NAME - YUG GARG
ENTRY NO.- 2019EE30607

ELL 881

Assignment - 3

Methodology

To use BERT for the Named Entity Recognition downstream tasks following methodology is followed:

1) Data Preprocessing

First, a dictionary corresponding to all the unique labels is made in which each label is assigned with some id. Following code is used for the same.

```
labels = [i.split() for i in data['labels'].values.tolist()]
unique_labels = set()

for lb in labels:
        [unique_labels.add(i) for i in lb if i not in unique_labels]
label_to_ids = {k: v for v, k in enumerate(sorted(unique_labels))}
ids_to_label = {v: k for v, k in enumerate(sorted(unique_labels))}
```

In order to use the data for BERT data preprocessing has to be done which includes two steps *a) Tokenization b) Adjusting the labels for matching tokenization* as after tokenizing the word, subword tokenization may happen for unknown words.
a) Tokenization
The sentence is first split into words and then tokenizer method from BertTokenizerFast is applied as shown in the following code. Here, I have chosen max length to be 512(i.e. the default size for Bert).

```
from transformers import BertTokenizerFast
tokenizer = BertTokenizerFast.from_pretrained('bert-base-cased')
tokenized_inputs = tokenizer(texts, padding='max_length', max_length=512,
truncation=True)
```

b) Adjusting the labels
As the tokenizer uses the word-piece tokenizer which is a sub-word tokenizer i.e., it might split one word into one or more meaningful sub-words. Therefore, there is a parity between the length of original sequence and the tokenizing sequence. Also, due to the addition of of special tokens after tokenization such as [CLS], [SEP], and [PAD], the position of the tokens might change i.e. [CLS] is the first token after the process and due to this the first word of the sentence is no longer the first sentence. These are some of the reasons due to which adjusting of labels have to be done.
word_ids are used for this process which gives the same id for the same sub-words and None to the

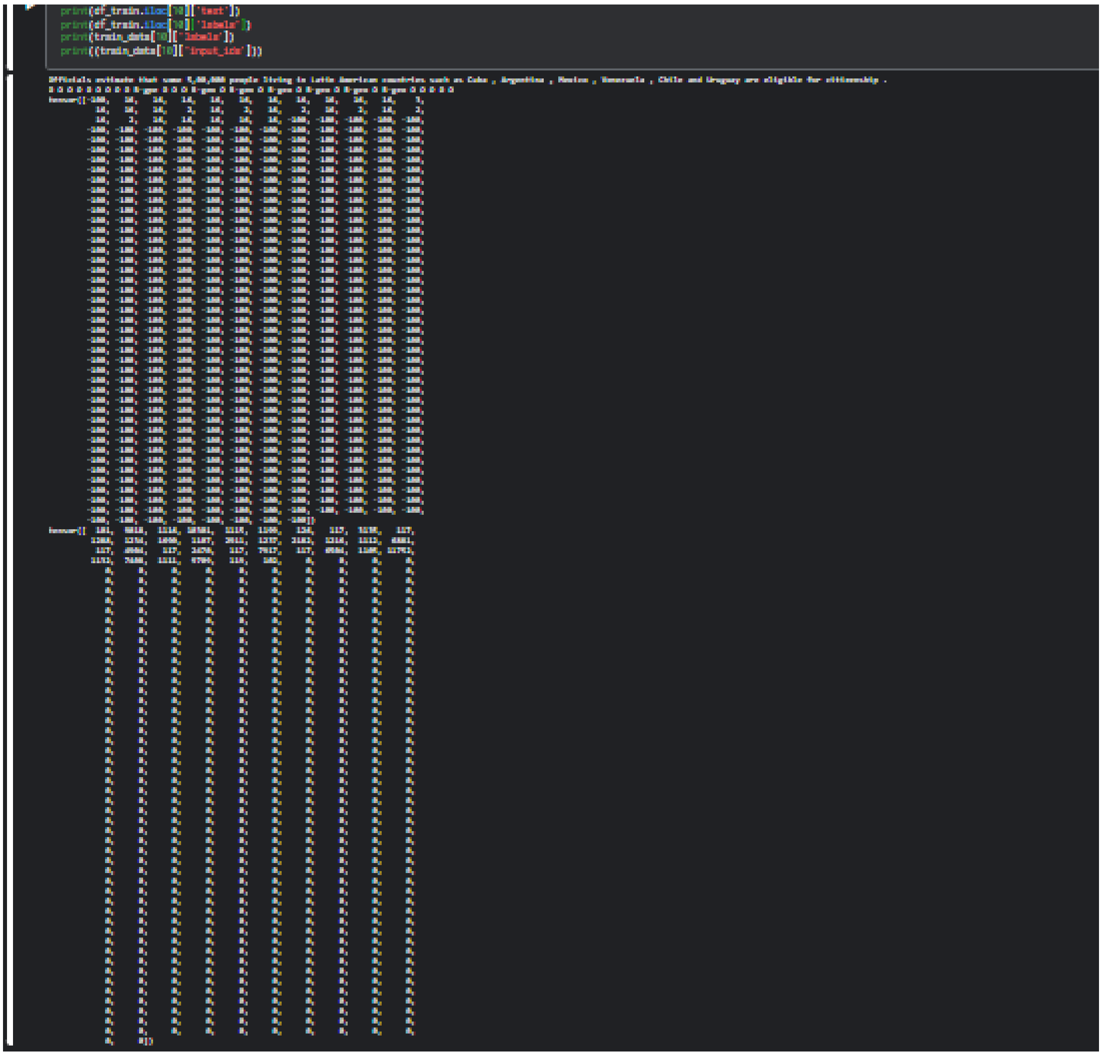
special tokens.
Through these ids alignment can be done through two ways:
a) Assigning label to the first sub-word of each splitted token. The continuation of the sub-word then will then be simply labelled '-100'. All tokens that don't have word_ids i.e special tokens will also be labeled with '-100'.
b) Assigning the same label to all of the sub-words that belong to the same token. All tokens that don't have word_ids will be labeled with '-100'.
The below code is used for the same.
label_all_tokens variable is used to choose between the two methods. When it is true method b) is chosen and when false method a).

```
label_all_tokens = True

def adjust_label(texts, labels):
    tokenized_inputs = tokenizer(texts, padding='max_length',
max_length=512, truncation=True)

    word_ids = tokenized_inputs.word_ids()

    previous_word_idx = None
    label_ids = []

    for word_idx in word_ids:

        if word_idx is None:
            label_ids.append(-100)

        elif word_idx != previous_word_idx:
            try:
                label_ids.append(label_to_ids[labels[word_idx]])
            except:
                label_ids.append(-100)
        else:
            try:
                label_ids.append(label_to_ids[labels[word_idx]] if
label_all_tokens else -100)
            except:
                label_ids.append(-100)
        previous_word_idx = word_idx

    return label_ids
```

2) Dataset Class and Model Building

A dataset class is created to generate and fetch a batch of the data for faster computation of the model. I have splitted the data in 80:10:10 ratio for training, validation and testing sets respectively.
The following is the code and one of the output on how the data is fetched from dataset class.

```
class Ner_Data(torch.utils.data.Dataset):
    def __init__(self, df):
        lb = [i.split() for i in df['labels'].values.tolist()]
        txt = df['text'].values.tolist()
        self.texts = [tokenizer(str(i), padding='max_length', max_length =
512, truncation=True) for i in txt]
        self.labels = [adjust_label(i,j) for i,j in zip(txt, lb)]
    def __len__(self):
        return len(self.labels)
    def get_batch_data(self, idx):
        return self.texts[idx]
    def get_batch_labels(self, idx):
        return torch.LongTensor(self.labels[idx])
    def __getitem__(self, idx):
        batch_data = self.get_batch_data(idx)
        batch_labels = self.get_batch_labels(idx)
        item = {key: torch.as_tensor(val) for key, val in
batch_data.items()}
        item['labels'] = batch_labels
        return item
```



Model Building
As our task is a classification task, therefore I have used the BertForTokenClassification pretrained model for the NER task. For maintaining the case senstivity, I have used the bert-bert-cased pretrained model.

```
from transformers import BertForTokenClassification
model = BertForTokenClassification.from_pretrained('bert-base-cased',
num_labels=len(label_to_ids))
model.to(device)
```

3) Training

Now, comes the most important step in this methodolgy i.e. training the model. The code for the same can be found in the notebook attached.
I have used the following hyperparameters for training:
learning_rate = 0.0001
batch_size = 16
epochs = 6
For backpropogation, I have used Adam optimizer for updating of parameters
Also, I have ignored special tokens like PAD, CLS and SEP for calculating accuracy i.e. val_with_accuracy.

Since, there is a huge abundance of 'O' tokens in the dataset. So, I have also calculated the accuracy without taking consideration of 'O' token (val_without_accuracy) .which can be seen in the below code.

```
            preds = model(input_ids=ids, attention_mask=mask ,labels =
labels)
    #         print(f"loss: {loss.item()}")
            loss = preds['loss']
            logits = preds['logits']
            train_loss+=loss.item()
            train_steps+=1
            # computing train accuracy
            flattened_targets = labels.view(-1)
            active_logits = logits.view(-1, model.num_labels)
            flattened_predictions = torch.argmax(active_logits, axis=1)

            # computing accuracy at active labels
            labels_without = labels
            active_accuracy = labels.view(-1) != -100 # shape (batch_size,
seq_len)

            labels = torch.masked_select(flattened_targets, active_accuracy)
            predictions = torch.masked_select(flattened_predictions,
active_accuracy)

            tmp_train_accuracy = accuracy_score(labels.cpu().numpy(),
predictions.cpu().numpy())
            train_with_accuracy += tmp_train_accuracy
            # computing accuracy at active labels excluding O
            active_without_accuracy = []
            for label in labels_without.view(-1):
                if(label == label_to_ids['O'] or label == -100):
                    active_without_accuracy.append(False)
                else:
                    active_without_accuracy.append(True)

            active_without_accuracy =
torch.as_tensor(active_without_accuracy)
            active_without_accuracy = active_without_accuracy.to(device)

            labels_without = torch.masked_select(flattened_targets,
active_without_accuracy)
            predictions_without = torch.masked_select(flattened_predictions,
active_without_accuracy)
```

```
            tmp_train_without_accuracy =
accuracy_score(labels_without.cpu().numpy(),
predictions_without.cpu().numpy())
            train_without_accuracy += tmp_train_without_accuracy
```

Same procedure is done to calculate validation accuracy.
The final step after training is evaluation of the model on the test dataset, the results of which can be found in the nexrr section.

Experimental Results

I have done mainly two experiments which I have also mentioned in the methodology section which are as follows:
a) Experiment - I
In this experiment methodology is same only difference is in data preprocessing. I have used method a) of aliging labels for it.
Following are the training and test results from it:

Training:

```
Epochs: 1
------------------------------
Train Loss per 1000 steps: 0.269061 [ 1000/2397.9375]
Train Loss per 1000 steps: 0.226578 [ 2000/2397.9375]
Validation loss per 100 evaluation steps: 0.16732494205236434
Validation loss per 100 evaluation steps: 0.16624102910980582
Validation loss per 100 evaluation steps: 0.16483251477281252
Total Train Loss: 0.21705353971232066
Total Train Accuracy With O: 0.9336092961694766
Total Train Accuracy Without O: 0.6828501795677945
Total Validation Loss: 0.16483251477281252
Total Validation Accuracy With O: 0.9475841599314698
Total Validation Accuracy Without O: 0.7438464128409162
Epochs: 2
------------------------------
Train Loss per 1000 steps: 0.149881 [ 1000/2397.9375]
Train Loss per 1000 steps: 0.142647 [ 2000/2397.9375]
Validation loss per 100 evaluation steps: 0.16365802075713873
Validation loss per 100 evaluation steps: 0.1625620689522475
Validation loss per 100 evaluation steps: 0.1614482051320374
Total Train Loss: 0.14045146165619152
Total Train Accuracy With O: 0.9552212412516871
Total Train Accuracy Without O: 0.7872091709823381
Total Validation Loss: 0.1614482051320374
Total Validation Accuracy With O: 0.9515258573684116
Total Validation Accuracy Without O: 0.7577831773252585
Epochs: 3
------------------------------
Train Loss per 1000 steps: 0.122629 [ 1000/2397.9375]
Train Loss per 1000 steps: 0.118516 [ 2000/2397.9375]
Validation loss per 100 evaluation steps: 0.1563493838906288
Validation loss per 100 evaluation steps: 0.15444016521796583
Validation loss per 100 evaluation steps: 0.15345680365959805
Total Train Loss: 0.11793271516153808
Total Train Accuracy With O: 0.9618660783974055
Total Train Accuracy Without O: 0.8209319563327583
Total Validation Loss: 0.15345680365959805
Epochs: 4
------------------------------
Train Loss per 1000 steps: 0.102306 [ 1000/2397.9375]
Train Loss per 1000 steps: 0.099592 [ 2000/2397.9375]
Validation loss per 100 evaluation steps: 0.16457560516893863
Validation loss per 100 evaluation steps: 0.16156427984125912
Validation loss per 100 evaluation steps: 0.15953611786787708
Total Train Loss: 0.09834922020136377
Total Train Accuracy With O: 0.9675153292424106
Total Train Accuracy Without O: 0.8482824552630105
Total Validation Loss: 0.15953611786787708
Total Validation Accuracy With O: 0.9548855857711934
Total Validation Accuracy Without O: 0.7846700839021589
Epochs: 5
------------------------------
Train Loss per 1000 steps: 0.088017 [ 1000/2397.9375]
Train Loss per 1000 steps: 0.085172 [ 2000/2397.9375]
Validation loss per 100 evaluation steps: 0.16095480801537632
Train Loss per 1000 steps: 0.075664 [ 1000/2397.9375]
Train Loss per 1000 steps: 0.076141 [ 2000/2397.9375]
Validation loss per 100 evaluation steps: 0.17052491534501313
Validation loss per 100 evaluation steps: 0.16843145601451398
Validation loss per 100 evaluation steps: 0.167963203197966
Total Train Loss: 0.07535196172020664
Total Train Accuracy With O: 0.9752086399142701
Total Train Accuracy Without O: 0.8859644854164269
Total Validation Loss: 0.167963203197966
Total Validation Accuracy With O: 0.9587978988103955
Total Validation Accuracy Without O: 0.8127963960468906
```

Test:

As, seen from the below results accuracy without O is significantly less than with O. This is due to the overpowering presence of 'O' in the dataset which has effected model learning.

```
Test Loss: 0.1638020795021373
Test Accuracy: 0.9603827639657017
Test Accuracy Without O: 0.815912740379854
```

b) Experiment - II

For this experiment method b) of aliging labels is used..

Following are the training and test results from it:

Training:

Validation loss and accuracy is improved in this case.

```
Epochs: 1
------------------------------
Train Loss per 1000 steps: 0.246665 [ 1000/2397.9375]
Train Loss per 1000 steps: 0.205349 [ 2000/2397.9375]
Validation loss per 100 evaluation steps: 0.14794150680303575
Validation loss per 100 evaluation steps: 0.14712351068854332
Validation loss per 100 evaluation steps: 0.14879861485213042
Total Train Loss: 0.196488976012527
Total Train Accuracy With O: 0.9399010747627506
Total Train Accuracy Without O: 0.6793964346812459
Total Validation Loss: 0.14879861485213042
Total Validation Accuracy With O: 0.9522722118207501
Total Validation Accuracy Without O: 0.7536994730591401
Epochs: 2
------------------------------
Train Loss per 1000 steps: 0.133655 [ 1000/2397.9375]
Train Loss per 1000 steps: 0.129069 [ 2000/2397.9375]
Validation loss per 100 evaluation steps: 0.13477992333471775
Validation loss per 100 evaluation steps: 0.13564114954322576
Validation loss per 100 evaluation steps: 0.1371049087991317
Total Train Loss: 0.12779463541843897
Total Train Accuracy With O: 0.958917228819887
Total Train Accuracy Without O: 0.7841444249933075
Total Validation Loss: 0.1371049087991317
Total Validation Accuracy With O: 0.9574312880289911
Total Validation Accuracy Without O: 0.7700392273646675
Epochs: 3
------------------------------
Train Loss per 1000 steps: 0.111230 [ 1000/2397.9375]
Train Loss per 1000 steps: 0.105534 [ 2000/2397.9375]
Validation loss per 100 evaluation steps: 0.1343186932057142
Validation loss per 100 evaluation steps: 0.13529235993977637
Validation loss per 100 evaluation steps: 0.13551968909489612
Total Train Loss: 0.10452239948353599
Total Train Accuracy With O: 0.9655422923817224
Total Train Accuracy Without O: 0.8200089197957443
Total Validation Loss: 0.13551968909489612
Total Validation Accuracy With O: 0.9600185850553906
Total Validation Accuracy Without O: 0.7788355701003432
```

```
Epochs: 4
------------------------------
Train Loss per 1000 steps: 0.093398 [ 1000/2397.9375]
Train Loss per 1000 steps: 0.091588 [ 2000/2397.9375]
Validation loss per 100 evaluation steps: 0.1412457975745201
Validation loss per 100 evaluation steps: 0.14211649749428035
Validation loss per 100 evaluation steps: 0.14105190861970185
Total Train Loss: 0.09059331273688026
Total Train Accuracy With O: 0.969866783441361
Total Train Accuracy Without O: 0.8441455873986892
Total Validation Loss: 0.14105190861970185
Total Validation Accuracy With O: 0.9588027270324776
Total Validation Accuracy Without O: 0.7752916602285052
Epochs: 5
------------------------------
Train Loss per 1000 steps: 0.086703 [ 1000/2397.9375]
Train Loss per 1000 steps: 0.083391 [ 2000/2397.9375]
Validation loss per 100 evaluation steps: 0.14253742039203643
Validation loss per 100 evaluation steps: 0.14136963131837546
Validation loss per 100 evaluation steps: 0.1393964004982263
Total Train Loss: 0.08192089112596687
Total Train Accuracy With O: 0.9728442163506116
Total Train Accuracy Without O: 0.8615830046159287
Total Validation Loss: 0.1393964004982263
Total Validation Accuracy With O: 0.9632010826491342
Total Validation Accuracy Without O: 0.8059281741557703
Epochs: 6
------------------------------
Train Loss per 1000 steps: 0.071500 [ 1000/2397.9375]
Train Loss per 1000 steps: 0.068578 [ 2000/2397.9375]
Validation loss per 100 evaluation steps: 0.14351764310151338
Validation loss per 100 evaluation steps: 0.14152177307754754
Validation loss per 100 evaluation steps: 0.1405267407745123
Total Train Loss: 0.06845852069760998
Total Train Accuracy With O: 0.9769524586234792
Total Train Accuracy Without O: 0.883641022339565
Total Validation Loss: 0.1405267407745123
Total Validation Accuracy With O: 0.9637565649175535
Total Validation Accuracy Without O: 0.805436881486579
```

Test:

Similar trend is seen in this case too. An improvement in loss and accuracy with O can be seen.

```
Test Loss: 0.14131849652683065
Test Accuracy: 0.9641340770597987
Test Accuracy Without O: 0.8061552460543501
```

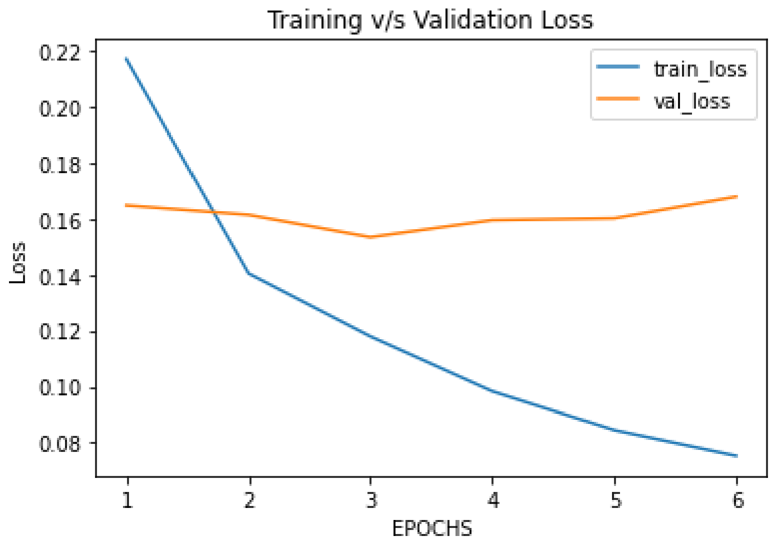
Analysis

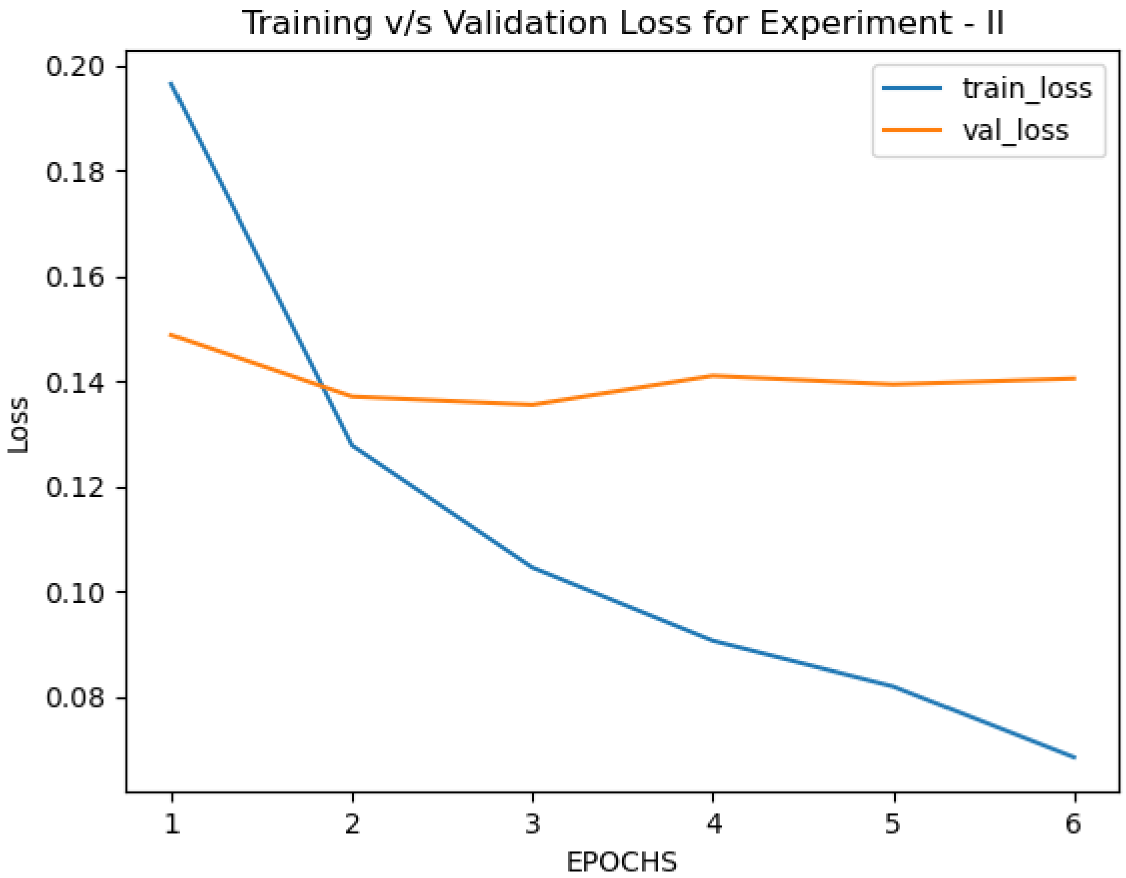
1) With and Without O

It can be clearly observed from the results that test accuracy is decreased when the label 'O' is not taken into consideration. This is due to the enormous amount of it in the dataset. Losses and accuracies seen in experiment 2 is better than 1 for both the cases because of the reduced redundancy of label prediction of the same subwords.

2) Training and Validation Loss

It can be clearly observed from the graphs that experiment II performs better over I. Furthermore, we can see from the graphs that validation is loss is almost constant for both the cases and training loss is reducing as the epochs is increasing. Validation loss can be reduced if model is trained for more epochs but due to the large size of the language model it is not possible to run on traditional devices. Similarly, training loss can be further decreased on training the model for more iteration.





3) Accuracy metrics

Below are the detailed classification reports for both the experiments. It can be seen that classification reports is more or less same for both the experiments.

Experiment - I

```
              precision    recall  f1-score   support

         art       0.30      0.13      0.18        55
         eve       0.17      0.12      0.14        17
         geo       0.80      0.87      0.83      4625
         gpe       0.96      0.89      0.93      1794
         nat       1.00      0.10      0.18        50
         org       0.71      0.65      0.68      2574
         per       0.73      0.76      0.74      2271
         tim       0.83      0.83      0.83      2115

   micro avg       0.79      0.80      0.80     13501
   macro avg       0.69      0.54      0.56     13501
weighted avg       0.79      0.80      0.79     13501
```

Experiment - II

```
              precision    recall  f1-score   support

         art       0.18      0.10      0.13        41
         eve       0.22      0.13      0.17        15
         geo       0.79      0.86      0.83      3788
         gpe       0.97      0.89      0.93      1636
         nat       1.00      0.14      0.25        28
         org       0.71      0.58      0.64      2003
         per       0.72      0.74      0.73      1676
         tim       0.86      0.83      0.84      2031

   micro avg       0.80      0.79      0.80     11218
   macro avg       0.68      0.54      0.56     11218
weighted avg       0.80      0.79      0.79     11218
```

Discussion

NER using BERT is successfully done in this assignment. As expected from the transfer learning paradigm, pretrained models when fine tuned on specific task gives much better results than the present SOTA neural language models.
We can see that validation loss is almost same for every epoch, this can be due to the smaller number of epochs model has been trained on. Since, the model is taking a long time to train on the full dataset 5-6 epoch came to be an optimal number for the model to train on.
Following is a sentence run through the trained model:

```
Sundar Pichai is the CEO of Google .
Tokenized Sentence: ['Sun', '##dar', 'Pi', '##cha', '##i', 'is', 'the', 'CEO', 'of', 'Google', '.']
Predicted Labels: ['B-per', 'B-per', 'I-per', 'I-per', 'I-per', 'O', 'O', 'O', 'O', 'B-org', 'O']
```

As,we can see from this example the sentence is first tokenized and the words which are OOV are further tokenized to subwords. Furthermore, model has correctly tagged the tokens with their respective entity i.e. the output is as expected on which is:

```
['B-per', 'I-per', 'O', 'O', 'O', 'O', 'B-org', 'O']
```

This matches with the model's output if we ignore the repetition of tokens through subwords. This effect of repetition is not seen in experiment 2 since we have only taken the first word of the subwords of a token to be in consideration and ignored the others.
The following example supports the arguement:

```
Sundar Pichai is the CEO of Google .
Tokenized Sentence: ['Sun', '##dar', 'Pi', '##cha', '##i', 'is', 'the', 'CEO', 'of', 'Google', '.']
Predicted Labels: ['B-per', 'I-per', 'O', 'O', 'O', 'O', 'B-org', 'O']
```

Jupyter Notebook Link

https://www.kaggle.com/code/gargguy/ell-881-a3

https://www.kaggle.com/code/yuggarg/ell881