import pandas as pd

import torch

import transformers

from transformers import AutoTokenizer

from datasets import Dataset

import numpy as np

In [2]:

*#display(FileLink(f'model\_v\_2\_0\_ep\_1.pt'))*

In [3]:

import os

import subprocess

from IPython.display import FileLink, display

In [4]:

linkcode

import nltk

from torch.nn.functional import softmax

*# Ensure you have the punkt tokenizer*

nltk.download('punkt')

pip install evaluate

real\_articles\_link = '/kaggle/input/fake-and-real-news-dataset/True.csv'

fake\_articles\_link = '/kaggle/input/fake-and-real-news-dataset/Fake.csv'

In [7]:

real\_df = pd.read\_csv(real\_articles\_link)

fake\_df = pd.read\_csv(fake\_articles\_link)

In [8]:

linkcode

real\_df

def remove\_mag(text):

text\_ls = '-'.join(text.split('-')[1:])[1:]

return text\_ls

In [10]:

real\_df['text'] = real\_df['text'].apply(remove\_mag)

In [11]:

linkcode

fake\_df.iloc[10000]['text']

*#remove\_mag(real\_df.iloc[1]['text'])*

In [13]:

linkcode

real\_df.iloc[10000]['text']

fake\_df

fake\_df.iloc[20]['text']

real\_df['label'] = 0

fake\_df['label'] = 1

df = pd.concat([real\_df, fake\_df])

df\_shuffled = df.sample(frac=1).reset\_index(drop=True)*#[:1000]*

*#real\_ds = Dataset.from\_pandas(df\_shuffled)*

In [17]:

*#real\_ds*

In [18]:

*# import torch*

*# import torch.nn as nn*

*# from transformers import BertModel, BertTokenizer*

*# class FakeNewsClassifier(nn.Module):*

*# def \_\_init\_\_(self, bert\_model\_name="bert-base-uncased"):*

*# super(FakeNewsClassifier, self).\_\_init\_\_()*

*# # BERT model for sentence embeddings*

*# self.bert = BertModel.from\_pretrained(bert\_model\_name)*

*# # Attention mechanism*

*# self.attention = nn.Sequential(*

*# nn.Linear(self.bert.config.hidden\_size, 128),*

*# nn.Tanh(),*

*# nn.Linear(128, 1),*

*# nn.Softmax(dim=1)*

*# )*

*# # Classifier*

*# self.classifier = nn.Linear(self.bert.config.hidden\_size, 2) # binary classification (fake, non-fake)*

*# def forward(self, input\_ids, attention\_mask):*

*# # Get embeddings for each sentence*

*# outputs = self.bert(input\_ids=input\_ids, attention\_mask=attention\_mask)*

*# print(outputs[0].shape)*

*# sentence\_embeddings = outputs.last\_hidden\_state[:, 0, :] # [batch\_size, num\_sentences, hidden\_size]*

*# print(sentence\_embeddings.shape)*

*# # Apply attention*

*# attention\_weights = self.attention(sentence\_embeddings) # [batch\_size, num\_sentences, 1]*

*# weighted\_sum = torch.sum(attention\_weights \* sentence\_embeddings, dim=1) # [batch\_size, hidden\_size]*

*# print(weighted\_sum.shape)*

*# # Pass through classifier*

*# logits = self.classifier(weighted\_sum) # [batch\_size, 2]*

*# return logits*

*# # Example Usage:*

*# tokenizer = BertTokenizer.from\_pretrained("bert-base-uncased")*

*# text = ["This is a sentence.", "This is another sentence."] # replace with your sentences*

*# inputs = tokenizer(text, return\_tensors="pt", padding=True, truncation=True)*

*# model = FakeNewsClassifier()*

*# logits = model(inputs["input\_ids"], inputs["attention\_mask"])*

In [19]:

import torch.nn as nn

from transformers import BertModel, BertTokenizer

from transformers import BertConfig

In [20]:

import math

class **PositionalEncoding**(nn.Module):

def \_\_init\_\_(self, d\_model, max\_len=5000):

super(PositionalEncoding, self).\_\_init\_\_()

*# Compute the positional encodings once in log space.*

pe = torch.zeros(max\_len, d\_model)

position = torch.arange(0., max\_len).unsqueeze(1)

div\_term = torch.exp(torch.arange(0., d\_model, 2) \* -(math.log(10000.0) / d\_model))

pe[:, 0::2] = torch.sin(position \* div\_term)

pe[:, 1::2] = torch.cos(position \* div\_term)

pe = pe.unsqueeze(0)

self.register\_buffer('pe', pe)

def forward(self, x):

*#print(x.size())*

*#print(self.pe.size())*

*#print(self.pe[:, :x.size(1)].size())*

x = x + self.pe[:, :x.size(0)]

return x