17) Sentiment Analysis on Movie Reviews

Vitor Kamada

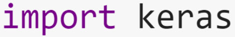
August 2019

Reference

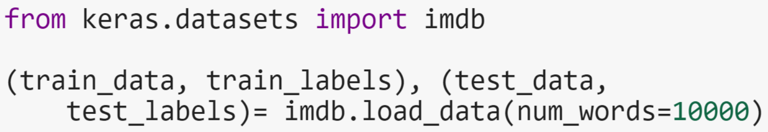
Chollet, F. (2018). **Deep Learning with Python**. Ch 3.

https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/3.5-classifying-movie-reviews.ipynb

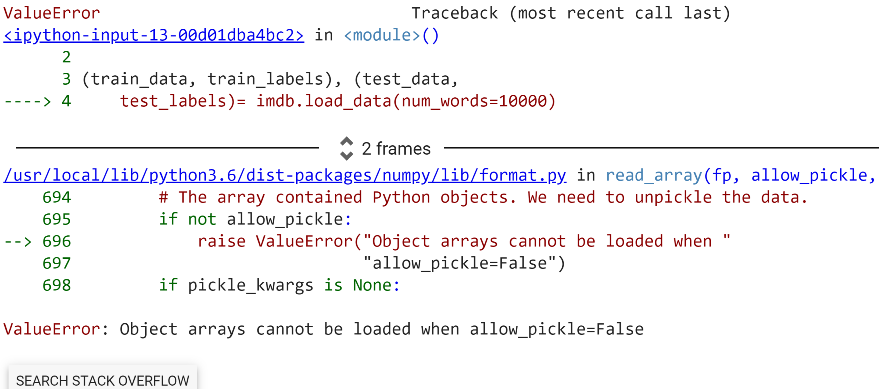
Internet Movie Database (IMDB)



```
import keras
```



```
from keras.datasets import imdb

(train_data, train_labels), (test_data,
    test_labels)= imdb.load_data(num_words=10000)
```



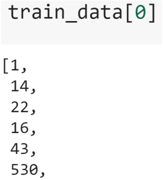
```
ValueError                                Traceback (most recent call last)
<ipython-input-13-00d01dba4bc2> in <module>()
      2
      3 (train_data, train_labels), (test_data,
----> 4     test_labels)= imdb.load_data(num_words=10000)

                                 2 frames

/usr/local/lib/python3.6/dist-packages/numpy/lib/format.py in read_array(fp, allow_pickle,
    694         # The array contained Python objects. We need to unpickle the data.
    695         if not allow_pickle:
--> 696             raise ValueError("Object arrays cannot be loaded when "
    697                              "allow_pickle=False")
    698         if pickle_kwargs is None:

ValueError: Object arrays cannot be loaded when allow_pickle=False
```

New Code to Load IMDB

```python
import numpy as np
# save np.load
np_load_old = np.load

# modify the default parameters of np.load
np.load = lambda *a,**k: np_load_old(*a,
                                 allow_pickle=True, **k)

# call load_data with allow_pickle implicitly set to true
(train_data, train_labels), (test_data,
          test_labels) = imdb.load_data(num_words=10000)

# restore np.load for future normal usage
np.load = np_load_old
```

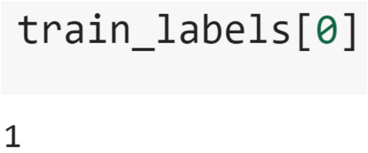
Reviews	**Total**	**Positive**	**Negative**
Training	25,000	50%	50%
Testing	25,000	50%	50%

Decode Back to English



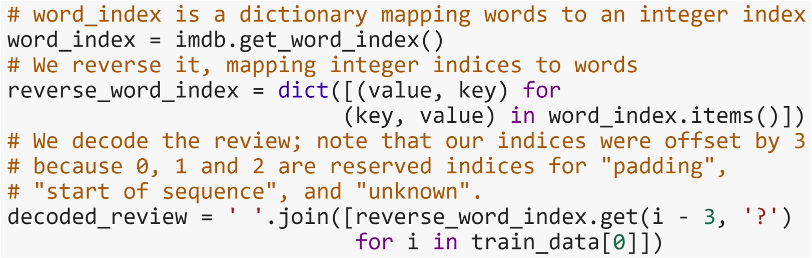
```
train_data[0]
```

```
[1,
 14,
 22,
 16,
 43,
 530,
```

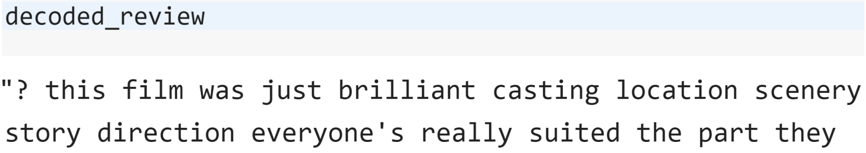


```
train_labels[0]
```

```
1
```



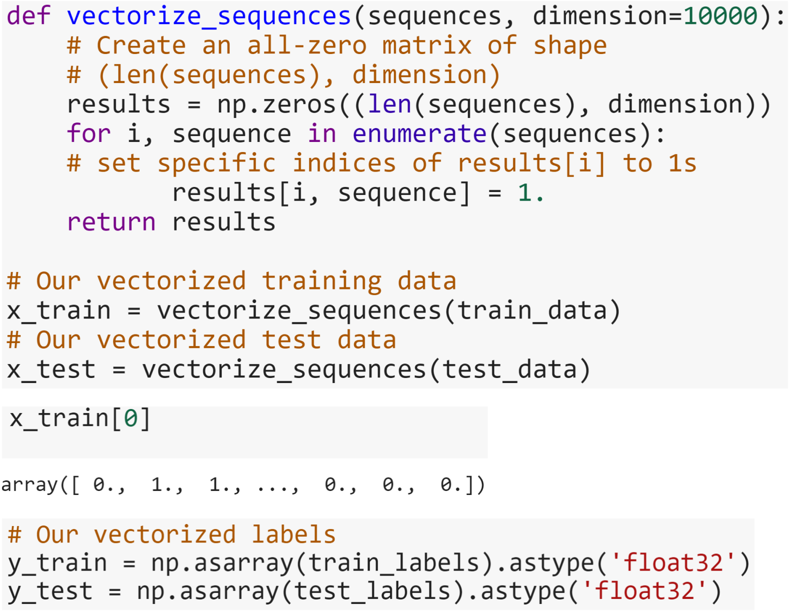
```
# word_index is a dictionary mapping words to an integer index
word_index = imdb.get_word_index()
# We reverse it, mapping integer indices to words
reverse_word_index = dict([(value, key) for
                          (key, value) in word_index.items()])
# We decode the review; note that our indices were offset by 3
# because 0, 1 and 2 are reserved indices for "padding",
# "start of sequence", and "unknown".
decoded_review = ' '.join([reverse_word_index.get(i - 3, '?')
                           for i in train_data[0]])
```



```
decoded_review
```

```
"? this film was just brilliant casting location scenery
 story direction everyone's really suited the part they
```

Vectorize the Data



```
def vectorize_sequences(sequences, dimension=10000):
    # Create an all-zero matrix of shape
    # (len(sequences), dimension)
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
    # set specific indices of results[i] to 1s
           results[i, sequence] = 1.
    return results

# Our vectorized training data
x_train = vectorize_sequences(train_data)
# Our vectorized test data
x_test = vectorize_sequences(test_data)
```

```
x_train[0]
```

```
array([ 0.,  1.,  1., ...,  0.,  0.,  0.])
```

```
# Our vectorized labels
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

Activation Function

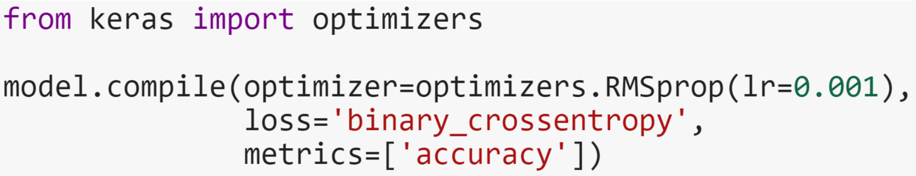
```
from keras import models
from keras import layers
model = models.Sequential()
model.add(layers.Dense(16, activation='relu',
                       input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

$$\sum_{k=1}^{16} z_{1k} = max(0, \sum_{j=1}^{10000} w_{1j}x_{1j} + b_1)$$

$$\sum_{l=1}^{16} z_{2l} = max(0, \sum_{k=1}^{16} w_{2k}z_{1k} + b_2)$$

$$y = 1/exp(-\sum_{l=1}^{16} w_{3l}z_{2l} + b_3)$$

Optimizer



```
from keras import optimizers

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

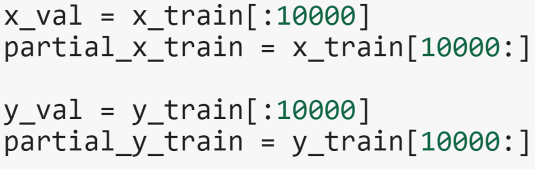
$$r \leftarrow \rho r + (1 - \rho) g \odot g$$

$$\Delta W = -\frac{\epsilon}{\sqrt{\delta + r}} \odot g$$

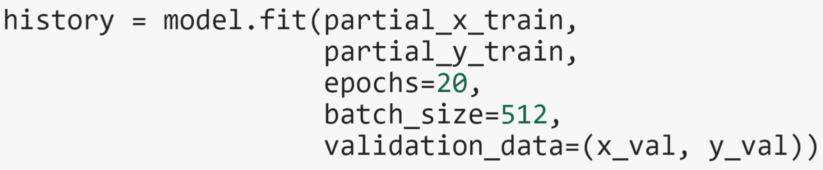
$$-\hat{p}_0 log(\hat{p}_0) - \hat{p}_1 log(\hat{p}_1)$$

$$\hat{p}_c = \frac{1}{N} \sum_{n=1}^{N} I(y_i = c), \qquad c = 0, 1$$

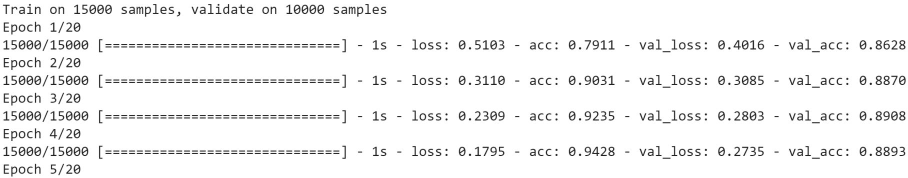
Validating the Approach



```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

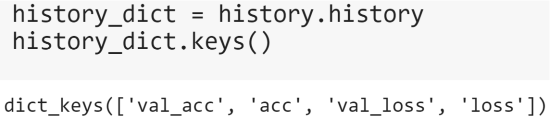


```
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```



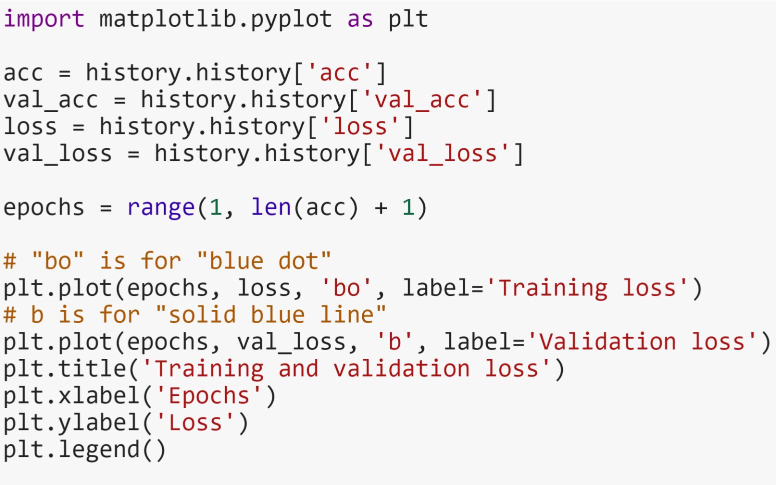
```
Train on 15000 samples, validate on 10000 samples
Epoch 1/20
15000/15000 [==============================] - 1s - loss: 0.5103 - acc: 0.7911 - val_loss: 0.4016 - val_acc: 0.8628
Epoch 2/20
15000/15000 [==============================] - 1s - loss: 0.3110 - acc: 0.9031 - val_loss: 0.3085 - val_acc: 0.8870
Epoch 3/20
15000/15000 [==============================] - 1s - loss: 0.2309 - acc: 0.9235 - val_loss: 0.2803 - val_acc: 0.8908
Epoch 4/20
15000/15000 [==============================] - 1s - loss: 0.1795 - acc: 0.9428 - val_loss: 0.2735 - val_acc: 0.8893
Epoch 5/20
```

History Object



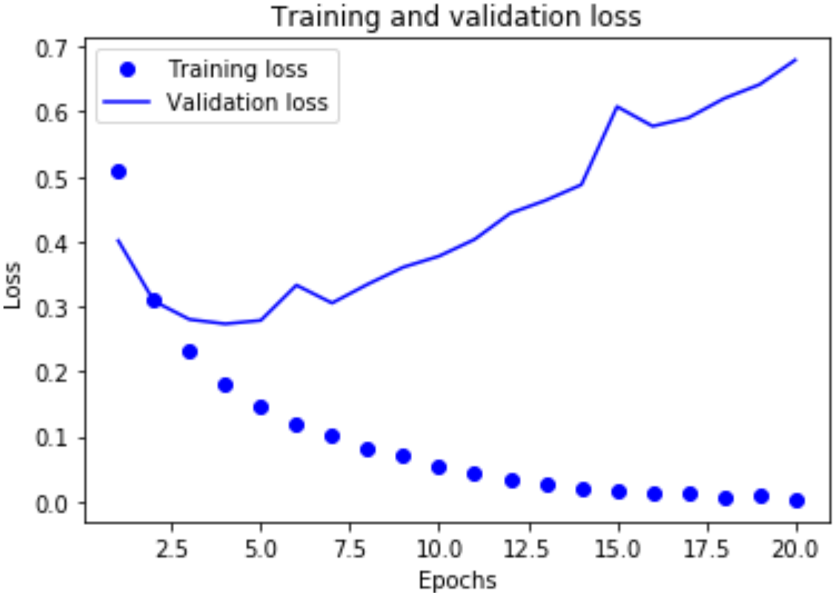
```
history_dict = history.history
history_dict.keys()
```

```
dict_keys(['val_acc', 'acc', 'val_loss', 'loss'])
```

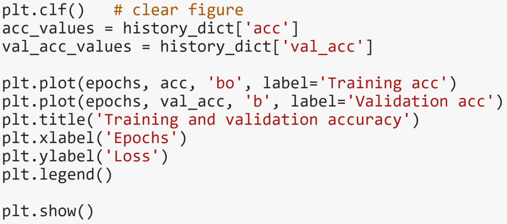


```
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

# "bo" is for "blue dot"
plt.plot(epochs, loss, 'bo', label='Training loss')
# b is for "solid blue line"
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

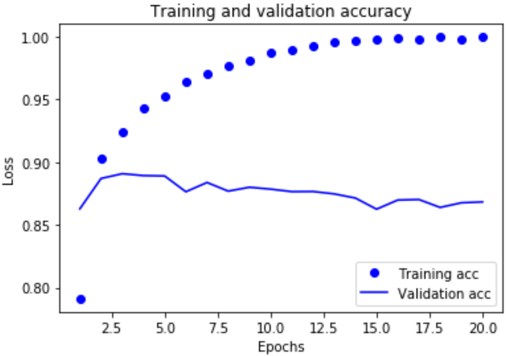
Training and Validation Loss



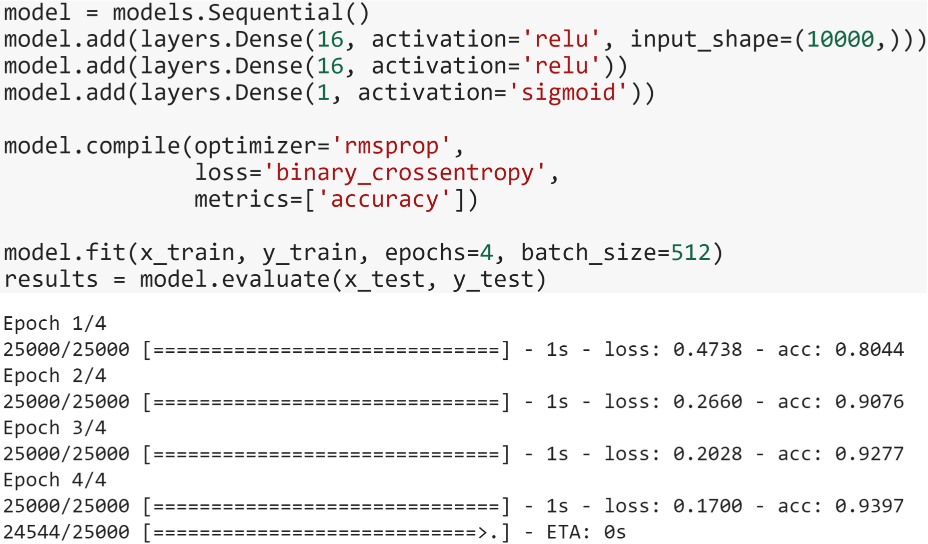
Training and Validation Accuracy



```
plt.clf()   # clear figure
acc_values = history_dict['acc']
val_acc_values = history_dict['val_acc']

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



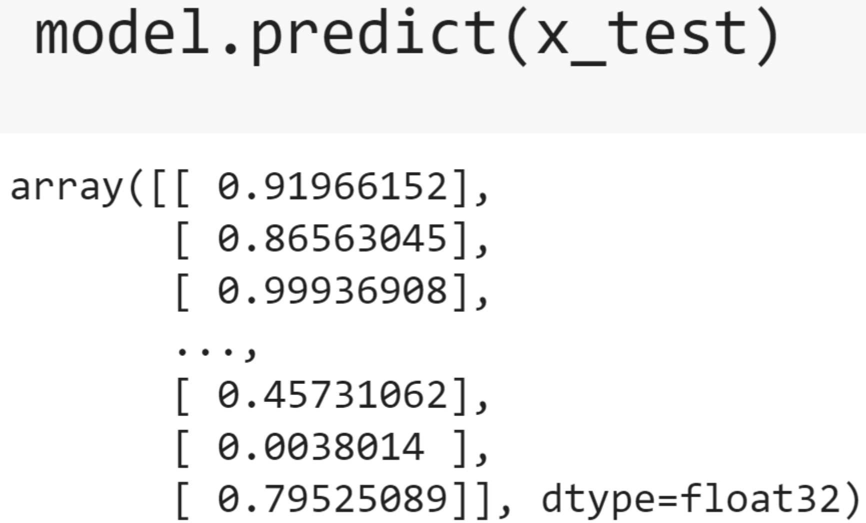
Evaluate on Test Data



```python
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'))

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

model.fit(x_train, y_train, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/4
25000/25000 [==============================] - 1s - loss: 0.4738 - acc: 0.8044
Epoch 2/4
25000/25000 [==============================] - 1s - loss: 0.2660 - acc: 0.9076
Epoch 3/4
25000/25000 [==============================] - 1s - loss: 0.2028 - acc: 0.9277
Epoch 4/4
25000/25000 [==============================] - 1s - loss: 0.1700 - acc: 0.9397
24544/25000 [============================>.] - ETA: 0s
```

Generate Predictions



```
model.predict(x_test)
```

```
array([[ 0.91966152],
       [ 0.86563045],
       [ 0.99936908],
       ...,
       [ 0.45731062],
       [ 0.0038014 ],
       [ 0.79525089]], dtype=float32)
```