INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

Introduction to NLP (CS7.401)

Assignment 4

Submitted by:

Bhanuj Gandhi

2022201068

April 27, 2023

Contents

1 ELMo

1.1 Introduction

ELMo (Embeddings from Language Models) is a deep contextualized word representation model. Unlike traditional word embedding models, ELMo is capable of capturing the contextual meaning of words by taking into account the entire sentence in which they appear.

The ELMo architecture contains forward and backward language models trained through its Bi-LSTMs. The stacked Bi-LSTM-based langauge model has a layer of non-contextual word embeddings (like word2vec, or a character convolutional network as in the original ELMo) at the input. While a forward language model can be trained by the next word prediction task, for a backward language model - to put it simply - it's nothing but previous word prediction task. In either of the cases, the LSTM model makes the word prediction based on the context. By context, we mean all the words preceeding (forward LM) or succeeding (backward LM) the word to be predicted in the sentence.

By training these Bi-LSTMs using the language modeling objective, we essen- tially pretrain the weights of the ELMo architecture, making it suitable for application on downstream tasks. To break this down, when we pretrain the model, we make the model learn the nature of the language itself. The model does not really learn any useful task in this process - instead it captures the gen- eral dependencies in the language, which makes it smarter to be able to solve any task thrown at it later quickly. This task is what we refer to as the down- stream task. On pretraining the model, we expect it to give good performance on a given NLP task with lesser training and data than it would take without pretraining.

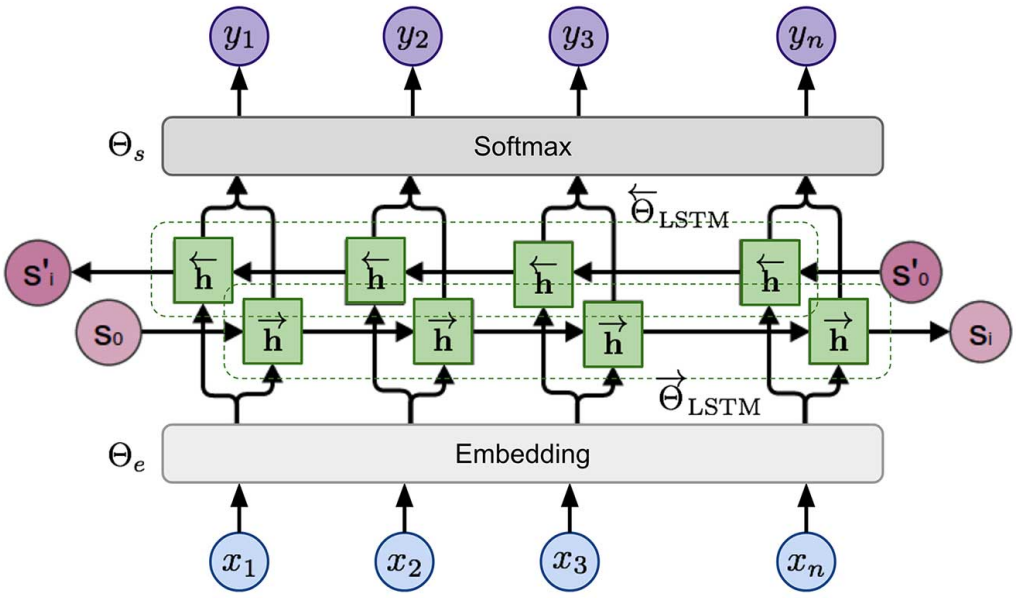

Figure 1: ELMo Architecture

1.2 Datasets

Datasets Used :

- **Stanford Sentiment Treebank:** It contains 8544 train, 1101 validation and 2210 test sentences. Each complete sentence is annotated with a foat label that indicates its level of positive sentiment from 0.0 to 1.0. To convert the given task into a binary sentiment classifcation task, we need to assign a label of either 0 or 1 to the generated elmo embedding using MLP classifer.
- **Multi-Genre NLI Corpus :** MultiNLI is a comprehensive dataset comprising more than 400,000 sentence pairs (hypothesis and premise), each of which is categorized as either entailment, contradiction, or neutral. Similarly in this corpus, we have to classify each elmo embedding of a particular sentence into three categories.

2 Implementation

2.1 Preprocessing

All the preprocessing steps are explained in the code comments below

```
def preprocess_sentence(sent: str) -> str:
    # Remove HTML
    soup = BeautifulSoup(sent, "html.parser")
    sent = soup.get_text(separator=" ")

    # Remove whitespaces
    sent = sent.strip()
    sent = " ".join(sent.split())

    # Lowercase
    sent = sent.lower()

    # Remove accent characters
    sent = unidecode.unidecode(sent)

    # Expand the contractions
    sent = contractions.fix(sent)

    # Remove punctuations and non-ASCII characters
    sent = re.sub(r"[^\w\s]", "", sent)
    sent = re.sub(r"[^\x00-\x7f]", "", sent)

    # Remove stopwords and lemmatize
    sent = " ".join(
        [lemmatizer.lemmatize(word) for word in sent.split() if word not in
    stop_words]
    )

    return sent
```

2.2 Build the Elmo model

I have used two Bi-LSTM stacked and passed the input. These layers are trained on the prediction of the next word.

```
class ELMo(nn.Module):
    def __init__(self, vocab_size, embedding_size, hidden_size, dropout,
    embeddings):
        super(ELMo, self).__init__()
        self.embedding = nn.Embedding.from_pretrained(embeddings)

        self.layer_1 = nn.LSTM(
            input_size=embedding_size,
            hidden_size=hidden_size,
            num_layers=1,
            batch_first=True,
            bidirectional=True,
        )

        self.layer_2 = nn.LSTM(
            input_size=2 * hidden_size,
```

```
            hidden_size=hidden_size,
            num_layers=1,
            batch_first=True,
            bidirectional=True,
        )

        self.dropout = nn.Dropout(dropout)

        self.linear = nn.Linear(hidden_size * 2, vocab_size)

    def forward(self, X):
        embeddings = self.embedding(X)

        lstm1_output, _ = self.layer_1(embeddings)

        lstm2_output, _ = self.layer_2(lstm1_output)
        lstm2_output = self.dropout(lstm2_output)

        output = self.linear(lstm2_output)
        output = torch.transpose(output, 1, 2)
        return output
```

2.3 PreTraining the Elmo model

```

class Sentiment_Classifier(nn.Module):
    def __init__(self, embedding_size):
        super(Sentiment_Classifier, self).__init__()

        self.s1 = nn.Parameter(torch.ones(1))
        self.s2 = nn.Parameter(torch.ones(1))
        self.s3 = nn.Parameter(torch.ones(1))
        self.alpha = nn.Parameter(torch.ones(1))

        self.linear = nn.Linear(embedding_size, 2)

        self.sigmoid = nn.Sigmoid()

    def forward(self, sentence):
        embeddings = elmo.embedding(sentence)
        out_1, _ = elmo.lstm1(embeddings)
        out_2, _ = elmo.lstm2(out_1)

        s_sum = self.s1 + self.s2 + self.s3

        output = self.alpha * (
            self.s1 / s_sum * embeddings
            + self.s2 / s_sum * out_1
            + self.s3 / s_sum * out_2
        ).to(torch.float32)

        output = self.linear(output)
        output = output.mean(dim=1)
        output = self.sigmoid(output)

        return output
```

2.4 Evaluate the Elmo model

```
1
2  sentiment_model = Sentiment_Classifier(100).to(device)
3  criterion = nn.CrossEntropyLoss().to(device)
4  optimizer = optim.Adam(sentiment_model.parameters(), lr=0.01)
5
6  prev_val = np.inf
7  for epoch in range(20):
8      total_loss = 0
9      total_loss_train = 0
10     for i, (sentence, label) in enumerate(tqdm(train_dataloader)):
11         sentiment_model.train()
12         sentence, label = sentence.to(device), label.to(device)
13
14         optimizer.zero_grad()
15         outputs = sentiment_model(sentence)
16
17         label = (label >= 0.5).long()
18         batch_loss = criterion(outputs, label)
19         total_loss_train += batch_loss.item()
20
21         batch_loss.backward()
22         optimizer.step()
23
24     print(f"Train Loss: {total_loss_train/len(train_dataloader)}")
25     with torch.no_grad():
26         sentiment_model.eval()
27         for inputs, targets in tqdm(test_dataloader):
28             model_input = inputs.to(device)
29             targets = targets.to(device)
30             out = sentiment_model(model_input)
31
32             targets = (targets >= 0.5).long()
33             loss = criterion(out, targets)
34             total_loss += loss.item()
35
36     if prev_val < total_loss:
37         break
38     else:
39         prev_val = total_loss
40
41     print(f"Validation Loss {total_loss/len(val_dataloader)}")
```

3 Visualize the results

3.1 SST Dataset

Classification Report:

	precision	recall	f1-score	support
0	0.70	0.78	0.74	1143
1	0.73	0.64	0.68	1067
accuracy			0.71	2210
macro avg	0.72	0.71	0.71	2210
weighted avg	0.71	0.71	0.71	2210

Figure 2: Classfication Report for SST

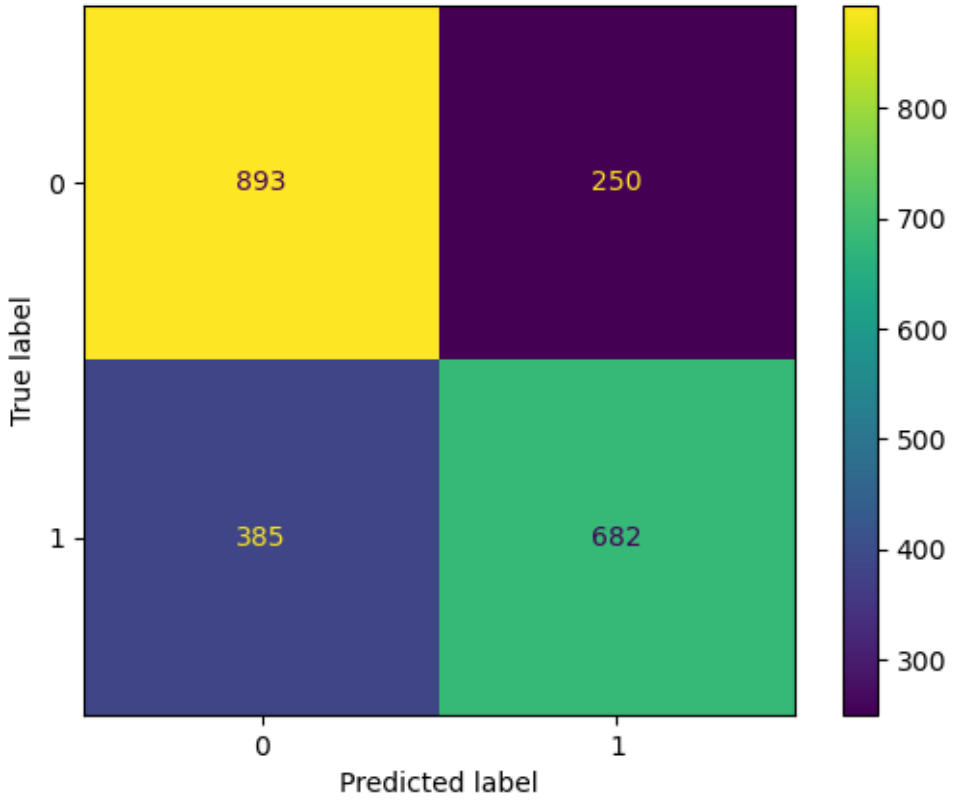

Figure 3: Confusion Matrix for SST

Positive Label - Positive Sentiment
Negative Label - Negative Sentiment

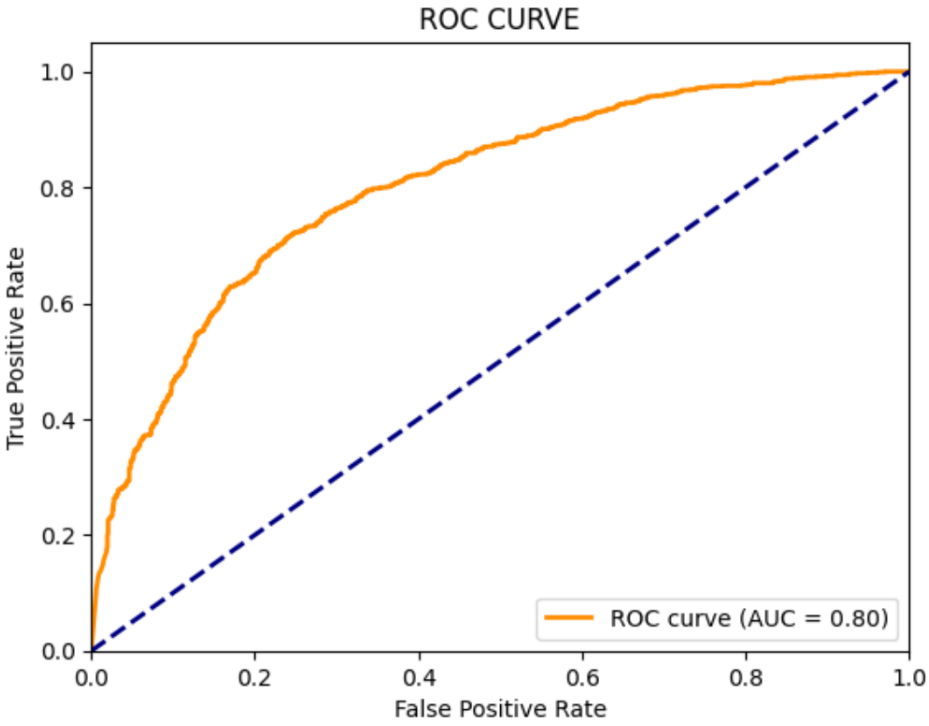

Figure 4: ROC Curve for SST Dataset

3.2 Multi NLI Dataset

```
Classification Report:
              precision    recall  f1-score   support

           0       0.46      0.36      0.40      3479
           1       0.42      0.47      0.45      3123
           2       0.48      0.54      0.51      3213

    accuracy                           0.46      9815
   macro avg       0.46      0.46      0.45      9815
weighted avg       0.46      0.46      0.45      9815
```

Figure 5: Classfication Report for Multi NLI

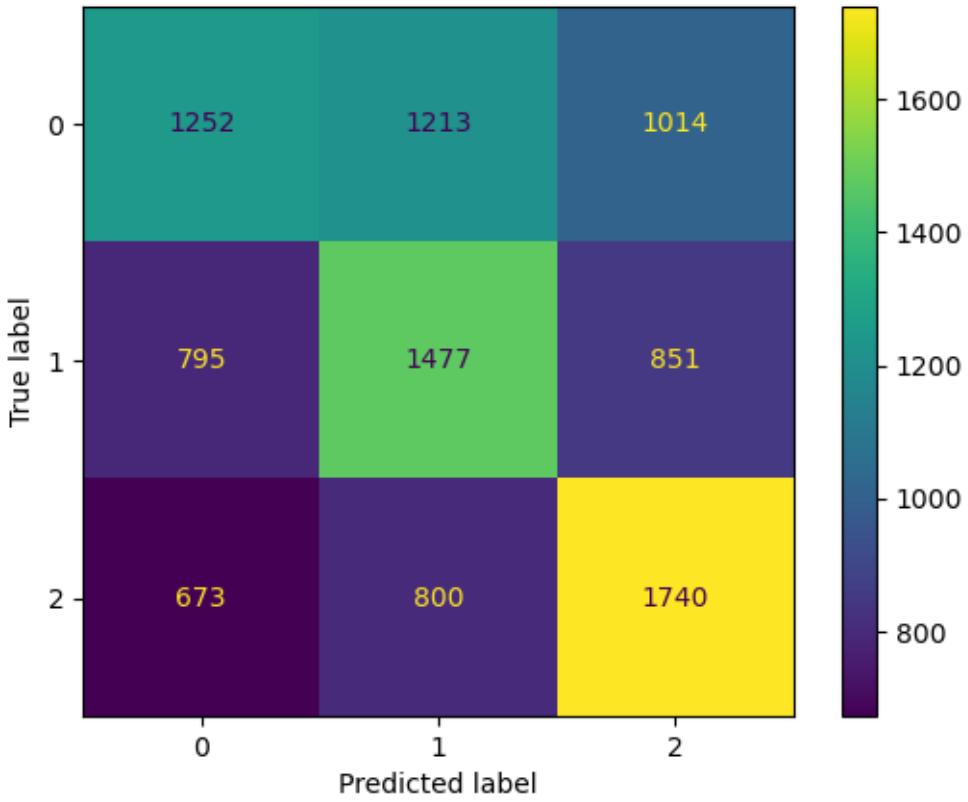

Figure 6: Confusion Matrix for Multi NLI

Positive Label - Entailment
Negative Label - Neutral & Contradiction

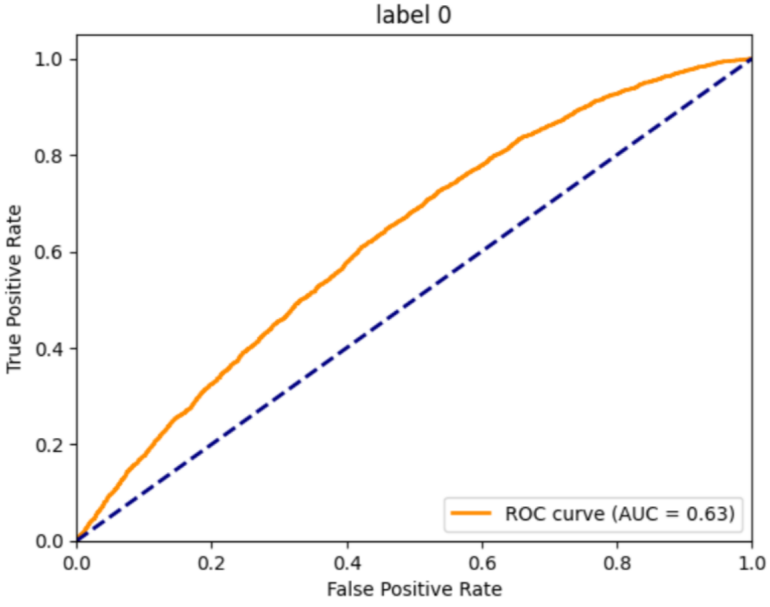

Figure 7: ROC Curve for MultiNLI Dataset

Positive Label - Neutral
Negative Label - Entailment & Contradiction

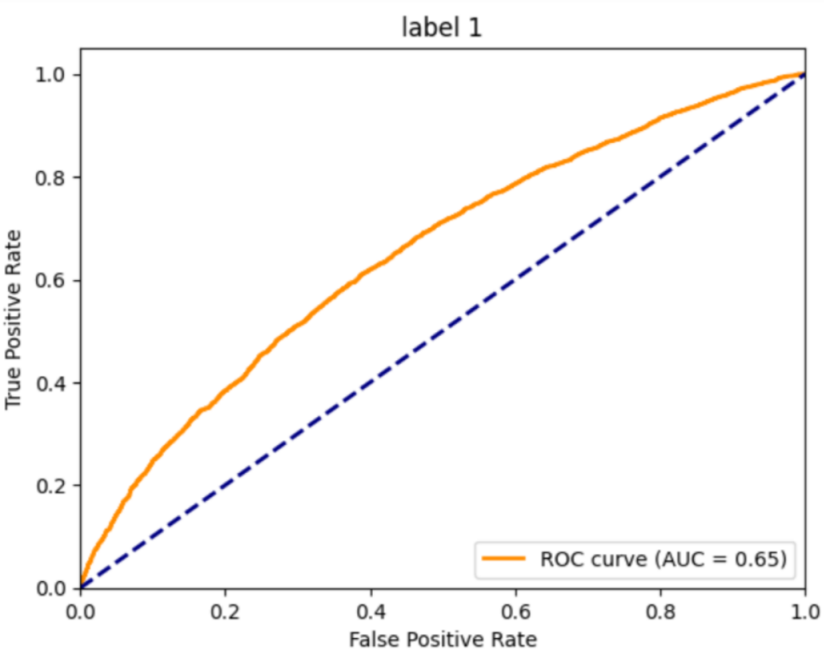

Figure 8: ROC Curve for MultiNLI Dataset

Positive Label - Contradiction
Negative Label - Entailment & Neutral

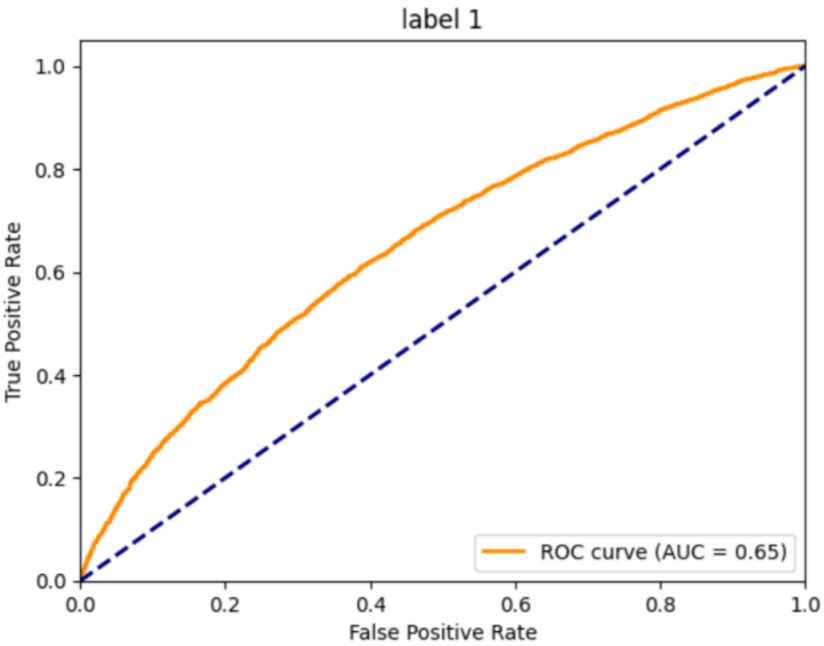

Figure 9: ROC Curve for MultiNLI Dataset

4 Conclusion

In this assignment, we analyze the performance of a deep learning model on two distinct NLP tasks: sentiment analysis and natural language inference (NLI). The aim is to assess the model's ability to accurately classify sentiment in movie reviews and determine the relationships between pairs of sentences. We utilize ELMo embeddings, which capture rich contextual information, to enhance the model's understanding of the text.

The sentiment analysis task involves classifying movie reviews as positive or negative. We evaluate the model's performance on the Stanford Sentiment Treebank (SST) dataset, which provides labeled movie reviews. For the NLI task, we use the Multi-NLI dataset, which requires determining the relationship (entailment, contradiction, or neutral) between two sentences.

Our experiments reveal that the sentiment analysis task achieves a remarkable test accuracy of 71% on the SST dataset. Additionally, the model exhibits an impressive AUC score of 0.80 for the receiver operating characteristic (ROC) curve. These findings demonstrate the model's ability to effectively distinguish between positive and negative movie reviews. The high accuracy and AUC score can be attributed to the clear distinction between the two sentiment classes in the dataset.

In contrast, the NLI task presents a more challenging scenario, where the model struggled to accurately classify the relationships between sentence pairs. The performance on the Multi-NLI dataset yields a lower test accuracy of 46% and an AUC score of 0.68 for the ROC curve. This suggests that the model faced difficulties in capturing the subtle differences that exist between sentences for entailment, contradiction, and neutral relationships. The complexity of the NLI task, coupled with the large number of samples in the dataset, presented a challenging learning environment for the model.

Our analysis highlights the effectiveness of ELMo embeddings in sentiment analysis, where the model excelled in accurately classifying positive and negative movie reviews. The contextual information captured by ELMo embeddings proved instrumental in distinguishing sentiment patterns. However, in the NLI task, the model struggled to achieve comparable performance, indicating the difficulty in accurately determining the relationships between sentences.

Further improvements could be explored, such as utilizing more advanced architectures or incorporating additional contextual information beyond ELMo embeddings. Additionally, exploring larger datasets or fine-tuning the model on domain-specific data might enhance its performance on the NLI task.