CSCI 5922 -Neural Networks and Deep Learning
Lab - 3 Solutions

Gowri Shankar Raju Kurapati
Student ID 110568555

October 10, 2022

1 Impact of RNN Architecture

In this section, we will explore the impact of various RNN architectures by analyzing spam/ ham classification of SMS messages. We have considered the dataset from the UCI repository(LINK) which consists of nearly 5.5k examples with varying lengths. We split the dataset into 70/30 train/test and create three models which are Vanilla RNN, LSTM and GRU. All the sentences are tokenized and the maximum length is set to 150. Standard Tokenizers are used for this experiment to vectorize the words in the message and to maintain the consistency of sentence lengths, padding is also applied to give a max length of 150 for batch size.

For all three models, the input dimension is the size of the vocabulary which is around 8k and the dense embedding is set to the size of 8. All the cell states have a dimension of 32. The output of the last cell state is passed through a fully connected layer and sigmoid layer to predict if it is spam(target label is 1) or not (target label is 0). The loss function used for back-propagating the gradients is Binary Cross Entropy.

Model / Metric	RNN	LSTM	GRU
Precision	0.94	0.97	**0.98**
Recall	0.92	**0.96**	0.95

The above table gives the precision and recall scores for the different models used. We can see a gradual increase in precision from RNNs to GRUs with a significant increase from RNN to LSTM/GRU. This is expected as the LSTM is more robust to take the information needed with selective read, write and forget gates. Also, the recall increases which means that the LSTM is better at classifying the spam messages correctly among all the ones that actually needed to be predicted as spam.
To explore how better the model is at classifying varying-length sentences, we have further divided the test set into three equal parts i.e short, medium, and long inputs. Below is the histogram for the length of the message and the corresponding count.

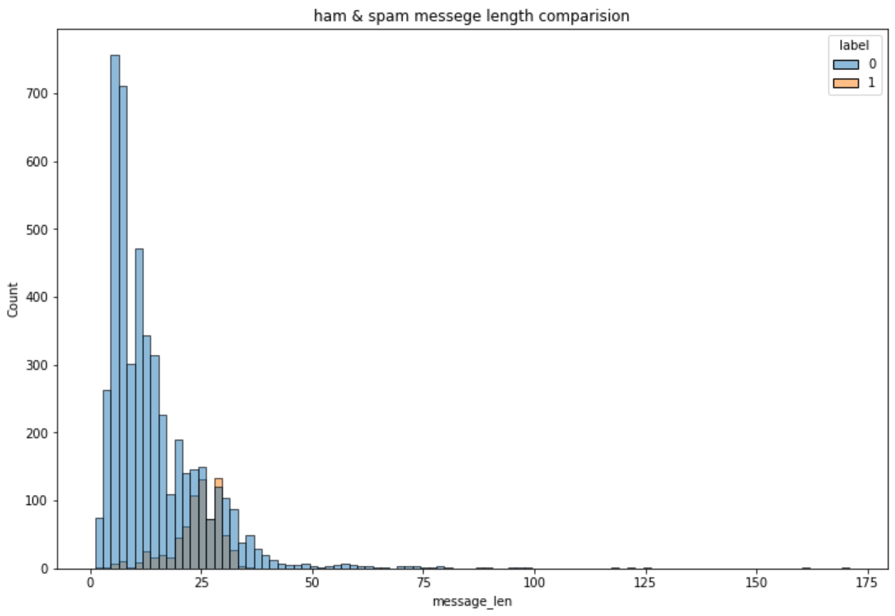


Figure 1: Histogram for message length and count in Dataset

As you can see from figure 1, the trisecting points to equally divide the test data are at message lengths of 8 and 18. All the messages of length less than or equal to 8 fall under the short category, between 8 and 18 fall under medium, and greater than 18 fall in the long input category. We have evaluated the models in these three categories and the metrics are tabulated below.

Short Input Lengths

Model / Metric	RNN	LSTM	GRU
Precision	0.98	1	1
Recall	0.25	0.5	0.5

Medium Input Lengths

Model / Metric	RNN	LSTM	GRU
Precision	0.99	1	1
Recall	0.75	0.88	0.88

Large Input Lengths

Model / Metric	RNN	LSTM	GRU
Precision	0.98	0.99	0.99
Recall	0.92	0.98	0.99

From *figure - 1* One important thing to observe is that most of the spam messages are of large input size. This holds true even for the training split as the splitting is done in a way to preserve class distribution. That is why the recall scores for large input sizes are very similar to the scores on the whole dataset.
The number of actual spam messages is 8 and 16 among the small and medium input sizes, out of which 6 and 4 are miss-classified by the RNN and LSTM/GRU, and 4 and 14 are mis-classified by LSTM/GRU respectively. That is why the recall scores are 0.25, 0.75 for small input size, and 0.5 and 0.88 for large input size. Since the data points are very scarce for spam messages, the inferences made here may not hold strongly but we can see that LSTM and GRU are classifying the spam messages

better compared to RNNs.

Also, we can see that the recall score gradually increased from RNN to LSTM and GRU, which states that LSTM is better at handling long-length sequences compared to vanilla RNNs. This observation can be directly tied back to the vanishing gradients problem in the RNNs which makes it the model hard to remember the information in the initial inputs.

There is not much difference in performance between LSTM and GRU models for any category of the input-sized dataset. Due to the skewness and limited size of the dataset, a clear comparison between the performances of LSTM and GRU cannot be made.

2 Impact of Pretrained Word Embedding

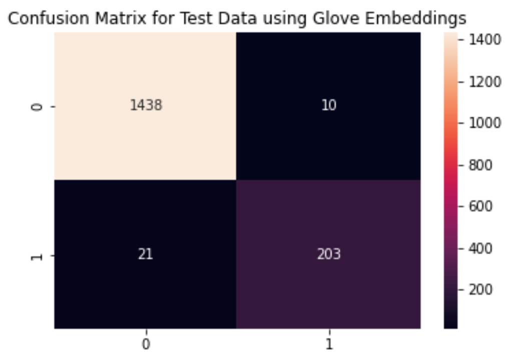
To understand the impact of the pre-trained word embeddings, we have trained two GRU models by using two different type of embeddings which are Glove embeddings(300 Dimension) and FastText(300 Dimension) Wiki embeddings. The GRU model consists of the embedding layer followed by the GRU Unit and later the output of the last layer of GRU is connected to fully connected network which has 2 hidden layers of 250 and 120 nodes with ReLu activation. The output node has the sigmoid activation to predict the class. We have used early stopping as regularizer and Adam optimizer with learning rate of 0.001, which weren't present in the models used for first part of the question. Also, the same loss as in first question which is Binary Cross Entropy is used.

We have also used to 100 dimensional Glove Embeddings to make the comparison of score on the impact of the size of the embeddings.The Precision and Recall for the models are given below.

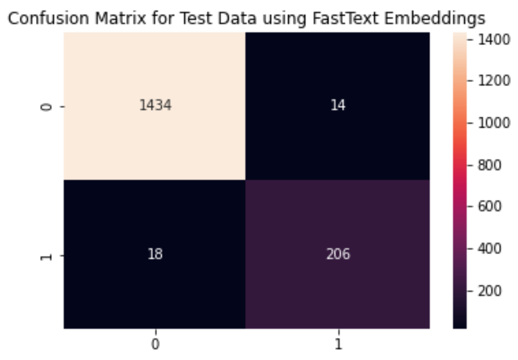
Model / Metric	GRU - Glove(100D)	GRU- Glove(300D)	FastText (300D)
Precision	0.968	**0.98**	0.95
Recall	0.88	0.91	**0.94**

As we can point out, there is a increase in both precision and recall score when we move from 100 dimensional glove embeddings to 300 dimensional glove embeddings but the increase is really not significant. Since the the dataset is sparse which consists of around just 8k words, 100 dimensional dense word embedding is able to capture the information needed to make the classification. Maybe if the dataset had many words and with closely related semantics, 300 dimensional embeddings would have a made difference when compared with 100 dimensional counter part. Glove-300 embeddings seem to perform better when competed against the FastText-300 Embeddings in precision scores. Glove-300 is slightly better at classifying the true spams better from all of the predicted spams.

Figure 2: Confusion Matrix for Test Data



(a) Using Glove-300 Embeddings



(b) Using Fasttext-300 Embeddings

From the Precision & Recall Table and also from the confusion matrices (*figure 2)*, though the precision is slightly better for Glove -300 compared to Fasttext-300, Fasttext is unconditionally better at recall by classifying the true spams better from all of the predicted spams. We usually wouldn't want to miss on ham messages and we would be okay to let a few spam messages in, but we would never want to miss out on a ham message. That is when precision is important compared to recall and we would go with Glove embeddings in that case as its precision is higher. But if the use case is other way around and we are stringent on having to remove the spam messages at the cost of leaving a few ham messages behind, the out emphasis would be on recall and choose FastText model in that scenario. But ideally, we would want both the precision and recall to be good enough. Though pretrained word embeddings aren't making much difference when compared to the tokenized input representation for our dataset, we can expect them to work better when the vocabulary of the dataset is large and includes words used under different semantics.

3 CODE

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from keras.layers import Dense , LSTM , Embedding , Dropout , Activation , Flatten, RNN, G
from sklearn.preprocessing import LabelEncoder
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential
from keras.preprocessing import sequence
from tensorflow.keras.utils import to_categorical
from keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.optimizers import RMSprop, Adam
import tensorflow as tf
%matplotlib inline
```

```
! unzip smsspamcollection.zip
```

```
df = pd.read_csv('SMSSpamCollection',sep ='\t',names=["label", "message"])
```

```
X = df['message']
Y = df['label']
le = LabelEncoder()
Y = le.fit_transform(Y)
Y = Y.reshape(-1,1)
print('Y shape: ', Y.shape)
print('Y unique values: ', np.unique(Y))
```

```
xtrain, xtest, ytrain, ytest = train_test_split(X, Y, test_size=0.3, stratify = Y)

# ytrain = to_categorical(ytrain)
# ytest = to_categorical(ytest)

# max
# print('Train data len:'+str(len(xtrain)))
# print('Class distribution'+str(Counter(ytrain)))


# print('Valid data len:'+str(len(xvalid)))
# print('Class distribution'+ str(Counter(yvalid)))

# print('Test data len:'+str(len(xtest)))
# print('Class distribution'+ str(Counter(ytest)))

# # print('Test data len:'+str(len(test_data['text'].tolist())))
# # print('Class distribution'+ str(Counter(test_data['label'].tolist())))


# train_dat =list(zip(ytrain,xtrain))
# valid_dat =list(zip(yvalid,xvalid))

max_words = 1000
max_len  =150
tok = Tokenizer(num_words= max_words)
tok.fit_on_texts(xtrain)
sequences = tok.texts_to_sequences(xtrain)
sequences_matrix = sequence.pad_sequences(sequences,maxlen=max_len)
print('sequences[2] length: ', len(sequences[2]))
print('sequences length: ', len(sequences))


def RNN():
    inputs = Input(name='inputs',shape=[max_len])
    layer = Embedding(max_words,8,input_length=max_len)(inputs)
    # Embedding (input_dim: size of vocabolary,
    # output_dim: dimension of dense embedding,
    # input_length: length of input sequence)
    layer = SimpleRNN(32)(layer)
    layer = Dense(1,name='out_layer')(layer)
    layer = Activation('sigmoid')(layer)
    model = Model(inputs=inputs,outputs=layer)
    return model

def LSTM1():
    inputs = Input(name='inputs',shape=[max_len])
    layer = Embedding(max_words,8,input_length=max_len)(inputs)
    # Embedding (input_dim: size of vocabolary,
    # output_dim: dimension of dense embedding,
    # input_length: length of input sequence)
    layer = LSTM(32)(layer)
    layer = Dense(1,name='out_layer')(layer)
    layer = Activation('sigmoid')(layer)
    model = Model(inputs=inputs,outputs=layer)
    return model



def GRU1():
    inputs = Input(name='inputs',shape=[max_len])
    layer = Embedding(max_words,8,input_length=max_len)(inputs)
    # Embedding (input_dim: size of vocabolary,
    # output_dim: dimension of dense embedding,
    # input_length: length of input sequence)
    layer = GRU(32)(layer)
    layer = Dense(1,name='out_layer')(layer)
    layer = Activation('sigmoid')(layer)
    model = Model(inputs=inputs,outputs=layer)
    return model


model = RNN()
model.summary()
model.compile(loss='binary_crossentropy',metrics=['accuracy'])

model2 = LSTM1()
```

```
model2.summary()
model2.compile(loss='binary_crossentropy',metrics=['accuracy'])


model3 = GRU1()
model3.summary()
model3.compile(loss='binary_crossentropy',metrics=['accuracy'])

history = model.fit(sequences_matrix,ytrain,batch_size=64,epochs=5,validation_split=0.2)
history2 = model2.fit(sequences_matrix,ytrain,batch_size=64,epochs=5,validation_split=0.2)
history3= model3.fit(sequences_matrix,ytrain,batch_size=64,epochs=5,validation_split=0.2)


test_sequences = tok.texts_to_sequences(xtest)
test_sequences_matrix = sequence.pad_sequences(test_sequences,maxlen=max_len)


test_sequences = tok.texts_to_sequences(df_large['message'])
test_sequences_matrix = sequence.pad_sequences(test_sequences,maxlen=max_len)
ytest = df_large['label']


accr = model.evaluate(test_sequences_matrix,ytest)
accr2 = model2.evaluate(test_sequences_matrix,ytest)
accr3 = model3.evaluate(test_sequences_matrix,ytest)


print('Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(accr[0],accr[1]))
print('Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(accr2[0],accr2[1]))
print('Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(accr3[0],accr3[1]))


ypred = model2.predict(test_sequences_matrix)
ypred = tf.math.round(ypred)


import matplotlib.pyplot as plt
cf_matrix = confusion_matrix(ytest, ypred)
print(cf_matrix)
from sklearn.metrics import precision_recall_fscore_support, classification_report
print(precision_recall_fscore_support(ytest, ypred, average='binary'))
print(classification_report(ytest, ypred))


import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import metrics

mat = metrics.confusion_matrix(ytest, ypred)
sns.heatmap(mat.T, square = True, annot = True, fmt = 'd', cbar = False)
plt.xlabel("True label")
plt.ylabel("Predicted label")


xtest
```

```
test = xtest

df = test.to_frame(name="message")


df["message_len"]=df["message"].str.count(' ') + 1
df['label'] = ytest


df.head(5)


import seaborn as sns
import matplotlib.pyplot as plt

fig=plt.figure(figsize=(12,8))
sns.histplot(
    x=df["message_len"]
)
plt.title("Histogram for count of messages with message length")
plt.savefig("Histogram.png")
plt.show()



df_small = df[df['message_len'] <= 8]
df_small.shape


df_medium= df[(df['message_len'] >8) & (df['message_len'] < 18)]
df_medium.shape


df_large = df[df['message_len'] > 18]
df_large.shape
```

Question 2

```
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip -q glove.6B.zip


embeddings_index = {}
f = open('glove.6B.300d.txt')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print('Found %s glove word vectors.' % len(embeddings_index))


max_word = 1000
max_len = 300
```

```
token = Tokenizer(num_words = max_word)
token.fit_on_texts(xtrain)
sequences = token.texts_to_sequences(xtrain)
seq_matrix = sequence.pad_sequences(sequences , maxlen = max_len)

sequences = token.texts_to_sequences(xtest)
seq_matrix_test = sequence.pad_sequences(sequences , maxlen = max_len)
word_index = token.word_index
len(word_index)
```

```
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))
```

```
embedding_matrix_glove = np.zeros((len(word_index) + 1, max_len))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix_glove[i] = embedding_vector
```

```
!wget --no-check-certificate \
     https://dl.fbaipublicfiles.com/fasttext/vectors-english/wiki-news-300d-1M.vec.zip \
     -O fasttext.1M.zip
```

```
! unzip fasttext.1M.zip
```

```
embeddings_index = {}
f = open('wiki-news-300d-1M.vec')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

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

```
embedding_matrix_fasttext = np.zeros((len(word_index) + 1, max_len))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
```

```
        if embedding_vector is not None:
            embedding_matrix_fasttext[i] = embedding_vector
```

```
BATCH = 128
EPOCHS = 10
```

```
from keras.models import Sequential
def build_model(model_name = 'RNN', embeddings = 'Glove'):
  model = Sequential()
  if embeddings == 'Glove':
    model.add(Embedding(input_dim=len(word_index) + 1,
                            output_dim=max_len,
                            weights=[embedding_matrix_glove],
                            input_length=max_len,
                            trainable=False))
  else:
    model.add(Embedding(input_dim=len(word_index) + 1,
                            output_dim=max_len,
                            weights=[embedding_matrix_fasttext],
                            input_length=max_len,
                            trainable=False))
  if model_name == 'LSTM':
    model.add(LSTM(64))
  elif model_name == 'GRU':
    model.add(GRU(64))

  model.add(Flatten())

  model.add(Dense(250, activation='relu'))
  model.add(Dropout(0.5))

  model.add(Dense(120, activation='relu'))

  model.add(Dense(1, activation='sigmoid'))

  return model
```

```
import keras

from keras import backend as K
def recall_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def precision_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision
```

```
def f1_m(y_true, y_pred):
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))
```

```
  glove = build_model('GRU', 'Glove')
  glove.compile(loss = 'binary_crossentropy' , optimizer = Adam(0.001) , metrics= ['accura
  es = EarlyStopping(monitor='val_loss', mode = 'min', patience = 2, verbose=1 , restore_b
  glove.fit(seq_matrix, ytrain, epochs = EPOCHS, batch_size = BATCH, validation_split=0.2,
```

```
  fast = build_model('GRU', 'Fasttext')
  fast.compile(loss = 'binary_crossentropy' , optimizer = Adam(0.001) , metrics= ['accurac
  es = EarlyStopping(monitor='val_loss', mode = 'min', patience = 2, verbose=1 , restore_b
  fast.fit(seq_matrix, ytrain, epochs = EPOCHS, batch_size = BATCH, validation_split=0.2,
```

```
from tensorflow import keras
metricsList= []
models =[]
models.append(glove)
models.append(fast)
for m in models:
  metrics = m.evaluate(seq_matrix_test, ytest, verbose=0)
  metricsList.append([metrics[2], metrics[3], metrics[4]])
  print(f"score of model {m} : f1 - {metrics[2]}, precision - {metrics[3]}, recall - {metr
```

```
pd.DataFrame(data = metricsList , columns=['F1', 'Precision', 'Recall'], index = ['Glove',
```

```
from sklearn.metrics import confusion_matrix
cms = []
for m in models:
  preds = m.predict(seq_matrix_test, batch_size = BATCH)
  preds = preds.round()
  cm=confusion_matrix(ytest,preds)
  cms.append(cm)
```

```
## Glove
ax = plt.axes()
sns.heatmap(cms[0], annot=True,fmt='.4g')
ax.set_title('Confusion Matrix for Test Data using Glove Embeddings')
plt.savefig("GloveE.png")
plt.show()
```

```
## Fasttext
ax = plt.axes()
sns.heatmap(cms[1], annot=True,fmt='.4g')
ax.set_title('Confusion Matrix for Test Data using FastText Embeddings')
plt.savefig("fastText.png")
plt.show()
```