerate(y_train_wos):
    y_train_wos[i] = label_mapping[label]
for i, label in enumerate(y_test_wos):
    y_test_wos[i] = label_mapping[label]
```

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

print(f'Number of Train Articles: {len(x_train_wos)}')
print(f'Number of Test Articles: {len(x_test_wos)}')

print('\nLabel Key:', wos_label)

print('\nSample Label and Articles:\n')
x = 107
for label, line in zip(y_train_wos[x:x+3], x_train_wos[x:x+3]):
    print(f'{label} - {wos_label[label]}: {line}')
```

```
Number of Train Articles: 1600
Number of Test Articles: 400

Label Key: {0: 'CS', 1: 'ECE', 2: 'Civil', 3: 'Medical'}

Sample Label and Articles:

0 - CS: An efficient procedure for calculating the electromagnetic fields in multilayered cylindrical structures is reported in this paper. Using symbolic computation, spectral Green's functions, suitable for numerical implementations are determined in compact and closed forms. Applications are presented for structures with two dielectric layers.

1 - ECE: A multifunctional platform based on the microhotplate was developed for applications including a Pirani vacuum gauge, temperature, and gas sensor. It consisted of a tungsten microhotplate and an on-chip operational amplifier. The platform was fabricated in a standard complementary metal oxide semiconductor (CMOS) process. A tungsten plug in standard CMOS process was specially designed as the serpentine resistor for the microhotplate, acting as both heater and thermister. With the sacrificial layer technology, the microhotplate was suspended over the silicon substrate with a 340 nm gap. The on-chip operational amplifier provided a bias current for the microhotplate. This platform has been used to develop different kinds of sensors. The first one was a Pirani vacuum gauge ranging from 10(-1) to 10(5) Pa. The second one was a temperature sensor ranging from -20 to 70 degrees C. The third one was a thermal-conductivity gas sensor, which could distinguish gases with different thermal conductivities in constant gas pressure and environment temperature. In the fourth application, with extra fabrication processes including the deposition of gas-sensitive film, the platform was used as a metal-oxide gas sensor for the detection of gas concentration.

2 - Civil: Artificial neural networks have been effectively used in various civil engineering fields, including construction management and labour productivity. In this study, the performance of the feed forward neural network (FFNN) was compared with radial basis neural network (RBNN) in modelling the productivity of masonry crews. A variety of input factors were incorporated and analysed. Mean absolute percentage error (MAPE) and correlation coefficient (R) were used to evaluate model performance. Research results indicated that the neural computing techniques could be successfully employed in modelling crew productivity. It was also found that successful models could be developed with different combinations of input factors, and several of the models which excluded one or more input factors turned out to be better than the baseline models. Based on the MAPE values obtained for the models, the RBNN technique was found to be better than the FFNN technique, although both slightly overestimated the masons' productivity.
```

Q1: Classification with LSTM [15pts]

We will be using an LSTM (Long Short-Term Memory Network) for classification. The architecture of our model looks like :



For more details on LSTM, please refer to class lectures.

Use an Embedding layer, followed by a LSTM layer, and a linear layer.

We will then classify the Clickbait and Web of science dataset for this task.

1.1 : Implementing the LSTM Model [10 pts]

In the **lstm.py** file complete the following functions:

- **__init__**
- **forward**

We have included local tests in 1.2.2 for you to test your implementation

1.1.1 : Pre-Processing Data [No Points]

Run the cell below to load functions for building vocabulary and tokenizing the sentences.

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from torchtext.data import get_tokenizer
from torchtext.vocab import build_vocab_from_iterator
import torchtext

tokenizer = get_tokenizer("basic_english")

def build_vocabulary(datasets):
    for dataset in datasets:
        for text in dataset:
            yield tokenizer(text)

vocab = build_vocab_from_iterator(build_vocabulary([x_train]), min_freq=1, specials=["<UNK>"])
vocab.set_default_index(vocab["<UNK>"])

vocab_wos = build_vocab_from_iterator(build_vocabulary([x_train_wos]), min_freq=1, specials=["<UNK>"])
vocab_wos.set_default_index(vocab["<UNK>"])
```

1.2 : Classifying Clickbait Dataset using LSTM [5 Pts]

Run the cells below to classify the Clickbait train and test datasets using the LSTM functions that you have implemented.

An accuracy of more than 85% is acceptable.

1.2.1 : Create the Dataloaders [No Points]

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from torch.utils.data import DataLoader

max_words = max(map(len, x_train))

def vectorize_batch(batch):
    Y, X = list(zip(*batch))
    X = [vocab(tokenizer(text)) for text in X] ## Tokenize and map tokens to indexes
```

```
    X_len = [len(text) for text in X]
    X = [tokens+([0]* (max_words-len(tokens))) if len(tokens)<max_words else tokens[:max_words] for tokens in X] ## Bringing all samples to max_words length.
    return torch.tensor(X, dtype=torch.int32), torch.tensor(X_len), torch.tensor(Y)

```

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

train_dataset = list(map(lambda y, x: (y, x), y_train, x_train))
test_dataset = list(map(lambda y, x: (y, x), y_test, x_test))

train_loader = DataLoader(train_dataset, batch_size=1024, collate_fn=vectorize_batch, shuffle=True)
test_loader  = DataLoader(test_dataset, batch_size=1024, collate_fn=vectorize_batch)
```

1.2.2: Local Tests for LSTM Functions [No Points]

You may test your implementation of the LSTM functions contained in **lstm.py** in the cell below. Feel free to comment out tests for functions that have not been completed yet. See Using the Local Tests for more details.

```
# ##############################
# ### DO NOT CHANGE THIS CELL ###
# ##############################

from lstm import LSTM
from local_tests.lstm_test import LSTM_Test
torch.manual_seed(10)

local_test = LSTM_Test()
lstm_model = LSTM(vocab, num_classes=2)
lstm_model.load_state_dict(torch.load('./local_tests/basic_lstm_model.pt'))    # upload default weights. If this errors out, make sure you have initialized the layers correctly.
lstm_model.eval()

# creating a vectorized batch from the sample datapoints
local_test_dataset = list(map(lambda y, x: (y, x), local_test.output_labels, local_test.input_sequences))
X_localtest, X_len_localtest, _ = vectorize_batch(local_test_dataset)

print('Local Tests for LSTM Implementation \n')

# Local test for forward
output = lstm_model.forward(X_localtest, X_len_localtest)
forward_test = (output.shape == local_test.output.shape) and (torch.allclose(output, local_test.output, rtol=0.0001, atol=0.0001))
print('Your forward works as expected:', forward_test)
```

```
Local Tests for LSTM Implementation

Your forward works as expected: True
```

1.2.3 : Train and Evaluate on the Clickbait dataset [5 Pts]

You must reach an accuracy of >= 85% to receive credit for this section..

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

# Use cuda if present
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Device available for running: ", device)
```

```
Device available for running:  cuda
```

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from lstm import LSTM
from tqdm import tqdm

NUM_CLASSES = 2

model = LSTM(vocab, num_classes=NUM_CLASSES)
model.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)
N_EPOCHS = 5 if not SHORT else 1

model.train()
for epoch in range(N_EPOCHS):
    total_loss = 0.0
    for X, X_len, Y in tqdm(train_loader):
        X = X.to(device)
        Y = Y.to(device)
        outputs = model(X, X_len)
        loss = criterion(outputs, Y)

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

        total_loss += loss.item()

    print("loss on epoch %i: %f" % (epoch, total_loss))
```

```
100%|██████████| 19/19 [00:01<00:00,  9.78it/s]
loss on epoch 0: 6.547718
```

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from sklearn.metrics import accuracy_score

with torch.no_grad():
    Y_truth, Y_preds = [],[]
    for X, X_len, Y in test_loader:
        X = X.to(device)
        outputs = model(X, X_len)

        Y_truth.append(Y)
        Y_preds.append(outputs)

    Y_truth = torch.cat(Y_truth)
    Y_preds = torch.cat(Y_preds)

print("Test Accuracy on Clickbait Dataset using LSTM  : {:.3f}".format(accuracy_score(Y_truth.cpu().detach().numpy(), F.softmax(Y_preds, dim=-1).argmax(d
```

```
Test Accuracy on Clickbait Dataset using LSTM  : 0.967
```

Submit these files to Gradescope

Run the cell below to save the state of your model. You will be required to upload the `best_lstm_model.pt` file that is saved in the `outputs folder` and the `lstm.py` file to Gradescope for accuracy evaluation. You must reach an accuracy of >= 85% on the above test dataset to receive credit for this section.

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

torch.save(model.state_dict(), './outputs/best_lstm_model.pt')
```

1.3 : Classifying Web of Science Dataset using LSTM [No Points]

Run the cells below to classify the Web of Science train and test datasets using the lstm functions that you have implemented. You should observe an accuracy of more than 40%

1.3.1 : Create the Dataloaders [No Points]

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

max_words = max(map(len, x_train_wos))

def vectorize_batch(batch):
    Y, X = list(zip(*batch))
    X = [vocab_wos(tokenizer(text)) for text in X] ## Tokenize and map tokens to indexes
    X_len = [len(text) for text in X]
    X = [tokens+([0]* (max_words-len(tokens))) if len(tokens)<max_words else tokens[:max_words] for tokens in X]
    return torch.tensor(X, dtype=torch.int32), torch.tensor(X_len), torch.tensor(Y)
```

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

train_dataset = list(map(lambda y, x: (y, x), y_train_wos, x_train_wos))
test_dataset = list(map(lambda y, x: (y, x), y_test_wos, x_test_wos))

train_loader = DataLoader(train_dataset, batch_size=128, collate_fn=vectorize_batch, shuffle=True)
test_loader  = DataLoader(test_dataset, batch_size=128, collate_fn=vectorize_batch)
```

1.3.2 Train and Evaluate on the Web of Science Dataset [No Points]

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from lstm import LSTM
from tqdm import tqdm

NUM_CLASSES = 4

model = LSTM(vocab_wos, num_classes=NUM_CLASSES)
model.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
N_EPOCHS = 20 if not SHORT else 1

model.train()
for epoch in range(N_EPOCHS):
    total_loss = 0.0
    for X, X_len, Y in tqdm(train_loader):
        X = X.to(device)
        Y = Y.to(device)
        outputs = model(X, X_len)
        loss = criterion(outputs, Y)

        optimizer.zero_grad()
        loss.backward()
```

```
        optimizer.step()

        total_loss += loss.item()

        if SHORT:
            break

    print("loss on epoch %i: %f" % (epoch, total_loss))
```

```
100%|██████████| 13/13 [00:02<00:00,  4.73it/s]
loss on epoch 0: 17.730795
100%|██████████| 13/13 [00:02<00:00,  6.23it/s]
loss on epoch 1: 17.067045
100%|██████████| 13/13 [00:02<00:00,  6.13it/s]
loss on epoch 2: 16.460056
100%|██████████| 13/13 [00:02<00:00,  6.49it/s]
loss on epoch 3: 15.760631
100%|██████████| 13/13 [00:02<00:00,  6.43it/s]
loss on epoch 4: 15.091935
100%|██████████| 13/13 [00:02<00:00,  6.12it/s]
loss on epoch 5: 14.114826
100%|██████████| 13/13 [00:02<00:00,  4.59it/s]
loss on epoch 6: 13.049757
100%|██████████| 13/13 [00:02<00:00,  4.45it/s]
loss on epoch 7: 11.780611
100%|██████████| 13/13 [00:02<00:00,  5.32it/s]
loss on epoch 8: 10.286781
100%|██████████| 13/13 [00:02<00:00,  6.14it/s]
loss on epoch 9: 9.146650
100%|██████████| 13/13 [00:02<00:00,  6.31it/s]
loss on epoch 10: 8.352035
100%|██████████| 13/13 [00:02<00:00,  6.33it/s]
loss on epoch 11: 7.511066
100%|██████████| 13/13 [00:02<00:00,  6.26it/s]
loss on epoch 12: 6.068389
100%|██████████| 13/13 [00:02<00:00,  5.31it/s]
loss on epoch 13: 5.045795
100%|██████████| 13/13 [00:02<00:00,  4.49it/s]
loss on epoch 14: 4.757475
100%|██████████| 13/13 [00:02<00:00,  4.77it/s]
loss on epoch 15: 4.005289
100%|██████████| 13/13 [00:02<00:00,  5.78it/s]
loss on epoch 16: 3.193915
100%|██████████| 13/13 [00:02<00:00,  6.38it/s]
loss on epoch 17: 2.588529
100%|██████████| 13/13 [00:02<00:00,  4.41it/s]
loss on epoch 18: 2.277907
100%|██████████| 13/13 [00:02<00:00,  4.61it/s]loss on epoch 19: 1.821465
```

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from sklearn.metrics import accuracy_score

with torch.no_grad():
    Y_truth, Y_preds = [],[]
    for X, X_len, Y in test_loader:
        X = X.to(device)
        outputs = model(X, X_len)

        Y_truth.append(Y)
        Y_preds.append(outputs)

        if SHORT:
            break

    Y_truth = torch.cat(Y_truth)
    Y_preds = torch.cat(Y_preds)

print("Test Accuracy on WoS Dataset using LSTM  : {:.3f}".format(accuracy_score(Y_truth.cpu().detach().numpy(), F.softmax(Y_preds, dim=-1).argmax(dim=-1).cpu().detach().numpy())))
```

```
Test Accuracy on WoS Dataset using LSTM  : 0.562
```

NOTE : LSTM alone is not able to perform good on the WoS dataset and that can be attributed to the very limited data with large vocabulary and lack of embedding structure.

Q2: Classification with LSTM + Attention [15pts]

A potential issue with a vanilla LSTM approach is that a neural network needs to be able to compress all of the necessary information of a source sentence into a fixed-length vector. This may make it difficult for the neural network to cope with long sentences, especially those that are longer than the sentences in the training corpus. The Attention mechanism helps to look at all of the hidden states from the sequence for making predictions unlike the vanilla approach.

In this task, we will be implementing LSTM with Attention.

Please refer to lecture for more details.

You will be extending the LSTM model and incorporating attention on top of it.

We will then classify the Clickbait and Web of science dataset for this task.

2.1 : Implementing the LSTM + Attention Model [10 Points]

In the **attention.py** file complete the following functions:

- **__init__**
- **forward_lstm**
- **forward_attention**
- **forward_context**
- **forward**

We have included local tests in 2.2.2 for you to test your implementation

2.2 : Classifying Clickbait Dataset using LSTM with Attention [No Points]

Run the cells below to classify the Clickbait train and test datasets using the attention functions that you have implemented. You should observe an accuracy of more than 88%.

2.2.1 : Create the Dataloaders [No Points]

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

max_words = max(map(len, x_train))

def vectorize_batch(batch):
    Y, X = list(zip(*batch))
    X = [vocab(tokenizer(text)) for text in X] ## Tokenize and map tokens to indexes
    X_len = [len(text) for text in X]
    X = [tokens+([0]* (max_words-len(tokens))) if len(tokens)<max_words else tokens[:max_words] for tokens in X] ## Bringing all samples to max_words length.
    return torch.tensor(X, dtype=torch.int32), torch.tensor(X_len), torch.tensor(Y)
```

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

train_dataset = list(map(lambda y, x: (y, x), y_train, x_train))
test_dataset = list(map(lambda y, x: (y, x), y_test, x_test))

train_loader = DataLoader(train_dataset, batch_size=1024, collate_fn=vectorize_batch, shuffle=True)
test_loader  = DataLoader(test_dataset, batch_size=1024, collate_fn=vectorize_batch)
```

2.2.2 : Local Tests for LSTM with Attention Functions [No Points]

You may test your implementation of the LSTM with Attention functions contained in **attention.py** in the cell below. Feel free to comment out tests for functions that have not been completed yet. See Using the Local Tests for more details.

```
# ##############################
# ### DO NOT CHANGE THIS CELL ###
# ##############################

from attention import Attention
from local_tests.attention_test import Attention_Test
torch.manual_seed(10)

local_test = Attention_Test()
attention_model = Attention(vocab, num_classes=2)
attention_model.load_state_dict(torch.load('./local_tests/basic_attention_model.pt'))   # upload default weights. If this errors out, make sure you have initialized the layers correct
attention_model.eval()

# creating a vectorized batch from the sample datapoints
local_test_dataset = list(map(lambda y, x: (y, x), local_test.output_labels, local_test.input_sequences))
X_localtest, X_len_localtest, _ = vectorize_batch(local_test_dataset)
# print(X_localtest.shape, X_len_localtest)

print('Local Tests for LSTM + Attention Implementation \n')

# Local test for forward_lstm
output, hidden = attention_model.forward_lstm(X_localtest, X_len_localtest)
forward_lstm_shape_test = (output.shape == local_test.lstm_output.shape) and (hidden.shape == local_test.lstm_hidden.shape)
forward_lstm_value_test = (torch.allclose(output, local_test.lstm_output, rtol=0.001, atol=0.001)) and (torch.allclose(hidden, local_test.lstm_hidden, rtol=0.0001, atol=0.0001))
print('Your forward_lstm works as expected: ', (forward_lstm_shape_test and forward_lstm_value_test))

# Local test for forward_attention
attn_weights = attention_model.forward_attention(local_test.lstm_output, local_test.lstm_hidden)
forward_attention_test = (attn_weights.shape == local_test.attention_weights.shape) and (torch.allclose(attn_weights, local_test.attention_weights, rtol=0.0001, atol=0.0001))
print('Your forward_attention works as expected: ', forward_attention_test)

# Local test for forward_context
context_output = attention_model.forward_context(local_test.lstm_output, local_test.attention_weights, local_test.lstm_hidden)
forward_context_test = (context_output.shape == local_test.context_output.shape) and (torch.allclose(context_output, local_test.context_output, rtol=0.0001, atol=0.0001))
print('Your forward_context works as expected: ', forward_context_test)

# Local test for forward
output = attention_model.forward(X_localtest, X_len_localtest)
forward_test = (output.shape == local_test.output.shape) and (torch.allclose(output, local_test.output, rtol=0.0001, atol=0.0001))
print('Your forward works as expected: ', forward_test)
```

```
Local Tests for LSTM + Attention Implementation

Your forward_lstm works as expected:  True
Your forward_attention works as expected:  True
Your forward_context works as expected:  True
Your forward works as expected:  True
```

2.2.3: Train and Evaluate on the Clickbait Dataset [No Points]

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

# Use cuda if present
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Device available for running: ", device)
```

```
Device available for running:  cuda
```

```
##############################
### DO NOT CHANGE THIS CELL ###
```

```
##############################

from attention import Attention
from tqdm import tqdm

NUM_CLASSES = 2

model = Attention(vocab, num_classes=NUM_CLASSES)
model.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)
N_EPOCHS = 5 if not SHORT else 1

model.train()
for epoch in range(N_EPOCHS):
    total_loss = 0.0
    for X, X_len, Y in tqdm(train_loader):
        X = X.to(device)
        Y = Y.to(device)
        outputs = model(X, X_len)
        loss = criterion(outputs, Y)

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

        total_loss += loss.item()

    print("loss on epoch %i: %f" % (epoch, total_loss))
```

```
100%|██████████| 19/19 [00:01<00:00, 14.02it/s]
loss on epoch 0: 5.596366
100%|██████████| 19/19 [00:01<00:00, 14.65it/s]
loss on epoch 1: 1.509814
100%|██████████| 19/19 [00:01<00:00, 13.38it/s]
loss on epoch 2: 0.468158
100%|██████████| 19/19 [00:01<00:00, 12.86it/s]
loss on epoch 3: 0.147131
100%|██████████| 19/19 [00:02<00:00,  7.43it/s]loss on epoch 4: 0.031196
```

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from sklearn.metrics import accuracy_score

with torch.no_grad():
    Y_truth, Y_preds = [],[]
    for X, X_len, Y in test_loader:
        X = X.to(device)
        outputs = model(X, X_len)

        Y_truth.append(Y)
        Y_preds.append(outputs)

    Y_truth = torch.cat(Y_truth)
    Y_preds = torch.cat(Y_preds)

print("Test Accuracy on Clickbait Dataset using LSTM with Attention  : {:.3f}".format(accuracy_score(Y_truth.cpu().detach().numpy(), F.softmax(Y_preds, dim=-1).argmax(dim=-1).cpu().de
```

```
Test Accuracy on Clickbait Dataset using LSTM with Attention  : 0.957
```

2.3 : Classifying Web of Science Dataset using LSTM with Attention [5 Pts]

Run the cells below to classify the Web of Science train and test datasets using the attention functions that you have implemented.

You must reach an accuracy of >=50% to receive credit for this section.

2.3.1 : Create the Dataloaders [No Points]

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

max_words = max(map(len, x_train_wos))

def vectorize_batch(batch):
    Y, X = list(zip(*batch))
    X = [vocab_wos(tokenizer(text)) for text in X] ## Tokenize and map tokens to indexes
    X_len = [len(text) for text in X]
    X = [tokens+([0]* (max_words-len(tokens))) if len(tokens)<max_words else tokens[:max_words] for tokens in X] ## Bringing all samples to max_words length.
    return torch.tensor(X, dtype=torch.int32), torch.tensor(X_len), torch.tensor(Y)
```

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

train_dataset = list(map(lambda y, x: (y, x), y_train_wos, x_train_wos))
test_dataset = list(map(lambda y, x: (y, x), y_test_wos, x_test_wos))

train_loader = DataLoader(train_dataset, batch_size=128, collate_fn=vectorize_batch, shuffle=True)
test_loader  = DataLoader(test_dataset, batch_size=128, collate_fn=vectorize_batch)
```

2.3.2 : Train and Evaluate on the Web of Science dataset [5 Pts]

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

```

```
# Use cuda if present
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Device available for running: ", device)
```

```
Device available for running:  cuda
```

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from attention import Attention
from tqdm import tqdm

NUM_CLASSES = 4

model = Attention(vocab_wos, num_classes=NUM_CLASSES)
model.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
N_EPOCHS = 25 if not SHORT else 1

model.train()
for epoch in range(N_EPOCHS):
    total_loss = 0.0
    for X, X_len, Y in tqdm(train_loader):
        X = X.to(device)
        Y = Y.to(device)
        outputs = model(X, X_len)
        loss = criterion(outputs, Y)

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

        total_loss += loss.item()

        if SHORT:
            break

    print("loss on epoch %i: %f" % (epoch, total_loss))
```

```
100%|██████████| 13/13 [00:02<00:00,  6.25it/s]
loss on epoch 0: 17.639267
100%|██████████| 13/13 [00:02<00:00,  6.39it/s]
loss on epoch 1: 16.107268
100%|██████████| 13/13 [00:02<00:00,  6.20it/s]
loss on epoch 2: 14.302524
100%|██████████| 13/13 [00:01<00:00,  6.53it/s]
loss on epoch 3: 12.863941
100%|██████████| 13/13 [00:02<00:00,  4.74it/s]
loss on epoch 4: 11.127718
100%|██████████| 13/13 [00:02<00:00,  4.48it/s]
loss on epoch 5: 9.498385
100%|██████████| 13/13 [00:02<00:00,  4.60it/s]
loss on epoch 6: 8.060333
100%|██████████| 13/13 [00:02<00:00,  6.19it/s]
loss on epoch 7: 6.614249
100%|██████████| 13/13 [00:02<00:00,  6.43it/s]
loss on epoch 8: 5.804770
100%|██████████| 13/13 [00:02<00:00,  6.44it/s]
loss on epoch 9: 4.907240
100%|██████████| 13/13 [00:02<00:00,  6.36it/s]
loss on epoch 10: 3.515091
100%|██████████| 13/13 [00:02<00:00,  5.93it/s]
loss on epoch 11: 2.663215
100%|██████████| 13/13 [00:02<00:00,  4.58it/s]
loss on epoch 12: 1.798664
100%|██████████| 13/13 [00:02<00:00,  4.67it/s]
loss on epoch 13: 1.200081
100%|██████████| 13/13 [00:02<00:00,  5.31it/s]
loss on epoch 14: 0.735106
100%|██████████| 13/13 [00:02<00:00,  6.38it/s]
loss on epoch 15: 0.411879
100%|██████████| 13/13 [00:02<00:00,  6.40it/s]
loss on epoch 16: 0.252141
100%|██████████| 13/13 [00:02<00:00,  6.31it/s]
loss on epoch 17: 0.199521
100%|██████████| 13/13 [00:02<00:00,  6.27it/s]
loss on epoch 18: 0.141240
100%|██████████| 13/13 [00:02<00:00,  5.11it/s]
loss on epoch 19: 0.130441
100%|██████████| 13/13 [00:02<00:00,  4.68it/s]
loss on epoch 20: 0.075788
100%|██████████| 13/13 [00:02<00:00,  4.78it/s]
loss on epoch 21: 0.046381
100%|██████████| 13/13 [00:02<00:00,  5.93it/s]
loss on epoch 22: 0.032549
100%|██████████| 13/13 [00:02<00:00,  6.36it/s]
loss on epoch 23: 0.024646
100%|██████████| 13/13 [00:02<00:00,  6.44it/s]loss on epoch 24: 0.021094
```

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from sklearn.metrics import accuracy_score

with torch.no_grad():
    Y_truth, Y_preds = [],[]
    for X, X_len, Y in test_loader:
        X = X.to(device)
        outputs = model(X, X_len)

        Y_truth.append(Y)
        Y_preds.append(outputs)

        if SHORT:
            break

    Y_truth = torch.cat(Y_truth)
    Y_preds = torch.cat(Y_preds)
```

```
print("Test Accuracy on WoS Dataset using LSTM with Attention : {:.3f}".format(accuracy_score(Y_truth.cpu().detach().numpy(), F.softmax(Y_preds, dim=-1).argmax(dim=-1).cpu().detach().
```

```
Test Accuracy on WoS Dataset using LSTM with Attention : 0.547
```

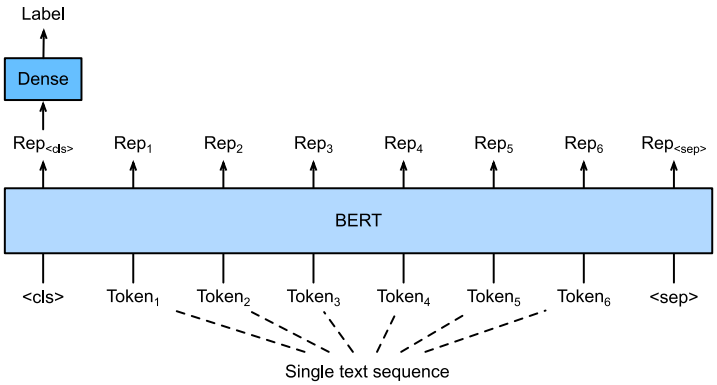
Submit these files to Gradescope

Run the cell below to save the state of your model. You will be required to upload the `best_attention_model.pt` file that is saved in the `outputs folder` and the `attention.py` file to Gradescope for accuracy evaluation. You must reach an accuracy of >= 50% on the above test dataset to receive credit for this section.

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

torch.save(model.state_dict(), './outputs/best_attention_model.pt')
```

Q3: Classification with BERT [15pts]

The transformer neural network is a novel architecture that aims to solve sequence-to-sequence tasks while handling long-range dependencies with ease. In a transformer, we can pass all the words of a sentence and determine the word embedding simultaneously.



We will be using BERT (Bidirectional Encoder Representations from Transformers) pre-trained models for embeddings. BERT architecture consists of several Transformer encoders stacked together. Each Transformer encoder encapsulates two sub-layers: a self-attention layer and a feed-forward layer.

The details on BERT can be referred from the paper : BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.

We will be using the BERT embeddings and a fully connected linear layer to perform classification.

We will then classify the Clickbait and Web of science dataset for this task.

3.1 : Implementing the BERT Classifier [10pts]

In the **bert.py** file complete the following functions:

- **__init__**
- **forward**

3.2: Local Tests for BERT Functions [No Points]

You may test your implementation of the BERT functions contained in **bert.py** in the cell below. Feel free to comment out tests for functions that have not been completed yet. See Using the Local Tests for more details.

```
# ##############################
# ### DO NOT CHANGE THIS CELL ###
# ##############################

from bert import BERTClassifier
from local_tests.bert_test import BERT_Test

local_test = BERT_Test()

print('Local Tests for BERT Functions \n')

# Instantiate BERTClassifier and load linear weights for local tests
torch.manual_seed(10)
student_bert = BERTClassifier(2)
basic_bert_linear = torch.load('./local_tests/basic_bert_linear.pt')

# If loading the state dict causes an error, then there may be an issue with your layer name or shape
student_bert.linear.load_state_dict(basic_bert_linear)

# Local test for forward
student_bert.eval()
outputs = student_bert(local_test.inputs, local_test.masks)
forward_test = (torch.allclose(outputs, local_test.outputs, rtol=0.0001, atol=0.0001) and (outputs.shape == local_test.outputs.shape))
print('Your forward works as expected:', forward_test)
```

```
Local Tests for BERT Functions

Downloading (…)lve/main/config.json: 100%                    570/570 [00:00<00:00, 21.5kB/s]

Downloading model.safetensors: 100%                          440M/440M [00:03<00:00, 81.0MB/s]

Your forward works as expected: True
```

3.3 : Initialize Tokenizer [No Points]

Run the below cells to initalizer tokenizer for the pretrained BERT model.

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from transformers import BertTokenizer

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

```
Downloading (…)okenizer_config.json: 100%  28.0/28.0 [00:00<00:00, 1.24kB/s]
Downloading (…)solve/main/vocab.txt: 100%  232k/232k [00:00<00:00, 3.98MB/s]
Downloading (…)/main/tokenizer.json: 100%  466k/466k [00:00<00:00, 9.81MB/s]
```

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

def vectorize_batch(batch):
    Y, X = list(zip(*batch))
    X = tokenizer(X, padding='max_length', max_length = 128, truncation=True, return_tensors="pt")
    input_ids, attention_mask = X['input_ids'], X['attention_mask']
    return input_ids, attention_mask, torch.tensor(Y)
```

3.4 : Classifying Clickbait Dataset using BERT [5pts]

Run the below cell to classify the Clickbait train and test dataset using the bert functions that you have already implemented in 3.

You must reach an accuracy of >= 90% to receive credit for this section.

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

train_dataset = list(map(lambda y, x: (y, x), y_train, x_train))
test_dataset = list(map(lambda y, x: (y, x), y_test, x_test))

train_loader = DataLoader(train_dataset, batch_size=32, collate_fn=vectorize_batch, shuffle=True)
test_loader  = DataLoader(test_dataset, batch_size=32, collate_fn=vectorize_batch)
```

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

# Use cuda if present
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Device available for running: ", device)
```

```
Device available for running:  cuda
```

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from bert import BERTClassifier
from tqdm import tqdm

NUM_CLASSES = 2

model = BERTClassifier(num_classes=NUM_CLASSES)
model.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001)
N_EPOCHS = 3 if not SHORT else 1

model.train()
for epoch in range(N_EPOCHS):
    total_loss = 0.0
    for X, X_mask, Y in tqdm(train_loader):
        X = X.to(device)
        X_mask = X_mask.to(device)
        Y = Y.to(device)
        outputs = model(X, X_mask)

        loss = criterion(outputs, Y)

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

        total_loss += loss.item()

        if SHORT:
            break

    print("loss on epoch %i: %f" % (epoch, total_loss))
```

```
100%|██████████| 600/600 [06:23<00:00,  1.57it/s]
loss on epoch 0: 45.047180
100%|██████████| 600/600 [06:30<00:00,  1.54it/s]
loss on epoch 1: 18.853489
100%|██████████| 600/600 [06:30<00:00,  1.54it/s]loss on epoch 2: 11.036154
```

```
##############################
### DO NOT CHANGE THIS CELL ###
```

```
##############################

from sklearn.metrics import accuracy_score

with torch.no_grad():
    Y_truth, Y_preds = [],[]
    for X, X_mask, Y in test_loader:
        X = X.to(device)
        X_mask = X_mask.to(device)
        outputs = model(X, X_mask)

        Y_truth.append(Y)
        Y_preds.append(outputs)

        if SHORT:
            break

    Y_truth = torch.cat(Y_truth)
    Y_preds = torch.cat(Y_preds)

print("Test Accuracy on Clickbait Dataset using BERT : {:.3f}".format(accuracy_score(Y_truth.cpu().detach().numpy(), F.softmax(Y_preds, dim=-1).argmax(dim=-1).cpu().detach().numpy()))
```

```
Test Accuracy on Clickbait Dataset using BERT : 0.985
```

Submit this file to Gradescope

Run the cell below to save the results of your model. You will be required to upload the `bert_clickbait.pkl` file that is saved in the `outputs folder` to Gradescope for accuracy evaluation. You must reach an accuracy of >= 90% on the above test dataset to receive credit for this section.

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

preds = F.softmax(Y_preds, dim=-1).argmax(dim=-1).cpu().detach().numpy()

with open('./outputs/bert_clickbait.pkl', 'wb') as fp:
    pickle.dump(preds, fp)
```

3.5 : Classifying Web of Science Dataset using BERT [5pts]

Run the below cell to classify the Web of Science train and test dataset using the bert functions that you have already implemented in 3.

You must reach an accuracy of >= 70% to receive credit for this section.

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

train_dataset = list(map(lambda y, x: (y, x), y_train_wos, x_train_wos))
test_dataset = list(map(lambda y, x: (y, x), y_test_wos, x_test_wos))

train_loader = DataLoader(train_dataset, batch_size=32, collate_fn=vectorize_batch, shuffle=True)
test_loader  = DataLoader(test_dataset, batch_size=32, collate_fn=vectorize_batch)
```

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

# Use cuda if present
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Device available for running: ", device)
```

```
Device available for running:  cuda
```

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from bert import BERTClassifier
from tqdm import tqdm

NUM_CLASSES = 4

model = BERTClassifier(num_classes=NUM_CLASSES)
model.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001)
N_EPOCHS = 3 if not SHORT else 1

model.train()
for epoch in range(N_EPOCHS):
    total_loss = 0.0
    for X, X_mask, Y in tqdm(train_loader):
        X = X.to(device)
        X_mask = X_mask.to(device)
        Y = Y.to(device)
        outputs = model(X, X_mask)

        loss = criterion(outputs, Y)

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

        total_loss += loss.item()

        if SHORT:
            break

    print("loss on epoch %i: %f" % (epoch, total_loss))
```

```
100%|██████████| 50/50 [00:42<00:00,  1.18it/s]
loss on epoch 0: 44.441428
100%|██████████| 50/50 [00:43<00:00,  1.16it/s]
loss on epoch 1: 22.020269
100%|██████████| 50/50 [00:42<00:00,  1.19it/s]loss on epoch 2: 11.977023
```

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from sklearn.metrics import accuracy_score

with torch.no_grad():
    Y_truth, Y_preds = [],[]
    for X, X_mask, Y in test_loader:
        X = X.to(device)
        X_mask = X_mask.to(device)
        outputs = model(X, X_mask)

        Y_truth.append(Y)
        Y_preds.append(outputs)

        if SHORT:
            break

    Y_truth = torch.cat(Y_truth)
    Y_preds = torch.cat(Y_preds)

print("Test Accuracy on Web of Science Dataset using BERT : {:.3f}".format(accuracy_score(Y_truth.cpu().detach().numpy(), F.softmax(Y_preds, dim=-1).argmax(dim=-1).cpu().detach().numpy
```

```
Test Accuracy on Web of Science Dataset using BERT : 0.848
```

Submit this file to Gradescope

Run the cell below to save the results of your model. You will be required to upload the `bert_wos.pkl` file that is saved in the `outputs folder` to Gradescope for accuracy evaluation. You must reach an accuracy of >= 70% on the above test dataset to receive credit for this section.

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

preds = F.softmax(Y_preds, dim=-1).argmax(dim=-1).cpu().detach().numpy()

with open('./outputs/bert_wos.pkl', 'wb') as fp:
    pickle.dump(preds, fp)
```

Q4: Sequence Labeling [15pts]

Part-of-speech (POS) tagging is a popular Natural Language Processing process which refers to categorizing words in a text (corpus) in correspondence with a particular part of speech, depending on the definition of the word and its context.

Named entity recognition (NER) seeks to locate and classify named entities in text into pre-defined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc.

We will be using BERT (Bidirectional Encoder Representations from Transformers) for sequence labeling. The architecture of the model is shown above in the diagram.

We will be using the BERT embeddings and a fully connected linear layer to perform classification.

We will then classify using the conll2003 dataset for this task.

4.1 : Implementing Sequence Labeling [5pts]

In the **sequenceLabeling.py** file complete the following functions:

- **__init__**
- **forward**

4.2 : Local Tests for Sequence Labeling Functions [No Points]

You may test your implementation of the Sequence Labeling functions contained in **sequenceLabeling.py** in the cell below. Feel free to comment out tests for functions that have not been completed yet. See Using the Local Tests for more details.

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from sequenceLabeling import SequenceLabeling
from local_tests.sequenceLabeling_test import SequenceLabeling_Test

local_test = SequenceLabeling_Test()

print('Local Tests for Sequence Labeling Functions \n')

# Instantiate SequenceLabeling and load linear weights for local tests
torch.manual_seed(10)
student_seq = SequenceLabeling(9)
basic_seq_linear = torch.load('./local_tests/basic_seq_linear.pt')

# If loading the state dict causes an error, then there may be an issue with your layer name or shape
student_seq.linear.load_state_dict(basic_seq_linear)

# Local test for forward
student_seq.eval()
outputs = student_seq(local_test.input_ids, local_test.masks, local_test.token_type_ids)
forward_test = torch.allclose(outputs, local_test.outputs, rtol=0.0001, atol=0.0001)
print('Your forward outputs the expected shape:', outputs.shape == local_test.outputs.shape)
print('Your forward outputs the expected values:', forward_test)
if not forward_test:
    print('If you forward values do not match the expected values but you are matching in shape and are getting the desired scores in 4.5, then no need to worry about the discrepancy.
```

```
Local Tests for Sequence Labeling Functions

Your forward outputs the expected shape: True
Your forward outputs the expected values: True
```

4.3 : Loading Dataset & Tokenizer [No Points]

Run the below cell to download the conll2003 dataset. Each word in the dataset has been put on a separate line and there is an empty line after each sentence. The first item on each line is a word, the second a part-of-speech (POS) tag, the third a syntactic chunk tag and the fourth the named entity tag. The chunk tags and the named entity tags have the format I-TYPE which means that the word is inside a phrase of type TYPE.

For more details on the structure and the supported labels, please refer : https://huggingface.co/datasets/conll2003.

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from datasets import load_dataset
from transformers import BertTokenizer

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
dataset = load_dataset("conll2003")

train = dataset['train']
val = dataset['validation']
test = dataset['test']
```

```
Downloading builder script: 100%  9.57k/9.57k [00:00<00:00, 619kB/s]
Downloading metadata: 100%  3.73k/3.73k [00:00<00:00, 225kB/s]
Downloading readme: 100%  12.3k/12.3k [00:00<00:00, 598kB/s]
Downloading data: 100%  983k/983k [00:00<00:00, 2.08MB/s]
Generating train split: 100%  14041/14041 [00:05<00:00, 2422.23 examples/s]
Generating validation split: 100%  3250/3250 [00:01<00:00, 1985.82 examples/s]
Generating test split: 100%  3453/3453 [00:00<00:00, 4949.93 examples/s]
```

4.4 : Pre-process Dataset [No Points]

Run the below cells to vectorize the dataset for tokenizers and to intialize the dataloaders.

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

### The code is used to vectorize the data for each batch as defined in the data loader.
### For data in the batch, the input tokens are encoded and
### then special tokens [CLS] (101) is added at the beginning and [SEP] (102) is added at the end.
### These tokens are added because the BERT model was pretrained with these tokens. So to get the same results for inference we need to add them.
### The input is padded with 0 if it is lesser than Max length.

MAX_LEN = 128

def vectorize_batch(batch):
    batch_input_ids = []
    batch_mask = []
    batch_token_type_ids = []
    batch_pos = []
    batch_ner = []

    for data in batch:
        target_pos = []
```

```
        target_ner = []
        inputs = []
        tokens = data['tokens']
        pos_tags = data['pos_tags']
        ner_tags = data['ner_tags']
        for i in range(len(tokens)):
            input = tokenizer.encode(tokens[i], add_special_tokens=False)
            input_len = len(input)
            target_pos.extend([pos_tags[i]] * input_len)
            target_ner.extend([ner_tags[i]] * input_len)
            inputs.extend(input)
        inputs = inputs[:MAX_LEN - 2]
        target_pos = target_pos[:MAX_LEN - 2]
        target_ner = target_ner[:MAX_LEN - 2]

        inputs = [101] + inputs + [102]
        target_pos = [0] + target_pos + [0]
        target_ner = [0] + target_ner + [0]

        mask = [1] * len(inputs)
        token_type_ids = [0] * len(inputs)

        padding_len = MAX_LEN - len(inputs)
        inputs = inputs + ([0] * padding_len)
        mask = mask + ([0] * padding_len)

        token_type_ids = token_type_ids + ([0] * padding_len)
        target_pos = target_pos + ([0] * padding_len)
        target_ner = target_ner + ([0] * padding_len)

        batch_input_ids.append(inputs)
        batch_mask.append(mask)
        batch_token_type_ids.append(token_type_ids)
        batch_pos.append(target_pos)
        batch_ner.append(target_ner)

    return torch.tensor(batch_input_ids, dtype=torch.long), torch.tensor(batch_mask, dtype=torch.long), torch.tensor(batch_token_type_ids, dtype=torch.long), torch.tensor(batch_pos, d
```

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

train_loader = DataLoader(train, batch_size=8, collate_fn=vectorize_batch, shuffle=True)
test_loader  = DataLoader(test, batch_size=1, collate_fn=vectorize_batch)
```

4.5 POS Tagging and NER using CoNLL-2003 dataset [10pts]

Run the below cell to classify the conll2003 dataset for POS Tagging and NER using the sequenceLabeling functions that you have already implemented in 4.

You must reach an F1-score of >= 90% in both tasks to receive credit for this section.

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

# Use cuda if present
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Device available for running: ", device)
```

```
Device available for running:  cuda
```

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from sequenceLabeling import SequenceLabeling
from tqdm import tqdm

NUM_CLASSES_POS = 47
NUM_CLASSES_NER = 9

model_pos = SequenceLabeling(NUM_CLASSES_POS)
model_pos.to(device)

model_ner = SequenceLabeling(NUM_CLASSES_NER)
model_ner.to(device)

criterion = nn.CrossEntropyLoss()
optimizer_pos = optim.Adam(model_pos.parameters(), lr=0.0001)
optimizer_ner = optim.Adam(model_ner.parameters(), lr=0.0001)
N_EPOCHS = 3 if not SHORT else 1

model_pos.train()
model_ner.train()

for epoch in range(N_EPOCHS):
    total_loss_pos = 0.0
    total_loss_ner = 0.0
    for input_ids, mask, token_type_ids, target_pos, target_ner in tqdm(train_loader):
        input_ids = input_ids.to(device)
        mask = mask.to(device)
        token_type_ids = token_type_ids.to(device)
        target_pos = target_pos.to(device)
        target_ner = target_ner.to(device)

        outputs_pos = model_pos(input_ids, mask, token_type_ids)
        outputs_ner = model_ner(input_ids, mask, token_type_ids)

        active_loss_pos = mask.view(-1) == 1
        active_logits_pos = outputs_pos.view(-1, NUM_CLASSES_POS)

        active_labels_pos = torch.where(
```

```
            active_loss_pos,
            target_pos.view(-1),
            torch.tensor(criterion.ignore_index).type_as(target_pos)
        )

        loss_pos = criterion(active_logits_pos, active_labels_pos)

        optimizer_pos.zero_grad()
        loss_pos.backward()
        optimizer_pos.step()

        active_loss_ner = mask.view(-1) == 1
        active_logits_ner = outputs_ner.view(-1, NUM_CLASSES_NER)

        active_labels_ner = torch.where(
            active_loss_ner,
            target_ner.view(-1),
            torch.tensor(criterion.ignore_index).type_as(target_ner)
        )

        loss_ner = criterion(active_logits_ner, active_labels_ner)

        optimizer_ner.zero_grad()
        loss_ner.backward()
        optimizer_ner.step()

        total_loss_pos += loss_pos.item()
        total_loss_ner += loss_ner.item()

        if SHORT:
            break

    print("POS Tagging loss on epoch %i: %f" % (epoch, total_loss_pos))
    print("NER loss on epoch %i: %f" % (epoch, total_loss_pos))
```

```
100%|██████████| 1756/1756 [11:40<00:00,  2.51it/s]
POS Tagging loss on epoch 0: 583.109677
NER loss on epoch 0: 583.109677
100%|██████████| 1756/1756 [11:42<00:00,  2.50it/s]
POS Tagging loss on epoch 1: 334.972757
NER loss on epoch 1: 334.972757
100%|██████████| 1756/1756 [11:48<00:00,  2.48it/s]POS Tagging loss on epoch 2: 284.867697
NER loss on epoch 2: 284.867697
```

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

with torch.no_grad():
    y_true_pos = []
    y_pred_pos = []
    y_true_ner = []
    y_pred_ner = []
    for input_ids, mask, token_type_ids, target_pos, target_ner in tqdm(test_loader):
        input_ids = input_ids.to(device)
        mask = mask.to(device)
        token_type_ids = token_type_ids.to(device)
        target_pos = target_pos.to(device).view(-1)
        target_ner = target_ner.to(device).view(-1)

        outputs_pos = model_pos(input_ids, mask, token_type_ids)
        predicted_pos = torch.argmax(outputs_pos, dim=-1)

        outputs_ner = model_ner(input_ids, mask, token_type_ids)
        predicted_ner = torch.argmax(outputs_ner, dim=-1)

        active_loss = mask == 1
        active_loss = active_loss.view(-1)
        predicted_pos = predicted_pos.view(-1)
        predicted_ner = predicted_ner.view(-1)

        for i in range(len(active_loss)):
            if not active_loss[i]:
                break
            y_true_pos.append(target_pos[i].cpu().detach().numpy())
            y_pred_pos.append(predicted_pos[i].cpu().detach().numpy())
            y_true_ner.append(target_ner[i].cpu().detach().numpy())
            y_pred_ner.append(predicted_ner[i].cpu().detach().numpy())

        if SHORT:
            break
from sklearn.metrics import f1_score

print("Test F1-score on Conll2003 Dataset for POS Tagging : {:.3f}".format(f1_score(y_true_pos, y_pred_pos, average='micro')))
print("Test F1-score on Conll2003 Dataset for NER : {:.3f}".format(f1_score(y_true_ner, y_pred_ner, average='micro')))
```

```
100%|██████████| 3453/3453 [01:58<00:00, 29.26it/s]
Test F1-score on Conll2003 Dataset for POS Tagging : 0.920
Test F1-score on Conll2003 Dataset for NER : 0.967
```

Submit these files to Gradescope

Run the cell below to save the results of your model. You will be required to upload the `bert_pos.pkl` and `bert_ner.pkl` files that are saved in the `outputs folder` to Gradescope for accuracy evaluation. You must reach an accuracy of >= 90% on both tasks to receive credit for this section.

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

with open('./outputs/bert_pos.pkl', 'wb') as fp:
    pickle.dump(np.array(y_pred_pos), fp)

with open('./outputs/bert_ner.pkl', 'wb') as fp:
    pickle.dump(np.array(y_pred_ner), fp)
```

Q5: Topic Modeling [20pts]

Topic Models are a type of statistical language models used for uncovering hidden structure in a collection of texts. There are several existing algorithms you can use to perform the topic modeling. In this HW, we will be exploring LDA (Latent Dirichlet Allocation) method. For more details please refer to class lectures and slides.

5.1: Latent Dirichlet Allocation [20pts]

In the **lda.py** file complete the following functions:

- **tokenize_words**
- **remove_stopwords**
- **create_dictionary**
- **build_LDAModel**

Latent Dirichlet Allocation (LDA) is a generative statistical model that explains a set of observations through unobserved groups, and each group explains why some parts of the data are similar. The LDA is an example of a topic model. In this, observations (e.g., words) are collected into documents, and each word's presence is attributable to one of the document's topics. Each document will contain a small number of topics.

5.2: Local Tests for LDA Functions [No Points]

You may test your implementation of the LDA functions contained in **lda.py** in the cell below. Feel free to comment out tests for functions that have not been completed yet. See Using the Local Tests for more details.

```
# ##############################
# ### DO NOT CHANGE THIS CELL ###
# ##############################

from lda import LDA
from local_tests.lda_test import LDA_Test

local_test = LDA_Test()

sample_data = local_test.input_sequences
stop_words = stopwords.words('english')
lda = LDA()

print('Local Tests for LDA Functions \n')

# Local test for tokenize_words
output = list(lda.tokenize_words(sample_data))
tokens_test = (output == local_test.output_tokenized)
print('Your tokenize_words works as expected:', tokens_test)

# Local test for remove_stopwords
filtered_output = lda.remove_stopwords(local_test.output_tokenized, stop_words)
remove_stopwords_test = (filtered_output == local_test.output_stopword_removal)
print('Your remove_stopwords works as expected:', remove_stopwords_test)

# Local test for create_dictionary
id2word_test, corpus_test  = lda.create_dictionary(local_test.output_stopword_removal)
id2word_corpus_test = ((id2word_test == local_test.output_id2word_dictionary) and (corpus_test == local_test.output_sample_corpus))
print('Your create_dictionary works as expected:', id2word_corpus_test)

# Instantiate build_LDAModel
lda_model_build = lda.build_LDAModel(id2word_test, corpus_test)
lda_model_test = isinstance(lda_model_build, gensim.models.LdaMulticore)
print("Your build_LDAModel works as expected:", lda_model_test)
```

```
WARNING:gensim.models.ldamulticore:too few updates, training might not converge; consider increasing the number of passes or iterations to improve accuracy
Local Tests for LDA Functions

Your tokenize_words works as expected: True
Your remove_stopwords works as expected: True
Your create_dictionary works as expected: True
Your build_LDAModel works as expected: True
```

5.3: Topic Modeling on Clickbait dataset using LDA [No Points]

Run the below cell to build LDA Model and visualize topics using gensim library.

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from lda import LDA

stop_words = stopwords.words('english')
lda = LDA()

data = list(lda.tokenize_words(x_train))
data = lda.remove_stopwords(data, stop_words)
id2word, corpus = lda.create_dictionary(data)
lda_model = lda.build_LDAModel(id2word, corpus)
```

```
# Instantiate build_LDAModel
lda_model_build = lda.build_LDAModel(id2word_test, corpus_test)
lda_model_test = isinstance(lda_model_build, gensim.models.LdaMulticore)
print("Your build_LDAModel works as expected:", lda_model_test)
```

```
WARNING:gensim.models.ldamulticore:too few updates, training might not converge; consider increasing the number of passes or iterations to improve accuracy
Local Tests for LDA Functions

Your tokenize_words works as expected: True
Your remove_stopwords works as expected: True
Your create_dictionary works as expected: True
Your build_LDAModel works as expected: True
```

5.3: Topic Modeling on Clickbait dataset using LDA [No Points]

Run the below cell to build LDA Model and visualize topics using gensim library.

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from lda import LDA

stop_words = stopwords.words('english')
lda = LDA()

data = list(lda.tokenize_words(x_train))
data = lda.remove_stopwords(data, stop_words)
id2word, corpus = lda.create_dictionary(data)
lda_model = lda.build_LDAModel(id2word, corpus)
```

```
WARNING:gensim.models.ldamulticore:too few updates, training might not converge; consider increasing the number of passes or iterations to improve accuracy
```

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

# Visualize the topics
pyLDAvis.enable_notebook()

LDAvis_prepared = pyLDAvis.gensim_models.prepare(lda_model, corpus, id2word)
LDAvis_prepared
```