BiLSTM for Sentiment Polarity

Introduction

I chose the word embeddings - BERT and XLNet to train the BiLSTM model for sentiment analysis.

I am using Twitter data from kaggle https://www.kaggle.com/datasets/crowdflower/twitter-airline-sentiment is used for training the BiLSTM models. I have used the same dataset to train the model in the previous assignment using NaiveBayes Classifier. Now, I can compare the performance of the Machine Learning model and the Deep Learning model for the sentiment classification problems.

Data Pre-Processing

Data Cleaning

- Since the training dataset are tweets, we need to remove usernames that start with @ and Twitter hashtags that start with #
- Performed a lowercase transformation to remove case sensitivity
- Also, removed Special Characters and Punctuation as it does not contribute to determining the sentiments

```
def returnCleanText(text):
        text = text.lower() # lowercase transformation
        text = re.sub(r'(@\w+.*?)','',text) # removing twitter name tags
        text = re.sub(r'(#\w+.*?)','',text) # removing twitter hash tags
        text = re.sub(r"\W+|_", ' ', text) # removing numbers, punctuations, and underscore in the text retaining only alphatecial words
        return text

# Now use the apply() method to run the function on each text
tweetsdf['clean_text'] = tweetsdf['text'].apply(returnCleanText)
```

Encoding Sentiment Variable

The sentiment variable ‘airline_sentiment’ is converted categorical variable before feeding it to the models.
‘Neutral’ sentiment -> 0
‘Negative’ sentiment -> 1
‘Positive’ sentiment -> 2

```
tweetsdf['airline_sentiment'].replace({'neutral':0, 'negative':1, 'positive':2}, inplace=True)
```

Stop Word Removal
A list of stop words is loaded from the nltk library and then all the stop words are removed from the corpus. We created a new column ‘nostop_words’ in the dataset that has tweets without stop words.

```
my_stops = stopwords.words('english') #loading all the stop words in english language provided by nltk package
stop_pat = r'\b(?:{})\b'.format('|'.join(my_stops)) # combining all the stop words by a delimiter '|' so that we can find
def removeStopWords(text):
  return re.sub(stop_pat, '', text) # remvoing all the stop words
```

```
df['nostop_words'] = df['clean_text'].apply(removeStopWords)
```

Lemmatization
Lemmatization was performed with the help of spacy. We have created a new column ‘lemmatized_text’ in the dataset that has tweets that are lemmatized.

```
import spacy
nlp = spacy.load('en_core_web_sm')
```

```
# time consuming. takes around 1-2 mins to lemmatize entire corpus
def lemmatizing_tweets(text):
  doc = nlp(text)
  return " ".join([token.lemma_ for token in doc])

df['lemmatized_text'] = df['clean_text'].apply(lemmatizing_tweets)
```

Model with Bert Embeddings (with stop words)

We require only ‘text’, and ‘airline_sentiment’ columns for training our models.

```
# take a peek at the data
df[['clean_text', 'airline_sentiment']].head().values
```

```
array([[' what said ', 0],
       [' plus you ve added commercials to the experience tacky ', 2],
       [' i didn t today must mean i need to take another trip ', 0],
       [' it s really aggressive to blast obnoxious entertainment in your guests faces amp they have little recourse',
        1],
       [' and it s a really big bad thing about it', 1]], dtype=object)
```

Splitting the dataset into training and testing with 75% of data as training and 25% as testing. Dimensions of both training and testing datasets are shown in the below code snippet.

```
train_x, test_x, train_y, test_y = train_test_split(clean_text,airline_sentiment, test_size = 0.25, random_state = 42)
len(train_x),len(test_x),len(train_y),len(test_y)
```

```
(10980, 3660, 10980, 3660)
```

The below code snippet shows loading the BERT Pre-trained model and Bert Tokenizer. Bert Tokenizer is needed to encode each token in the corpus with word indexes. Bert model is needed to determine the embedding for each word index.

```
# Load pre-trained model tokenizer (vocabulary)
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased',do_lower_case = True)
bert_model = TFBertModel.from_pretrained('bert-base-uncased')
```

The below code snippet shows tokenizing training dataset using Bert batch encoder. It also pads the input data to the sentence with the maximum length found in the corpus if we do not provide the 'max_length' parameter to the tokenizer.

```
def bert_tokenizer(data):
  encoding = tokenizer.batch_encode_plus(data,padding=True)
  return tf.constant(encoding['input_ids'])
```

```
train_tokenized = bert_tokenizer(train_x)
test_tokenized = bert_tokenizer(test_x)
```

Dependent variable - airline_sentiment is one hot encoded with dimension as 3 because we have 3 sentiments in the training corpus.

```
# convert class vectors to binary class matrices- one hot encoding
from tensorflow import keras

y_train = keras.utils.to_categorical(train_y, num_classes)
y_test = keras.utils.to_categorical(test_y, num_classes)
```

Building the BERT model:

- Input layer will have training data encoded with BERT Tokenizer
- Tokenized words are passed to the Bert model to get the embeddings
- We have drop out layer to avoid model overfitting
- We have used BiLSTM with 50 (25*2) neurons
- We have used Softmax activation function and loss function is categorical cross entropy because we are trying to solve multiclass classification (0-neutral,1-negative,2-positive)
- We have used Adam optimizer which is quiet commonly used optimizer for classification problems

```
def build_bertmodel():
    word_inputs = tf.keras.Input(shape=(None,),dtype=tf.int32)
    last_hidden_states = bert_model(word_inputs)[0]
    x = tf.keras.layers.SpatialDropout1D(0.2)(last_hidden_states)
    x = tf.keras.layers.Bidirectional(LSTM(25, dropout=0.2, recurrent_dropout=0.2))(x)
    outputs = tf.keras.layers.Dense(num_classes, activation='softmax')(x)
    model = tf.keras.Model(word_inputs, outputs)
    return model
```

```
base_bertmodel = build_bertmodel()
base_bertmodel.compile(loss='categorical_crossentropy',optimizer="adam",metrics=['accuracy'])
```

Below is the model summary showing the number of parameters in each layer and total trainable parameters:

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
```

```
=================================================================
 input_1 (InputLayer)        [(None, None)]            0

 tf_bert_model (TFBertModel)  TFBaseModelOutputWithPoo  109482240
                             lingAndCrossAttentions(l
                             ast_hidden_state=(None,
                             None, 768),
                              pooler_output=(None, 76
                             8),
                              past_key_values=None, h
                             idden_states=None, atten
                             tions=None, cross_attent
                             ions=None)

 spatial_dropout1d (SpatialD  (None, None, 768)        0
 ropout1D)

 bidirectional (Bidirectiona  (None, 50)               158800
 l)

 dense (Dense)               (None, 3)                 153

=================================================================
Total params: 109,641,193
Trainable params: 109,641,193
Non-trainable params: 0
_________________________________________________________________
```

We ran the model for 1 epoch because it was time-consuming to run many epochs. Validation accuracy found to be 64%

```
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/bert/pooler/dense/kernel:0', 'tf_bert_model/bert/pooler/dense/bias:0'] when mir
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/bert/pooler/dense/kernel:0', 'tf_bert_model/bert/pooler/dense/bias:0'] when mir
344/344 [==============================] - ETA: 0s - loss: 0.9363 - accuracy: 0.6187WARNING:tensorflow:Early stopping conditioned on metric `val_acc` v
344/344 [==============================] - 241s 640ms/step - loss: 0.9363 - accuracy: 0.6187 - val_loss: 0.9025 - val_accuracy: 0.6393 - lr: 0.0010
```

Model with XLNet Embeddings (with stop words)

The below code snippet shows loading the XLNet Pre-trained model and XLNet Tokenizer. XLNet Tokenizer is needed to encode each token in the corpus with word indexes. XLNet model is needed to determine the embedding for each word index.

```
from numpy.core.fromnumeric import size

xlnet_tokenizer = XLNetTokenizer.from_pretrained('xlnet-base-cased')
xlnet_model = TFXLNetModel.from_pretrained('xlnet-base-cased')
```

The below code snippet shows tokenizing training dataset using XLNet batch encoder. It also pads the input data to the sentence with a maximum length of 120.

```
def xlnet_encode(text):
  train_tokenized = xlnet_tokenizer.batch_encode_plus(text,max_length= max_len, padding='max_length',return_tensors="pt")
  return tf.constant(train_tokenized['input_ids'])
```

```
train_encoded = xlnet_encode(train_x)
test_encoded = xlnet_encode(test_x)
```

Building the XLNet model:

- Input layer will have training data encoded with XLNet Tokenizer
- Tokenized words are passed to the XLNet pre trained model to get the embeddings
- We have drop out layer to avoid model overfitting

- We have used BiLSTM with 50 (25*2) neurons
- We have used Softmax activation function and loss function is categorical cross entropy because we are trying to solve multiclass classification (0-neutral,1-negative,2-positive)
- We have used Adam optimizer which is quiet commonly used optimizer for classification problems

```
def build_xlnetmodel():
  word_inputs = tf.keras.Input(shape=(max_len,), dtype='int32')

  # Call XLNet model
  xlnet_encodings = xlnet_model(word_inputs)[0]
  doc_encoding = tf.keras.layers.SpatialDropout1D(0.2)(xlnet_encodings)
  x = tf.keras.layers.Bidirectional(LSTM(25, dropout=0.2, recurrent_dropout=0.2))(doc_encoding)

  # Final output
  outputs = tf.keras.layers.Dense(num_classes, activation='softmax', name='outputs')(x)

  # Compile model
  model = tf.keras.Model(inputs=[word_inputs], outputs=[outputs])
  model.compile(optimizer="adam", loss='categorical_crossentropy', metrics=['accuracy'])
  return model
```

```
base_xlmodel = build_xlnetmodel()
base_xlmodel.summary()
```

We ran the model for 1 epoch because it was time-consuming to run many epochs. Validation accuracy found to be 64%

```
WARNING:tensorflow:Gradients do not exist for variables ['tfxl_net_model/transformer/mask_emb:0', 'tfxl_net_model/transformer/layer_._0/rel_attn/r_s_bi
WARNING:tensorflow:Gradients do not exist for variables ['tfxl_net_model/transformer/mask_emb:0', 'tfxl_net_model/transformer/layer_._0/rel_attn/r_s_bi
344/344 [==============================] - 654s 2s/step - loss: 0.9359 - accuracy: 0.6175 - val_loss: 0.9027 - val_accuracy: 0.6393 - lr: 0.0010
```

Model with Bert Embeddings (without stop words)

Now, we are training the model with tweets that do not have any stop words.

```
# take a peek at the data
df[['nostop_words', 'airline_sentiment']].head().values
```

Summary of the model is shown below:

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 120)]             0

 tfxl_net_model (TFXLNetMode  TFXLNetModelOutput(last_  116718336
 l)                          hidden_state=(None, 120,
                              768),
                              mems=((120, None, 768),
                              (120, None, 768),
                              (120, None, 768),
                              (120, None, 768),
                              (120, None, 768),
                              (120, None, 768),
                              (120, None, 768),
                              (120, None, 768),
                              (120, None, 768),
                              (120, None, 768),
```

```
                                (120, None, 768)),
                                hidden_states=None, att
                               entions=None)

 spatial_dropout1d_1 (Spatia   (None, 120, 768)         0
 lDropout1D)

 bidirectional_1 (Bidirectio   (None, 50)              158800
 nal)

 outputs (Dense)              (None, 3)                153

=================================================================
Total params: 116,877,289
Trainable params: 116,877,289
Non-trainable params: 0
_________________________________________________________________
```

We ran the model for 1 epoch because it was time-consuming to run many epochs. Validation accuracy found to be 64%

```
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/bert/pooler/dense/kernel:0', 'tf_bert_model/bert/pooler/dense/bias:0'] when mir
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/bert/pooler/dense/kernel:0', 'tf_bert_model/bert/pooler/dense/bias:0'] when mir
344/344 [==============================] - 226s 599ms/step - loss: 0.9263 - accuracy: 0.6209 - val_loss: 0.9025 - val_accuracy: 0.6393 - lr: 0.0010
<keras.callbacks.History at 0x7f00968d6350>
```

Model with XLNet Embeddings (without stop words)

Now, we trained the XLNet model without stop words to check the performance. Below is the summary of the model.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_6 (InputLayer)        [(None, 120)]             0

 tfxl_net_model (TFXLNetMode  TFXLNetModelOutput(last_  116718336
 l)                          hidden_state=(None, 120,
                              768),
                              mems=((120, None, 768),
                              (120, None, 768),
                              (120, None, 768),
                              (120, None, 768),
                              (120, None, 768),
                              (120, None, 768),
                              (120, None, 768),
                              (120, None, 768),
                              (120, None, 768),
                              (120, None, 768),
                              (120, None, 768)),
                              hidden_states=None, att
                             entions=None)

 spatial_dropout1d_5 (Spatia  (None, 120, 768)         0
 lDropout1D)

 bidirectional_5 (Bidirectio  (None, 50)               158800
 nal)

 outputs (Dense)             (None, 3)                 153

=================================================================
Total params: 116,877,289
Trainable params: 116,877,289
Non-trainable params: 0
```

We ran the model for 1 epoch and found the validation accuracy to be 64%.

```
WARNING:tensorflow:Gradients do not exist for variables ['tfxl_net_model/transformer/mask_emb:0', 'tfxl_net_model/transformer/layer_._0/rel_attn/r_s_b:
WARNING:tensorflow:Gradients do not exist for variables ['tfxl_net_model/transformer/mask_emb:0', 'tfxl_net_model/transformer/layer_._0/rel_attn/r_s_b:
344/344 [==============================] - 657s 2s/step - loss: 0.9290 - accuracy: 0.6192 - val_loss: 0.9027 - val_accuracy: 0.6393 - lr: 0.0010
<keras.callbacks.History at 0x7f02f9e1b9d0>
```

Summarizing Performace of the Models so far.

Parameter Settings	Accuracy
BERT Word Embedding + StopWords	64%
XLNet Word Embedding + StopWords	64%
BERT Word Embedding + No StopWords	64%
XLNet Word Embedding + No StopWords	64%

From the above Stats, all the models have similar performance hence we can choose any of the models for further training. Using `XLNet Model with Stop Words` to train the model with lemmatization.

Model with XLNet Embeddings (Lemmatization with stop words)

Using ‘lemmatized_text’ as an independent variable and ‘airline_sentiment’ as the dependent variable.

```
lemmatized_text = df['lemmatized_text'].values.tolist()
airline_sentiment = df['airline_sentiment'].values.tolist()
```

```
train_x, test_x, train_y, test_y = train_test_split(clean_text,airline_sentiment, test_size = 0.25, random_state = 42)
len(train_x),len(test_x),len(train_y),len(test_y)
```

```
(10980, 3660, 10980, 3660)
```

Tokenizing the lemmatized training data using XLNet tokenizer.

```
train_xlnet_lemmatized = xlnet_encode(train_x)
test_xlnet_lemmatized = xlnet_encode(test_x)
```

Below is the summary of the model.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_7 (InputLayer)        [(None, 120)]             0

 tfxl_net_model (TFXLNetMode  TFXLNetModelOutput(last_  116718336
 l)                          hidden_state=(None, 120,
                              768),
                              mems=((120, None, 768),
                              (120, None, 768),
                              (120, None, 768),
                              (120, None, 768),
                              (120, None, 768),
                              (120, None, 768),
                              (120, None, 768),
                              (120, None, 768),
                              (120, None, 768),
                              (120, None, 768),
                              (120, None, 768),
                              (120, None, 768)),
                              hidden_states=None, att
                             entions=None)

 spatial_dropout1d_6 (Spatia  (None, 120, 768)         0
 lDropout1D)

 bidirectional_6 (Bidirectio  (None, 50)               158800
 nal)

 outputs (Dense)             (None, 3)                 153

=================================================================
Total params: 116,877,289
Trainable params: 116,877,289
Non-trainable params: 0
_________________________________________________________________
```

We trained the model for 1 epoch and found the accuracy to be 64%

```
WARNING:tensorflow:Gradients do not exist for variables ['tfxl_net_model/transformer/mask_emb:0', 'tfxl_net_model/transformer/layer_._0/rel_attn/r_s_bi
WARNING:tensorflow:Gradients do not exist for variables ['tfxl_net_model/transformer/mask_emb:0', 'tfxl_net_model/transformer/layer_._0/rel_attn/r_s_bi
344/344 [==============================] - 657s 2s/step - loss: 0.9265 - accuracy: 0.6214 - val_loss: 0.9022 - val_accuracy: 0.6393 - lr: 0.0010
<keras.callbacks.History at 0x7f02f8e5c550>
```

Hyperparameter Tuning

Since all the models have similar performance, we are using XLNet Word Embedding + StopWords to tune the model with different **batch sizes** and drop out of 50%.

Batch sizes - 64, 128, and 256 along with 50% drop out.

```
base_xlmodel.fit(
    train_xlnet_dataset,
    batch_size=64,
    epochs=1,
    validation_data=test_xlnet_dataset,
    verbose=1,
    callbacks = callbacks)
```

```
344/344 [==============================] - 637s 2s/step - loss: 0.9211 - accuracy: 0.6228 - val_loss: 0.9019 - val_accuracy: 0.6393 - lr: 0.0010
<keras.callbacks.History at 0x7f0093d7b350>
```

This model (batch size = 64) took approximately 10mins to completely train. The validation accuracy of this model is 64% and the Validation Loss is 0.9 seems to be pretty high. Validation loss closer to 0 is considered pretty good for the model's performance.

```
base_xlmodel.fit(
    train_xlnet_dataset,
    batch_size=128,
    epochs=1,
    validation_data=test_xlnet_dataset,
    verbose=1,
    callbacks = callbacks)
```

```
344/344 [==============================] - 638s 2s/step - loss: 0.9211 - accuracy: 0.6228 - val_loss: 0.9022 - val_accuracy: 0.6393 - lr: 0.0010
<keras.callbacks.History at 0x7f0093d818d0>
```

This model (batch size = 128) also took approximately 10 mins to completely train. The validation accuracy of this model is 64% and the Validation Loss is 0.9 seems to be pretty high.

```
base_xlmodel.fit(
    train_xlnet_dataset,
    batch_size=256,
    epochs=1,
    validation_data=test_xlnet_dataset,
    verbose=1,
    callbacks = callbacks)
```

```
344/344 [==============================] - 635s 2s/step - loss: 0.9211 - accuracy: 0.6228 - val_loss: 0.9024 - val_accuracy: 0.6393 - lr: 0.0010
<keras.callbacks.History at 0x7f0093d83190>
```

This model (batch size = 256) also took approximately 10 mins to completely train. The validation accuracy of this model is 64% and the Validation Loss is 0.9 seems to be pretty high.

The accuracy of the models was found to be the same for different batch sizes.

Summarizing the models

All the models are showing similar performance with an accuracy of 64%. Stop words, and Lemmatization do not seem to have much effect on sentiment predictions.

parameter settings	Accuracy
BERT Word Embedding + StopWords	64%
XLNet Word Embedding + StopWords	64%
BERT Word Embedding + No StopWords	64%
XLNet Word Embedding + No StopWords	64%
XLNet Word Embedding + StopWords + Lemmatization	64%

Various data pre-processing steps like stop word removal, and lemmatization does not seem to have an impact on the BiLSTM model. Even with hyperparameter tuning, the model accuracy stayed the same at 64%.

It can be due to the fact that LSTM models do not perform well on sentences having lengths greater than 100. We could try attention models and transformers for better performance.

XLNet should have performed better than BERT however the performance of both the models was similar to the Twitter dataset. This needs to be further investigated.

Prediction

Predicting the sentiment for 2 lakh sentences in the corpus is resource intense and was unable to perform. Hence, predicted the sentiment polarity for 200 news articles from the corpus.

```
nltk.download('punkt')
# iterating over each news article to predict the sentiment
for index,row in tempdf.iterrows():
  sents = nltk.sent_tokenize(row['text'])
  num_pos_score = 0
  num_neg_score = 0
  num_neutral_score = 0
  for sent in sents:
    print(sent)
    try:
      encoding = get_inputs(sent,xlnet_tokenizer)
      pred_arr = base_xlmodel.predict(encoding, verbose=False) # predicting the sentiment
      # picking the array index having highest probability
      argIndex = np.argmax(pred_arr[0],axis=-1)
    except:
      continue
    if argIndex == 2:
        num_pos_score = num_pos_score + 1
    elif argIndex == 1:
        num_neg_score = num_neg_score + 1
    elif argIndex == 0:
        num_neutral_score = num_neutral_score + 1
  tempdf.loc[index, 'the number of positive sentences'] = num_pos_score
  tempdf.loc[index, 'the number of negative sentences'] = num_neg_score
  tempdf.loc[index, 'the number of neutral sentences'] = num_neutral_score
```