```
#@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");

Open in Colab

```
import csv
import tensorflow as tf
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

!wget --no-check-certificate \
    https://storage.googleapis.com/laurencemoroney-blog.appspot.com/bbc-text.csv \
    -O /tmp/bbc-text.csv
```

```
--2020-09-19 19:42:51--  https://storage.googleapis.com/laurencemoroney-blog.appspot.coı
Resolving storage.googleapis.com (storage.googleapis.com)... 173.194.214.128, 173.194.2:
Connecting to storage.googleapis.com (storage.googleapis.com)|173.194.214.128|:443... cc
HTTP request sent, awaiting response... 200 OK
Length: 5057493 (4.8M) [application/octet-stream]
Saving to: ‘/tmp/bbc-text.csv’

/tmp/bbc-text.csv   100%[===================>]   4.82M  --.-KB/s    in 0.05s

2020-09-19 19:42:52 (105 MB/s) - ‘/tmp/bbc-text.csv’ saved [5057493/5057493]
```

```
vocab_size = 10000
embedding_dim = 16
max_length = 120
trunc_type = "post"
padding_type = "post"
oov_tok = "<OOV>"
training_portion = .8
```

```
sentences = []
labels = []
stopwords = [ "a", "about", "above", "after", "again", "against", "all", "am", "an", "and", ".
print(len(stopwords))
```

```
# Expected Output
# 153
```

```
153
```

```
with open("/tmp/bbc-text.csv", 'r') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    next(reader)
    for label, sentence in reader:
      labels.append(label)
      sentence_to_remove_stop_words = sentence
      for stop_word in stopwords:
        word_token = " " + stop_word + " "
        sentence_to_remove_stop_words = sentence_to_remove_stop_words.replace(word_token, " "
      sentences.append(sentence_to_remove_stop_words)


print(len(labels))
print(len(sentences))
print(sentences[0])
# Expected Output
# 2225
# 2225
# tv future hands viewers home theatre systems  plasma high-definition tvs  digital video rec
```

```
2225
2225
tv future hands viewers home theatre systems  plasma high-definition tvs  digital video
```

```
#truncate decimals
train_size = int(len(sentences) * training_portion)

train_sentences = sentences[:train_size]
train_labels = labels[:train_size]

validation_sentences = sentences[train_size:]
validation_labels = labels[train_size:]

print(train_size)
print(len(train_sentences))
print(len(train_labels))
print(len(validation_sentences))
print(len(validation_labels))

# Expected output (if training_portion=.8)
# 1780
# 1780
# 1780
# 445
# 445
```

```
1780
1780
1780
445
445
```

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

train_sequences = tokenizer.texts_to_sequences(train_sentences)
train_padded = pad_sequences(train_sequences, padding=padding_type ,maxlen=max_length, trunca

print(len(train_sequences[0]))
print(len(train_padded[0]))

print(len(train_sequences[1]))
print(len(train_padded[1]))

print(len(train_sequences[10]))
print(len(train_padded[10]))

# Expected Ouput
# 449
# 120
# 200
# 120
# 192
# 120
```

```
449
120
200
120
192
120
```

```
validation_sequences = tokenizer.texts_to_sequences(validation_sentences)
validation_padded = pad_sequences(validation_sequences, padding=padding_type, maxlen=max_leng

print(len(validation_sequences))
print(validation_padded.shape)

# Expected output
# 445
# (445, 120)
```

```
445
(445, 120)
```

```
label_tokenizer = Tokenizer()
label_tokenizer.fit_on_texts(labels)

training_label_seq = np.array(label_tokenizer.texts_to_sequences(train_labels))
validation_label_seq = np.array(label_tokenizer.texts_to_sequences(validation_labels))

print(training_label_seq[0])
print(training_label_seq[1])
print(training_label_seq[2])
print(training_label_seq.shape)

print(validation_label_seq[0])
print(validation_label_seq[1])
print(validation_label_seq[2])
print(validation_label_seq.shape)

# Expected output
# [4]
# [2]
# [1]
# (1780, 1)
# [5]
# [4]
# [3]
# (445, 1)
```

```
[4]
[2]
[1]
(1780, 1)
[5]
[4]
[3]
(445, 1)
```

```
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(6, activation='softmax')
])
model.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()

# Expected Output
# Layer (type)                 Output Shape              Param #
# =================================================================
# embedding (Embedding)        (None, 120, 16)           16000
# _________________________________________________________________
# global_average_pooling1d (Gl (None, 16)                0
# _________________________________________________________________
# dense (Dense)                (None, 24)                408
# _________________________________________________________________
# dense_1 (Dense)              (None, 6)                 150
# =================================================================
# Total params: 16,558
# Trainable params: 16,558
# Non-trainable params: 0
```

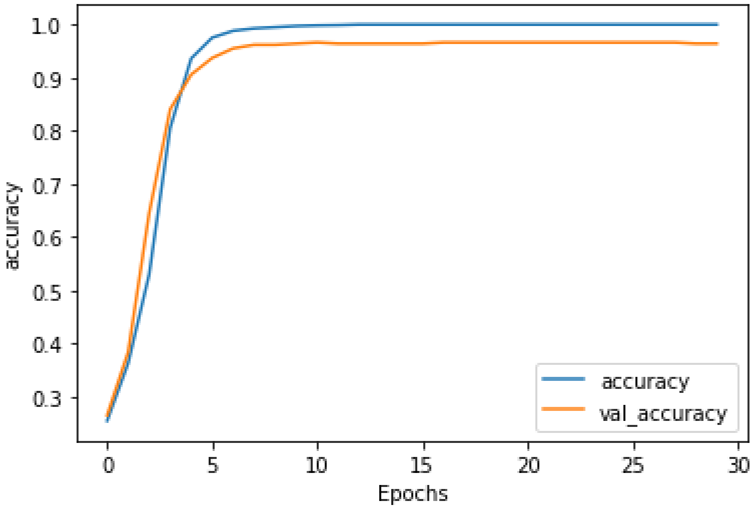
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 120, 16)           160000
_________________________________________________________________
global_average_pooling1d (Gl (None, 16)                0
_________________________________________________________________
dense (Dense)                (None, 24)                408
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 150
=================================================================
Total params: 160,558
Trainable params: 160,558
Non-trainable params: 0
_________________________________________________________________
```

```
num_epochs = 30
history = model.fit(train_padded, training_label_seq, epochs=num_epochs, validation_data=(val
     Epoch 1/30
     56/56 [==============================] - 1s 9ms/step - loss: 1.7548 - accuracy: 0.2551 -
     Epoch 2/30
     56/56 [==============================] - 0s 5ms/step - loss: 1.6230 - accuracy: 0.3635 -
     Epoch 3/30
     56/56 [==============================] - 0s 5ms/step - loss: 1.3826 - accuracy: 0.5303 -
     Epoch 4/30
     56/56 [==============================] - 0s 5ms/step - loss: 1.0676 - accuracy: 0.8051 -
     Epoch 5/30
     56/56 [==============================] - 0s 5ms/step - loss: 0.7557 - accuracy: 0.9360 -
     Epoch 6/30
     56/56 [==============================] - 0s 4ms/step - loss: 0.4980 - accuracy: 0.9753 -
     Epoch 7/30
     56/56 [==============================] - 0s 5ms/step - loss: 0.3178 - accuracy: 0.9882 -
     Epoch 8/30
     56/56 [==============================] - 0s 5ms/step - loss: 0.2067 - accuracy: 0.9927 -
     Epoch 9/30
     56/56 [==============================] - 0s 5ms/step - loss: 0.1407 - accuracy: 0.9949 -
     Epoch 10/30
     56/56 [==============================] - 0s 4ms/step - loss: 0.1000 - accuracy: 0.9972 -
     Epoch 11/30
     56/56 [==============================] - 0s 5ms/step - loss: 0.0740 - accuracy: 0.9983 -
     Epoch 12/30
     56/56 [==============================] - 0s 5ms/step - loss: 0.0562 - accuracy: 0.9989 -
     Epoch 13/30
     56/56 [==============================] - 0s 5ms/step - loss: 0.0438 - accuracy: 1.0000 -
     Epoch 14/30
     56/56 [==============================] - 0s 5ms/step - loss: 0.0348 - accuracy: 1.0000 -
     Epoch 15/30
     56/56 [==============================] - 0s 6ms/step - loss: 0.0282 - accuracy: 1.0000 -
     Epoch 16/30
     56/56 [==============================] - 0s 5ms/step - loss: 0.0232 - accuracy: 1.0000 -
```

```
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")
```



1.75
1.50
1.25
1.00
loss
loss
val_loss

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

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

Epochs

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

# Expected output
# (1000, 16)
```

```
(10000, 16)
```

```
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()
```

```
try:
  from google.colab import files
except ImportError:
  pass
else:
  files.download('vecs.tsv')
  files.download('meta.tsv')
```