

**UNIVERSITY OF SCIENCE – VIETNAM NATIONAL UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY**



REPORT FINAL PROJECT

SUBJECT: STATISTICAL MACHINE LEARNING - CSC15004

**TOPIC:
SENTIMENT ANALYSIS ON IMDB MOVIE REVIEW WITH BERT**

[Student Information]

20127049 – Nguyễn Đức Minh

20127028 – Võ Văn Hoàng

20127092 – Nguyễn Minh Tuấn

[Lecturers]

Theory Lecturers – Ngô Minh Nhựt

Lab Instructor – Lê Long Quốc

Ho Chi Minh City, August, 2023

TABLE OF CONTENT

I. Student Evaluation	2
II. Problem Statement and benefits	2
1. Problem Statement.....	2
2. Benefits	2
III. About the Dataset	3
1. General information.....	3
2. Data exploration	3
IV. Related information	5
1. What is Transformers?	5
2. What is BERT?.....	10
3. BERT Overview.....	11
4. BERT versions.....	12
5. Main work in BERT.....	12
6. Bert Use.....	14
V. Model Description	14
1. File explore.ipynb	14
2. File train_eval.ipynb	14
3. File utils.py	19
4. File config.py	19
5. File config.json.....	19
6. File app.py.....	19
VI. Evaluation	20
VII. Application	22
1. Display of the web-base application	22
2. Something that can be modified	22
3. About the main function.....	24
VIII. References.....	26

I. Student Evaluation

Student ID	Full name	Work	Rate
20127049	Nguyễn Đức Minh	Research about BERT, write report, implement code, build web-based application.	100%
20127028	Võ Văn Hoàng	Research about BERT, write report, implement code, build web-based application.	100%
20127092	Nguyễn Minh Tuấn	Research about BERT, write report, implement code, build web-based application.	100%

II. Problem Statement and benefits

1. Problem Statement

- Have you ever wished to compile a list of all the movie reviews so you could quickly determine if a certain film is worth seeing? This project's goal is to create a model that can correctly forecast whether a movie review will be favorable or not. A straightforward Streamlit App will make the model accessible for interaction.
- We have a set of sentiment on movie reviews from IMDb (IMDb is an online database of information related to films, television series, podcasts...). By using Natural Language Processing Transformer Model, we should retrain the Transformer Model and Evaluate it to classify sentiment labels for reviews. Along with that, we have created an application that users can visually utilize the Model prediction.

2. Benefits

- Sentiment IMDB analysis is very useful for companies to understand how their user base reacts to a given product or service. Through quantifying, monitoring, and automating sentiments, gathered through feedback method, companies can quickly react to possible faulty features or services. In addition, enterprises are able to make more informed decisions quicker and more accurately.
- Sentiment IMDB analysis allows for data sorting at scale. Furthermore, sentiment analysis is done in real-time, giving organizations valuable insights on key metrics like churn or customer satisfaction rates. Lastly, companies have access to constant real-time critique, enabling a positive feedback loop and faster product iterations to address unhappy users.

III. About the Dataset

1. General information

- Our group has chosen the [IMDB Dataset of 50K Movie Reviews](#) to retrain our Sentiment Analysis model. [Here](#) is more information about this large dataset. This set contains 50000 rows, which is sufficiently large for us to split it into train/validate/test sets. It has around 410 duplicated rows, about under 0.5% duplicated values on the whole set. The dataset includes 2 columns ["review"] and ["sentiment"]; For each row, the "sentiment value" corresponds to the attitude to the "review" of that row. With 2 unique values of column ["sentiment"]: **positive** and **negative**, which is to express the positive or negative watching experience by the review. Our mission is to train a Text Classification model to recognize the sentiment for a new review given.

2. Data exploration

a. Check the percentage of duplicates

```
print(f'Number of samples:{len(df.review)}')
print(f'Number of unique reviews: {df.review.nunique()}')
print(f'Percentage of duplicates: {(len(df.review) - df.review.nunique()) /
len(df.review)*100}%')
```

```
Number of samples:50000
Number of unique reviews: 49582
Percentage of duplicates: 0.836%
```

- About 0.836% data is duplicated, we will have some methods to handle this in the model description part.

b. Check the content of 2 columns in the dataset

- First of all, we read the 10 first lines of this dataset by: `df.head(10)` and we got the result:

	review	sentiment
0	One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked. They are right, as this is exactly what happened with me. The first thing that struck me...	positive
1	A wonderful little production. The filming technique is very unassuming- very old-time-BBC fashion and gives a comforting, and sometimes discomforting, sense of realism to the entire p...	positive
2	I thought this was a wonderful way to spend time on a too hot summer weekend, sitting in the air conditioned theater and watching a light-hearted comedy. The plot is simplistic, but the dialogue i...	positive
3	Basically there's a family where a little boy (Jake) thinks there's a zombie in his closet & his parents are fighting all the time. This movie is slower than a soap opera... and suddenl...	negative
4	Petter Mattei's "Love in the Time of Money" is a visually stunning film to watch. Mr. Mattei offers us a vivid portrait about human relations. This is a movie that seems to be telling us what mone...	positive
5	Probably my all-time favorite movie, a story of selflessness, sacrifice and dedication to a noble cause, but it's not preachy or boring. It just never gets old, despite my having seen it some 15 o...	positive
6	I sure would like to see a resurrection of a up dated Seahunt series with the tech they have today it would bring back the kid excitement in me.I grew up on black and white TV and Seahunt with Gun...	positive
7	This show was an amazing, fresh & innovative idea in the 70's when it first aired. The first 7 or 8 years were brilliant, but things dropped off after that. By 1990, the show was not really funny ...	negative
8	Encouraged by the positive comments about this film on here I was looking forward to watching this film. Bad mistake. I've seen 950+ films and this is truly one of the worst of them - it's awful i...	negative
9	If you like original gut wrenching laughter you will like this movie. If you are young or old then you will love this movie, hell even my mom liked it. Great Camp!!!	positive

- It seems there are many rows with symbol in html such as: `
`, so we will count to know whether there are many rows with that symbol or not by:

```
df = pd.read_csv('IMDB_dataset.csv')
br_count = df['review'].str.contains('<br />').sum()
print(f"Number of rows with <br /> in 'review' column: {br_count}")
Number of rows with <br /> in 'review' column: 29200
```

- The result is 29200, it is a big value, so that we will have some method to process it.

c. *Check if there are any missing values*

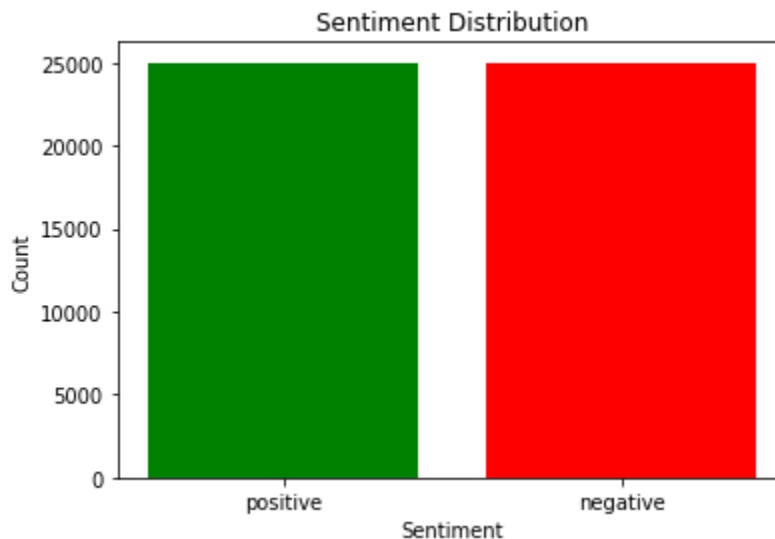
```
df.isna().sum()
```

```
review      0
sentiment   0
dtype: int64
```

- It is a good signal, nothing in the dataset is missing.

d. *Examine the balance of data*

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('IMDB_dataset.csv')
sentiment_counts = df['sentiment'].value_counts()
plt.bar(sentiment_counts.index, sentiment_counts.values, color=['green',
'red'])
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.title('Sentiment Distribution')
plt.show()
```



- This IDBM dataset has a good equality, both the “positive” and “negative” has 25000 samples in the sum of 50000.

→ Conclusion:

- There are HTML tags in the second review; and probably in many other reviews. We'll need to perform some cleaning for the whole dataset.
- It looks like there are duplicated reviews. We'll remove these to avoid overly optimistic estimation of model performance (which will happen if same reviews appear in both training and test sets).
- The target is in string format (negative, positive) and will need to be converted into integers (0, 1).
- No need to worry about data imbalance.

IV. Related information

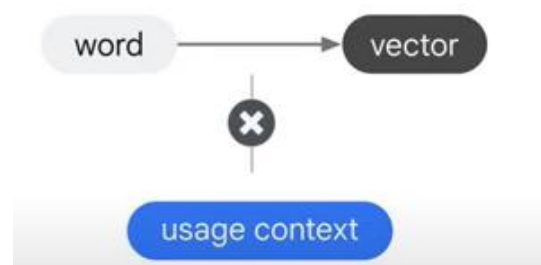
- Language modeling has evolved over the years. The recent breakthrough in the past ten years include the usage of neural networks to represent text such as Word2vec and N-grams in 2013. In 2014, the development of sequence to sequence models such as RNN's and LSTM's helped improve the performance of ML models on NLP tasks such as translation and text classification. In 2015, the excitement came with attention mechanisms and the models built based on it, such as Transformers and the Bert model.



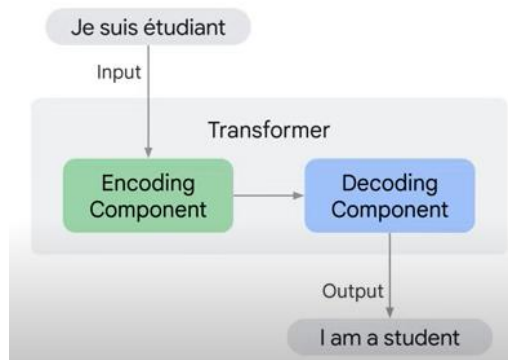
Language Model History

1. What is Transformers?

- Transformers is based on a 2017 paper named **Attention As All You Need**. Although all the models before Transformers were able to represent words as vectors, these vectors did not contain the context and the usage of words changes based on the context.

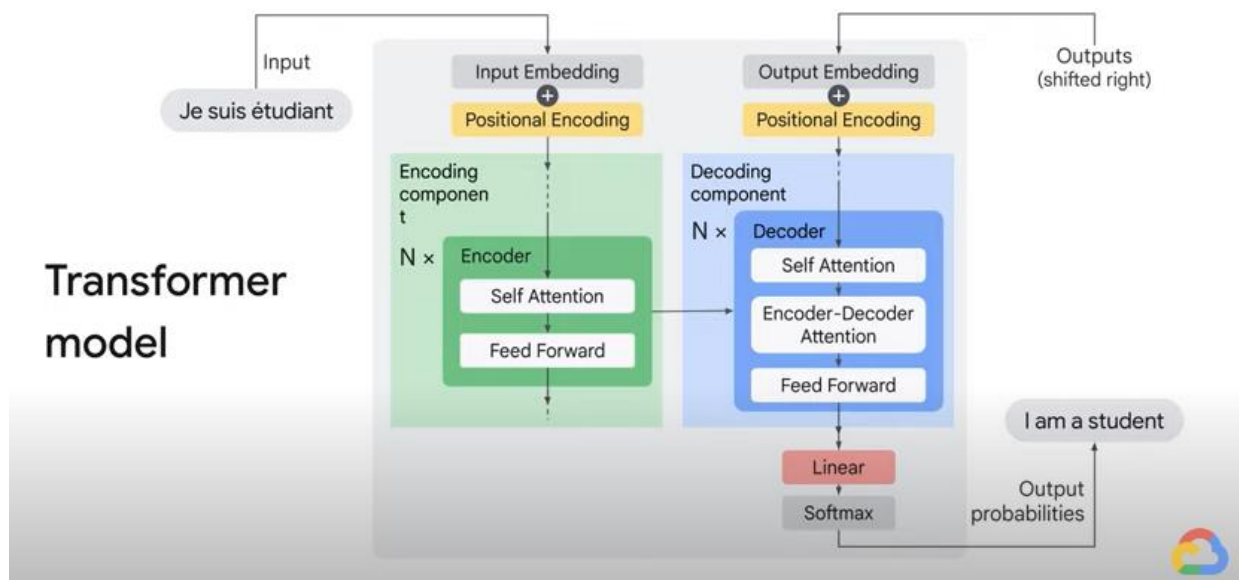


- For example, bank and riverbank, versus bank in bank robber might have the same vector representation before attention mechanisms came about.

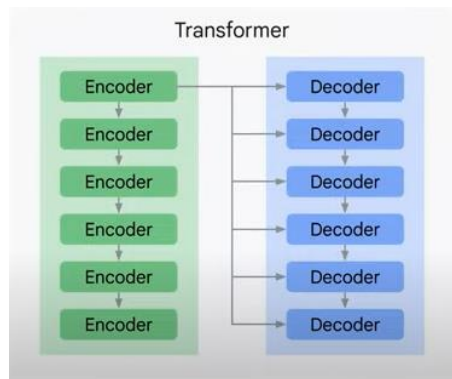


- A Transformer is an encoder – decoder model that uses the attention mechanism. It can take advantage of pluralization and also process a large amount of data at the same time because of its model architecture.

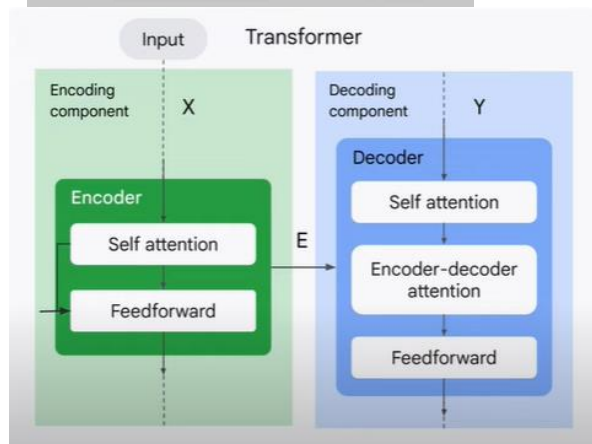
- Attention mechanism helps improve the performance of machine translation applications. Transformer models were built using attention mechanisms at the core.



- A transformer model consists of encoder and decoder. The encoder encodes the input sequence and passes it to the decoder, and the decoder decodes the representation for a relevant task.



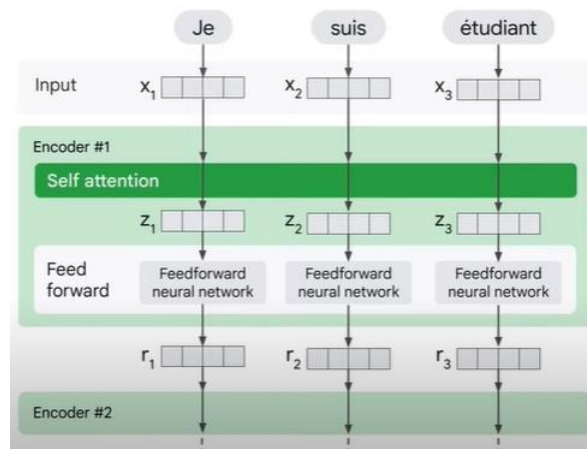
- The encoding component is a stack of encoders of the same number.
- The research paper that introduced Transformers stack six encoders on top of each other. Six is not a magical number, it is just a hyperparameter.

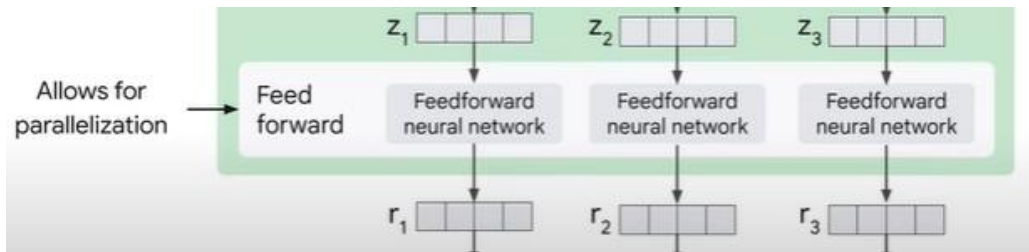


- The encoders are all identical in structure, but with different weights. Each encoder can be broken down into two sub layers.
- The first layer is called self-attention. The input of the encode are first flows through a self-attention layer which helps to encode or look at relevant parts of the words as it encodes a central word in the input sentence.
- The second layer is called a feedforward layer, the output of the self-attention layer is

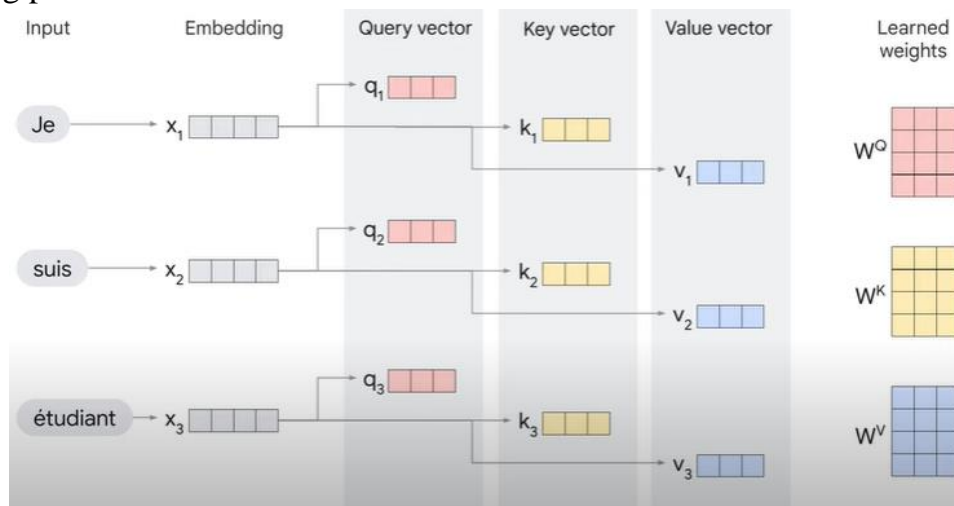
fed to the feedforward neural network. The exact same feedforward neural network is independently applied to each position. The decoder has both the self-attention and the feedforward layer, but between them is the encoder – decoder attention layer that helps a decoder focus on relevant parts of the input sentence.

- After embedding the words in the input sequence, each of the embedding vector flows through the two layers of the encoder. The word at each position passes through a self-attention process. Then it passes through a feedforward neural network, the exact same network with each vector flowing through it separately. Dependencies exist between these paths in this self-attention layer.
- However, the feedforward layer does not have these dependencies and therefore various paths can be executed in parallel while they flow through the feedforward layer.

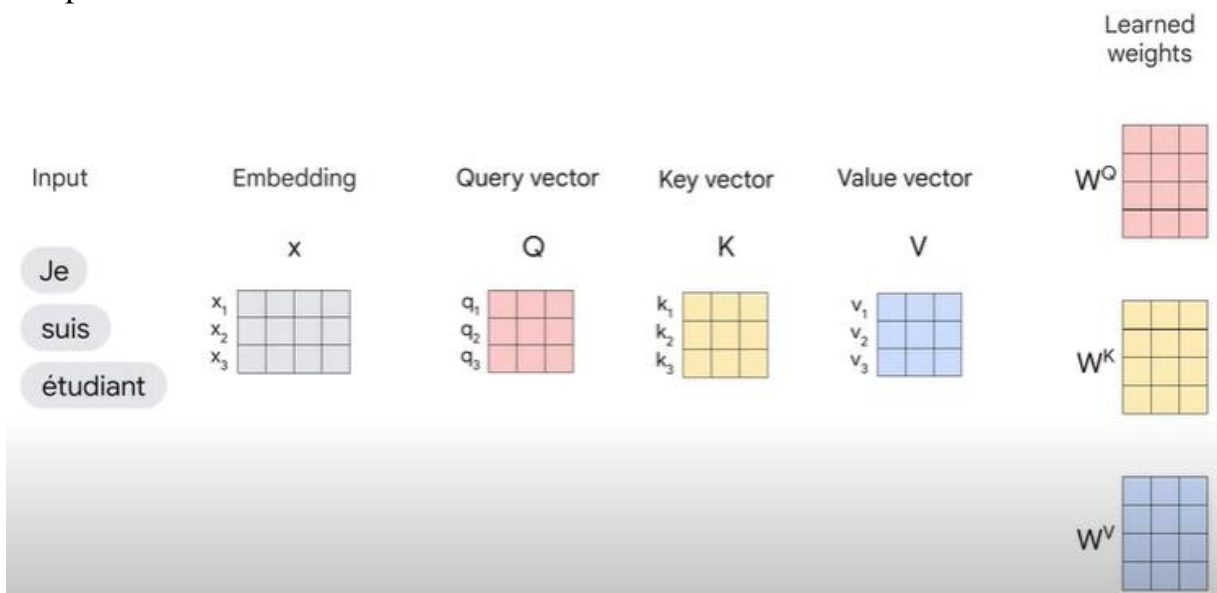




- In the self-attention layer, the input embedding is broken up into query, key, and value vectors. These vectors are computed using weights that the transformer learns during the training process.



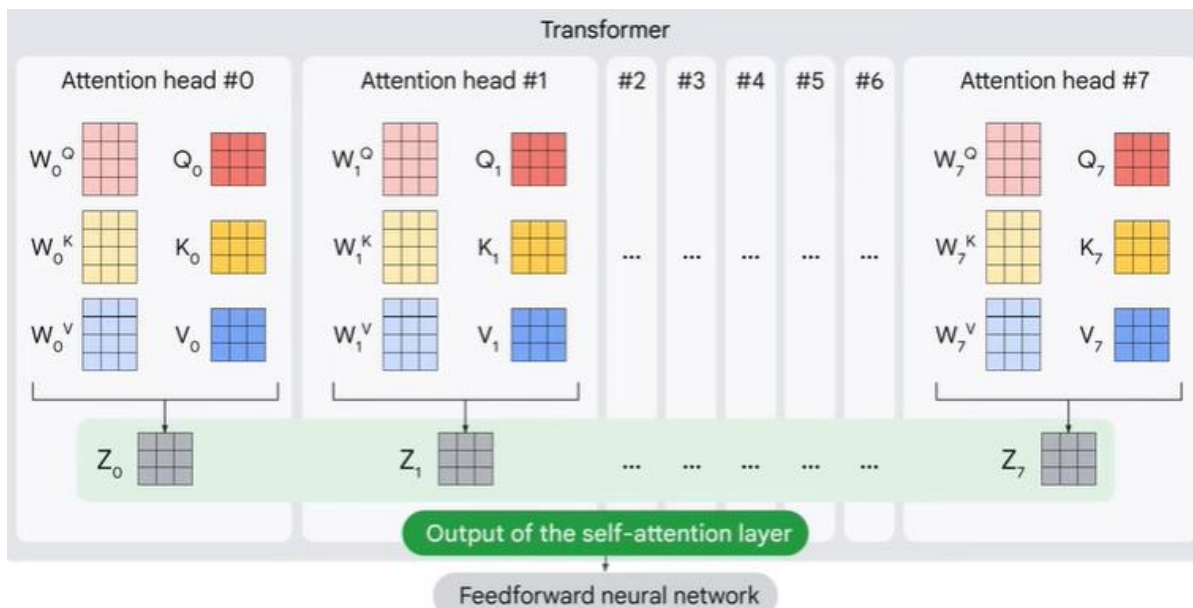
- All of these computations happen in parallel in the model, in the form of matrix computation.



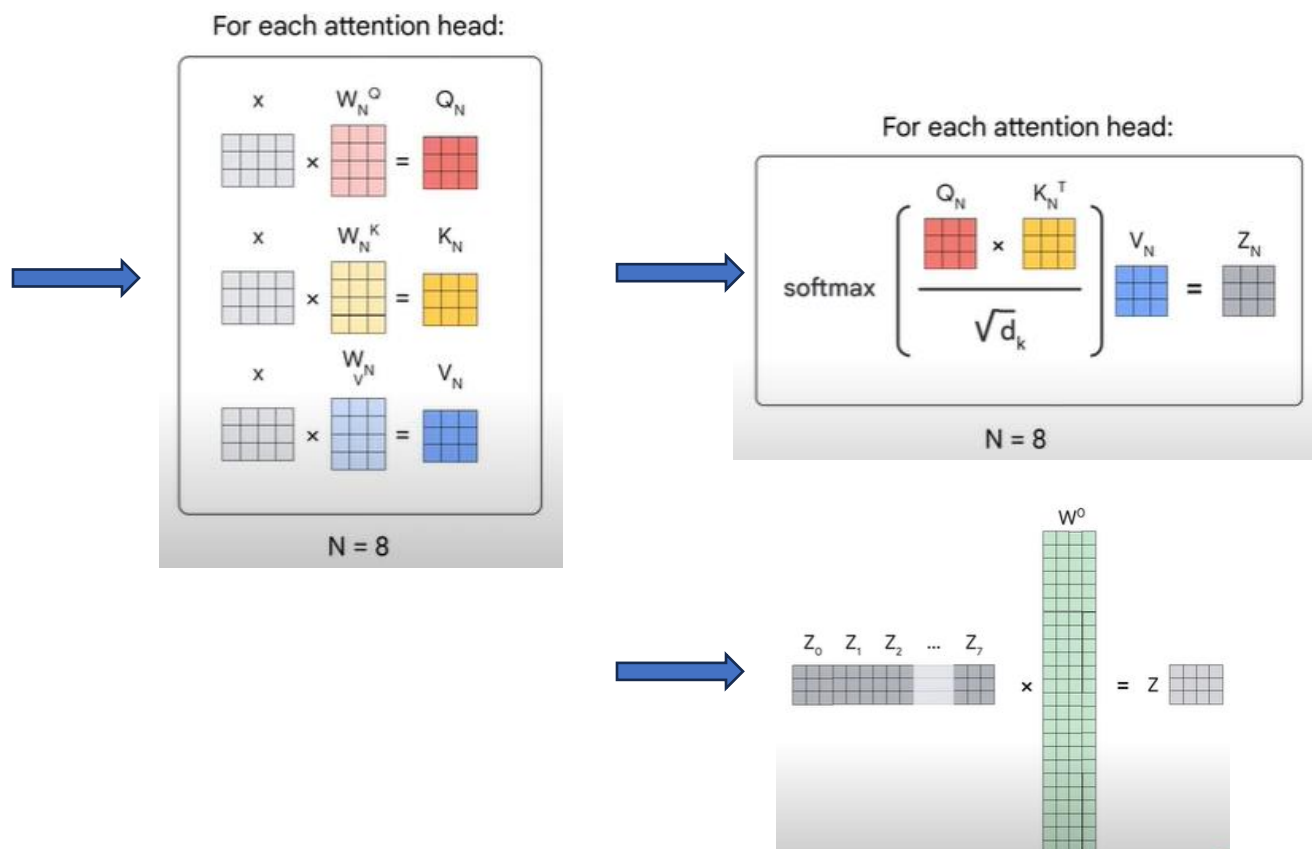
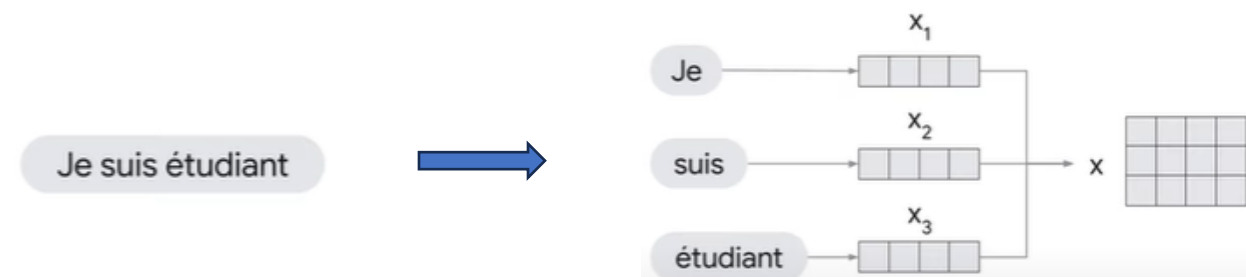
- Once we have the query key and value vectors, the next step is to multiply each value vector by the SoftMax score preparation to sum them up. The intention here is to keep intact the values of the words we want to focus on and leave out irrelevant words by multiplying them by tiny numbers like 0.001 for example.

$$\text{softmax} \times \left(\frac{Q \times K^T}{\sqrt{d_k}} \right) \times V = Z$$

- Next, we have to sum up the weighted value vectors which produces the output of the self-attention layer at this position. For the first word, we can send along the resulting vector to the feedforward neural network.



- To sum up this process of getting the final embeddings, these are the steps that we take. We start with the natural language sentence \rightarrow embed each word in the sentence \rightarrow after that, we perform multi-headed attention 8 times in this case and multiply this embedded word with the respective weighted matrices \rightarrow we then calculate the attention using the resulting Q, K, V matrices \rightarrow finally, we can concatenate the matrices to produce the output matrix, which is the same dimension as the final matrix that this layer initially got.

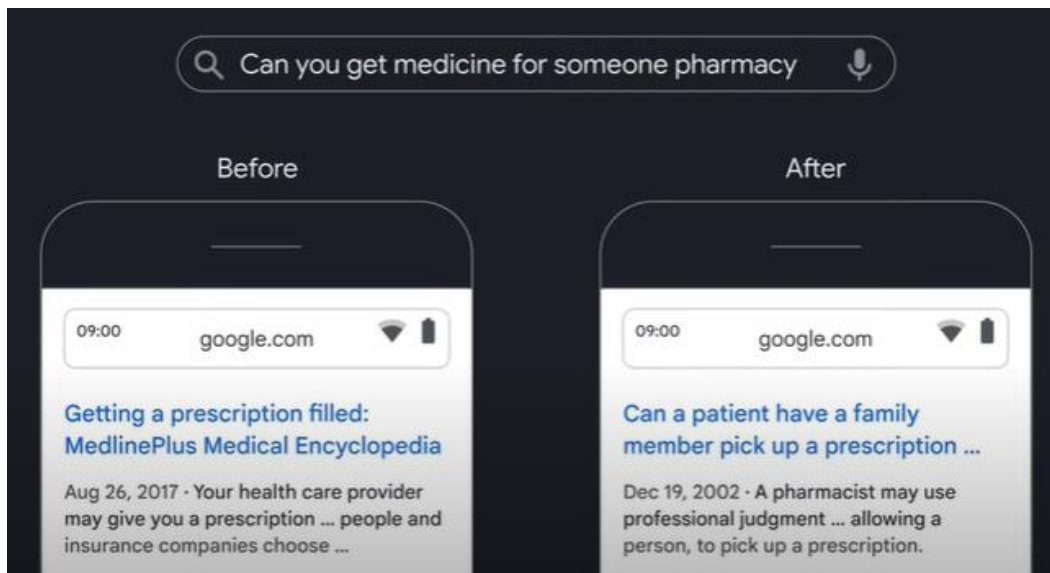


2. What is BERT?

- There is multiple variations of transformers out there now.

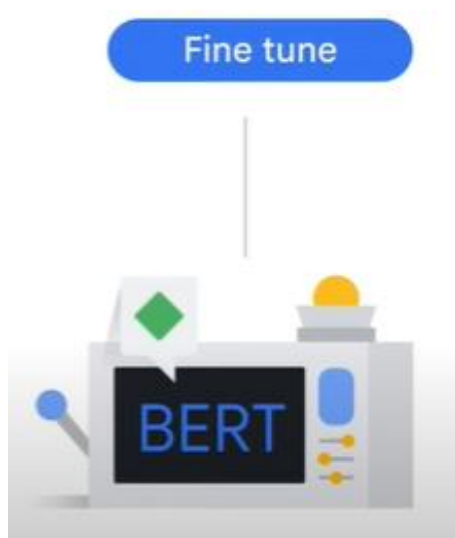


- Some use both the encoder and the decoder component from the original architecture, some use only the encoder and some use only the decoder. A popular encoder only architecture is BERT.
- BERT is one of the trained transformer models. **BERT** stands for **Bidirectional Encoder Representations** from Transformers and it is a state-of-the-art machine learning model used for NLP tasks. Jacob Devlin and his colleagues developed BERT at Google in 2018.
- Since then, multiple variations of Bert have been built. Today, Bert powers Google Search.



The different results provided by Bert are for the same search query before and after.

3. BERT Overview



- There are two pre-trained general BERT variations: The Bert base model is a 12-layer, 768-hidden, 12-heads, 110M parameter neural network architecture, whereas the Bert large model is a 24-layer, 1024-hidden, 16-heads, 340M parameter neural network architecture.
- The Bert model is powerful because it can handle long input context.
- It was trained the BERT on English Wikipedia (2,500M words) and BooksCorpus (800M words) and achieved the best accuracies for some of the NLP tasks in 2018 by Devlin and his colleagues.
- The Bert model was trained for 1 million steps.
- Bert is trained on different tasks, which means it has multi-task objective. This makes BERT very powerful.

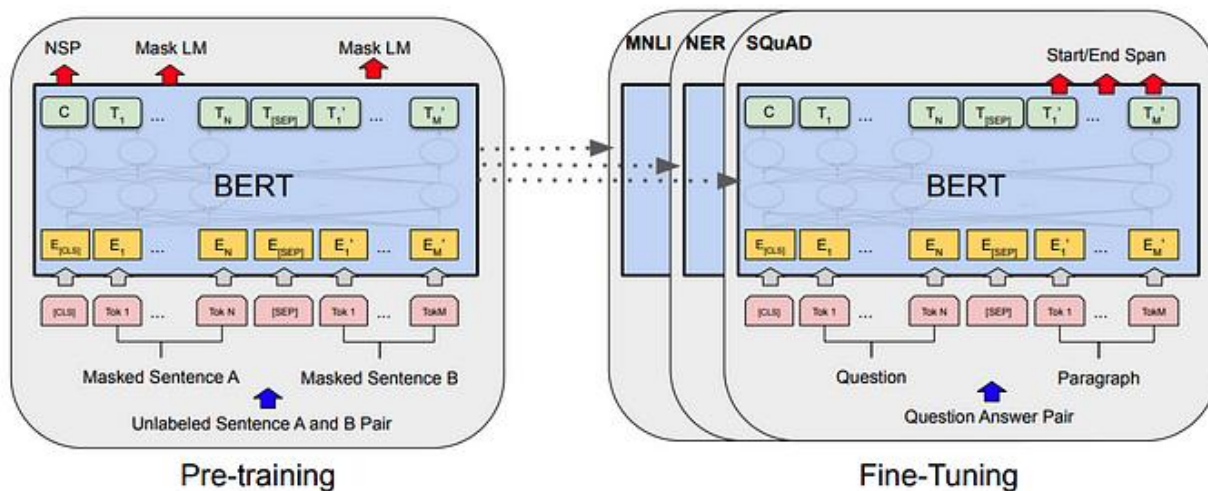
- Because of the kind of tasks it was trained on, it works at both a sentence level and at a token level.

4. BERT versions

	BERT _{BASE}	BERT _{LARGE}	Transformer
Layers	12	24	6
Feedforward networks (hidden units)	768	1024	512
Attention heads	12	16	8

- These are the two different versions of Bert that were originally released. One is Bert base, which had 12 layers, whereas Bert large had 24 layers and compared to the original transformer, which had 6 layers.

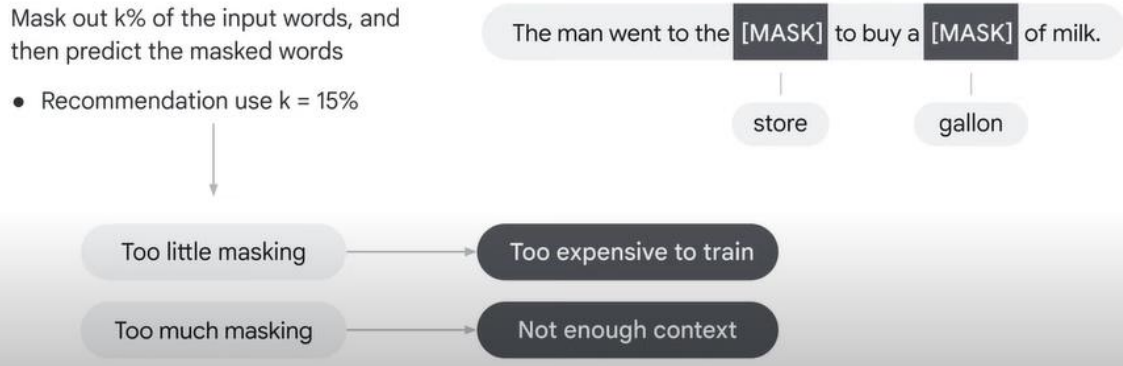
5. Main work in BERT



Picture about the overall pre-training and fine-tuning procedures for BERT

a. Masked language modeling (MLM)

- The way that Bert works is that it was trained on two different tasks. Task 1 is called a masked language model, where the sentences are masked and the model is trained to predict the masked words.
- If we were to train Bert from scratch, we would have to mask a certain percentage of the words in our corpus. The recommended percentage for masking is 15%. The masking percentage achieves a balance between too little and too much masking. Too little masking makes the training process extremely expensive, and too much masking removes the context of the model requires.



b. Next sentence prediction (NSP) (binary classification task)

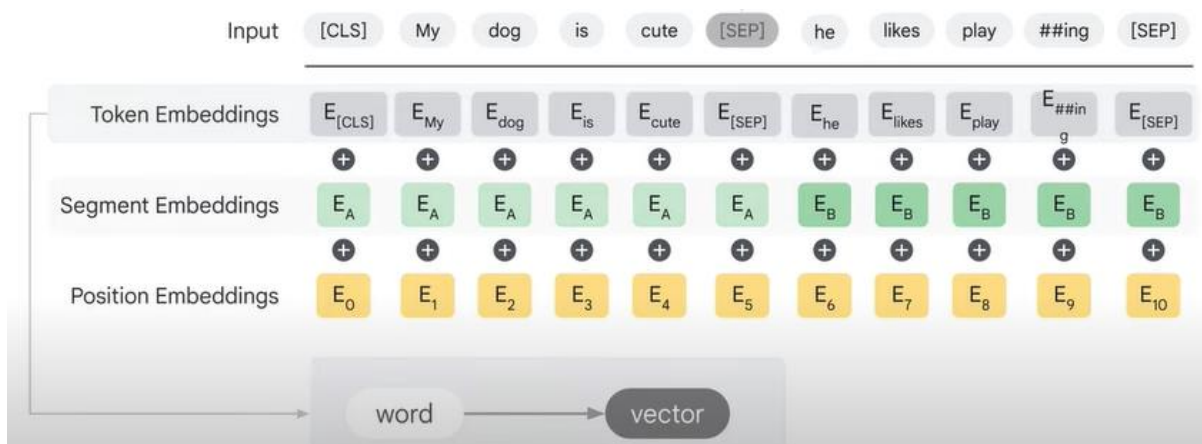
- The second task is to predict the next sentence. For example, the model is given two set of sentences. Bert aims to learn the relationships between sentences and predict the next sentence given the first one.



- For example, sentence A could be “The man went to the store” and sentence B is “he bought a gallon of milk”. Bert is responsible for classifying if sentence B is in next sentence after sentence A.
- This is a binary classification task. This helps Bert perform at a sentence level in order to train Bert.

c. BERT input embeddings

- In order to train Bert, we need to feed 3 different kinds of embeddings to the model, for the input sentence we get 3 different embeddings token, segment and position embeddings. The token embedding is a representation of each token as an embedding in the input sentence. The words are transformed into vector representations of certain dimensions.



- Bert can solve NLP tasks that involve text classification as well. An example is to classify whether 2 sentences say “My dog is cute and he likes playing” are semantically similar. The pairs of input texts are simply concatenated and fed into the model. How does bert distinguish the input in a given pair? The answer is to use segment embeddings. There is a special token represented by SEP that separates the 2 different splits of the sentence.
- Another problem is to learn the order of the words in the sentence. As we know, Bert consists of a stack of transformers. Bert is designed to process input sequences up to a length of 512. The order of the input sequence is incorporated into the position embeddings. This allows Bert to learn a vector representation for each position

6. Bert Use

- Bert can be used for different downstream tasks. Although Bert was trained on mass language modeling and single sentence classification, it can be used for popular NLP tasks like single sentence classification, sentence pair classification, question answering and single sentence tagging tasks.

V. Model Description

1. File explore.ipynb

- The main purpose of this file is for exploratory data analysis that we mentioned clearly in part III.

2. File train_eval.ipynb

- In this file, we redefined every configuration and functions again for easy usage.

```
pd.set_option('display.max_colwidth', 200)
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(device)
```

cpu

- This line of code is to check if there GPU available. It is a priority to use GPU, if there is no GPU, we will use CPU.

```
DATA_PATH = '../data/IMDB_Dataset.csv'
SAVED_MODEL_PATH = '../model'
BERT_CHECKPOINT = 'bert-base-uncased'
MAX_LEN = 128
BATCH_SIZE = 32
NUM_CLASSES = 2
LEARNING_RATE = 2e-5
NUM_EPOCHS= 5
```

- We defined file directory, checkpoint and configurations.

```
# For cleaning reviews
def clean_text(text):
    """Removes extra whitespaces and html tags from text."""
    # remove weird spaces
    text = " ".join(text.split())
    # remove html tags
    text = re.sub(r'<.*?>', '', text)
    return text
```

- This function is to remove html tags and also white spaces.

```
# Class for custom dataset
class CustomDataset(Dataset):
    def __init__(self, review, target, tokenizer, max_len, clean_text=None):
        self.clean_text = clean_text
        self.review = review
        self.target = target
        self.tokenizer = tokenizer
        self.max_len = max_len
    def __len__(self):
        return len(self.review)
    def __getitem__(self, idx):
        y = torch.tensor(self.target[idx], dtype=torch.long)
        X = str(self.review[idx])
        if self.clean_text:
            X = self.clean_text(X)
        encoded_X = self.tokenizer(
            X,
            return_tensors = 'pt',
            max_length = self.max_len,
            truncation=True,
            padding = 'max_length'
```


- ```

)
 return {'input_ids': encoded_X['input_ids'].squeeze(),
 'attention_mask': encoded_X['attention_mask'].squeeze(),
 'labels': y}

```
- This class is to initialize the features id and token encoder from a dataset.
- ```

# Traing loop for one epoch
def train_epoch(model, dataloader, optimizer, scheduler, device, progress_bar):
    losses = []
    accuracies = []
    model.train()
    for batch in dataloader:
        optimizer.zero_grad()
        batch = {k:v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        loss = outputs.loss
        loss.backward()
        optimizer.step()
        scheduler.step()
        preds = torch.argmax(outputs.logits, dim=1)
        acc = torch.sum(preds == batch['labels']) / len(preds)
        accuracies.append(acc)
        losses.append(loss)
        progress_bar.update(1)
    return torch.tensor(losses, dtype=torch.float).mean().item(),
           torch.tensor(accuracies).mean().item()

```
- This function is to calculate loss and accuracy for one time epoch training, this calculation by calculating mean of all batches.

```

# Evaluation loop
def eval_epoch(model, dataloader, device):
    losses = []
    accuracies = []
    model.eval()
    with torch.no_grad():
        for batch in dataloader:
            batch = {k:v.to(device) for k, v in batch.items()}
            outputs = model(**batch)
            loss = outputs.loss
            preds = torch.argmax(outputs.logits, dim=1)
            acc = torch.sum(preds == batch['labels']) / len(preds)
            accuracies.append(acc)
            losses.append(loss)

```

- ```

 return torch.tensor(losses, dtype=torch.float).mean().item(),
 torch.tensor(accuracies).mean().item()

```
- This is also to calculate the loss and accuracy. However, this function is for evaluation set.

```

For final evaluation on test set
def test(model, dataloader, device):
 y_preds = []
 y_true = []
 model.eval()
 with torch.no_grad():
 for batch in dataloader:
 batch = {k:v.to(device) for k, v in batch.items()}
 outputs = model(**batch)
 y_preds.extend(torch.argmax(outputs.logits, dim=1))
 y_true.extend(batch['labels'])
 return y_preds, y_true

```

- This is for evaluation on test set on time only, it returns predicted and actual values to estimate model accuracy by Machine Learning metrics.

```

transform targets to integers
data['sentiment'] = data['sentiment'].apply(lambda x: 0 if x == "negative" else 1)
data.head()

```

- At first, we convert positive as 1 and negative as 0 that the model can comprehend as an classification task.

```

Train, validation and test splits
train_df, test_val_df = train_test_split(data, test_size=0.3,
 stratify=data['sentiment'], random_state=20)
val_df, test_df = train_test_split(test_val_df, test_size=0.5,
 stratify=test_val_df['sentiment'], random_state=20)
train_df.reset_index(drop=True, inplace=True)
val_df.reset_index(drop=True, inplace=True)
test_df.reset_index(drop=True, inplace=True)
print(f'Number of samples in train set: {len(train_df)}')
print(f'Number of samples in validation set: {len(val_df)}')
print(f'Number of samples in test set: {len(test_df)}')

```

```

Number of samples in train set: 34707
Number of samples in validation set: 7437
Number of samples in test set: 7438

```

- First we split 70% data to be trained. The remaining 30% is splitted in half, first half for validation and the remaining for testing.

```
tokenizer = BertTokenizer.from_pretrained(BERT_CHECKPOINT)
```

- Initialize tokenizer.

```
dfs = {'train':train_df, 'val':val_df, 'test':test_df}
dataloaders = {}
for df in dfs:
 should_shuffle = True if df == 'train' else False
 dataloaders[df] = DataLoader(
 CustomDataset(dfs[df]['review'], dfs[df]['sentiment'], tokenizer=tokenizer,
 max_len=MAX_LEN, clean_text=clean_text),
 batch_size=BATCH_SIZE, shuffle=should_shuffle
)
```

- With each set, we custom its data by CustomData class.

```
model
model = AutoModelForSequenceClassification.from_pretrained(BERT_CHECKPOINT,
 num_labels=NUM_CLASSES)
model.to(device)
optimizer
optimizer = torch.optim.AdamW(model.parameters(), lr = LEARNING_RATE)
scheduler
num_training_steps = NUM_EPOCHS * len(dataloaders['train'])
scheduler = get_scheduler(
 'linear',
 optimizer=optimizer,
 num_warmup_steps=0,
 num_training_steps=num_training_steps
)
```

- This is to initialize the configuration for model optimizer and scheduler.

```
Training, evaluation
progress_bar = tqdm(range(num_training_steps))
history = {'train_loss':[], 'train_acc':[], 'val_loss':[], 'val_acc': []}
best_accuracy = 0
for epoch in range(NUM_EPOCHS):
 train_loss, train_acc = train_epoch(model, dataloaders['train'], optimizer,
 scheduler, device, progress_bar)
 print(f'Train Loss: {train_loss :.4f} | Accuracy: {train_acc*100 :.2f}')
 val_loss, val_acc = eval_epoch(model, dataloaders['val'], device)
 print(f'Eval Loss: {val_loss :.4f} | Accuracy: {val_acc*100 :.2f}')
```

```

history['train_loss'].append(train_loss)
history['train_acc'].append(train_acc)
history['val_loss'].append(val_loss)
history['val_acc'].append(val_acc)
save best model
if val_acc > best_accuracy:
 model.save_pretrained(SAVED_MODEL_PATH)
 best_accuracy = val_acc
print('-'*50)

```

- With each epoch training, we calculate loss and accuracy for both train and validate set. Then, we store the best model which has the highest accuracy on validate set.

### 3. File **utils.py**

- The same with the explanation in file `train_eval.ipynb`.


### 4. File **config.py**

- This file is only to store the configurations, file directory and type of BERT model used in the `app.py` file.

### 5. File **config.json**

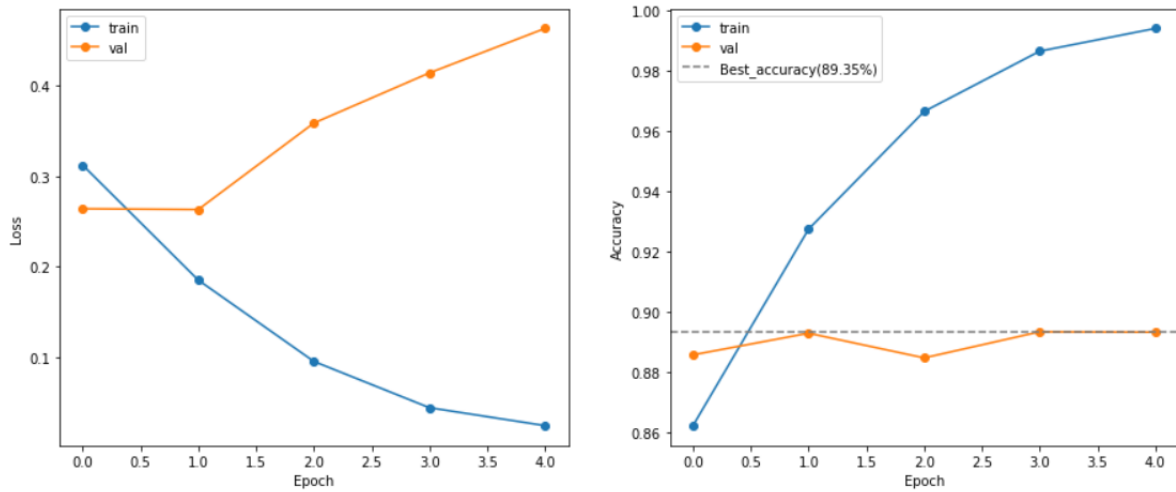
- There are some specific configurations of our BERT model using such as model name, architecture, hidden size, vocab size, problem type,...

### 6. File **app.py**

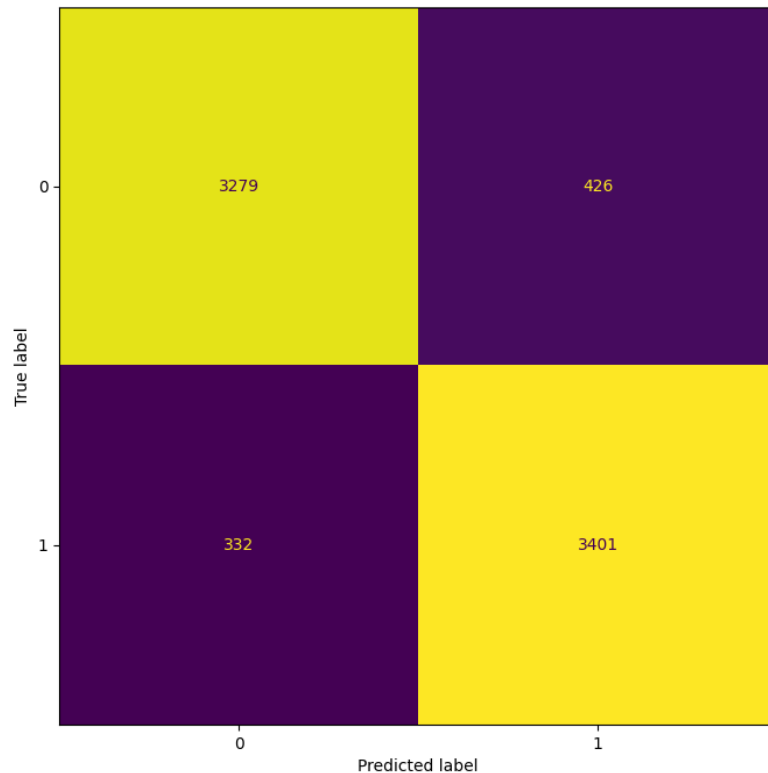
- The main purpose of this file is to create a Graphical User Interface that clients could easily utilize. At first, it loads the model and tokenizer that have been fine-tuned. After setting up the model from the stored `pytorch_model.bin` file, we uploaded the model on GUI. With the user interface, it contains a select box where we could choose English or Vietnamese as the language. However, when we write an input in a text box, the model only supports the English language. We had an “Process” button below to start classifying the sentiment of that input. Finally, GUI will return the sentiment prediction (positive or negative) with confidence score. Along with that, we decorated the sentiment with emoticon “

19

## VI. Evaluation



- We plotted out a line chart to present loss on different numbers of epochs. Although the higher number of epoch helps reduce the loss on train set considerably with nearly 0 loss on the fourth time epoch training, the loss on validate set increases to under 0.5 when we grow the number of epochs. Parallely, the accuracy of the train set presents the model fine-tuned very well with about 100% accuracy with 4 times epoch training. However, it seems the accuracy of the validate set converges at around 90% with every number of epochs. In conclusion, with the large dataset, it is not necessary to train the model many times, because a small number of epochs presents that loss is tiny and accuracy also optimized.




- To make an evaluation, we create a confusion matrix of prediction on the test set. We have a number of samples in the test set: 7438. In actual values, both positive and negative classes have approximately equal number of records (3733 and 3705, respectively). As we can observe from the Confusion Matrix, the model can predict really well with 3279 out of 3705 of negative class and 3401 out of 3722 of positive class.

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.91      | 0.89   | 0.90     | 3705    |
| 1            | 0.89      | 0.91   | 0.90     | 3733    |
| accuracy     |           |        | 0.90     | 7438    |
| macro avg    | 0.90      | 0.90   | 0.90     | 7438    |
| weighted avg | 0.90      | 0.90   | 0.90     | 7438    |

- To support our model, we created a classification report with some reliable metrics to evaluate such as precision, recall, f1-score, accuracy. We have mentioned in the above paragraph, with negative class and positive class, these metrics indicate around 90%, which means the model has been trained and fine-tuned very well to classify the sentiment analysis of reviews correctly. It shows that the model is trustable to be utilized to realize the sentiment impact of a movie.

## VII. Application



### 1. Display of the web-base application


Select language 

English

## Movie Reviews Classifier



Instructions: Enter a movie review  and click the 'Process' button to classify it .


 Enter your movie review:

Process

Created by a group of K20 students in HCMUS in statistical machine learning

### 2. Something that can be modified

#### a. Language

Select language 

English

Vietnamese

English

- We can choose the display in Vietnamese or English, if we choose Vietnamese, the screen can be as the image bellow:

Select language 🌐

Vietnamese

## Phân loại đánh giá phim 🎬 🎬 🎬



Hướng dẫn: Nhập đánh giá phim 🗣️ và nhấp vào nút 'Xử lý' để phân loại 🚀.

📝 Nhập đánh giá phim của bạn

Xử lý

Được tạo ra bởi nhóm sinh viên K20 trường HCMUS trong môn Học Thống kê

### b. Color display



Rerun R

Settings

Record a screencast

Report a bug

Get help

About

Developer options

Clear cache C

Deploy this app

Streamlit Cloud

Report a Streamlit bug

Visit Streamlit docs

Visit Streamlit forums

- We can also change some different things in Setting, such as: wide mode or not, or the types of theme as the following picture:

#### Settings



##### DEVELOPMENT

☐ Run on save

Automatically updates the app when the underlying code is updated.

##### APPEARANCE

☒ Wide mode

Turn on to make this app occupy the entire width of the screen

##### Theme

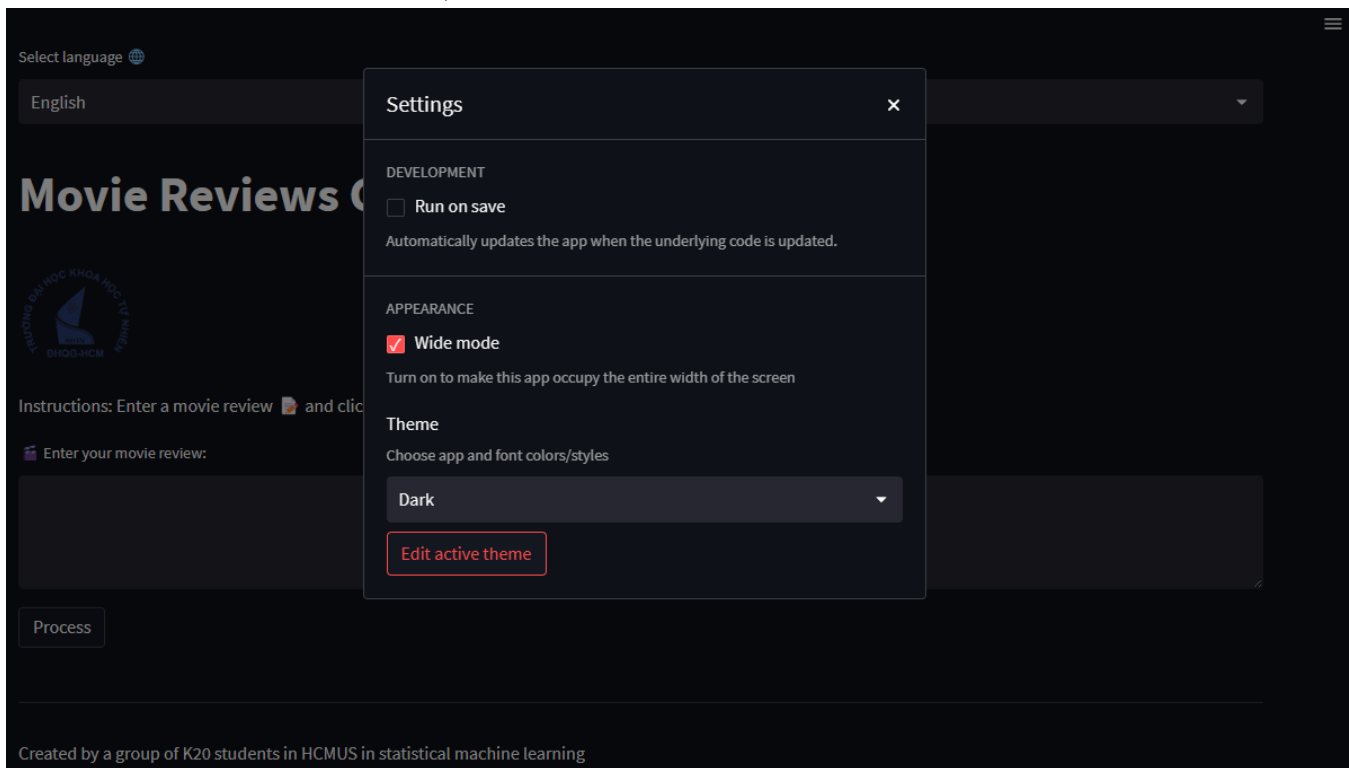
Choose app and font colors/styles

Light

Edit active theme



- If we choose dark theme, the screen can be:



### 3. About the main function

- If we do not enter anything in the movie review space and click *Process* button
  - The result will be: “Please enter a review before clicking “Process”



b. *If we enter a good review about film*

- The result can be:



Instructions: Enter a movie review 📄 and click the 'Process' button to classify it 🚀.

📄 Enter your movie review:

This is the most emotional film that I have ever seen. It makes me love this life more and more.

Process

## Review Classification Result

The review is **positive** 🌟 with a confidence of **100.00%**.

c. *If we enter a bad review about film*



Instructions: Enter a movie review 📄 and click the 'Process' button to classify it 🚀.

📄 Enter your movie review:

This film is boring, there is nothing special scene in the whole movie.

Process

## Review Classification Result

The review is **negative** 🟡 with a confidence of **100.00%**.

## VIII. References

- [1]: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding: <https://arxiv.org/pdf/1810.04805.pdf>
- [2]: How to Fine-Tune BERT for Text Classification?: <https://arxiv.org/pdf/1905.05583v3.pdf>
- [3]: Sentiment Analysis in 10 Minutes with BERT and TensorFlow: <https://towardsdatascience.com/sentiment-analysis-in-10-minutes-with-bert-and-hugging-face-294e8a04b671>
- [4]: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding: <https://arxiv.org/pdf/1810.04805.pdf>
- [5]: How to Fine-Tune BERT With NSP: <https://towardsdatascience.com/how-to-fine-tune-bert-with-nsp-8b5615468e12>
- [6]: Masked-Language Modeling With BERT: <https://towardsdatascience.com/masked-language-modelling-with-bert-7d49793e5d2c>
- [7]: Named Entity Recognition with BERT in PyTorch: <https://towardsdatascience.com/named-entity-recognition-with-bert-in-pytorch-a454405e0b6a>
- [8]: Question Answering on SQuAD with BERT: <https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/reports/default/15792151.pdf>
- [9]: Transformer models and BERT model: Overview: [https://www.youtube.com/watch?v=t45S\\_MwAcOw](https://www.youtube.com/watch?v=t45S_MwAcOw)