Natural Language Processing

Week-1 (Words based encodings using apis)

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.text import Tokenizer

sentences = [
    'I love my dog',
    'I love my cat'
]

tokenizer = Tokenizer(num_words = 100)
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index
print(word_index)
```

```
{'i': 1, 'my': 3, 'dog': 4, 'cat': 5, 'love': 2}
```

Text to sequence

```
from tensorflow.keras.preprocessing.text import Tokenizer

sentences = [
    'I love my dog',
    'I love my cat',
    'You love my dog!',
    'Do you think my dog is amazing?'
]

tokenizer = Tokenizer(num_words = 100)
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index

sequences = tokenizer.texts_to_sequences(sentences)

print(word_index)
print(sequences)
```

```
{'amazing': 10, 'dog': 3, 'you': 5, 'cat': 6,
 'think': 8, 'i': 4, 'is': 9, 'my': 1, 'do': 7,
 'love': 2}


[[4, 2, 1, 3], [4, 2, 1, 6], [5, 2, 1, 3], [7, 5,
8, 1, 3, 9, 10]]
```

```
test_data = [
    'i really love my dog',
    'my dog loves my manatee'
]

test_seq = tokenizer.texts_to_sequences(test_data)
print(test_seq)


[[4, 2, 1, 3], [1, 3, 1]]

{'think': 8, 'amazing': 10, 'my': 1, 'love': 2, 'dog': 3, 'is': 9, 'you': 5, 'do': 7,
'cat': 6, 'i': 4}
```

00V (Out of Vocabulary)

```
from tensorflow.keras.preprocessing.text import Tokenizer

sentences = [
    'I love my dog',
    'I love my cat',
    'You love my dog!',
    'Do you think my dog is amazing?'
]

tokenizer = Tokenizer(num_words = 100, oov_token="<OOV>")
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index

sequences = tokenizer.texts_to_sequences(sentences)

test_data = [
    'i really love my dog',
    'my dog loves my manatee'
]

test_seq = tokenizer.texts_to_sequences(test_data)
print(test_seq)
```

```
[[5, 1, 3, 2, 4], [2, 4, 1, 2, 1]]
```

```
{'think': 9, 'amazing': 11, 'dog': 4, 'do': 8, 'i': 5, 'cat': 7,
 'you': 6, 'love': 3, '<OOV>': 1, 'my': 2, 'is': 10}
```

Week-2 (Word Embeddings)

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']
```

The values for S and I are tensors, so by calling their NumPy method, I'll actually extract their value.

```
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())
```

When training, my labels are expected to be NumPy arrays. So I'll turn the list of labels that I've just created into NumPy arrays with this code.

```
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)
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Alternatively, you can use a Global Average Pooling 1D like below, which averages across the vector to flatten it out. Over 10 epochs with global average pooling, I got an accuracy of 0.9664 on training and 0.8187 on test, taking about 6.2 seconds per epoch. With flatten, my accuracy was 1.0 and my validation about 0.83 taking about 6.5 seconds per epoch. So it was a little slower, but a bit more accurate.

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    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()
```

```
num_epochs = 10
model.fit(padded,
          training_labels_final,
          epochs=num_epochs,
          validation_data=(testing_padded, testing_labels_final))
```

Demonstrate the embeddings

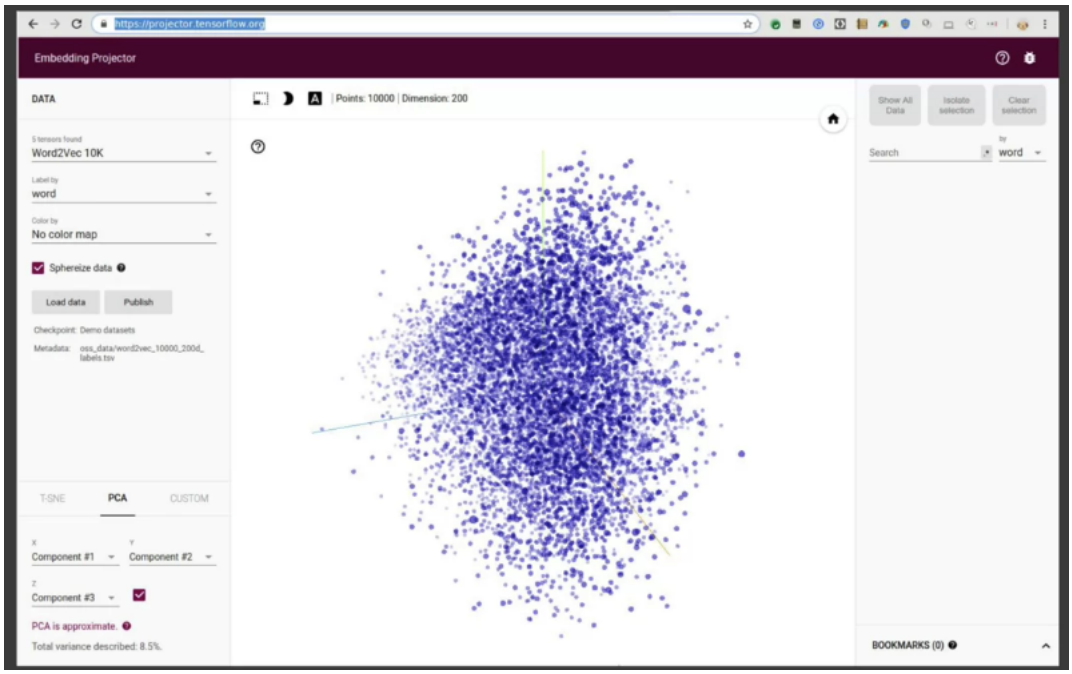
```
e = model.layers[0]
weights = e.get_weights()[0]
print(weights.shape) # shape: (vocab_size, embedding_dim)


(10000, 16)
```

```
Hello : 1
World : 2
How   : 3
Are   : 4
You   : 5


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


1 : Hello
2 : World
3 : How
4 : Are
5 : You
```

Now it's time to write the vectors and their metadata auto files. The TensorFlow Projector reads this file type and uses it to plot the vectors in 3D space so we can visualize them. To the vectors file, we simply write out the value of each of the items in the array of embeddings, i.e, the co-efficient of each dimension on the vector for this word. To the metadata array, we just write out the words.

```
import io

out_v = io.open('vecs.tsv', 'w', encoding='utf-8')
out_m = io.open('meta.tsv', 'w', encoding='utf-8')
for word_num in range(1, vocab_size):
  word = reverse_word_index[word_num]
  embeddings = weights[word_num]
  out_m.write(word + "\n")
  out_v.write('\t'.join([str(x) for x in embeddings]) + "\n")
out_v.close()
out_m.close()
```

To now render the results, go to the TensorFlow Embedding Projector on projector.tensorflow.org, press the ''Load data'' button on the left. You'll see a dialog asking you to load data from your computer. Use vector.TSV for the first one, and meta.TSV for the second. Once they're loaded, you should see something like this. Click this ''sphereize data'' checkbox on the top left, and you'll see the binary clustering of the data.



Pre-tokenized datasets

We'll take a look at a version of the IMDb dataset that has been pre-tokenized for you, but the tokenization is done on sub words. We'll use that to demonstrate how text classification can have some unique issues, namely that the sequence of words can be just as important as their existence.

https://github.com/tensorflow/datasets/blob/master/docs/catalog/imdb_reviews.md

```
import tensorflow_datasets as tfds
imdb, info = tfds.load("imdb_reviews/subwords8k", with_info=True, as_supervised=True)
```

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

```
tokenizer = info.features['text'].encoder
```

tensorflow.org/datasets/api_docs/python/tfds/features/text/SubwordTextEncoder

```
print(tokenizer.subwords)

['the_', ', ', '. ', 'a_', 'and_', 'of_', 'to_', 's_', 'is_',
'br', 'in_', 'I_', 'that_', 'this_', 'it_', … ]
```

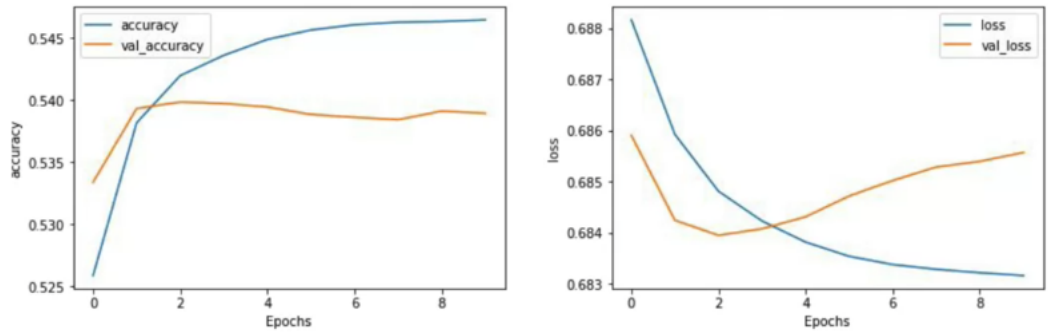
```
sample_string = 'TensorFlow, from basics to mastery'

tokenized_string = tokenizer.encode(sample_string)
print ('Tokenized string is {}'.format(tokenized_string))

original_string = tokenizer.decode(tokenized_string)
print ('The original string: {}'.format(original_string))


Tokenized string is [6307, 2327, 4043, 2120, 2, 48, 4249, 4429,
7, 2652, 8050]

The original string: TensorFlow, from basics to mastery
```

```
for ts in tokenized_string:
  print ('{} ----> {}'.format(ts, tokenizer.decode([ts])))

6307 ----> Ten
2327 ----> sor
4043 ----> Fl
2120 ----> ow
2 ----> ,
48 ----> from
4249 ----> basi
4429 ----> cs
7 ----> to
2652 ----> master
8050 ----> y
```

One thing to take into account, is the shape of the vectors coming from the tokenizer through the embedding, and it's not easily flattened. So we'll use Global Average Pooling 1D instead. Trying to flatten them, will cause a TensorFlow crash.

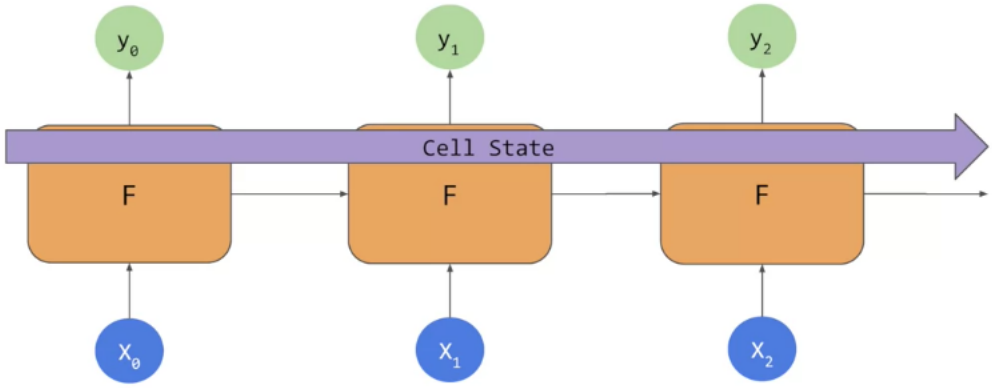
```
embedding_dim = 64
model = tf.keras.Sequential([
  tf.keras.layers.Embedding(tokenizer.vocab_size, embedding_dim),
  tf.keras.layers.GlobalAveragePooling1D(),
  tf.keras.layers.Dense(6, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])

model.summary()
```

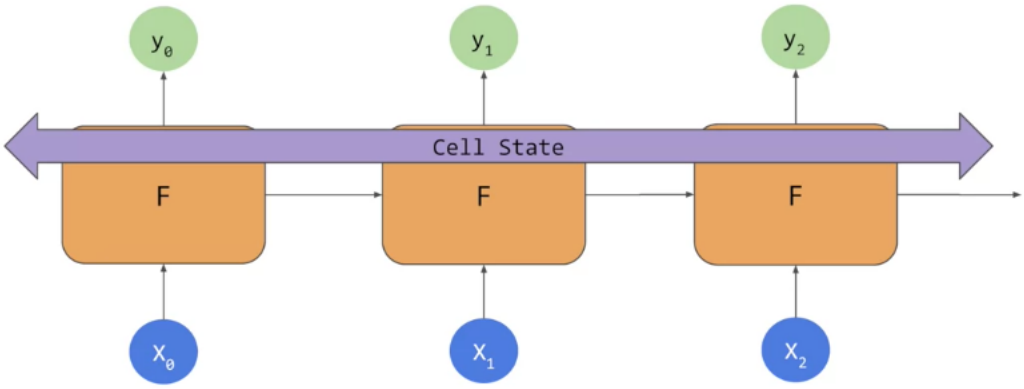
```
num_epochs = 10

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

history = model.fit(train_data,
                    epochs=num_epochs,
                    validation_data=test_data)
```



While losses decreasing, it's decreasing in a very small way. So why do you think that might be? Well, the keys in the fact that we're using sub-words and not for-words, sub-word meanings are often nonsensical and it's only when we put them together in sequences that they have meaningful semantics. Thus, some way from learning from sequences would be a great way forward, and that's exactly what you're going to do with recurrent neural networks.

Week-3 (LSTMs)





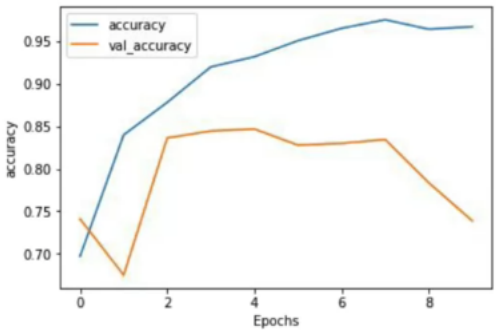

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(tokenizer.vocab_size, 64),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, None, 64)          523840
_________________________________________________________________
bidirectional_1 (Bidirection (None, 128)               66048
_________________________________________________________________
dense_4 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 65
=================================================================
Total params: 598,209
Trainable params: 598,209
Non-trainable params: 0
```

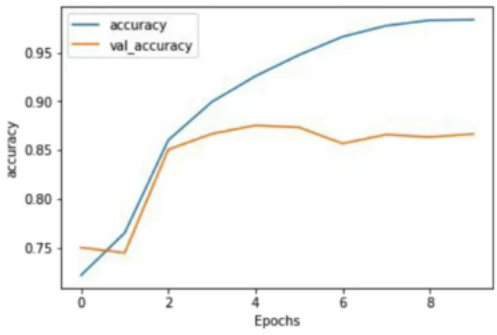
You can also stack LSTMs like any other keras layer by using code like below. But when you feed an LSTM into another one, you do have to put the return sequences equal true parameter into the first one. This ensures that the outputs of the LSTM match the desired inputs of the next one.

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(tokenizer.vocab_size, 64),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64, return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

10 Epochs : Accuracy Measurement

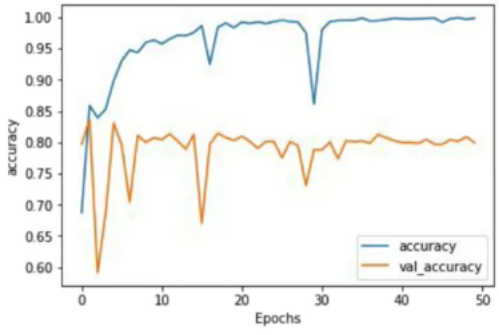


1 Layer LSTM

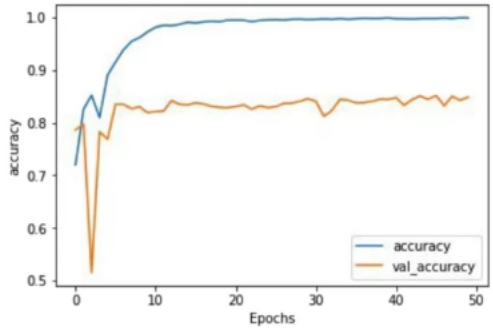


2 Layer LSTM

50 Epochs : Accuracy Measurement



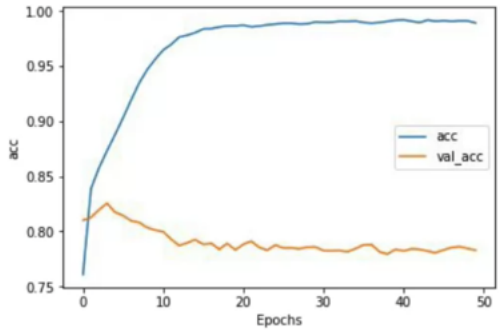
1 Layer LSTM

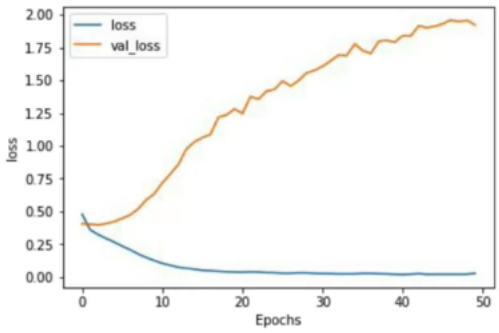


2 Layer LSTM

Using a convolutional network

```
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.GlobalMaxPooling1D(),
    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

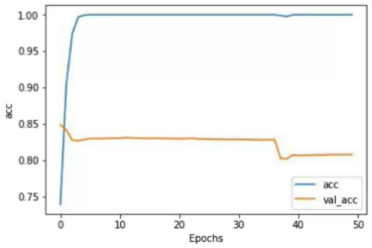


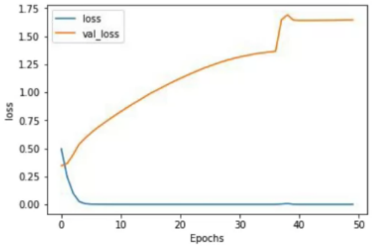



```
max_length = 120

tf.keras.layers.Conv1D(128, 5, activation='relu'),


Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 120, 16)           16000
_________________________________________________________________
conv1d (Conv1D)              (None, 116, 128)          10368
_________________________________________________________________
global_max_pooling1d (Global (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 24)                3096
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 25
=================================================================
Total params: 29,489
Trainable params: 29,489
Non-trainable params: 0
```

IMDB Dataset –

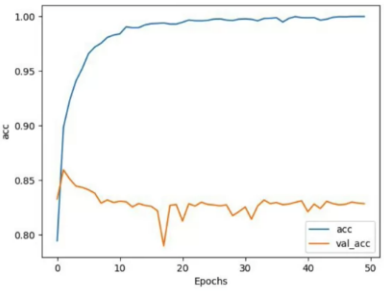
```
imdb, info = tfds.load("imdb_reviews", with_info=True, as_supervised=True)



model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Flatten(),
    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()
```

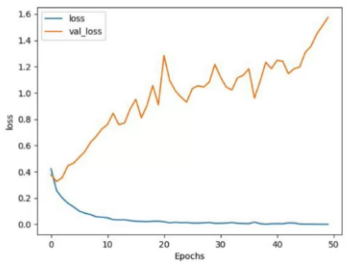




IMDB with Embedding-only : ~ 5s per epoch

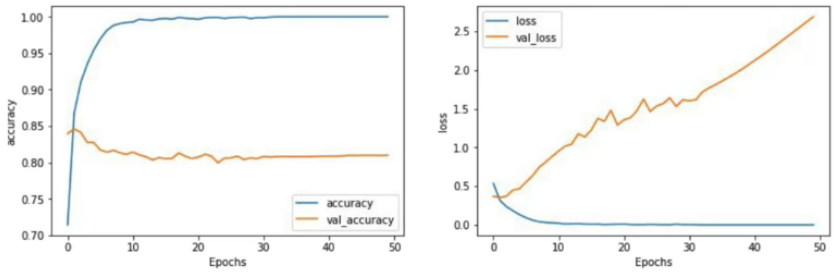
```
imdb, info = tfds.load("imdb_reviews", with_info=True, as_supervised=True)



# 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()
```





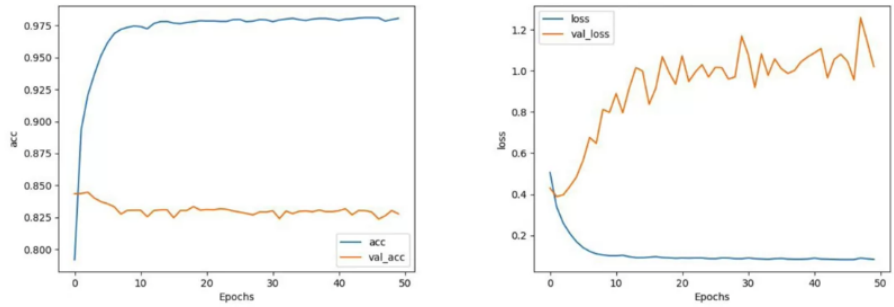
IMDB with LSTM ~43s per epoch

```
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()
```



IMDB with GRU : ~ 20s per epoch

```
# 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()
```



IMDB with CNN : ~ 6s per epoch

Week-4 (Text Generation)

```
tokenizer = Tokenizer()

data="In the town of Athy one Jeremy Lanigan \n Battered away ... ..."
corpus = data.lower().split("\n")

tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1
```

```
input_sequences = []
for line in corpus:
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
        input_sequences.append(n_gram_sequence)
```

```
In the town of Athy one Jeremy Lanigan

        ⬇

[4  2 66  8 67 68 69 70]
```

```
Line:                                   Input Sequences:

[4 2 66 8 67 68 69 70]                  [4 2]

                                        [4 2 66]

                                        [4 2 66 8]

                                        [4 2 66 8 67]

                                        [4 2 66 8 67 68]

                                        [4 2 66 8 67 68 69]

                                        [4 2 66 8 67 68 69 70]
```

```
max_sequence_len = max([len(x) for x in input_sequences])
```

```
input_sequences =
    np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))
```

```
Line:                          Padded Input Sequences:

[4 2 66 8 67 68 69 70]         [0 0 0 0 0 0 0 0 0 0 0 4 2]
                               [0 0 0 0 0 0 0 0 0 0 4 2 66]
                               [0 0 0 0 0 0 0 0 0 4 2 66 8]
                               [0 0 0 0 0 0 0 0 4 2 66 8 67]
                               [0 0 0 0 0 0 0 4 2 66 8 67 68]
                               [0 0 0 0 0 0 4 2 66 8 67 68 69]
                               [0 0 0 0 0 4 2 66 8 67 68 69 70]
```

```
                Padded Input Sequences:
Input (X)                                      Label (Y)
          [0 0 0 0 0 0 0 0 0 0 0 4 2]
          [0 0 0 0 0 0 0 0 0 0 4 2 66]
          [0 0 0 0 0 0 0 0 0 4 2 66 8]
          [0 0 0 0 0 0 0 0 4 2 66 8 67]
          [0 0 0 0 0 0 0 4 2 66 8 67 68]
          [0 0 0 0 0 0 4 2 66 8 67 68 69]
          [0 0 0 0 0 4 2 66 8 67 68 69 70]
```

```python
xs = input_sequences[:,:-1]
labels = input_sequences[:,-1]
```

```python
ys = tf.keras.utils.to_categorical(labels, num_classes=total_words)
```

```
Sentence: [0 0 0 0 4 2 66 8 67 68 69 70]

      X:[0 0 0 0 4 2 66 8 67 68 69]

  Label:[ 70 ]

     Y:[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
         0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
         0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
         0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.
         0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
         0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
         0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
         0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
         0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
         0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
         0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
         0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
         0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
         0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
```

```python
model = Sequential()
model.add(Embedding(total_words, 64, input_length=max_sequence_len - 1))
model.add((LSTM(20)))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(xs, ys, epochs=500, verbose=1)
```

We subtract one because we cropped off the last word of each sequence to get the label, so our sequences will be one less than the maximum sequence length.

```
model = Sequential()
model.add(Embedding(total_words, 64, input_length=max_sequence_len - 1))
model.add(Bidirectional(LSTM(20)))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(xs, ys, epochs=500, verbose=1)
```

Let's take a look at what happens if we change the code to be bidirectional. By adding this line simply defining the LSTM is bidirectional, and then retraining, I can see that I do converge a bit quicker as you'll see in this chart.

Predicting a word

```
seed_text = "Laurence went to dublin"
next_words = 10

for _ in range(next_words):
  token_list = tokenizer.texts_to_sequences([seed_text])[0]
  token_list = pad_sequences([token_list], maxlen=max_sequence_len - 1, padding='pre')
  predicted = model.predict_classes(token_list, verbose=0)
  output_word = ""
  for word, index in tokenizer.word_index.items():
    if index == predicted:
      output_word = word
      Break
  seed_text += " " + output_word
print(seed_text)
```

Updated the model make it work better with a larger corpus of work

Three things that you can experiment with. First, is the dimensionality of the embedding, 100 is purely arbitrary. Similarly, I increased the number of LSTN units to 150. Again, you can try different values or you can see how it behaves if you remove the bidirectional.

```
model = Sequential()
model.add(Embedding(total_words, 100, input_length=max_sequence_len-1))
model.add(Bidirectional(LSTM(150)))
model.add(Dense(total_words, activation='softmax'))
adam = Adam(lr=0.01)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
history = model.fit(xs, ys, epochs=100, verbose=1)
```