Open in Colab

Text Classification challenge

You are required to train a deep learning model on the IMDB reviews dataset and classify a set of new reviews as positive(1) or negative(0) using the trained model.

```
##import the required libraries and APIs
import numpy as np
import tensorflow as tf
import tensorflow_datasets as tfds
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

print(tf.__version__)
```

```
2.4.1
```

Downloading the TensorFlow `imdb_review` dataset

> Make sure tensorflow_datasets is installed

```
##load the imdb reviews dataset
data, info = tfds.load("imdb_reviews", with_info=True, as_supervised=True)
```

Segregating training and testing sets

```
##segregate training and test set
train_data, test_data = data['train'], data['test']

##create empty list to store sentences and labels
train_sentences = []
test_sentences = []

train_labels = []
test_labels = []


##iterate over the train data to extract sentences and labels
for sent, label in train_data:
    train_sentences.append(str(sent.numpy().decode('utf8')))
    train_labels.append(label.numpy())

##iterate over the test set to extract sentences and labels
for sent, label in test_data:
    test_sentences.append(str(sent.numpy().decode('utf8')))
    test_labels.append(label.numpy())



##convert lists into numpy array
train_labels = np.array(train_labels)
test_labels = np.array(test_labels)
```

Data preparation - setting up the tokenizer

```
##define the parameters for the tokenizing and padding
vocab_size = 10000
embedding_dim = 16
max_length = 120
trunc_type='post'
oov_tok = "<OOV>"



tokenizer = Tokenizer(num_words = vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(train_sentences)
word_index = tokenizer.word_index

##training sequences and labels
train_seqs = tokenizer.texts_to_sequences(train_sentences)
train_padded = pad_sequences(train_seqs, maxlen=max_length, truncating=trunc_type)

##testing sequences and labels
test_seqs = tokenizer.texts_to_sequences(test_sentences)
test_padded = pad_sequences(test_seqs,maxlen=max_length)
```

Define the Neural Network with Embedding layer

1. Use the Sequential API.
2. Add an embedding input layer of input size equal to vocabulary size.
3. Add a flatten layer, and two dense layers.

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

##compile the model with loss function, optimizer and metrics
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 120, 16)           160000
_________________________________________________________________
global_average_pooling1d_1 ( (None, 16)                0
_________________________________________________________________
dense_2 (Dense)              (None, 24)                408
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 25
=================================================================
Total params: 160,433
Trainable params: 160,433
Non-trainable params: 0
_________________________________________________________________
```

Model Training

```
num_epochs = 10

##train the model with training and validation set
history = model.fit(
    train_padded,   #training sequence
    train_labels, # training labels
    epochs=num_epochs,
    validation_data=(test_padded, test_labels) # test data
    )
```

```
Epoch 1/10
782/782 [==============================] - 5s 5ms/step - loss: 0.6110 - accuracy: 0.6855 - val_loss: 0.
Epoch 2/10
782/782 [==============================] - 4s 5ms/step - loss: 0.3120 - accuracy: 0.8697 - val_loss: 0.
Epoch 3/10
782/782 [==============================] - 4s 5ms/step - loss: 0.2491 - accuracy: 0.9032 - val_loss: 0.
Epoch 4/10
782/782 [==============================] - 4s 5ms/step - loss: 0.2047 - accuracy: 0.9244 - val_loss: 0.
Epoch 5/10
782/782 [==============================] - 4s 5ms/step - loss: 0.1730 - accuracy: 0.9394 - val_loss: 0.
Epoch 6/10
782/782 [==============================] - 4s 5ms/step - loss: 0.1535 - accuracy: 0.9463 - val_loss: 0.
Epoch 7/10
782/782 [==============================] - 4s 5ms/step - loss: 0.1426 - accuracy: 0.9512 - val_loss: 0.
Epoch 8/10
782/782 [==============================] - 4s 5ms/step - loss: 0.1209 - accuracy: 0.9614 - val_loss: 0.
Epoch 9/10
782/782 [==============================] - 4s 5ms/step - loss: 0.1095 - accuracy: 0.9659 - val_loss: 0.
Epoch 10/10
782/782 [==============================] - 4s 5ms/step - loss: 0.1001 - accuracy: 0.9701 - val_loss: 0.
```

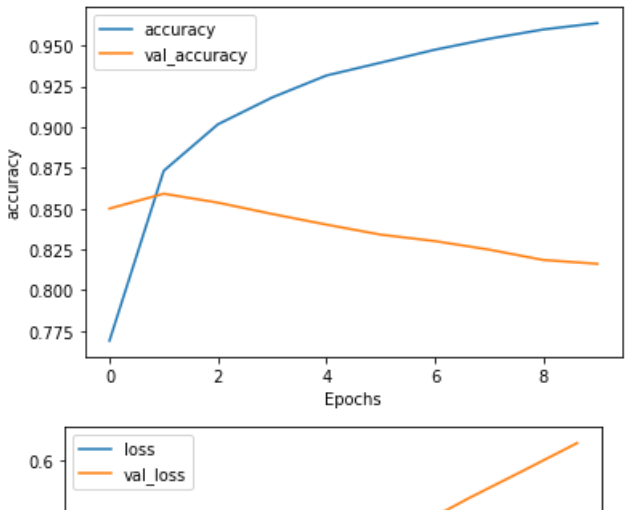
Visualise the train & validation accuracy and loss

```
import matplotlib.pyplot as plt

##plot the scores from history
def plot_metrics(history, metric):
  plt.plot(history.history[metric])
  plt.plot(history.history['val_'+metric])
  plt.legend([metric, 'val_'+metric])
  plt.xlabel("Epochs")
  plt.ylabel(metric)
  plt.show()

##plot accuracy
plot_metrics(history, "accuracy")

##plot loss
plot_metrics(history, "loss")
```



Classify new reviews

```
sentence = ["The first part of the movie was dull and boring!", "We watched Queen's Gambit, all seven hours

##prepare the sequences of the sentences in question
sequences = tokenizer.texts_to_sequences(sentence)
padded_seqs = pad_sequences(sequences, maxlen=max_length, truncating=trunc_type)

##print the classification score
print(model.predict(padded_seqs))
```

```
[[0.15088856]
 [0.81281435]]
```