CODE: Sentiment Classification - Impact of RNN Architecture

```
import numpy as np

import tensorflow_datasets as tfds
import tensorflow as tf

data, info = tfds.load('imdb_reviews', with_info=True, as_supervised=True)
train, test = data['train'], data['test']
```

```
Downloading and preparing dataset imdb_reviews/plain_text/1.0.0 (download: 80.23
Dl Completed...:    0/0 [00:00<?, ? url/s]

Dl Size...:    0/0 [00:00<?, ? MiB/s]


Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/plain_te
100%                                          24999/25000 [00:00<00:00, 105187.70 examples/s]
Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/plain_te
100%                                          24999/25000 [00:00<00:00, 117584.58 examples/s]
Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/plain_te
100%                                          49999/50000 [00:00<00:00, 133195.65 examples/s]
WARNING:absl:Dataset is using deprecated text encoder API which will be removed s
Dataset imdb_reviews downloaded and prepared to /root/tensorflow_datasets/imdb_re
```

```
VOCAB_SIZE = 10000
BATCH_SIZE = 64
embedding_dim=16
max_length = 120
trunc_type= 'post'
UNK="<UNK>"
```

```
## Processing the data

train_input = []  #X_train
test_input = []   #X_test
train_labels = [] #y_train
test_labels = []  #y_test

for i,j in train:
    train_input.append(str(i.numpy()))
    train_labels.append(j.numpy())

for i,j in test:
    test_input.append(str(i.numpy()))
    test_labels.append(j.numpy())


train_labels = np.array(train_labels)
test_labels = np.array(test_labels)


from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

tokenizer = Tokenizer(num_words = VOCAB_SIZE, oov_token=UNK)
tokenizer.fit_on_texts(train_input)
word_index = tokenizer.word_index


seq = tokenizer.texts_to_sequences(train_input)
padding = pad_sequences(seq, maxlen=max_length, truncating = trunc_type)
test_seq = tokenizer.texts_to_sequences(test_input)
test_padded = pad_sequences(test_seq, maxlen=max_length)


print("train : ", len(seq[0]))
print("test : ", len(test_seq[0]))
```

```
train :  118
test :  173
```

```
## Vanilla RNN Model

model = tf.keras.Sequential([
    tf.keras.layers.Embedding(VOCAB_SIZE, embedding_dim, input_length=max_length),
    tf.keras.layers.SimpleRNN(64),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 120, 16)           160000

 simple_rnn_1 (SimpleRNN)    (None, 64)                5184

 dense_2 (Dense)             (None, 10)                650

 dense_3 (Dense)             (None, 1)                 11
```

```
=================================================================
Total params: 165,845
Trainable params: 165,845
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history=model.fit(padding, train_labels, epochs=10, validation_data = (test_padded, te
```

```
Epoch 1/10
782/782 [==============================] - 140s 174ms/step - loss: 0.6951 - accu
Epoch 2/10
782/782 [==============================] - 134s 172ms/step - loss: 0.6949 - accu
Epoch 3/10
782/782 [==============================] - 134s 172ms/step - loss: 0.6942 - accu
Epoch 4/10
782/782 [==============================] - 137s 175ms/step - loss: 0.6914 - accu
Epoch 5/10
782/782 [==============================] - 133s 170ms/step - loss: 0.6677 - accu
Epoch 6/10
782/782 [==============================] - 134s 172ms/step - loss: 0.5931 - accu
Epoch 7/10
782/782 [==============================] - 135s 173ms/step - loss: 0.6056 - accu
Epoch 8/10
782/782 [==============================] - 134s 171ms/step - loss: 0.5453 - accu
Epoch 9/10
782/782 [==============================] - 133s 170ms/step - loss: 0.5762 - accu
Epoch 10/10
782/782 [==============================] - 132s 169ms/step - loss: 0.5765 - accu
```

```
from sklearn import metrics
from sklearn.metrics import precision_score, recall_score

predicted_1 = model.predict(test_padded)

pred_1 = np.zeros(len(predicted_1))
for i, score in enumerate(predicted_1):
    if score > 0.5:
      pred_1[i] = 1
    else:
      predicted_1[i] = 0

print(metrics.classification_report(test_labels, pred_1))
print("Precision = ",precision_score(test_labels, pred_1))
print("Recall = ", recall_score(test_labels, pred_1))
```

```
              precision    recall  f1-score   support

           0       0.58      0.41      0.48     12500
           1       0.55      0.70      0.61     12500

    accuracy                           0.56     25000
```

```
       macro avg       0.56      0.56      0.55     25000
    weighted avg       0.56      0.56      0.55     25000

    Precision =  0.5451332920024798
    Recall =  0.70344
```

```
import matplotlib.pyplot as plt

def plot_graphs(history, input):
    plt.plot(history.history[input])
    plt.plot(history.history['val_'+input])
    plt.xlabel("Epochs")
    plt.ylabel(input)
    plt.legend([input, 'val_'+input])
    plt.show()

plot_graphs(history, 'accuracy')
plot_graphs(history, 'loss')
```

```
    ---------------------------------------------------------------------------
    NameError                                 Traceback (most recent call last)
    <ipython-input-2-924d1a230a6c> in <module>()
          9     plt.show()
         10 
    ---> 11 plot_graphs(history, 'accuracy')
         12 plot_graphs(history, 'loss')

    NameError: name 'history' is not defined
```

SEARCH STACK OVERFLOW

```
## GRU Model

model_3 = tf.keras.Sequential([
    tf.keras.layers.Embedding(VOCAB_SIZE, embedding_dim, input_length=max_length),
    tf.keras.layers.Bidirectional(tf.keras.layers.GRU(64)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model_3.summary()
```

```
    Model: "sequential_2"
    _________________________________________________________________
     Layer (type)                Output Shape              Param #   
    =================================================================
     embedding_3 (Embedding)     (None, 120, 16)           160000    
                                                                     
     bidirectional (Bidirectiona  (None, 128)              31488     
     l)                                                              
                                                                     
     dense_4 (Dense)             (None, 32)                4128      
```

```
 dense_5 (Dense)             (None, 1)                 33

=================================================================
Total params: 195,649
Trainable params: 195,649
Non-trainable params: 0
_________________________________________________________________
```

```
model_3.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model_3.fit(padding, train_labels, epochs=10, validation_data = (test_padded
```

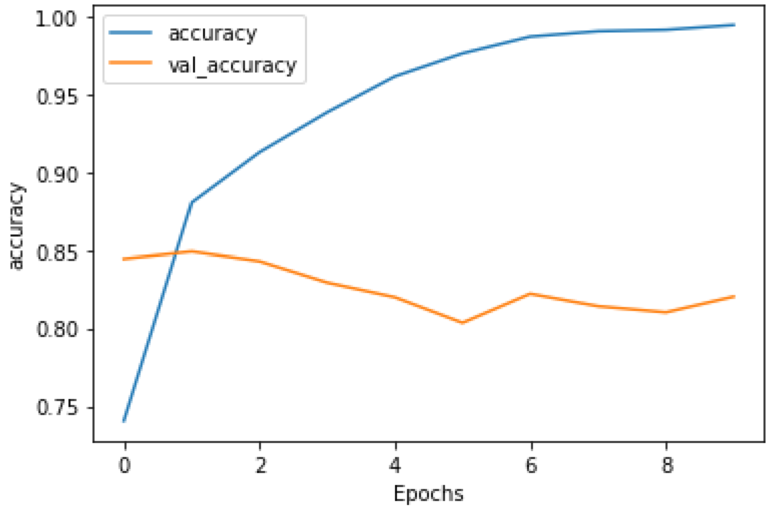
```
Epoch 1/10
782/782 [==============================] - 59s 69ms/step - loss: 0.4952 - accurac
Epoch 2/10
782/782 [==============================] - 53s 68ms/step - loss: 0.2936 - accurac
Epoch 3/10
782/782 [==============================] - 53s 68ms/step - loss: 0.2275 - accurac
Epoch 4/10
782/782 [==============================] - 52s 67ms/step - loss: 0.1679 - accurac
Epoch 5/10
782/782 [==============================] - 53s 67ms/step - loss: 0.1124 - accurac
Epoch 6/10
782/782 [==============================] - 53s 68ms/step - loss: 0.0711 - accurac
Epoch 7/10
782/782 [==============================] - 53s 68ms/step - loss: 0.0401 - accurac
Epoch 8/10
782/782 [==============================] - 53s 68ms/step - loss: 0.0272 - accurac
Epoch 9/10
782/782 [==============================] - 54s 69ms/step - loss: 0.0236 - accurac
Epoch 10/10
782/782 [==============================] - 54s 69ms/step - loss: 0.0146 - accurac
```

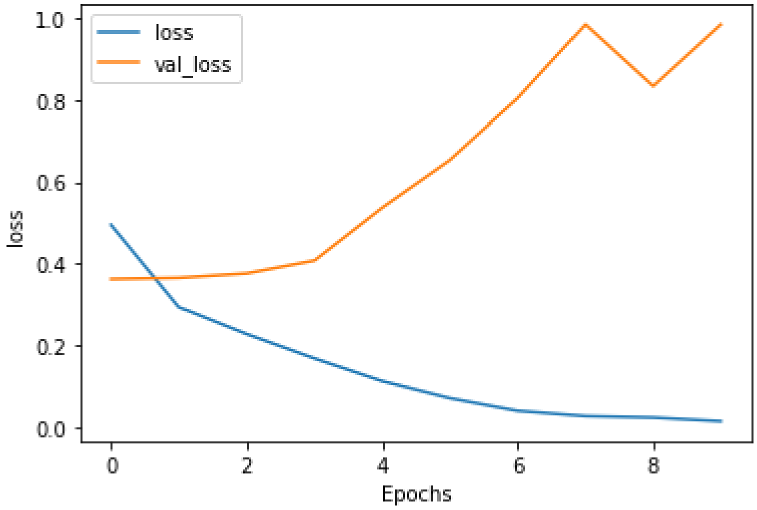
```
from sklearn import metrics
from sklearn.metrics import precision_score, recall_score

predicted_3 = model_3.predict(test_padded)

pred_3 = np.zeros(len(predicted_3))
for i, score in enumerate(predicted_3):
    if score > 0.5:
      pred_3[i] = 1
    else:
      pred_3[i] = 0

print(metrics.classification_report(test_labels, pred_3))
print("Precision = ",precision_score(test_labels, pred_3))
print("Recall = ", recall_score(test_labels, pred_3))
```

```
              precision    recall  f1-score   support

           0       0.81      0.83      0.82     12500
           1       0.83      0.81      0.82     12500
```

```
        accuracy                           0.82     25000
       macro avg       0.82      0.82      0.82     25000
    weighted avg       0.82      0.82      0.82     25000

    Precision =  0.826363414037662
    Recall =  0.81096
```

```
plot_graphs(history, 'accuracy')
plot_graphs(history, 'loss')
```





```
## LSTM Model

model_2 = tf.keras.Sequential([
    tf.keras.layers.Embedding(VOCAB_SIZE, embedding_dim, input_length=max_length),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model_2.summary()
```

```
    Model: "sequential_3"
    _________________________________________________________________
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 120, 16)           160000

 bidirectional_1 (Bidirectio  (None, 128)              41472
 nal)

 dense_6 (Dense)             (None, 32)                4128

 dense_7 (Dense)             (None, 1)                 33

=================================================================
Total params: 205,633
Trainable params: 205,633
Non-trainable params: 0
_________________________________________________________________
```

```
model_2.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model_2.fit(padding, train_labels, epochs=10, validation_data = (test_padde
```

```
Epoch 1/10
782/782 [==============================] - 61s 73ms/step - loss: 0.4469 - accura
Epoch 2/10
782/782 [==============================] - 56s 71ms/step - loss: 0.2918 - accura
Epoch 3/10
782/782 [==============================] - 55s 71ms/step - loss: 0.2358 - accura
Epoch 4/10
782/782 [==============================] - 55s 70ms/step - loss: 0.1899 - accura
Epoch 5/10
782/782 [==============================] - 55s 71ms/step - loss: 0.1535 - accura
Epoch 6/10
782/782 [==============================] - 55s 71ms/step - loss: 0.1118 - accura
Epoch 7/10
782/782 [==============================] - 55s 71ms/step - loss: 0.0831 - accura
Epoch 8/10
782/782 [==============================] - 55s 71ms/step - loss: 0.0628 - accura
Epoch 9/10
782/782 [==============================] - 56s 71ms/step - loss: 0.0453 - accura
Epoch 10/10
782/782 [==============================] - 56s 72ms/step - loss: 0.0332 - accura
```

```
from sklearn import metrics
from sklearn.metrics import precision_score, recall_score

predicted_2 = model_2.predict(test_padded)

pred_2 = np.zeros(len(predicted_2))
for i, score in enumerate(predicted_2):
    if score > 0.5:
      pred_2[i] = 1
    else:
      pred_2[i] = 0
```

```
print(metrics.classification_report(test_labels, pred_2))
print("Precision = ",precision_score(test_labels, pred_2))
print("Recall = ", recall_score(test_labels, pred_2))
```

```
              precision    recall  f1-score   support

           0       0.86      0.74      0.79     12500
           1       0.77      0.88      0.82     12500

    accuracy                           0.81     25000
   macro avg       0.82      0.81      0.81     25000
weighted avg       0.82      0.81      0.81     25000

Precision =  0.7695146577536384
Recall =  0.88408
```

```
plot_graphs(history, 'accuracy')
plot_graphs(history, 'loss')
```

Small, Medium and Long Inputs

```
ind_len = {}
for i in range(len(test_seq)):
  ind_len[i] = len(test_seq[i])
```

```
print(ind_len)
sorted_ind_len = dict(sorted(ind_len.items(), key=lambda val: val[1]))
print(sorted_ind_len)
sorted_list_ind = []

for key, val in sorted_ind_len.items():
  sorted_list_ind.append(key)

print(sorted_list_ind)
print(sorted_list_ind[2])
```

```
{0: 173, 1: 294, 2: 555, 3: 284, 4: 82, 5: 156, 6: 104, 7: 108, 8: 213, 9: 435,
{8712: 7, 11653: 7, 15391: 9, 17746: 10, 163: 11, 4981: 11, 11377: 13, 3076: 15,
[8712, 11653, 15391, 17746, 163, 4981, 11377, 3076, 12768, 9294, 24771, 1481, 10
15391
```

```
import math

split_size = math.floor(len(test_seq)/3)

small_test = []
med_test = []
long_test = []
```

```
for i in range(0,split_size):
  small_test.append(test_labels[sorted_list_ind[i]])

for i in range(split_size, split_size*2):
  med_test.append(test_labels[sorted_list_ind[i]])

for i in range(split_size*2, len(test_labels)):
  long_test.append(test_labels[sorted_list_ind[i]])

print(len(small_test))
print(len(med_test))
print(len(long_test))
```

```
    8333
    8333
    8334
```

```
## Model - 1 : Vanilla RNN

small_pred_1 = []
med_pred_1 = []
long_pred_1 = []

for i in range(0,split_size):
  small_pred_1.append(pred_1[sorted_list_ind[i]])

for i in range(split_size, split_size*2):
  med_pred_1.append(pred_1[sorted_list_ind[i]])

for i in range(split_size*2, len(test_labels)):
  long_pred_1.append(pred_1[sorted_list_ind[i]])


## Small Input
print("=====RNN: SMALL INPUT======")
print("Precision = ",precision_score(small_test, small_pred_1))
print("Recall = ", recall_score(small_test, small_pred_1))

## Medium Input
print("=====RNN: MEDIUM INPUT======")
print("Precision = ",precision_score(med_test, med_pred_1))
print("Recall = ", recall_score(med_test, med_pred_1))

## Long Input
print("=====RNN: LONG INPUT======")
print("Precision = ",precision_score(long_test, long_pred_1))
print("Recall = ", recall_score(long_test, long_pred_1))
```

```
    =====RNN: SMALL INPUT======
    Precision =  0.5822124071471592
    Recall =  0.6708304418228083
    =====RNN: MEDIUM INPUT======
    Precision =  0.5303951367781155
    Recall =  0.6880236569738788
    =====RNN: LONG INPUT======
    Precision =  0.5269328802039083
    Recall =  0.7528526341344987
```

```
## Model - 2 : LSTM

small_pred_2 = []
med_pred_2 = []
long_pred_2 = []

for i in range(0,split_size):
  small_pred_2.append(pred_2[sorted_list_ind[i]])

for i in range(split_size, split_size*2):
  med_pred_2.append(pred_2[sorted_list_ind[i]])

for i in range(split_size*2, len(test_labels)):
  long_pred_2.append(pred_2[sorted_list_ind[i]])


## Small Input
print("=====RNN: SMALL INPUT======")
print("Precision = ",precision_score(small_test, small_pred_2))
print("Recall = ", recall_score(small_test, small_pred_2))

## Medium Input
print("=====RNN: MEDIUM INPUT======")
print("Precision = ",precision_score(med_test, med_pred_2))
print("Recall = ", recall_score(med_test, med_pred_2))

## Long Input
print("=====RNN: LONG INPUT======")
print("Precision = ",precision_score(long_test, long_pred_2))
print("Recall = ", recall_score(long_test, long_pred_2))
```

```
    =====RNN: SMALL INPUT======
    Precision =  0.8214360477287
    Recall =  0.9077029840388618
    =====RNN: MEDIUM INPUT======
    Precision =  0.7567911714770797
    Recall =  0.8787580088713652
    =====RNN: LONG INPUT======
    Precision =  0.7309113300492611
    Recall =  0.864530225782957
```

```
## Model - 3 : GRU

small_pred_3 = []
```

```
small_pred_3 = []
med_pred_3 = []
long_pred_3 = []

for i in range(0,split_size):
  small_pred_3.append(pred_3[sorted_list_ind[i]])

for i in range(split_size, split_size*2):
  med_pred_3.append(pred_3[sorted_list_ind[i]])

for i in range(split_size*2, len(test_labels)):
  long_pred_3.append(pred_3[sorted_list_ind[i]])


## Small Input
print("=====RNN: SMALL INPUT======")
print("Precision = ",precision_score(small_test, small_pred_3))
print("Recall = ", recall_score(small_test, small_pred_3))

## Medium Input
print("=====RNN: MEDIUM INPUT======")
print("Precision = ",precision_score(med_test, med_pred_3))
print("Recall = ", recall_score(med_test, med_pred_3))

## Long Input
print("=====RNN: LONG INPUT======")
print("Precision = ",precision_score(long_test, long_pred_3))
print("Recall = ", recall_score(long_test, long_pred_3))
```

```
=====RNN: SMALL INPUT======
Precision =  0.868986454927604
Recall =  0.8607448531112654
=====RNN: MEDIUM INPUT======
Precision =  0.8142352347299344
Recall =  0.7949728930507639
=====RNN: LONG INPUT======
Precision =  0.7929405915983098
Recall =  0.7744598203447439
```

0s completed at 8:29 AM