```
[ ] #@title Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    # https://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.
```

Licensed under the Apache License, Version 2.0 (the "License");

```
[ ] import tensorflow as tf
    print(tf.__version__)

    # !pip install -q tensorflow-datasets
```

```
[ ] import tensorflow_datasets as tfds
    imdb, info = tfds.load("imdb_reviews", with_info=True, as_supervised=True)
```

```
[ ] import numpy as np

    train_data, test_data = imdb['train'], imdb['test']

    training_sentences = []
    training_labels = []

    testing_sentences = []
    testing_labels = []

    # str(s.tonumpy()) is needed in Python3 instead of just s.numpy()
    for s,l in train_data:
      training_sentences.append(str(s.numpy()))
      training_labels.append(l.numpy())

    for s,l in test_data:
      testing_sentences.append(str(s.numpy()))
      testing_labels.append(l.numpy())

    training_labels_final = np.array(training_labels)
    testing_labels_final = np.array(testing_labels)
```

```
[ ] vocab_size = 10000
    embedding_dim = 16
    max_length = 120
    trunc_type='post'
    oov_tok = "<OOV>"


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

    tokenizer = Tokenizer(num_words = vocab_size, oov_token=oov_tok)
    tokenizer.fit_on_texts(training_sentences)
    word_index = tokenizer.word_index
    sequences = tokenizer.texts_to_sequences(training_sentences)
    padded = pad_sequences(sequences,maxlen=max_length, truncating=trunc_type)

    testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
    testing_padded = pad_sequences(testing_sequences,maxlen=max_length)
```

```
[ ] reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])

    def decode_review(text):
        return ' '.join([reverse_word_index.get(i, '?') for i in text])

    print(decode_review(padded[1]))
    print(training_sentences[1])
```

```
[ ] model = tf.keras.Sequential([
        tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
        tf.keras.layers.Bidirectional(tf.keras.layers.GRU(32)),
        tf.keras.layers.Dense(6, activation='relu'),
        tf.keras.layers.Dense(1, activation='sigmoid')
    ])
    model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
    model.summary()
```

```
[ ] num_epochs = 50
    history = model.fit(padded, training_labels_final, epochs=num_epochs, validation_data=(testing_padded, testing_labels_final))
```

```
    import matplotlib.pyplot as plt


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

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

[ ]
```
# Model Definition with LSTM
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()
```

[ ]
```
# Model Definition with Conv1D
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Conv1D(128, 5, activation='relu'),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()
```