untitled7

May 3, 2023

[1]:
```
from keras.datasets import imdb

# Load the data, keeping only 10,000 of the most frequently occuring words
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words␣
 ↪= 10000)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/imdb.npz
17464789/17464789 [==============================] - 4s 0us/step
```

[2]:
```
# Since we restricted ourselves to the top 10000 frequent words, no word index␣
 ↪should exceed 10000
# we'll verify this below

# Here is a list of maximum indexes in every review --- we search the maximum␣
 ↪index in this list of max indexes
print(type([max(sequence) for sequence in train_data]))

# Find the maximum of all max indexes
max([max(sequence) for sequence in train_data])
```

```
<class 'list'>
```

[2]:
```
9999
```

[3]:
```
import numpy as np

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))    # Creates an all zero␣
 ↪matrix of shape (len(sequences),10K)
    for i,sequence in enumerate(sequences):
        results[i,sequence] = 1                        # Sets specific indices␣
 ↪of results[i] to 1s
    return results

# Vectorize training Data
X_train = vectorize_sequences(train_data)
```

```
# Vectorize testing Data
X_test = vectorize_sequences(test_data)
```

```
[4]: X_train[0]
```

```
[4]: array([0., 1., 1., …, 0., 0., 0.])
```

```
[5]: X_train.shape
```

```
[5]: (25000, 10000)
```

```
[6]: y_train = np.asarray(train_labels).astype('float32')
     y_test  = np.asarray(test_labels).astype('float32')
```

```
[13]: import keras.models
      from keras.layers import Activation, Dense
      from keras.models import Sequential
      model=Sequential()
      model.add(Dense(16, activation='relu'))
```

```
[14]: Dense(16, activation='relu')
```

```
[14]: <keras.layers.core.dense.Dense at 0x268e3acdee0>
```

```
[15]: model.add(Activation('relu'))
```

```
[20]: from keras import models
      from keras import layers

      model = models.Sequential()
      model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
      model.add(layers.Dense(16, activation='relu'))
      model.add(layers.Dense(1, activation='sigmoid'))
```

```
[21]: from keras import optimizers
      from keras import losses
      from keras import metrics

      model.compile(optimizer=optimizers.RMSprop(lr=0.001),
                    loss = losses.binary_crossentropy,
                    metrics = [metrics.binary_accuracy])
```

```
C:\Users\avcom\anaconda3\lib\site-
packages\keras\optimizers\legacy\rmsprop.py:143: UserWarning: The `lr` argument
is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
```

[22]:
```
# Input for Validation
X_val = X_train[:10000]
partial_X_train = X_train[10000:]

# Labels for validation
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

[23]:
```
history = model.fit(partial_X_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(X_val, y_val))
```

```
Epoch 1/20
30/30 [==============================] - 1s 26ms/step - loss: 0.5204 -
binary_accuracy: 0.7866 - val_loss: 0.4062 - val_binary_accuracy: 0.8498
Epoch 2/20
30/30 [==============================] - 0s 16ms/step - loss: 0.3112 -
binary_accuracy: 0.8989 - val_loss: 0.3076 - val_binary_accuracy: 0.8854
Epoch 3/20
30/30 [==============================] - 0s 15ms/step - loss: 0.2249 -
binary_accuracy: 0.9251 - val_loss: 0.2817 - val_binary_accuracy: 0.8898
Epoch 4/20
30/30 [==============================] - 0s 14ms/step - loss: 0.1736 -
binary_accuracy: 0.9447 - val_loss: 0.2769 - val_binary_accuracy: 0.8892
Epoch 5/20
30/30 [==============================] - 0s 15ms/step - loss: 0.1440 -
binary_accuracy: 0.9539 - val_loss: 0.2819 - val_binary_accuracy: 0.8859
Epoch 6/20
30/30 [==============================] - 0s 14ms/step - loss: 0.1136 -
binary_accuracy: 0.9665 - val_loss: 0.3262 - val_binary_accuracy: 0.8792
Epoch 7/20
30/30 [==============================] - 0s 15ms/step - loss: 0.0940 -
binary_accuracy: 0.9739 - val_loss: 0.3346 - val_binary_accuracy: 0.8760
Epoch 8/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0766 -
binary_accuracy: 0.9808 - val_loss: 0.3398 - val_binary_accuracy: 0.8825
Epoch 9/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0627 -
binary_accuracy: 0.9839 - val_loss: 0.3620 - val_binary_accuracy: 0.8814
Epoch 10/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0511 -
binary_accuracy: 0.9881 - val_loss: 0.3767 - val_binary_accuracy: 0.8776
Epoch 11/20
30/30 [==============================] - 0s 14ms/step - loss: 0.0412 -
binary_accuracy: 0.9910 - val_loss: 0.4066 - val_binary_accuracy: 0.8777
```

```
Epoch 12/20
30/30 [==============================] - 0s 15ms/step - loss: 0.0308 -
binary_accuracy: 0.9943 - val_loss: 0.4354 - val_binary_accuracy: 0.8708
Epoch 13/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0245 -
binary_accuracy: 0.9962 - val_loss: 0.5063 - val_binary_accuracy: 0.8731
Epoch 14/20
30/30 [==============================] - 0s 14ms/step - loss: 0.0206 -
binary_accuracy: 0.9965 - val_loss: 0.4988 - val_binary_accuracy: 0.8736
Epoch 15/20
30/30 [==============================] - 0s 14ms/step - loss: 0.0157 -
binary_accuracy: 0.9969 - val_loss: 0.5302 - val_binary_accuracy: 0.8680
Epoch 16/20
30/30 [==============================] - 0s 14ms/step - loss: 0.0091 -
binary_accuracy: 0.9995 - val_loss: 0.5672 - val_binary_accuracy: 0.8716
Epoch 17/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0095 -
binary_accuracy: 0.9989 - val_loss: 0.5856 - val_binary_accuracy: 0.8679
Epoch 18/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0105 -
binary_accuracy: 0.9975 - val_loss: 0.6258 - val_binary_accuracy: 0.8695
Epoch 19/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0037 -
binary_accuracy: 0.9997 - val_loss: 0.6506 - val_binary_accuracy: 0.8674
Epoch 20/20
30/30 [==============================] - 0s 12ms/step - loss: 0.0028 -
binary_accuracy: 1.0000 - val_loss: 0.7914 - val_binary_accuracy: 0.8620
```

[24]:
```
history_dict = history.history
history_dict.keys()
```

[24]:
```
dict_keys(['loss', 'binary_accuracy', 'val_loss', 'val_binary_accuracy'])
```

[25]:
```
model.fit(partial_X_train,
                   partial_y_train,
                   epochs=3,
                   batch_size=512,
                   validation_data=(X_val, y_val))
```

```
Epoch 1/3
30/30 [==============================] - 1s 21ms/step - loss: 0.0048 -
binary_accuracy: 0.9991 - val_loss: 0.7242 - val_binary_accuracy: 0.8663
Epoch 2/3
30/30 [==============================] - 0s 16ms/step - loss: 0.0016 -
binary_accuracy: 1.0000 - val_loss: 0.7717 - val_binary_accuracy: 0.8659
Epoch 3/3
30/30 [==============================] - 0s 15ms/step - loss: 0.0044 -
binary_accuracy: 0.9989 - val_loss: 0.8017 - val_binary_accuracy: 0.8644
```

```
[25]: <keras.callbacks.History at 0x2688f96f4c0>
```

```
[26]: # Making Predictions for testing data
      np.set_printoptions(suppress=True)
      result = model.predict(X_test)
```

```
782/782 [==============================] - 1s 893us/step
```

```
[27]: result
```

```
[27]: array([[0.00323082],
              [1.        ],
              [0.99471897],
              …,
              [0.00112891],
              [0.01904037],
              [0.81179774]], dtype=float32)
```

```
[ ]:
```