Intro to NLP

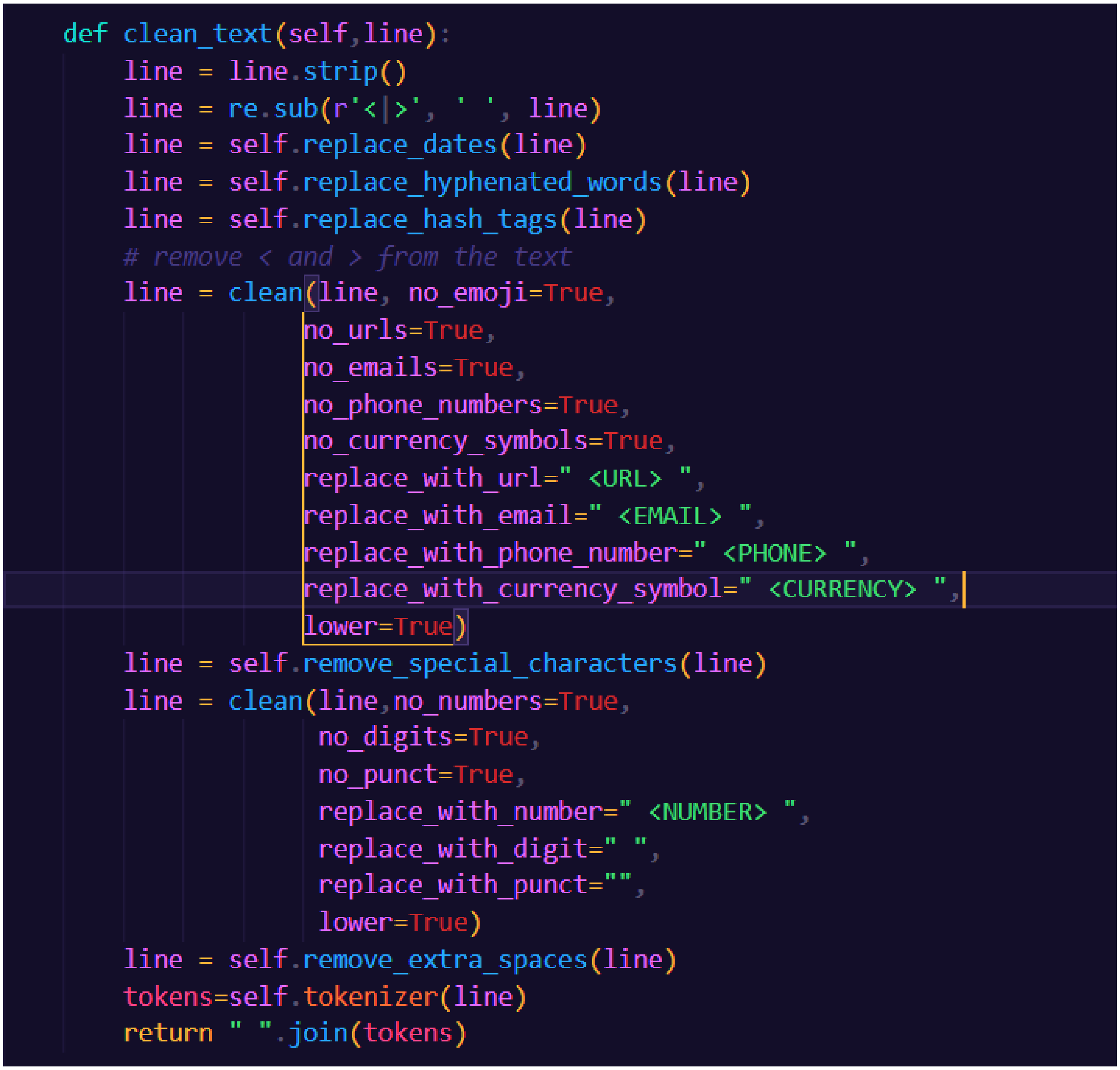
Assignment - 4

Soumodipta Bose

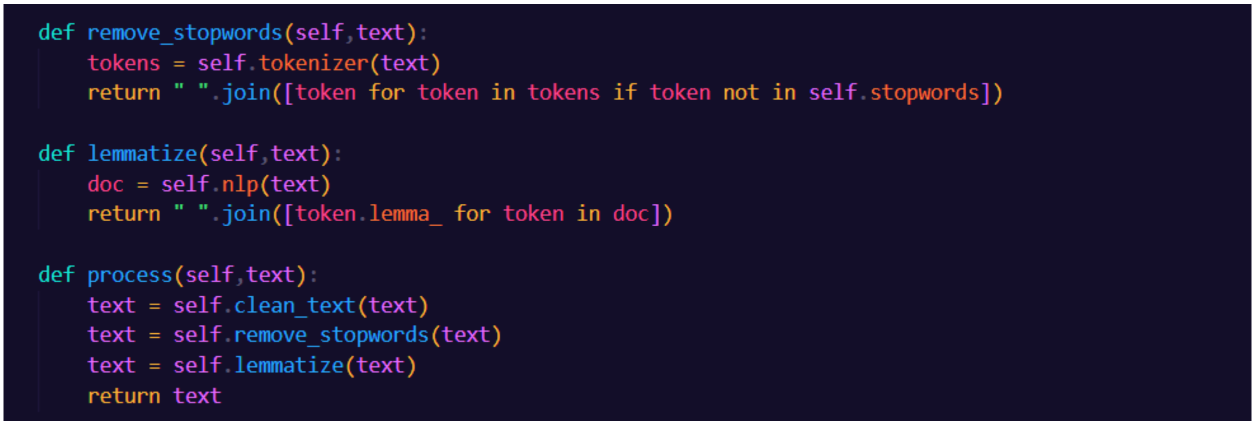
2021201086

Exploring Data and Pre-processing

The first step is cleaning the data, for that i am doing the following :

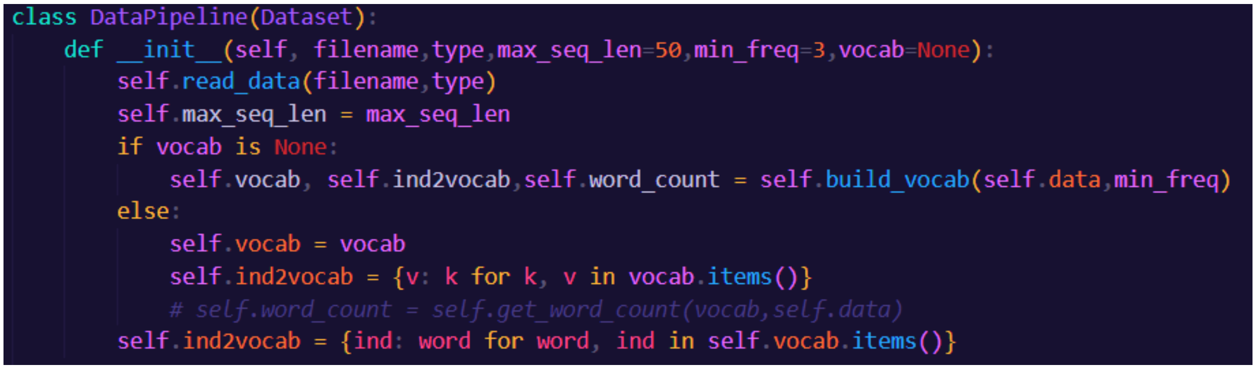


```
    def clean_text(self,line):
        line = line.strip()
        line = re.sub(r'<|>', ' ', line)
        line = self.replace_dates(line)
        line = self.replace_hyphenated_words(line)
        line = self.replace_hash_tags(line)
        # remove < and > from the text
        line = clean(line, no_emoji=True,
                    no_urls=True,
                    no_emails=True,
                    no_phone_numbers=True,
                    no_currency_symbols=True,
                    replace_with_url=" <URL> ",
                    replace_with_email=" <EMAIL> ",
                    replace_with_phone_number=" <PHONE> ",
                    replace_with_currency_symbol=" <CURRENCY> ",
                    lower=True)
        line = self.remove_special_characters(line)
        line = clean(line,no_numbers=True,
                     no_digits=True,
                     no_punct=True,
                     replace_with_number=" <NUMBER> ",
                     replace_with_digit=" ",
                     replace_with_punct="",
                     lower=True)
        line = self.remove_extra_spaces(line)
        tokens=self.tokenizer(line)
        return " ".join(tokens)
```



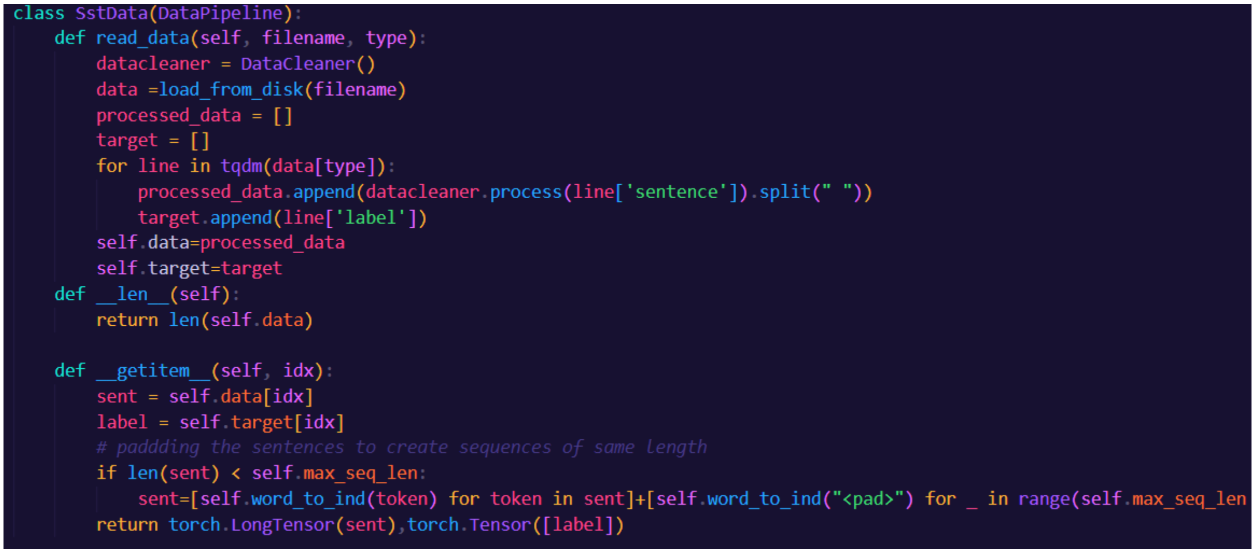
```python
def remove_stopwords(self,text):
    tokens = self.tokenizer(text)
    return " ".join([token for token in tokens if token not in self.stopwords])

def lemmatize(self,text):
    doc = self.nlp(text)
    return " ".join([token.lemma_ for token in doc])

def process(self,text):
    text = self.clean_text(text)
    text = self.remove_stopwords(text)
    text = self.lemmatize(text)
    return text
```

Next i am removing the stop words and lemmatizing the sentences.



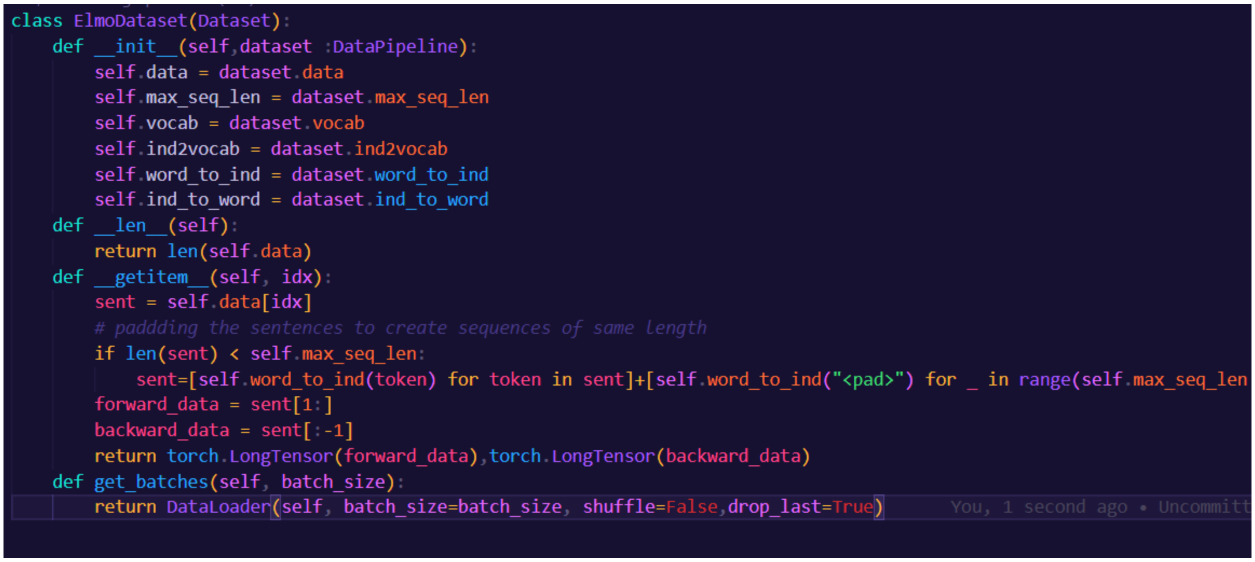
```python
class DataPipeline(Dataset):
    def __init__(self, filename,type,max_seq_len=50,min_freq=3,vocab=None):
        self.read_data(filename,type)
        self.max_seq_len = max_seq_len
        if vocab is None:
            self.vocab, self.ind2vocab,self.word_count = self.build_vocab(self.data,min_freq)
        else:
            self.vocab = vocab
            self.ind2vocab = {v: k for k, v in vocab.items()}
            # self.word_count = self.get_word_count(vocab,self.data)
        self.ind2vocab = {ind: word for word, ind in self.vocab.items()}
```

Using a common DataPipeline class from which i am creating two Dataset Specific dataset files for example SST data :



```python
class SstData(DataPipeline):
    def read_data(self, filename, type):
        datacleaner = DataCleaner()
        data =load_from_disk(filename)
        processed_data = []
        target = []
        for line in tqdm(data[type]):
            processed_data.append(datacleaner.process(line['sentence']).split(" "))
            target.append(line['label'])
        self.data=processed_data
        self.target=target
    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        sent = self.data[idx]
        label = self.target[idx]
        # paddding the sentences to create sequences of same length
        if len(sent) < self.max_seq_len:
            sent=[self.word_to_ind(token) for token in sent]+[self.word_to_ind("<pad>") for _ in range(self.max_seq_len
        return torch.LongTensor(sent),torch.Tensor([label])
```

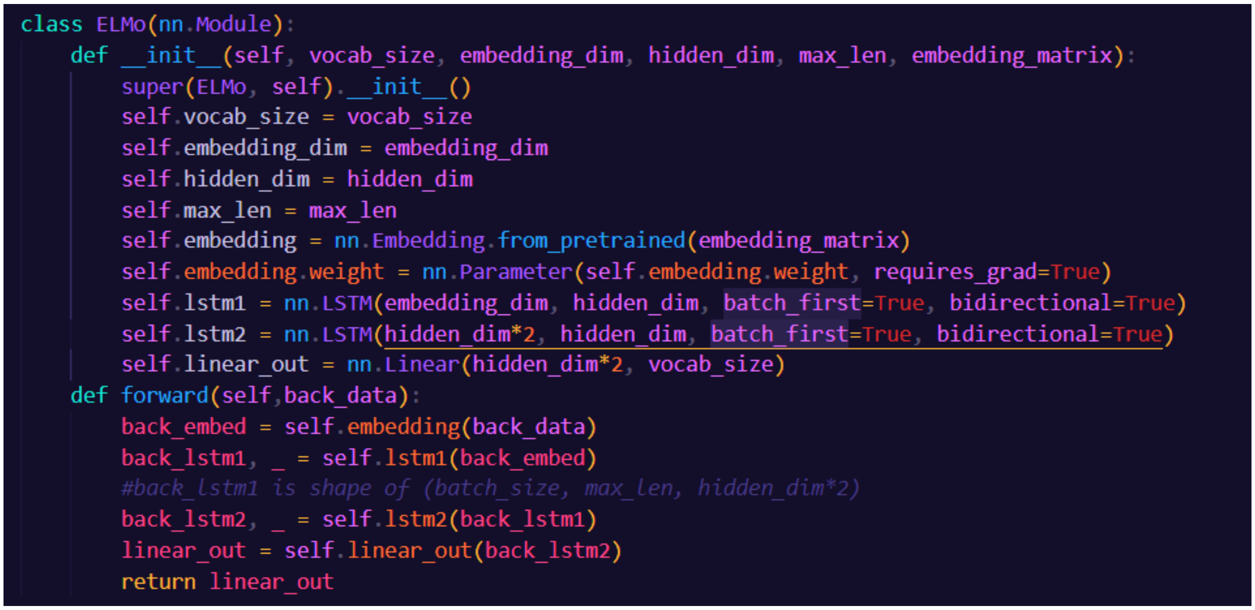
Since the Elmo Dataset needs to be created for both the dataset i am creating a common interface to convert the dataset to a elmo compatible dataset for training.



```
class ElmoDataset(Dataset):
    def __init__(self,dataset :DataPipeline):
        self.data = dataset.data
        self.max_seq_len = dataset.max_seq_len
        self.vocab = dataset.vocab
        self.ind2vocab = dataset.ind2vocab
        self.word_to_ind = dataset.word_to_ind
        self.ind_to_word = dataset.ind_to_word
    def __len__(self):
        return len(self.data)
    def __getitem__(self, idx):
        sent = self.data[idx]
        # paddding the sentences to create sequences of same length
        if len(sent) < self.max_seq_len:
            sent=[self.word_to_ind(token) for token in sent]+[self.word_to_ind("<pad>") for _ in range(self.max_seq_len
        forward_data = sent[1:]
        backward_data = sent[:-1]
        return torch.LongTensor(forward_data),torch.LongTensor(backward_data)
    def get_batches(self, batch_size):
        return DataLoader(self, batch_size=batch_size, shuffle=False,drop_last=True)      You, 1 second ago • Uncommitt
```

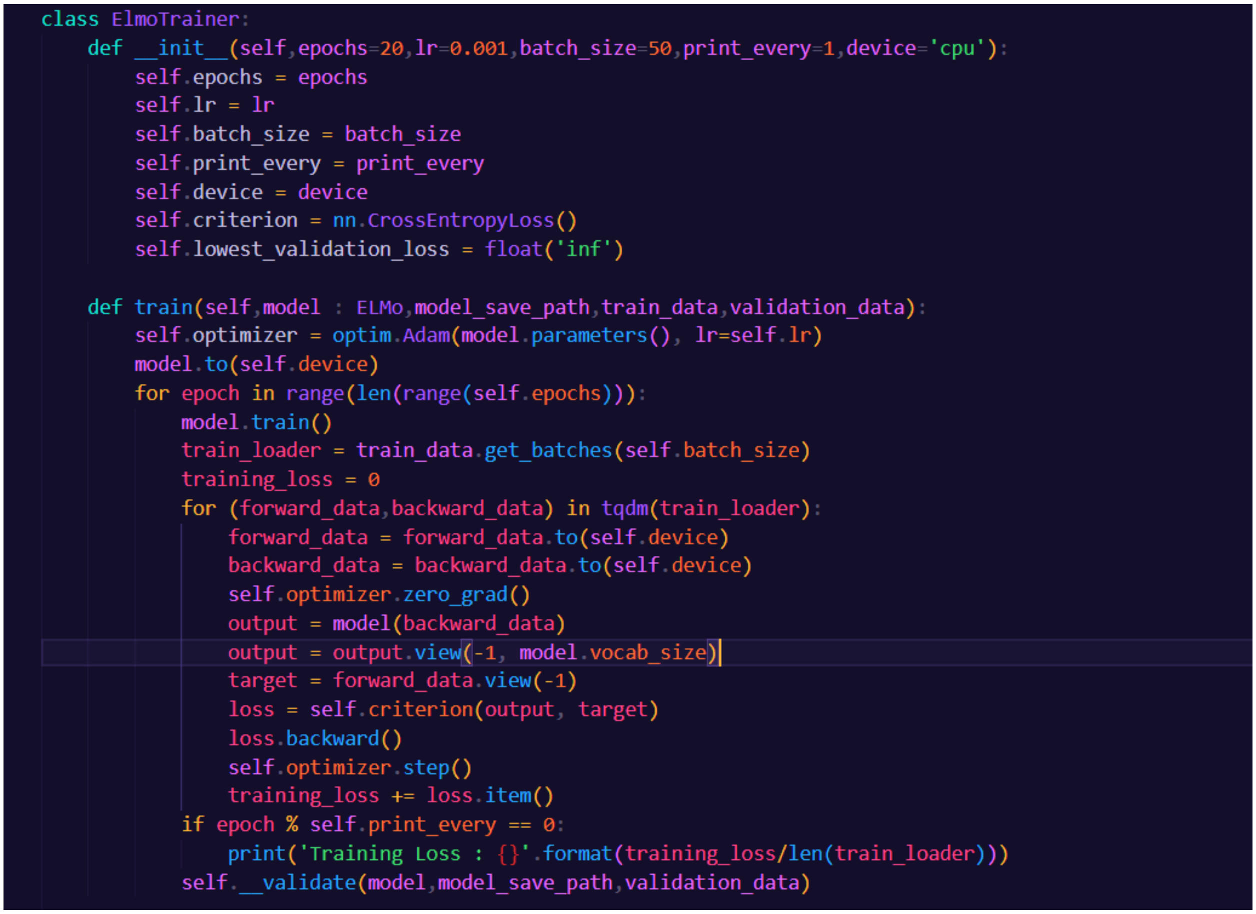
Model Creation

The model was created using Pytorch. The model consisted of an initial embedding layer that used pre-trained embedding embeddings using Glove. Then was fed into an LSTM and finally a Linear Fully connected layer with output dimensions of the tag set size.

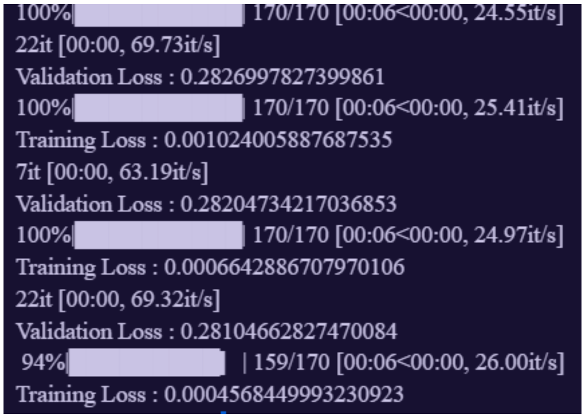


```
class ELMo(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, max_len, embedding_matrix):
        super(ELMo, self).__init__()
        self.vocab_size = vocab_size
        self.embedding_dim = embedding_dim
        self.hidden_dim = hidden_dim
        self.max_len = max_len
        self.embedding = nn.Embedding.from_pretrained(embedding_matrix)
        self.embedding.weight = nn.Parameter(self.embedding.weight, requires_grad=True)
        self.lstm1 = nn.LSTM(embedding_dim, hidden_dim, batch_first=True, bidirectional=True)
        self.lstm2 = nn.LSTM(hidden_dim*2, hidden_dim, batch_first=True, bidirectional=True)
        self.linear_out = nn.Linear(hidden_dim*2, vocab_size)
    def forward(self,back_data):
        back_embed = self.embedding(back_data)
        back_lstm1, _ = self.lstm1(back_embed)
        #back_lstm1 is shape of (batch_size, max_len, hidden_dim*2)
        back_lstm2, _ = self.lstm2(back_lstm1)
        linear_out = self.linear_out(back_lstm2)
        return linear_out
```

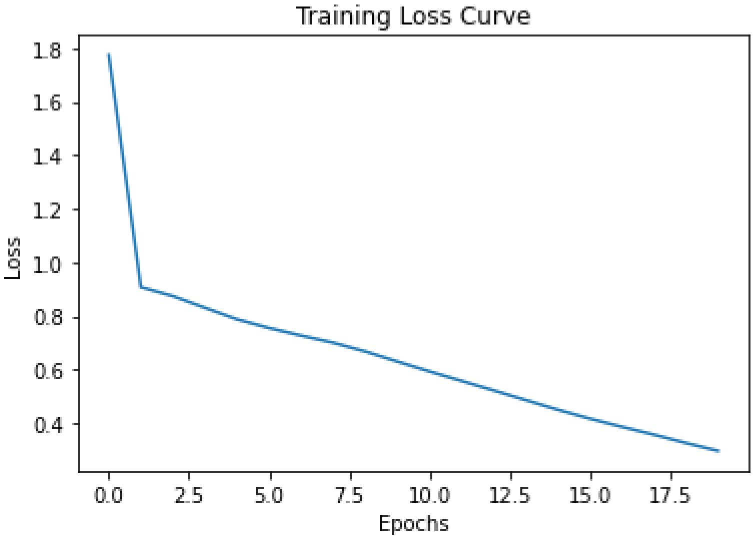
Training the model



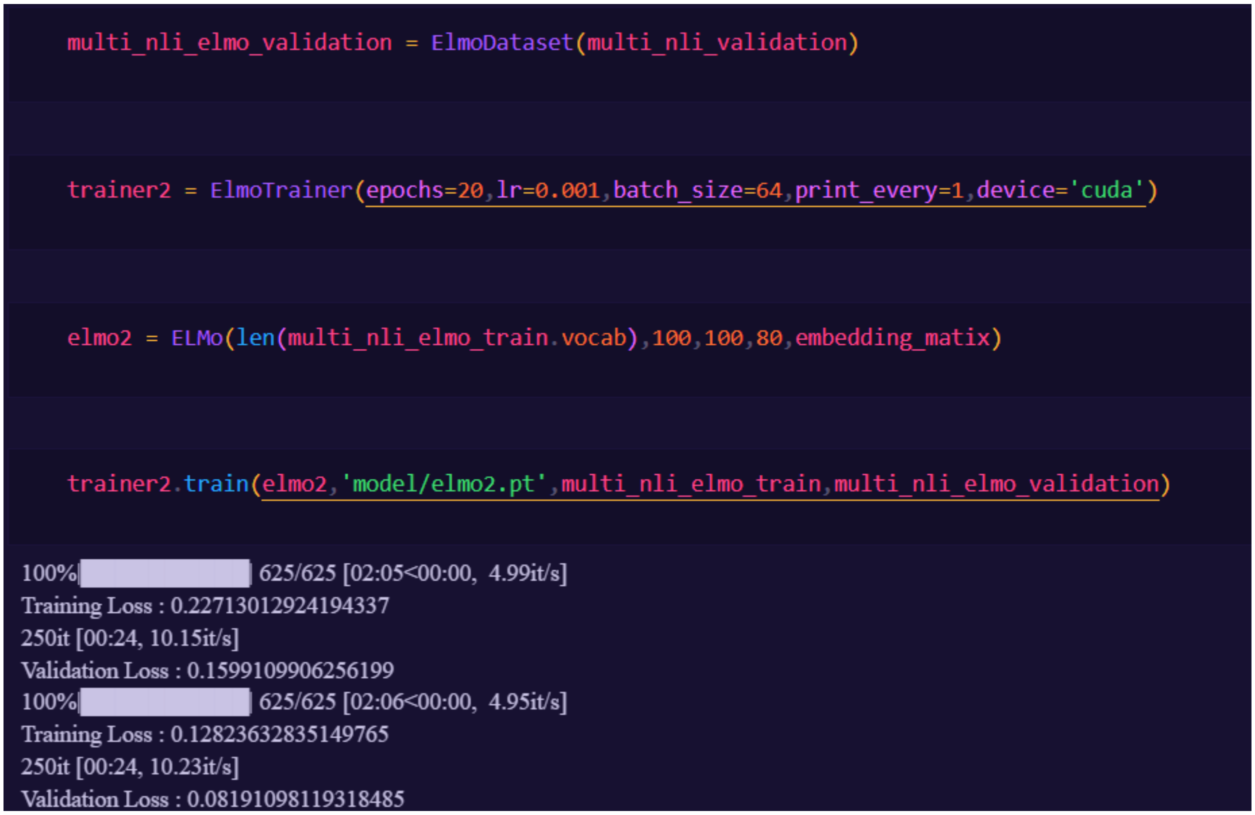
```
class ElmoTrainer:
    def __init__(self,epochs=20,lr=0.001,batch_size=50,print_every=1,device='cpu'):
        self.epochs = epochs
        self.lr = lr
        self.batch_size = batch_size
        self.print_every = print_every
        self.device = device
        self.criterion = nn.CrossEntropyLoss()
        self.lowest_validation_loss = float('inf')

    def train(self,model : ELMo,model_save_path,train_data,validation_data):
        self.optimizer = optim.Adam(model.parameters(), lr=self.lr)
        model.to(self.device)
        for epoch in range(len(range(self.epochs))):
            model.train()
            train_loader = train_data.get_batches(self.batch_size)
            training_loss = 0
            for (forward_data,backward_data) in tqdm(train_loader):
                forward_data = forward_data.to(self.device)
                backward_data = backward_data.to(self.device)
                self.optimizer.zero_grad()
                output = model(backward_data)
                output = output.view(-1, model.vocab_size)
                target = forward_data.view(-1)
                loss = self.criterion(output, target)
                loss.backward()
                self.optimizer.step()
                training_loss += loss.item()
            if epoch % self.print_every == 0:
                print('Training Loss : {}'.format(training_loss/len(train_loader)))
            self.__validate(model,model_save_path,validation_data)
```



```
100%|██████████| 170/170 [00:06<00:00, 24.55it/s]
22it [00:00, 69.73it/s]
Validation Loss : 0.2826997827399861
100%|██████████| 170/170 [00:06<00:00, 25.41it/s]
Training Loss : 0.001024005887687535
7it [00:00, 63.19it/s]
Validation Loss : 0.28204734217036853
100%|██████████| 170/170 [00:06<00:00, 24.97it/s]
Training Loss : 0.0006642886707970106
22it [00:00, 69.32it/s]
Validation Loss : 0.28104662827470084
 94%|█████████  | 159/170 [00:06<00:00, 26.00it/s]
Training Loss : 0.0004568449993230923
```



We will do the same for the other dataset i.e. Multi NLI

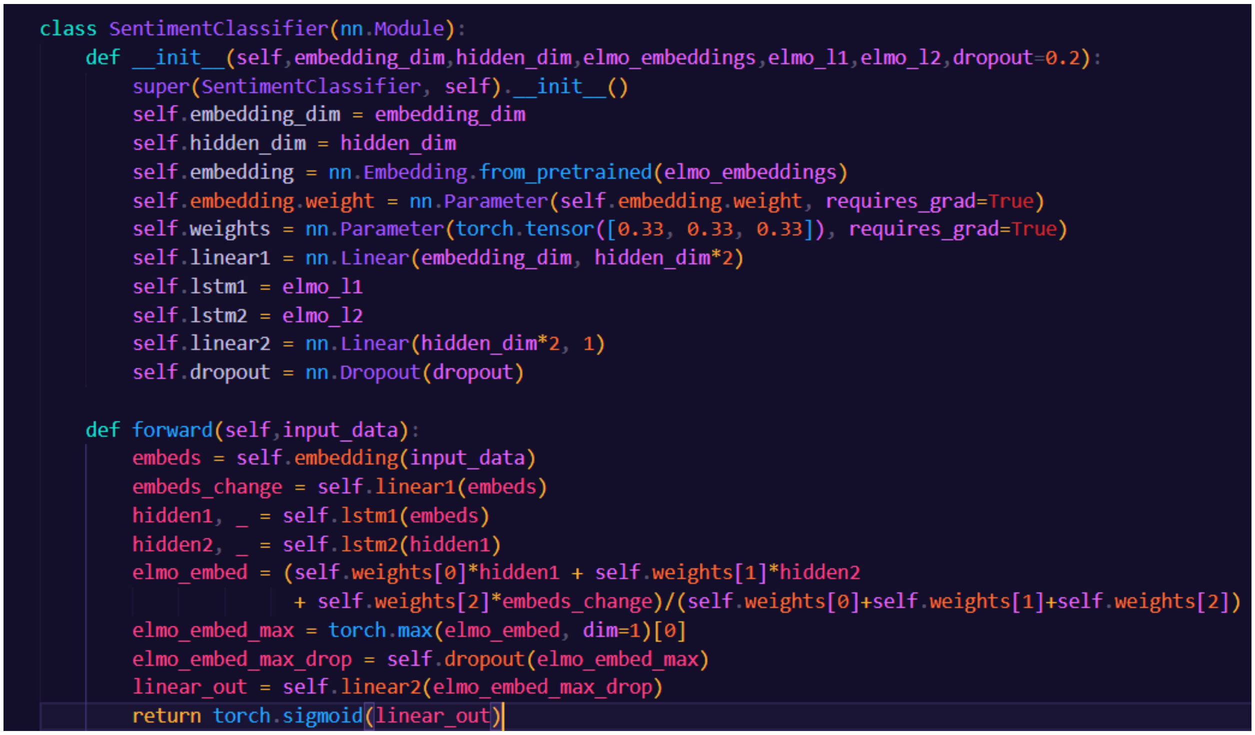


```
multi_nli_elmo_validation = ElmoDataset(multi_nli_validation)

trainer2 = ElmoTrainer(epochs=20,lr=0.001,batch_size=64,print_every=1,device='cuda')

elmo2 = ELMo(len(multi_nli_elmo_train.vocab),100,100,80,embedding_matix)

trainer2.train(elmo2,'model/elmo2.pt',multi_nli_elmo_train,multi_nli_elmo_validation)
```

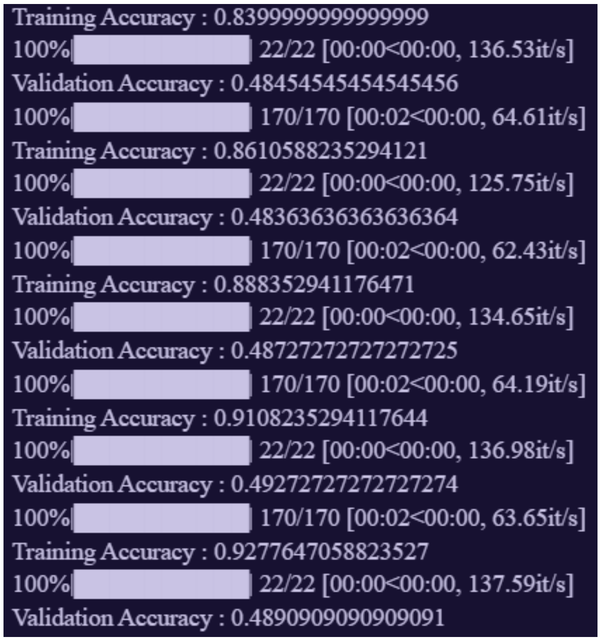
```
100%|██████████| 625/625 [02:05<00:00,  4.99it/s]
Training Loss : 0.22713012924194337
250it [00:24, 10.15it/s]
Validation Loss : 0.1599109906256199
100%|██████████| 625/625 [02:06<00:00,  4.95it/s]
Training Loss : 0.12823632835149765
250it [00:24, 10.23it/s]
Validation Loss : 0.08191098119318485
```

Sentiment Analysis Task

We use the elmo embeddings and the corresponding lstm layers in our Sentiment Classification model and train it after fiddling with the hyper-parameters.

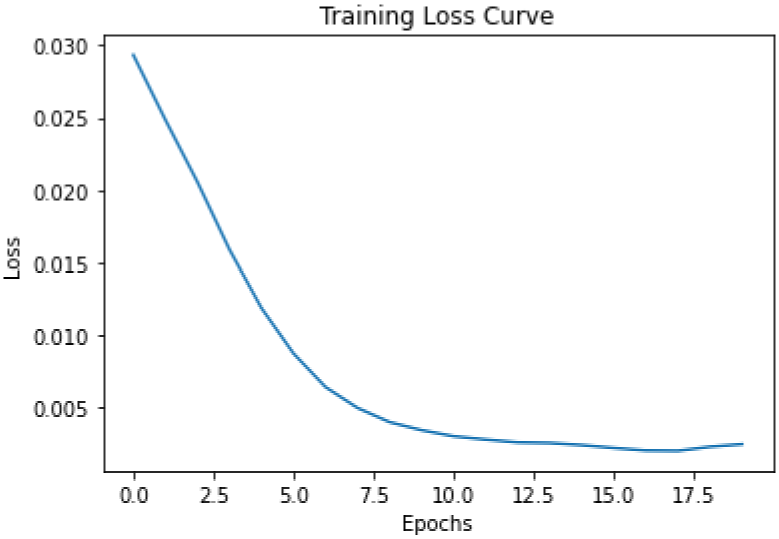


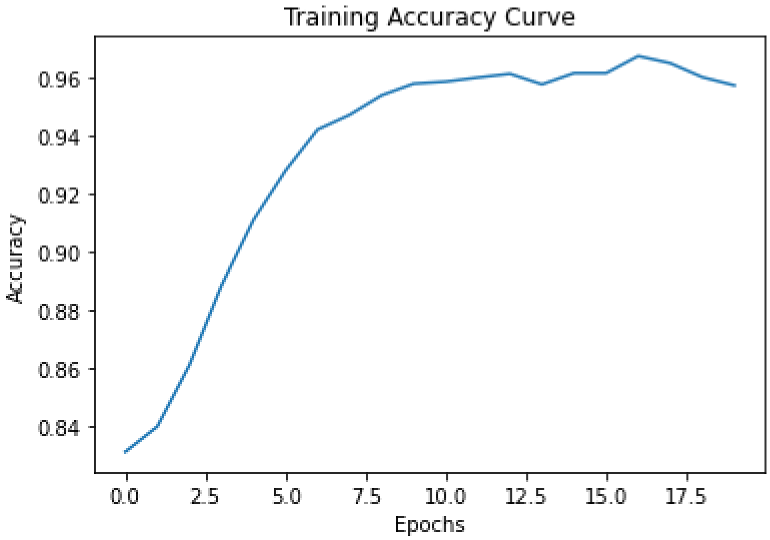
```python
class SentimentClassifier(nn.Module):
    def __init__(self,embedding_dim,hidden_dim,elmo_embeddings,elmo_l1,elmo_l2,dropout=0.2):
        super(SentimentClassifier, self).__init__()
        self.embedding_dim = embedding_dim
        self.hidden_dim = hidden_dim
        self.embedding = nn.Embedding.from_pretrained(elmo_embeddings)
        self.embedding.weight = nn.Parameter(self.embedding.weight, requires_grad=True)
        self.weights = nn.Parameter(torch.tensor([0.33, 0.33, 0.33]), requires_grad=True)
        self.linear1 = nn.Linear(embedding_dim, hidden_dim*2)
        self.lstm1 = elmo_l1
        self.lstm2 = elmo_l2
        self.linear2 = nn.Linear(hidden_dim*2, 1)
        self.dropout = nn.Dropout(dropout)

    def forward(self,input_data):
        embeds = self.embedding(input_data)
        embeds_change = self.linear1(embeds)
        hidden1, _ = self.lstm1(embeds)
        hidden2, _ = self.lstm2(hidden1)
        elmo_embed = (self.weights[0]*hidden1 + self.weights[1]*hidden2
                      + self.weights[2]*embeds_change)/(self.weights[0]+self.weights[1]+self.weights[2])
        elmo_embed_max = torch.max(elmo_embed, dim=1)[0]
        elmo_embed_max_drop = self.dropout(elmo_embed_max)
        linear_out = self.linear2(elmo_embed_max_drop)
        return torch.sigmoid(linear_out)
```



```
Training Accuracy : 0.8399999999999999
100%|██████████| 22/22 [00:00<00:00, 136.53it/s]
Validation Accuracy : 0.48454545454545456
100%|██████████| 170/170 [00:02<00:00, 64.61it/s]
Training Accuracy : 0.8610588235294121
100%|██████████| 22/22 [00:00<00:00, 125.75it/s]
Validation Accuracy : 0.48363636363636364
100%|██████████| 170/170 [00:02<00:00, 62.43it/s]
Training Accuracy : 0.888352941176471
100%|██████████| 22/22 [00:00<00:00, 134.65it/s]
Validation Accuracy : 0.48727272727272725
100%|██████████| 170/170 [00:02<00:00, 64.19it/s]
Training Accuracy : 0.9108235294117644
100%|██████████| 22/22 [00:00<00:00, 136.98it/s]
Validation Accuracy : 0.49272727272727274
100%|██████████| 170/170 [00:02<00:00, 63.65it/s]
Training Accuracy : 0.9277647058823527
100%|██████████| 22/22 [00:00<00:00, 137.59it/s]
Validation Accuracy : 0.4890909090909091
```

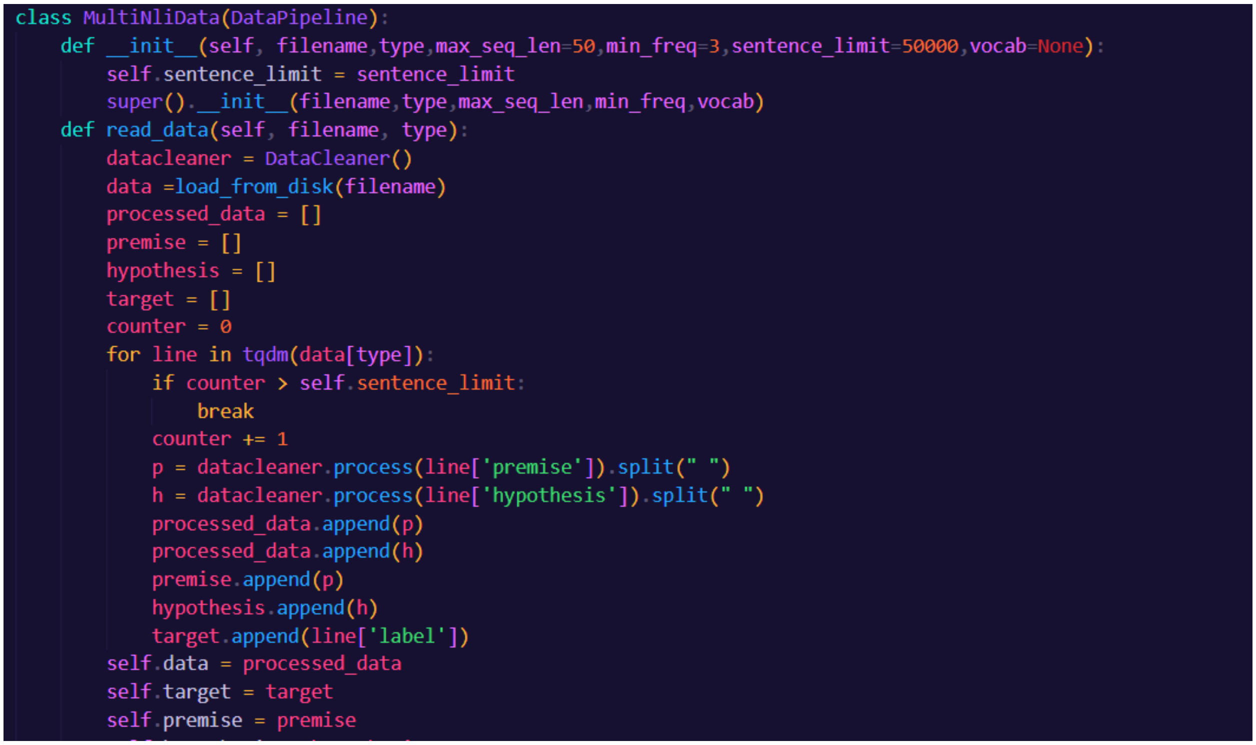
We plotted the training losses and accuracy. Classification report not possible as inititally it was a regression task so instead approximated using a threshold.



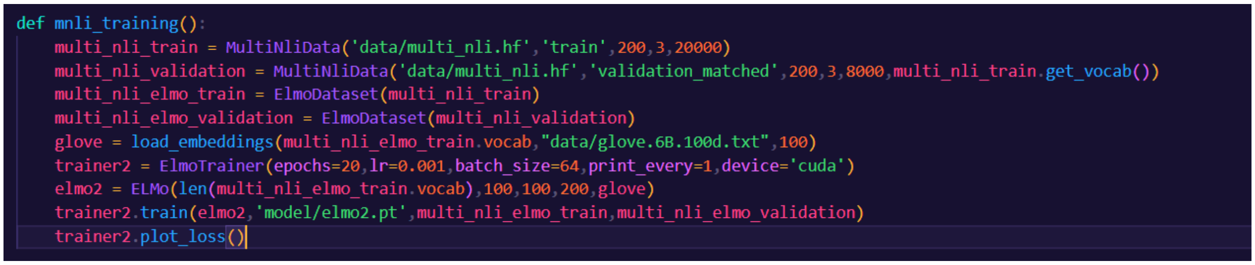

Training Accuracy Curve
Accuracy
0.96
0.94
0.92
0.90
0.88
0.86
0.84
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epochs

Natural Language Inference

For this task I have just created the Elmo Embeddings not created any model.



```python
class MultiNliData(DataPipeline):
    def __init__(self, filename,type,max_seq_len=50,min_freq=3,sentence_limit=50000,vocab=None):
        self.sentence_limit = sentence_limit
        super().__init__(filename,type,max_seq_len,min_freq,vocab)
    def read_data(self, filename, type):
        datacleaner = DataCleaner()
        data =load_from_disk(filename)
        processed_data = []
        premise = []
        hypothesis = []
        target = []
        counter = 0
        for line in tqdm(data[type]):
            if counter > self.sentence_limit:
                break
            counter += 1
            p = datacleaner.process(line['premise']).split(" ")
            h = datacleaner.process(line['hypothesis']).split(" ")
            processed_data.append(p)
            processed_data.append(h)
            premise.append(p)
            hypothesis.append(h)
            target.append(line['label'])
        self.data = processed_data
        self.target = target
        self.premise = premise
```



```python
def mnli_training():
    multi_nli_train = MultiNliData('data/multi_nli.hf','train',200,3,20000)
    multi_nli_validation = MultiNliData('data/multi_nli.hf','validation_matched',200,3,8000,multi_nli_train.get_vocab())
    multi_nli_elmo_train = ElmoDataset(multi_nli_train)
    multi_nli_elmo_validation = ElmoDataset(multi_nli_validation)
    glove = load_embeddings(multi_nli_elmo_train.vocab,"data/glove.6B.100d.txt",100)
    trainer2 = ElmoTrainer(epochs=20,lr=0.001,batch_size=64,print_every=1,device='cuda')
    elmo2 = ELMo(len(multi_nli_elmo_train.vocab),100,100,200,glove)
    trainer2.train(elmo2,'model/elmo2.pt',multi_nli_elmo_train,multi_nli_elmo_validation)
    trainer2.plot_loss()
```

Conclusion

This assignment taught us a lot.