Reid Glaze
CSCI 5922

Lab Assignment 3

1. Impact of RNN Architecture

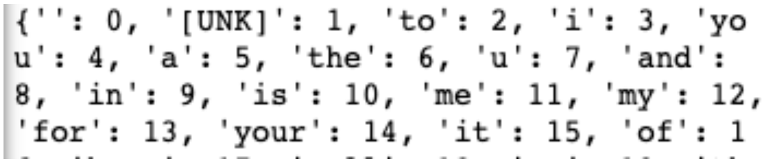
Methods

The goal of part 1 was to create three different neural networks and to assess the precision and recall of each one on a spam classifier. Vanilla RNN, LSTM, and GRU neural networks were examined. For each of these, precision and recall were obtained using the entire test set. Additionally, precision and recall were obtained for three different equal sized subsets of the test set. These subsets were made up of short, medium, and long inputs.

For the spam classifier, the "sms_spam" dataset was chosen. This dataset can be imported from the datasets library. It includes 5574 samples. It is made up of a "sms" feature and a "label" output. This feature includes a string for each sample that represents a text message. These strings vary greatly in length. The output is binary, made up of 0s and 1s, indicating whether the text message is spam or not. After splitting the data into training and test sets using a 70/30 split, there are 3901 training samples and 1673 test samples.

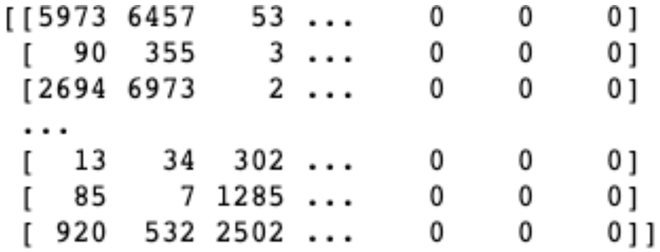
To run the feature data through the neural networks, it is necessary to convert the string data to integer data, and the first step is determining the size of the integer matrices. The string data was tokenized using Keras and the tokenizer was fit with the training samples. An integer sequence was created and the length of the longest string was determined.

After the max length was determined, TextVectorization from Keras was used to create a word index for up to 10,000 words in the training samples. The indexing of some of these words is shown in Figure 1. [UNK] represents words that were not vectorized. Additionally, this vectorizer was used to convert the string data into integer matrices with several columns that each reflect the length of the longest string in the training data. 0’s are used for indices after each shorter vectorized string. This is known as “padding”.The resulting matrix is shown in Figure 2. Argsort was used from the Numpy library to sort these sequences from shortest to longest. The resulting matrix is shown in Figure 3. This was done so that the training data could be split into three different subsets, representing short, medium, and long sequences.



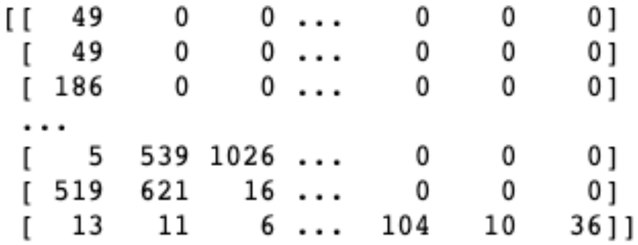
```
{'': 0, '[UNK]': 1, 'to': 2, 'i': 3, 'yo
u': 4, 'a': 5, 'the': 6, 'u': 7, 'and':
8, 'in': 9, 'is': 10, 'me': 11, 'my': 12,
'for': 13, 'your': 14, 'it': 15, 'of': 1
```

Figure 1 - word index



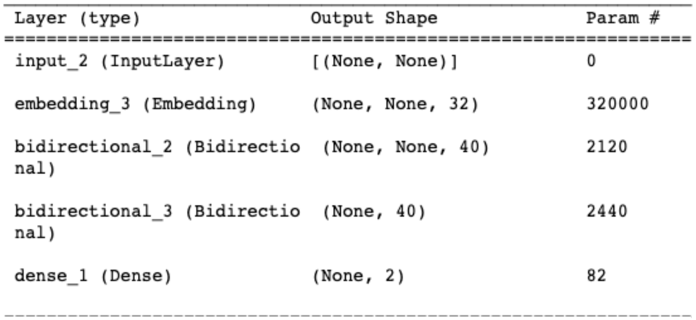
```
[[5973 6457   53 ...    0    0    0]
 [  90  355    3 ...    0    0    0]
 [2694 6973    2 ...    0    0    0]
 ...
 [  13   34  302 ...    0    0    0]
 [  85    7 1285 ...    0    0    0]
 [ 920  532 2502 ...    0    0    0]]
```

Figure 2 - padded and vectorized matrix



```
[[  49    0    0 ...    0    0    0]
 [  49    0    0 ...    0    0    0]
 [ 186    0    0 ...    0    0    0]
 ...
 [   5  539 1026 ...    0    0    0]
 [ 519  621   16 ...    0    0    0]
 [  13   11    6 ...  104   10   36]]
```

Figure 3 - sorted sequences

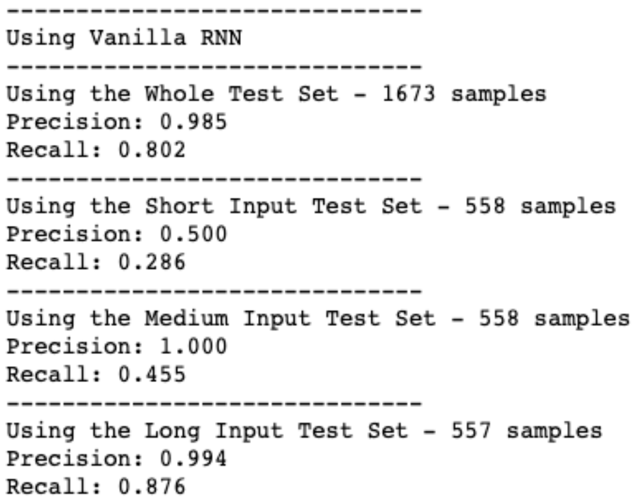


```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, None)]            0

 embedding_3 (Embedding)     (None, None, 32)          320000

 bidirectional_2 (Bidirectio  (None, None, 40)         2120
 nal)

 bidirectional_3 (Bidirectio  (None, 40)               2440
 nal)

 dense_1 (Dense)             (None, 2)                 82

_________________________________________________________________
```

Figure 4 - format for each model

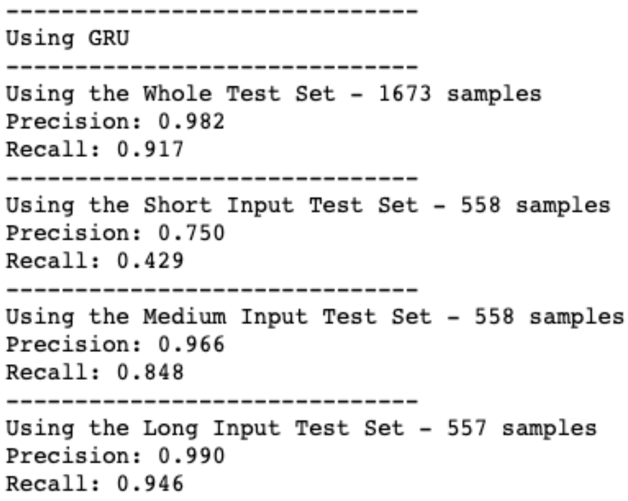
The neural networks were created using an input layer, an embedding layer, two Bidirectional RNN layers, and a Dense layer. The format for each model is shown in Figure 4. For the three different nueral networks, either SimpleRNN (vanilla RNN), LSTM, or GRU were used for the RNN layers. The model was compiled and fit with the training data. After training, each model was tested on 4 different test sets. The whole test set, the short sequences, the medium sequences, and the long sequences were all tested. Scikit-learn was used to determine the precision and recall scores for each of these trials.

In these models, several hyperparameters were kept the same. The embedding layers had a max token size of 10,000 and a node size of 32. Each of the RNN layers had a node size of 20. The Dense layer had a size of 2, based on the number of classes. For compiling, sparse categorical crossentropy was used for loss and an adam optimizer was used.



```
------------------------------
Using Vanilla RNN
------------------------------
Using the Whole Test Set - 1673 samples
Precision: 0.985
Recall: 0.802
------------------------------
Using the Short Input Test Set - 558 samples
Precision: 0.500
Recall: 0.286
------------------------------
Using the Medium Input Test Set - 558 samples
Precision: 1.000
Recall: 0.455
------------------------------
Using the Long Input Test Set - 557 samples
Precision: 0.994
Recall: 0.876
```

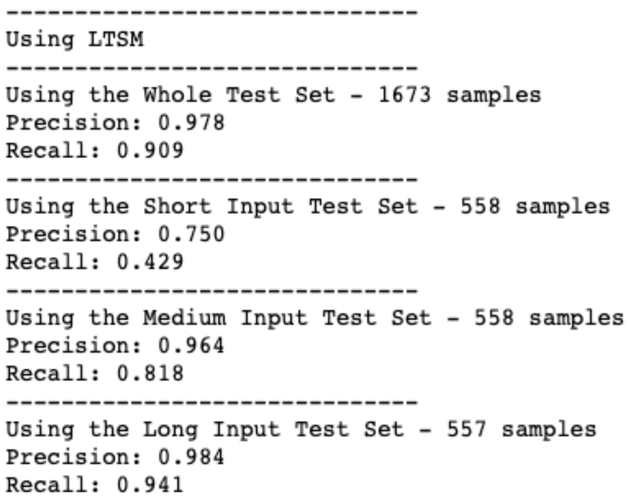
Figure 5 - Vanilla RNN Results



```
------------------------------
Using GRU
------------------------------
Using the Whole Test Set - 1673 samples
Precision: 0.982
Recall: 0.917
------------------------------
Using the Short Input Test Set - 558 samples
Precision: 0.750
Recall: 0.429
------------------------------
Using the Medium Input Test Set - 558 samples
Precision: 0.966
Recall: 0.848
------------------------------
Using the Long Input Test Set - 557 samples
Precision: 0.990
Recall: 0.946
```

Figure 6 - LSTM Results

Results



```
------------------------------
Using LTSM
------------------------------
Using the Whole Test Set - 1673 samples
Precision: 0.978
Recall: 0.909
------------------------------
Using the Short Input Test Set - 558 samples
Precision: 0.750
Recall: 0.429
------------------------------
Using the Medium Input Test Set - 558 samples
Precision: 0.964
Recall: 0.818
------------------------------
Using the Long Input Test Set - 557 samples
Precision: 0.984
Recall: 0.941
```

Figure 7 - GRU Results

The results for precision and recall using vanilla RNN are shown in Figure 5. These values are shown for the whole test and each of the 3 subsets categorized by input length. Figure 6 shows the precision and recall values obtained from these four test sets on the LSTM model. Figure 7 shows the precision and recall values obtained from the four test sets in the GRU model. Additionally, the number of samples in each test set is shown in the three figures.

Analysis

One trend that can be observed is that it is easier to obtain a higher Precision and Recall value when the test input is longer. For the most part, this trend is consistent across all 3 models created. There is only one exception to this trend. In the Vanilla RNN, the medium input test set has a precision value of 1.0 and the long test input has a precision value of 0.994. This difference is minimal, and the long input has a recall value of 0.876, much higher than the medium test input recall value of 0.429. Precision and Recall can be used together to get a good idea of how accurate the model is. From this information, we can determine that a longer test input is going to be more accurate in correctly classifying spam, regardless of the type of RNN used.

Another trend that can be observed is that the difference between the decision and recall is different depending on the model used. In the Vanilla RNN, the precision is always significantly higher than the recall. In GRU, it appears that the differences between the two is the smallest. The differences between precision and recall values in LSTM are only slightly larger than in GRU. When classifying spam, it is important to be selective when determining which email is classified as Spam. A high precision value is more important than a high recall value when choosing a model for this purpose.

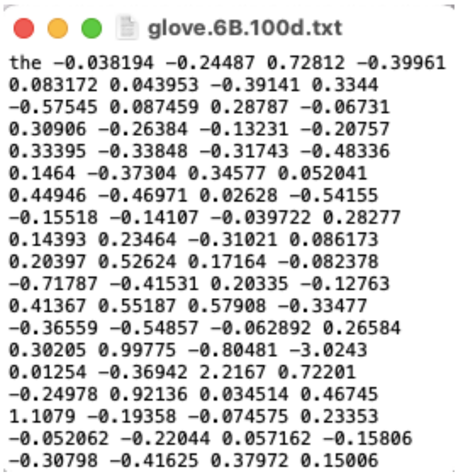
When selecting which model to use for spam detection, the precision value is weighted heavily, but should not be relied on exclusively. The vanilla RNN model has the highest precision value of 0.985 when the whole test set is used. The GRU model has a precision value of 0.982, which is not much lower. The LSTM model has the lowest precision value of 0.978. Since precision is the most important metric to consider, one may think that the vanilla RNN model would be the most appropriate one. However, the recall value of the GRU model using the whole test set is 0.917, much higher than the recall value of 0.802 obtained using vanilla RNN. There is a good argument that the small loss in precision value when using the GRU model would be worth the large increase in the recall value. Conveniently, the LSTM model has a lower recall value than GRU. For this reason, the GRU model should be selected as the most suitable one to use for spam classification using this dataset.

2. Impact of RNN Architecture

Methods

The goal of part 2 was to use two different pre-trained word embeddings and to test them on the most efficient type of RNN determined from part 1. It was determined that the GRU RNN would be appropriate to use based off the precision and recall values obtained. The Stanford GloVe model was used for pretrained word embeddings. First, the 100d GloVe embeddings were used. Afterwards, the 200d GloVe embeddings were used. For each of these two trials, precision and recall values were obtained, along with a confusion matrix.

The GRU models used in part 2 are very similar to the GRU models used in part 1. They are made up of an input layer, an embedding layer, two bidirectional GRU layers, and a Dense layer. The main difference is that a pretrained embedding layer is used in part 2, whereas a simple Keras embedding layer is used in part 1. The hyperparameters are kept the same aside from the ones in the embedding layer. Only the whole test set is analyzed in part 2 and the smaller subsets based on string lengths are not considered.



```
the -0.038194 -0.24487 0.72812 -0.39961
0.083172 0.043953 -0.39141 0.3344
-0.57545 0.087459 0.28787 -0.06731
0.30906 -0.26384 -0.13231 -0.20757
0.33395 -0.33848 -0.31743 -0.48336
0.1464 -0.37304 0.34577 0.052041
0.44946 -0.46971 0.02628 -0.54155
-0.15518 -0.14107 -0.039722 0.28277
0.14393 0.23464 -0.31021 0.086173
0.20397 0.52624 0.17164 -0.082378
-0.71787 -0.41531 0.20335 -0.12763
0.41367 0.55187 0.57908 -0.33477
-0.36559 -0.54857 -0.062892 0.26584
0.30205 0.99775 -0.80481 -3.0243
0.01254 -0.36942 2.2167 0.72201
-0.24978 0.92136 0.034514 0.46745
1.1079 -0.19358 -0.074575 0.23353
-0.052062 -0.22044 0.057162 -0.15806
-0.30798 -0.41625 0.37972 0.15006
```

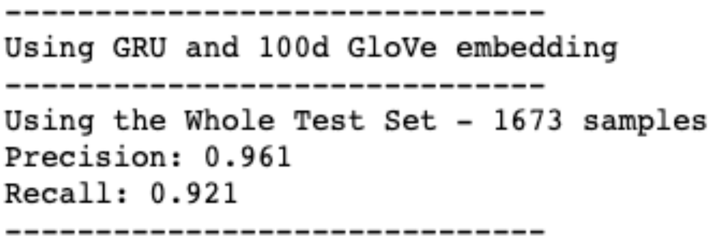
Figure 8 - 100d embedding dictionary

The first step to using the GloVe pretrained embeddings is downloading the GloVe model. The download includes several text files, which include several embedding dictionaries. The text files contain coefficients for words. The 100d embedding ditionary is shown in Figure 8. For each trial, the respective text file is imported and a list of 400,000 word vectors are obtained. Next, an embedding matrix is made for the list and it is associated with the dataset to get coefficients for these words. In each trial, 5491 words are converted and 2405 are missed. This means that 2405 words present in the dataset are not identified in the downloaded dictionary. In each trial, an embedding layer is created from the resulting embedding matrix.

After the GRU model is trained and fitted with the test data, the precision and recall scores are obtained along with a confusion matrix. Scikit-Learn is used to obtain the precision and recall scores. The Seaborn and Matplotlib libraries are used to create the confusion matrices. For each trial, the number of true negatives, false positives, false negatives, and true positives are all printed along with the percent allocated to each category.

Results

The precision and recall values obtained using the GRU model with the 100d GloVe embeddings are shown in Figure 9. The resulting confusion matrix is shown in Figure 10. The precision and recall values obtained using the GRU model with the 200d GloVe embddings are shown in figure 11. The resulting confusion matrix is shown in figure 12.



```
------------------------------
Using GRU and 100d GloVe embedding
------------------------------
Using the Whole Test Set - 1673 samples
Precision: 0.961
Recall: 0.921
------------------------------
```

Figure 9 - 100d GloVe embedding results

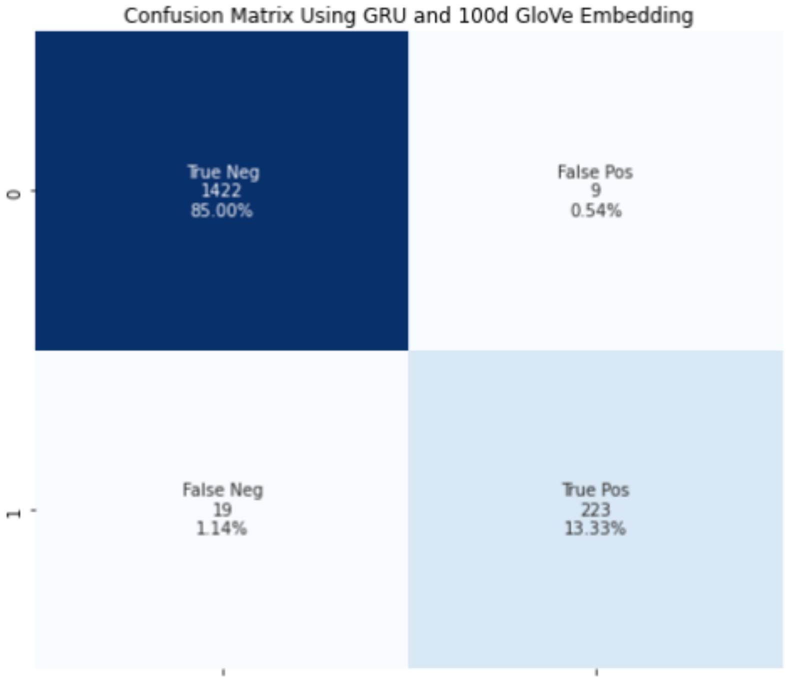
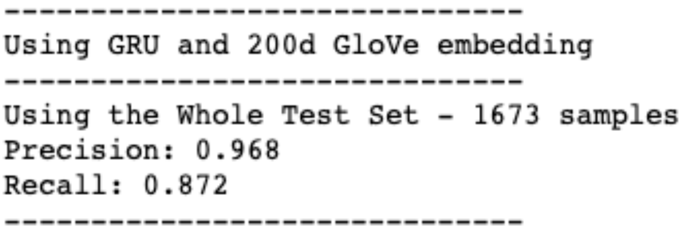


Figure 10 - 100d GloVe embedding confusion matrix

Analysis



```
------------------------------
Using GRU and 200d GloVe embedding
------------------------------
Using the Whole Test Set - 1673 samples
Precision: 0.968
Recall: 0.872
------------------------------
```

Figure 11 - 200d GloVe embedding results

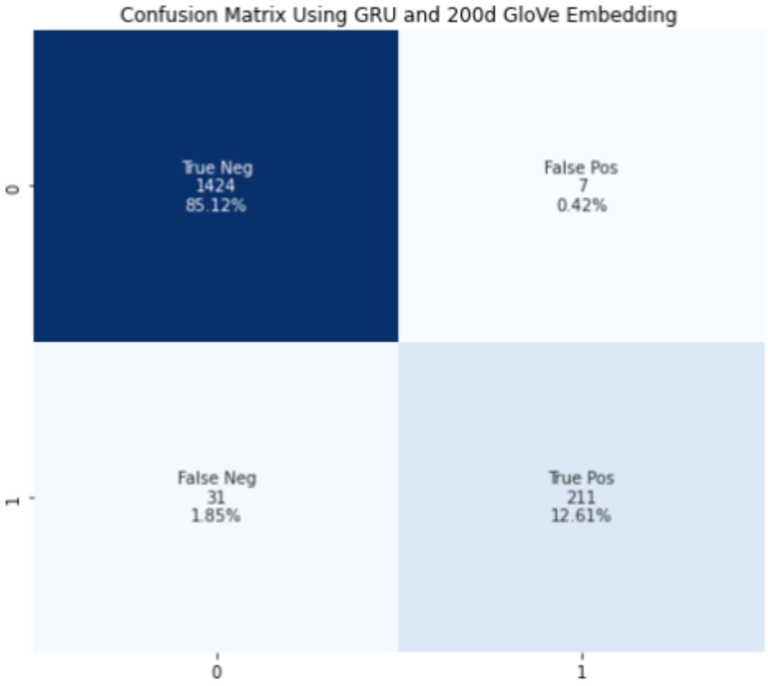
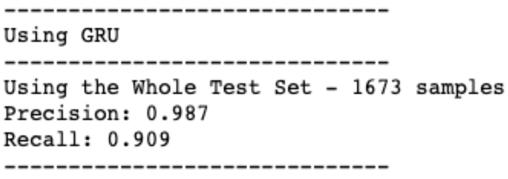


Figure 12 - 200d GloVe embedding confusion matrix

One trend that can be observed is that both models appear less optimal than the GRU model obtained in part 1. The main reason is because the precision value for GRU in part 1 is 0.982, is higher than the precision values of 0.961 (100d) and 0.968 (200d) obtained in part 2. While this difference may not seem significant, the precision value is very important in spam classification because incorrectly classifying email as spam is not good. Even though the recall for the 100d model in part 2 is slightly higher than the value obtained in part 1, the gain in recall is not worth the loss in precision. Based on the results, it would be better to use a simple embedding layer than to use 100d or 200d pretrained GloVe embedding layers.

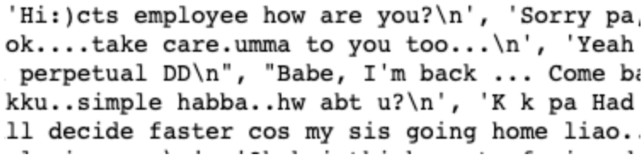
One factor that can explain the results may be the dimensionality of the embedding layers. In part 1, the embedding layer in the GRU model has 32 dimensions. This is shown in figure 4. In part 2, 100 dimensions and 200 dimensions are used. The purpose of an embedding layer is to map a high dimensional input into a lower dimensional space so that it works better with the other layers in the model. Often, a higher level of dimensions allows for more specificity. However, the bidirectional RNN layers are made up of only 20 nodes. An embedding layer with more than 20 nodes may be optimal, but a number much higher than this could be detrimental. It could be argued that the models in part 2 are less accurate because the dimensionality of the embedding layer is too high when compared to the dimensionality of the RNN layers.



```
------------------------------
Using GRU
------------------------------
Using the Whole Test Set - 1673 samples
Precision: 0.987
Recall: 0.909
------------------------------
```

Figure 13 - Normal Embedding using 100 nodes

In theory, a model with GloVe embedding should be more accurate than a model with regular embedding. To test this, the GRU model in part 1 was re-created using 100 nodes instead of 32 nodes. The results are shown in figure 13. The precision was more accurate than when 32 nodes were used. This counters the theory that an embedding layer dimensionality of 100 is too high. The precision value of 0.987 in figure 13 is higher than either of the precision values obtained using GloVe embedding. From this information, there is a reasonable argument that GloVe embedding simply does not work well with this dataset. This could be because this dataset is made up of text messages shown in Figure 14, which are very informal in terms of spelling and grammar. This explains why there are 2405 misses when converting the input data using the GloVe dictionary. It can be inferred that the GloVe embedding does not work well on this dataset because of the informality of the messages.



```
'Hi:)cts employee how are you?\n', 'Sorry pa
ok....take care.umma to you too...\n', 'Yeah
 perpetual DD\n", "Babe, I'm back ... Come b
kku..simple habba..hw abt u?\n', 'K k pa Had
ll decide faster cos my sis going home liao..
```

Figure 14 - Informal Input data

Reid Glaze - Lab Report 3

October 10, 2022

[8]:

```
#Reid Glaze
#CSCI 5922
#Lab Report 3

import numpy as np
from datasets import list_datasets, load_dataset

# Load a dataset and print the first example in the training set
dataset = load_dataset('sms_spam')
train = dataset['train']

## Prep the train dataset to samples (input) and labels (output)
train_sms = [x['sms'] for x in train]
train_labels = [x['label'] for x in train]
classes = list(set(train_labels))
print("Classes:", np.unique(train_labels))
from sklearn.model_selection import train_test_split
X_train_samples, X_test_samples, y_train_labels, y_test_labels =␣
 ↪train_test_split(train_sms, train_labels, test_size=0.3, random_state=0)
print("Number of samples in train:", len(X_train_samples))
print("Number of samples in test:", len(X_test_samples))
```

```
Found cached dataset sms_spam (/Users/reidglaze/.cache/huggingface/datasets/sms_
spam/plain_text/1.0.0/53f051d3b5f62d99d61792c91acefe4f1577ad3e4c216fb0ad39e30b9f
20019c)
```

```
  0%|          | 0/1 [00:00<?, ?it/s]
```

```
Classes: [0 1]
Number of samples in train: 3901
Number of samples in test: 1673
```

[9]:

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.layers import TextVectorization
import tensorflow as tf

num_words = 10000
oov_token = '<UNK>'
```

```
pad_type = 'post'
trunc_type = 'post'

# Tokenize our training data
tokenizer = Tokenizer(num_words=num_words, oov_token=oov_token)
tokenizer.fit_on_texts(X_train_samples)

# Get our training data word index
#word_index = tokenizer.word_index

# Encode training data sentences into sequences
train_sequences = tokenizer.texts_to_sequences(X_train_samples)

# Get max training sequence length
maxlen = max([len(x) for x in train_sequences])
print(maxlen)


# Create our Text Vectorizer to index our vocabulary based on the train samples
vectorizer = TextVectorization(max_tokens=10000, output_sequence_length=maxlen)
text_ds = tf.data.Dataset.from_tensor_slices(X_train_samples).batch(128) ##␣
 ↪Read batches of 128 samples
vectorizer.adapt(text_ds)

# Create a map to get the unique list of the vocabulary
voc = vectorizer.get_vocabulary()
word_index = dict(zip(voc, range(len(voc))))
#print(word_index)

# Vectorize our data (Convert the string data to integer data)
X_train = vectorizer(np.array([[s] for s in X_train_samples])).numpy()
X_test = vectorizer(np.array([[s] for s in X_test_samples])).numpy()
#print(X_train)
# from list to numpy array
y_train = np.array(y_train_labels)
y_test = np.array(y_test_labels)
# Create short, long, and medium test sets
indices = (X_test != 0).sum(axis=1).argsort()
X_sorted = X_test[indices]
y_sorted = y_test[indices]
#print(X_sorted)
ind = np.arange(X_sorted.shape[0])
short, medium, long = (np.array_split(ind, 3))
X_test_short = X_sorted[short]
X_test_medium = X_sorted[medium]
X_test_long = X_sorted[long]
y_test_short = y_sorted[short]
```

```
y_test_medium = y_sorted[medium]
y_test_long = y_sorted[long]
```

```
162
```

[10]:

```
from keras.layers import Embedding
from keras.initializers import Constant
from keras import layers, Input, Model
from sklearn.metrics import precision_score, recall_score

embedding_layer = Embedding(10000, 32,
                            input_length=X_train.shape[1])
```

[83]:

```
print("Using Vanilla RNN")
# create model
int_sequences_input = Input(shape=(None,), dtype="int64")
embedded_sequences = embedding_layer(int_sequences_input)
x = layers.Bidirectional(layers.SimpleRNN(20,␣
 ↪return_sequences=True))(embedded_sequences)
x = layers.Bidirectional(layers.SimpleRNN(20))(x)
preds = layers.Dense(len(classes), activation="softmax")(x)
model = Model(int_sequences_input, preds)
model.summary()
# Train and fit the model
model.compile(loss="sparse_categorical_crossentropy", optimizer="adam",␣
 ↪metrics=["acc"])
model.fit(X_train, y_train, batch_size=128, epochs=20)
y_predict = model.predict(X_test, batch_size=128, verbose=0)
y_pred = np.argmax(y_predict, axis=1)
# Find precision and recall for each test set
print("-------------------------------")
print("Using Vanilla RNN")
print("-------------------------------")
print("Using the Whole Test Set -", y_test.shape[0], "samples")
print('Precision: %.3f' % precision_score(y_test, y_pred))
print('Recall: %.3f' % recall_score(y_test, y_pred))
y_predict = model.predict(X_test_short, batch_size=128, verbose=0)
y_pred = np.argmax(y_predict, axis=1)
print("-------------------------------")
print("Using the Short Input Test Set -", y_test_short.shape[0], "samples")
print('Precision: %.3f' % precision_score(y_test_short, y_pred))
print('Recall: %.3f' % recall_score(y_test_short, y_pred))
y_predict = model.predict(X_test_medium, batch_size=128, verbose=0)
y_pred = np.argmax(y_predict, axis=1)
print("-------------------------------")
print("Using the Medium Input Test Set -", y_test_medium.shape[0], "samples")
print('Precision: %.3f' % precision_score(y_test_medium, y_pred))
```

```
print('Recall: %.3f' % recall_score(y_test_medium, y_pred))
y_predict = model.predict(X_test_long, batch_size=128, verbose=0)
y_pred = np.argmax(y_predict, axis=1)
print("------------------------------")
print("Using the Long Input Test Set -", y_test_long.shape[0], "samples")
print('Precision: %.3f' % precision_score(y_test_long, y_pred))
print('Recall: %.3f' % recall_score(y_test_long, y_pred))
```

```
Using Vanilla RNN
Model: "model_1"
-----------------------------------------------------------------
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, None)]            0

 embedding_3 (Embedding)     (None, None, 32)          320000

 bidirectional_2 (Bidirectio  (None, None, 40)        2120
 nal)

 bidirectional_3 (Bidirectio  (None, 40)              2440
 nal)

 dense_1 (Dense)             (None, 2)                 82

=================================================================
Total params: 324,642
Trainable params: 324,642
Non-trainable params: 0
-----------------------------------------------------------------
Epoch 1/20
31/31 [==============================] - 8s 147ms/step - loss: 0.3566 - acc:
0.8636
Epoch 2/20
31/31 [==============================] - 4s 141ms/step - loss: 0.1217 - acc:
0.9639
Epoch 3/20
31/31 [==============================] - 4s 143ms/step - loss: 0.0369 - acc:
0.9946
Epoch 4/20
31/31 [==============================] - 5s 157ms/step - loss: 0.0161 - acc:
0.9977
Epoch 5/20
31/31 [==============================] - 5s 169ms/step - loss: 0.0077 - acc:
0.9995
Epoch 6/20
31/31 [==============================] - 5s 161ms/step - loss: 0.0029 - acc:
1.0000
```

```
Epoch 7/20
31/31 [==============================] - 4s 130ms/step - loss: 0.0017 - acc:
1.0000
Epoch 8/20
31/31 [==============================] - 5s 152ms/step - loss: 0.0012 - acc:
1.0000
Epoch 9/20
31/31 [==============================] - 5s 154ms/step - loss: 9.5198e-04 - acc:
1.0000
Epoch 10/20
31/31 [==============================] - 5s 150ms/step - loss: 7.6624e-04 - acc:
1.0000
Epoch 11/20
31/31 [==============================] - 4s 140ms/step - loss: 6.4437e-04 - acc:
1.0000
Epoch 12/20
31/31 [==============================] - 4s 138ms/step - loss: 5.6118e-04 - acc:
1.0000
Epoch 13/20
31/31 [==============================] - 4s 141ms/step - loss: 4.9175e-04 - acc:
1.0000
Epoch 14/20
31/31 [==============================] - 4s 138ms/step - loss: 4.3206e-04 - acc:
1.0000
Epoch 15/20
31/31 [==============================] - 5s 153ms/step - loss: 3.8601e-04 - acc:
1.0000
Epoch 16/20
31/31 [==============================] - 5s 168ms/step - loss: 3.4416e-04 - acc:
1.0000
Epoch 17/20
31/31 [==============================] - 5s 156ms/step - loss: 3.1111e-04 - acc:
1.0000
Epoch 18/20
31/31 [==============================] - 4s 144ms/step - loss: 2.8869e-04 - acc:
1.0000
Epoch 19/20
31/31 [==============================] - 5s 159ms/step - loss: 2.6292e-04 - acc:
1.0000
Epoch 20/20
31/31 [==============================] - 4s 140ms/step - loss: 2.3915e-04 - acc:
1.0000
------------------------------
Using Vanilla RNN
------------------------------
Using the Whole Test Set - 1673 samples
Precision: 0.985
Recall: 0.802
```

```
------------------------------
Using the Short Input Test Set - 558 samples
Precision: 0.500
Recall: 0.286
------------------------------
Using the Medium Input Test Set - 558 samples
Precision: 1.000
Recall: 0.455
------------------------------
Using the Long Input Test Set - 557 samples
Precision: 0.994
Recall: 0.876
```

[84]:

```python
print("Using LTSM")
# create model
int_sequences_input = Input(shape=(None,), dtype="int64")
embedded_sequences = embedding_layer(int_sequences_input)
x = layers.Bidirectional(layers.LSTM(20,␣
 ↪return_sequences=True))(embedded_sequences)
x = layers.Bidirectional(layers.LSTM(20))(x)
preds = layers.Dense(len(classes), activation="softmax")(x)
model = Model(int_sequences_input, preds)
model.summary()
# Train and fit the model
model.compile(loss="sparse_categorical_crossentropy", optimizer="adam",␣
 ↪metrics=["acc"])
model.fit(X_train, y_train, batch_size=128, epochs=20)
y_predict = model.predict(X_test, batch_size=128, verbose=0)
y_pred = np.argmax(y_predict, axis=1)
# Find precision and recall for each test set
print("------------------------------")
print("Using LTSM")
print("------------------------------")
print("Using the Whole Test Set -", y_test.shape[0], "samples")
print('Precision: %.3f' % precision_score(y_test, y_pred))
print('Recall: %.3f' % recall_score(y_test, y_pred))
y_predict = model.predict(X_test_short, batch_size=128, verbose=0)
y_pred = np.argmax(y_predict, axis=1)
print("------------------------------")
print("Using the Short Input Test Set -", y_test_short.shape[0], "samples")
print('Precision: %.3f' % precision_score(y_test_short, y_pred))
print('Recall: %.3f' % recall_score(y_test_short, y_pred))
y_predict = model.predict(X_test_medium, batch_size=128, verbose=0)
y_pred = np.argmax(y_predict, axis=1)
print("------------------------------")
print("Using the Medium Input Test Set -", y_test_medium.shape[0], "samples")
print('Precision: %.3f' % precision_score(y_test_medium, y_pred))
```

```python
print('Recall: %.3f' % recall_score(y_test_medium, y_pred))
y_predict = model.predict(X_test_long, batch_size=128, verbose=0)
y_pred = np.argmax(y_predict, axis=1)
print("------------------------------")
print("Using the Long Input Test Set -", y_test_long.shape[0], "samples")
print('Precision: %.3f' % precision_score(y_test_long, y_pred))
print('Recall: %.3f' % recall_score(y_test_long, y_pred))
```

```
Using LTSM
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, None)]            0

 embedding_3 (Embedding)     (None, None, 32)          320000

 bidirectional_4 (Bidirectio  (None, None, 40)         8480
 nal)

 bidirectional_5 (Bidirectio  (None, 40)               9760
 nal)

 dense_2 (Dense)             (None, 2)                 82

=================================================================
Total params: 338,322
Trainable params: 338,322
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
31/31 [==============================] - 17s 318ms/step - loss: 0.4852 - acc:
0.8441
Epoch 2/20
31/31 [==============================] - 9s 307ms/step - loss: 0.2384 - acc:
0.8885
Epoch 3/20
31/31 [==============================] - 8s 258ms/step - loss: 0.0660 - acc:
0.9874
Epoch 4/20
31/31 [==============================] - 8s 273ms/step - loss: 0.0280 - acc:
0.9956
Epoch 5/20
31/31 [==============================] - 8s 266ms/step - loss: 0.0155 - acc:
0.9982
Epoch 6/20
31/31 [==============================] - 10s 315ms/step - loss: 0.0103 - acc:
0.9985
```

```
Epoch 7/20
31/31 [==============================] - 9s 276ms/step - loss: 0.0055 - acc:
0.9992
Epoch 8/20
31/31 [==============================] - 9s 277ms/step - loss: 0.0036 - acc:
0.9992
Epoch 9/20
31/31 [==============================] - 8s 267ms/step - loss: 0.0017 - acc:
1.0000
Epoch 10/20
31/31 [==============================] - 8s 260ms/step - loss: 0.0011 - acc:
1.0000
Epoch 11/20
31/31 [==============================] - 8s 258ms/step - loss: 0.0024 - acc:
0.9997
Epoch 12/20
31/31 [==============================] - 8s 256ms/step - loss: 8.6348e-04 - acc:
1.0000
Epoch 13/20
31/31 [==============================] - 9s 306ms/step - loss: 6.4786e-04 - acc:
1.0000
Epoch 14/20
31/31 [==============================] - 15s 472ms/step - loss: 5.0946e-04 -
acc: 1.0000
Epoch 15/20
31/31 [==============================] - 8s 260ms/step - loss: 4.2384e-04 - acc:
1.0000
Epoch 16/20
31/31 [==============================] - 8s 258ms/step - loss: 3.6389e-04 - acc:
1.0000
Epoch 17/20
31/31 [==============================] - 8s 249ms/step - loss: 3.1742e-04 - acc:
1.0000
Epoch 18/20
31/31 [==============================] - 8s 255ms/step - loss: 2.8065e-04 - acc:
1.0000
Epoch 19/20
31/31 [==============================] - 9s 292ms/step - loss: 2.5023e-04 - acc:
1.0000
Epoch 20/20
31/31 [==============================] - 8s 257ms/step - loss: 2.2465e-04 - acc:
1.0000
------------------------------
Using LTSM
------------------------------
Using the Whole Test Set - 1673 samples
Precision: 0.978
Recall: 0.909
```

```
------------------------------
Using the Short Input Test Set - 558 samples
Precision: 0.750
Recall: 0.429
------------------------------
Using the Medium Input Test Set - 558 samples
Precision: 0.964
Recall: 0.818
------------------------------
Using the Long Input Test Set - 557 samples
Precision: 0.984
Recall: 0.941
```

[85]:

```python
print("Using GRU")
# create model
int_sequences_input = Input(shape=(None,), dtype="int64")
embedded_sequences = embedding_layer(int_sequences_input)
x = layers.Bidirectional(layers.GRU(20,␣
 ↪return_sequences=True))(embedded_sequences)
x = layers.Bidirectional(layers.GRU(20))(x)
preds = layers.Dense(len(classes), activation="softmax")(x)
model = Model(int_sequences_input, preds)
model.summary()
# Train and fit the model
model.compile(loss="sparse_categorical_crossentropy", optimizer="adam",␣
 ↪metrics=["acc"])
model.fit(X_train, y_train, batch_size=128, epochs=20)
y_predict = model.predict(X_test, batch_size=128, verbose=0)
y_pred = np.argmax(y_predict, axis=1)
# Find precision and recall for each test set
print("------------------------------")
print("Using GRU")
print("------------------------------")
print("Using the Whole Test Set -", y_test.shape[0], "samples")
print('Precision: %.3f' % precision_score(y_test, y_pred))
print('Recall: %.3f' % recall_score(y_test, y_pred))
y_predict = model.predict(X_test_short, batch_size=128, verbose=0)
y_pred = np.argmax(y_predict, axis=1)
print("------------------------------")
print("Using the Short Input Test Set -", y_test_short.shape[0], "samples")
print('Precision: %.3f' % precision_score(y_test_short, y_pred))
print('Recall: %.3f' % recall_score(y_test_short, y_pred))
y_predict = model.predict(X_test_medium, batch_size=128, verbose=0)
y_pred = np.argmax(y_predict, axis=1)
print("------------------------------")
print("Using the Medium Input Test Set -", y_test_medium.shape[0], "samples")
print('Precision: %.3f' % precision_score(y_test_medium, y_pred))
```

```
print('Recall: %.3f' % recall_score(y_test_medium, y_pred))
y_predict = model.predict(X_test_long, batch_size=128, verbose=0)
y_pred = np.argmax(y_predict, axis=1)
print("------------------------------")
print("Using the Long Input Test Set -", y_test_long.shape[0], "samples")
print('Precision: %.3f' % precision_score(y_test_long, y_pred))
print('Recall: %.3f' % recall_score(y_test_long, y_pred))
```

```
Using GRU
Model: "model_3"
-----------------------------------------------------------------
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, None)]            0

 embedding_3 (Embedding)     (None, None, 32)          320000

 bidirectional_6 (Bidirectio  (None, None, 40)         6480
 nal)

 bidirectional_7 (Bidirectio  (None, 40)               7440
 nal)

 dense_3 (Dense)             (None, 2)                 82

=================================================================
Total params: 334,002
Trainable params: 334,002
Non-trainable params: 0
-----------------------------------------------------------------
Epoch 1/20
31/31 [==============================] - 18s 307ms/step - loss: 0.4408 - acc:
0.8621
Epoch 2/20
31/31 [==============================] - 8s 270ms/step - loss: 0.1529 - acc:
0.9251
Epoch 3/20
31/31 [==============================] - 8s 260ms/step - loss: 0.0197 - acc:
0.9959
Epoch 4/20
31/31 [==============================] - 8s 267ms/step - loss: 0.0073 - acc:
0.9987
Epoch 5/20
31/31 [==============================] - 8s 254ms/step - loss: 0.0045 - acc:
0.9990
Epoch 6/20
31/31 [==============================] - 9s 294ms/step - loss: 0.0037 - acc:
0.9995
```

```
Epoch 7/20
31/31 [==============================] - 8s 255ms/step - loss: 0.0020 - acc:
0.9997
Epoch 8/20
31/31 [==============================] - 8s 255ms/step - loss: 0.0014 - acc:
0.9997
Epoch 9/20
31/31 [==============================] - 8s 258ms/step - loss: 0.0010 - acc:
1.0000
Epoch 10/20
31/31 [==============================] - 8s 265ms/step - loss: 8.0499e-04 - acc:
1.0000
Epoch 11/20
31/31 [==============================] - 7s 241ms/step - loss: 6.4653e-04 - acc:
1.0000
Epoch 12/20
31/31 [==============================] - 9s 280ms/step - loss: 5.4070e-04 - acc:
1.0000
Epoch 13/20
31/31 [==============================] - 10s 332ms/step - loss: 4.6097e-04 -
acc: 1.0000
Epoch 14/20
31/31 [==============================] - 9s 285ms/step - loss: 4.0218e-04 - acc:
1.0000
Epoch 15/20
31/31 [==============================] - 8s 245ms/step - loss: 3.4938e-04 - acc:
1.0000
Epoch 16/20
31/31 [==============================] - 8s 256ms/step - loss: 3.1031e-04 - acc:
1.0000
Epoch 17/20
31/31 [==============================] - 8s 271ms/step - loss: 2.7700e-04 - acc:
1.0000
Epoch 18/20
31/31 [==============================] - 8s 264ms/step - loss: 2.5236e-04 - acc:
1.0000
Epoch 19/20
31/31 [==============================] - 8s 255ms/step - loss: 2.3236e-04 - acc:
1.0000
Epoch 20/20
31/31 [==============================] - 8s 253ms/step - loss: 2.1114e-04 - acc:
1.0000
------------------------------
Using GRU
------------------------------
Using the Whole Test Set - 1673 samples
Precision: 0.982
Recall: 0.917
```

```
------------------------------
Using the Short Input Test Set - 558 samples
Precision: 0.750
Recall: 0.429
------------------------------
Using the Medium Input Test Set - 558 samples
Precision: 0.966
Recall: 0.848
------------------------------
Using the Long Input Test Set - 557 samples
Precision: 0.990
Recall: 0.946
```

[86]:

```
##9. Download and unzip the Stanford GloVe model (pretrained word embeddings)
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip -q glove.6B.zip
```

```
--2022-10-08 21:25:30--  http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)… 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80…
connected.
HTTP request sent, awaiting response… 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2022-10-08 21:25:30--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443…
connected.
HTTP request sent, awaiting response… 301 Moved Permanently
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2022-10-08 21:25:30--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)… 171.64.64.22
Connecting to downloads.cs.stanford.edu
(downloads.cs.stanford.edu)|171.64.64.22|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: 'glove.6B.zip'

glove.6B.zip        100%[===================>] 822.24M  4.04MB/s    in 3m 54s

2022-10-08 21:29:24 (3.52 MB/s) - 'glove.6B.zip' saved [862182613/862182613]

replace glove.6B.100d.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: ^C
```

[54]:

```
#. Read the embeddings in the pretrained model (we are using the 100D version␣
↪of GloVe)
import os
path_to_glove_file = "glove.6B.100d.txt"

embeddings_index = {}
```

```
with open(path_to_glove_file) as f:
    for line in f:
        word, coefs = line.split(maxsplit=1)
        coefs = np.fromstring(coefs, "f", sep=" ")
        embeddings_index[word] = coefs

print("Found %s word vectors." % len(embeddings_index))
```

```
Found 400000 word vectors.
```

[47]:

```
## Create "embedding_matrix" to index our vocabulary using the GloVe model
num_tokens = len(voc)
embedding_dim = 100 ## 100 dimensions
hits = 0 ## number of words that were found in the pretrained model
misses = 0 ## number of words that were missing in the pretrained model

# Prepare embedding matrix for our word list
embedding_matrix = np.zeros((num_tokens, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # Words not found in embedding index will be all-zeros.
        # This includes the representation for "padding" and "OOV"
        embedding_matrix[i] = embedding_vector
        hits += 1
    else:
        misses += 1
print("Converted %d words (%d misses)" % (hits, misses))
#create embedding layer
embedding_layer = Embedding(num_tokens, embedding_dim,
                            embeddings_initializer= Constant(embedding_matrix),
                            trainable=False,
)
```

```
Converted 5491 words (2405 misses)
```

[48]:

```
print("Using GRU and 100d GloVe embedding")
# create model
int_sequences_input = Input(shape=(None,), dtype="int64")
embedded_sequences = embedding_layer(int_sequences_input)
x = layers.Bidirectional(layers.GRU(20,␣
 ↪return_sequences=True))(embedded_sequences)
x = layers.Bidirectional(layers.GRU(20))(x)
preds = layers.Dense(len(classes), activation="softmax")(x)
model = Model(int_sequences_input, preds)
model.summary()
# Train and fit the model
```

```
model.compile(loss="sparse_categorical_crossentropy", optimizer="adam",␣
 ↪metrics=["acc"])
model.fit(X_train, y_train, batch_size=128, epochs=20)
y_predict = model.predict(X_test, batch_size=128, verbose=0)
y_pred = np.argmax(y_predict, axis=1)
# Find precision and recall for each test set
print("------------------------------")
print("Using GRU and 100d GloVe embedding")
print("------------------------------")
print("Using the Whole Test Set -", y_test.shape[0], "samples")
print('Precision: %.3f' % precision_score(y_test, y_pred))
print('Recall: %.3f' % recall_score(y_test, y_pred))
print("------------------------------")
```

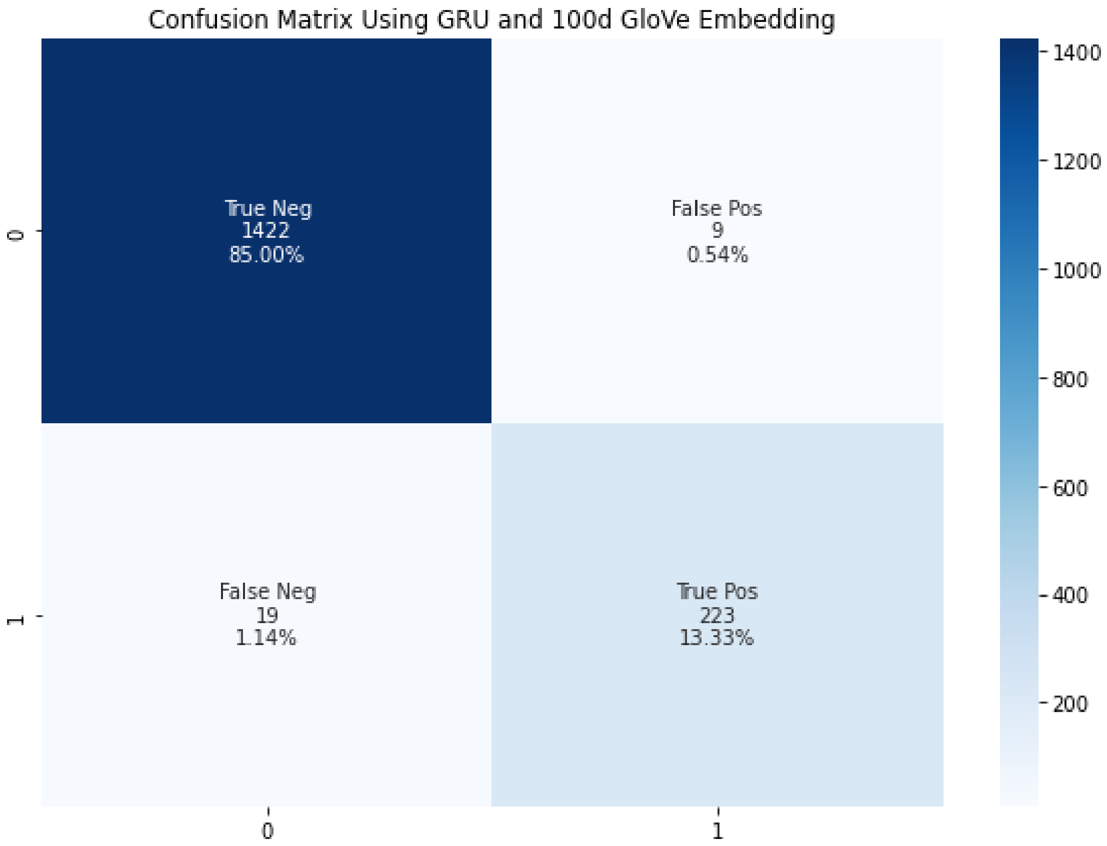
```
Using GRU and 100d GloVe embedding
Model: "model_2"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, None)]            0

 embedding_3 (Embedding)     (None, None, 100)         789600

 bidirectional_4 (Bidirectio  (None, None, 40)         14640
 nal)

 bidirectional_5 (Bidirectio  (None, 40)               7440
 nal)

 dense_2 (Dense)             (None, 2)                 82

=================================================================
Total params: 811,762
Trainable params: 22,162
Non-trainable params: 789,600

_________________________________________________________________
Epoch 1/20
31/31 [==============================] - 20s 340ms/step - loss: 0.4198 - acc:
0.8239
Epoch 2/20
31/31 [==============================] - 9s 295ms/step - loss: 0.2998 - acc:
0.8795
Epoch 3/20
31/31 [==============================] - 11s 354ms/step - loss: 0.2068 - acc:
0.9221
Epoch 4/20
31/31 [==============================] - 10s 307ms/step - loss: 0.1123 - acc:
0.9603
```

```
Epoch 5/20
31/31 [==============================] - 11s 347ms/step - loss: 0.0848 - acc:
0.9718
Epoch 6/20
31/31 [==============================] - 11s 346ms/step - loss: 0.0713 - acc:
0.9780
Epoch 7/20
31/31 [==============================] - 11s 339ms/step - loss: 0.0664 - acc:
0.9792
Epoch 8/20
31/31 [==============================] - 10s 311ms/step - loss: 0.0564 - acc:
0.9841
Epoch 9/20
31/31 [==============================] - 9s 301ms/step - loss: 0.0539 - acc:
0.9839
Epoch 10/20
31/31 [==============================] - 11s 351ms/step - loss: 0.0472 - acc:
0.9877
Epoch 11/20
31/31 [==============================] - 11s 351ms/step - loss: 0.0449 - acc:
0.9872
Epoch 12/20
31/31 [==============================] - 10s 310ms/step - loss: 0.0381 - acc:
0.9890
Epoch 13/20
31/31 [==============================] - 8s 271ms/step - loss: 0.0343 - acc:
0.9892
Epoch 14/20
31/31 [==============================] - 10s 322ms/step - loss: 0.0320 - acc:
0.9908
Epoch 15/20
31/31 [==============================] - 9s 286ms/step - loss: 0.0259 - acc:
0.9931
Epoch 16/20
31/31 [==============================] - 11s 341ms/step - loss: 0.0211 - acc:
0.9956
Epoch 17/20
31/31 [==============================] - 11s 365ms/step - loss: 0.0187 - acc:
0.9951
Epoch 18/20
31/31 [==============================] - 9s 278ms/step - loss: 0.0161 - acc:
0.9964
Epoch 19/20
31/31 [==============================] - 10s 311ms/step - loss: 0.0141 - acc:
0.9972
Epoch 20/20
31/31 [==============================] - 9s 299ms/step - loss: 0.0126 - acc:
0.9974
```

```
------------------------------
Using GRU and 100d GloVe embedding
------------------------------
Using the Whole Test Set - 1673 samples
Precision: 0.961
Recall: 0.921
------------------------------
```

[49]:

```python
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

cf_matrix = confusion_matrix(y_test, y_pred)
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
plt.figure(figsize = (10,7))
plt.title("Confusion Matrix Using GRU and 100d GloVe Embedding")
sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')
plt.show()
```



[50]:

```
path_to_glove_file = "glove.6B.200d.txt"

embeddings_index = {}
with open(path_to_glove_file) as f:
    for line in f:
        word, coefs = line.split(maxsplit=1)
        coefs = np.fromstring(coefs, "f", sep=" ")
        embeddings_index[word] = coefs

print("Found %s word vectors." % len(embeddings_index))
```

```
Found 400000 word vectors.
```

[51]:

```
## Create "embedding_matrix" to index our vocabulary using the GloVe model
num_tokens = len(voc)
embedding_dim = 200 ## 100 dimensions
hits = 0 ## number of words that were found in the pretrained model
misses = 0 ## number of words that were missing in the pretrained model

# Prepare embedding matrix for our word list
```

```
embedding_matrix = np.zeros((num_tokens, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # Words not found in embedding index will be all-zeros.
        # This includes the representation for "padding" and "OOV"
        embedding_matrix[i] = embedding_vector
        hits += 1
    else:
        misses += 1
print("Converted %d words (%d misses)" % (hits, misses))
#create embedding layer
embedding_layer = Embedding(num_tokens, embedding_dim,
                            embeddings_initializer= Constant(embedding_matrix),
                            trainable=False,
)
```

```
Converted 5491 words (2405 misses)
```

[52]:
```
print("Using GRU and 200d GloVe embedding")
# create model
int_sequences_input = Input(shape=(None,), dtype="int64")
embedded_sequences = embedding_layer(int_sequences_input)
x = layers.Bidirectional(layers.GRU(20,␣
 ↪return_sequences=True))(embedded_sequences)
x = layers.Bidirectional(layers.GRU(20))(x)
preds = layers.Dense(len(classes), activation="softmax")(x)
model = Model(int_sequences_input, preds)
model.summary()
# Train and fit the model
model.compile(loss="sparse_categorical_crossentropy", optimizer="adam",␣
 ↪metrics=["acc"])
model.fit(X_train, y_train, batch_size=128, epochs=20)
y_predict = model.predict(X_test, batch_size=128, verbose=0)
y_pred = np.argmax(y_predict, axis=1)
# Find precision and recall for each test set
print("-------------------------------")
print("Using GRU and 200d GloVe embedding")
print("-------------------------------")
print("Using the Whole Test Set -", y_test.shape[0], "samples")
print('Precision: %.3f' % precision_score(y_test, y_pred))
print('Recall: %.3f' % recall_score(y_test, y_pred))
print("-------------------------------")
```

```
Using GRU and 200d GloVe embedding
Model: "model_3"

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

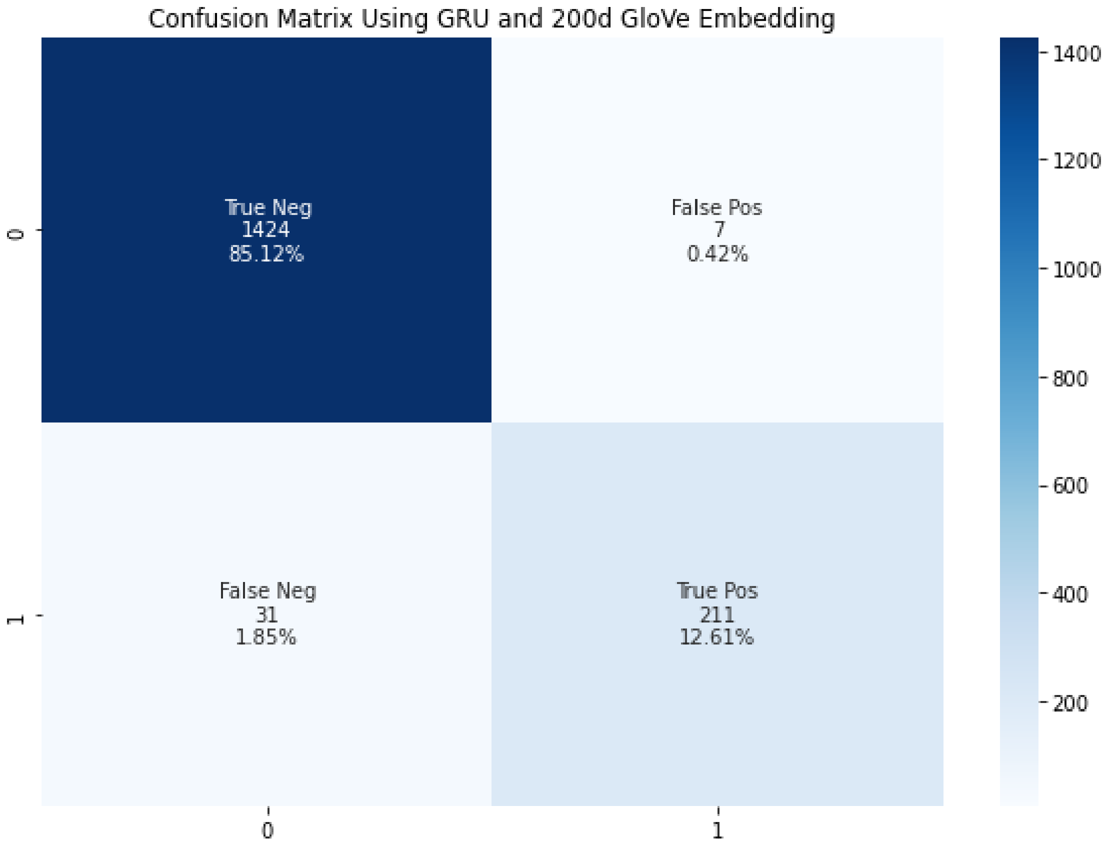
```
=================================================================
 input_4 (InputLayer)        [(None, None)]            0

 embedding_4 (Embedding)     (None, None, 200)         1579200

 bidirectional_6 (Bidirectio  (None, None, 40)         26640
 nal)

 bidirectional_7 (Bidirectio  (None, 40)               7440
 nal)

 dense_3 (Dense)             (None, 2)                 82

=================================================================
Total params: 1,613,362
Trainable params: 34,162
Non-trainable params: 1,579,200
_________________________________________________________________
Epoch 1/20
31/31 [==============================] - 26s 440ms/step - loss: 0.4068 - acc:
0.8475
Epoch 2/20
31/31 [==============================] - 11s 337ms/step - loss: 0.2908 - acc:
0.8882
Epoch 3/20
31/31 [==============================] - 11s 341ms/step - loss: 0.1876 - acc:
0.9367
Epoch 4/20
31/31 [==============================] - 13s 436ms/step - loss: 0.1068 - acc:
0.9677
Epoch 5/20
31/31 [==============================] - 14s 445ms/step - loss: 0.0776 - acc:
0.9759
Epoch 6/20
31/31 [==============================] - 11s 352ms/step - loss: 0.0596 - acc:
0.9828
Epoch 7/20
31/31 [==============================] - 13s 421ms/step - loss: 0.0504 - acc:
0.9854
Epoch 8/20
31/31 [==============================] - 13s 429ms/step - loss: 0.0478 - acc:
0.9846
Epoch 9/20
31/31 [==============================] - 14s 447ms/step - loss: 0.0367 - acc:
0.9897
Epoch 10/20
31/31 [==============================] - 13s 409ms/step - loss: 0.0305 - acc:
0.9926
```

```
Epoch 11/20
31/31 [==============================] - 13s 422ms/step - loss: 0.0261 - acc:
0.9944
Epoch 12/20
31/31 [==============================] - 12s 381ms/step - loss: 0.0227 - acc:
0.9946
Epoch 13/20
31/31 [==============================] - 14s 441ms/step - loss: 0.0184 - acc:
0.9962
Epoch 14/20
31/31 [==============================] - 12s 389ms/step - loss: 0.0207 - acc:
0.9946
Epoch 15/20
31/31 [==============================] - 12s 383ms/step - loss: 0.0159 - acc:
0.9964
Epoch 16/20
31/31 [==============================] - 11s 369ms/step - loss: 0.0166 - acc:
0.9962
Epoch 17/20
31/31 [==============================] - 12s 386ms/step - loss: 0.0138 - acc:
0.9977
Epoch 18/20
31/31 [==============================] - 12s 380ms/step - loss: 0.0121 - acc:
0.9977
Epoch 19/20
31/31 [==============================] - 12s 378ms/step - loss: 0.0199 - acc:
0.9951
Epoch 20/20
31/31 [==============================] - 11s 359ms/step - loss: 0.0198 - acc:
0.9944
------------------------------
Using GRU and 200d GloVe embedding
------------------------------
Using the Whole Test Set - 1673 samples
Precision: 0.968
Recall: 0.872
------------------------------
```

[53]:

```python
cf_matrix = confusion_matrix(y_test, y_pred)
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
```

```
plt.figure(figsize = (10,7))
plt.title("Confusion Matrix Using GRU and 200d GloVe Embedding")
sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')
plt.show()
```



[11]:

```
embedding_layer = Embedding(10000, 100,
                            input_length=X_train.shape[1])
```

[12]:

```
print("Using GRU")
# create model
int_sequences_input = Input(shape=(None,), dtype="int64")
embedded_sequences = embedding_layer(int_sequences_input)
x = layers.Bidirectional(layers.GRU(20,␣
 ↪return_sequences=True))(embedded_sequences)
x = layers.Bidirectional(layers.GRU(20))(x)
preds = layers.Dense(len(classes), activation="softmax")(x)
model = Model(int_sequences_input, preds)
model.summary()
# Train and fit the model
```

```
model.compile(loss="sparse_categorical_crossentropy", optimizer="adam",␣
 ↪metrics=["acc"])
model.fit(X_train, y_train, batch_size=128, epochs=20)
y_predict = model.predict(X_test, batch_size=128, verbose=0)
y_pred = np.argmax(y_predict, axis=1)
# Find precision and recall for each test set
print("------------------------------")
print("Using GRU")
print("------------------------------")
print("Using the Whole Test Set -", y_test.shape[0], "samples")
print('Precision: %.3f' % precision_score(y_test, y_pred))
print('Recall: %.3f' % recall_score(y_test, y_pred))

print("------------------------------")
```

```
Using GRU
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, None)]            0

 embedding_2 (Embedding)     (None, None, 100)         1000000

 bidirectional_6 (Bidirectio  (None, None, 40)         14640
 nal)

 bidirectional_7 (Bidirectio  (None, 40)               7440
 nal)

 dense_3 (Dense)             (None, 2)                 82

=================================================================
Total params: 1,022,162
Trainable params: 1,022,162
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
31/31 [==============================] - 19s 342ms/step - loss: 0.4414 - acc:
0.8634
Epoch 2/20
31/31 [==============================] - 10s 314ms/step - loss: 0.1737 - acc:
0.9216
Epoch 3/20
31/31 [==============================] - 10s 336ms/step - loss: 0.0385 - acc:
0.9880
Epoch 4/20
31/31 [==============================] - 10s 332ms/step - loss: 0.0150 - acc:
```

```
0.9969
Epoch 5/20
31/31 [==============================] - 11s 344ms/step - loss: 0.0072 - acc:
0.9987
Epoch 6/20
31/31 [==============================] - 10s 325ms/step - loss: 0.0043 - acc:
0.9992
Epoch 7/20
31/31 [==============================] - 10s 324ms/step - loss: 0.0023 - acc:
0.9995
Epoch 8/20
31/31 [==============================] - 12s 391ms/step - loss: 0.0012 - acc:
1.0000
Epoch 9/20
31/31 [==============================] - 11s 349ms/step - loss: 0.0012 - acc:
0.9995
Epoch 10/20
31/31 [==============================] - 10s 316ms/step - loss: 6.9992e-04 -
acc: 1.0000
Epoch 11/20
31/31 [==============================] - 11s 347ms/step - loss: 4.0889e-04 -
acc: 1.0000
Epoch 12/20
31/31 [==============================] - 10s 317ms/step - loss: 3.3288e-04 -
acc: 1.0000
Epoch 13/20
31/31 [==============================] - 10s 332ms/step - loss: 2.8185e-04 -
acc: 1.0000
Epoch 14/20
31/31 [==============================] - 11s 358ms/step - loss: 2.4130e-04 -
acc: 1.0000
Epoch 15/20
31/31 [==============================] - 10s 328ms/step - loss: 2.1267e-04 -
acc: 1.0000
Epoch 16/20
31/31 [==============================] - 12s 390ms/step - loss: 1.8801e-04 -
acc: 1.0000
Epoch 17/20
31/31 [==============================] - 12s 375ms/step - loss: 1.6900e-04 -
acc: 1.0000
Epoch 18/20
31/31 [==============================] - 10s 321ms/step - loss: 1.5298e-04 -
acc: 1.0000
Epoch 19/20
31/31 [==============================] - 11s 355ms/step - loss: 1.3968e-04 -
acc: 1.0000
Epoch 20/20
31/31 [==============================] - 10s 336ms/step - loss: 1.2768e-04 -
```

```
acc: 1.0000
------------------------------
Using GRU
------------------------------
Using the Whole Test Set - 1673 samples
Precision: 0.987
Recall: 0.909
------------------------------
```

[ ]:

```

```