IMDB_In_Keras

September 15, 2019

1 Analyzing IMDB Data in Keras

```
In [1]: # Imports
        import numpy as np
        import keras
        from keras.datasets import imdb
        from keras.models import Sequential
        from keras.layers import Dense, Dropout, Activation
        from keras.preprocessing.text import Tokenizer
        import matplotlib.pyplot as plt
        %matplotlib inline

        np.random.seed(42)
```

```
Using TensorFlow backend.
```

1.1 1. Loading the data

This dataset comes preloaded with Keras, so one simple command will get us training and testing data. There is a parameter for how many words we want to look at. We've set it at 1000, but feel free to experiment.

```
In [2]: # Loading the data (it's preloaded in Keras)
        (x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=1000)

        print(x_train.shape)
        print(x_test.shape)
```

```
Downloading data from https://s3.amazonaws.com/text-datasets/imdb.npz
17465344/17464789 [==============================] - 0s 0us/step
(25000,)
(25000,)
```

1.2 2. Examining the data

Notice that the data has been already pre-processed, where all the words have numbers, and the reviews come in as a vector with the words that the review contains. For example, if the word 'the' is the first one in our dictionary, and a review contains the word 'the', then there is a 1 in the corresponding vector.

The output comes as a vector of 1's and 0's, where 1 is a positive sentiment for the review, and 0 is negative.

```
In [3]: print(x_train[0])
        print(y_train[0])
```

```
[1, 11, 2, 11, 4, 2, 745, 2, 299, 2, 590, 2, 2, 37, 47, 27, 2, 2, 2, 19, 6, 2, 15, 2, 2, 17, 2,
1
```

1.3 3. One-hot encoding the output

Here, we'll turn the input vectors into (0,1)-vectors. For example, if the pre-processed vector contains the number 14, then in the processed vector, the 14th entry will be 1.

```
In [4]: # One-hot encoding the output into vector mode, each of length 1000
        tokenizer = Tokenizer(num_words=1000)
        x_train = tokenizer.sequences_to_matrix(x_train, mode='binary')
        x_test = tokenizer.sequences_to_matrix(x_test, mode='binary')
        print(x_train[0])
```

```
[ 0.  1.  1.  0.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
  1.  1.  0.  1.  1.  1.  1.  0.  1.  1.  0.  1.  1.  0.  0.  1.  1.  1.
  1.  1.  0.  1.  1.  0.  0.  0.  1.  0.  1.  1.  1.  1.  1.  0.  1.  0.
  1.  1.  1.  0.  1.  0.  0.  1.  1.  0.  0.  1.  1.  0.  1.  1.  1.  0.
  0.  0.  0.  0.  0.  1.  0.  1.  0.  0.  1.  0.  1.  0.  1.  0.  1.  0.
  0.  0.  0.  1.  0.  0.  0.  0.  1.  0.  1.  1.  0.  0.  0.  0.  0.  1.
  0.  0.  0.  0.  1.  0.  0.  0.  0.  1.  1.  0.  0.  1.  1.  1.  0.  1.
  0.  1.  0.  0.  0.  0.  0.  1.  0.  0.  1.  0.  1.  0.  0.  0.  1.  0.
  1.  0.  1.  0.  1.  0.  1.  0.  1.  0.  0.  0.  0.  1.  0.  1.  0.  0.
  1.  0.  0.  0.  0.  1.  0.  1.  1.  0.  1.  0.  1.  0.  0.  1.  0.  0.
  1.  0.  0.  0.  0.  0.  0.  1.  1.  0.  0.  0.  1.  0.  0.  0.  0.  0.
  1.  0.  0.  0.  0.  0.  1.  1.  1.  0.  1.  0.  0.  0.  0.  0.  0.  0.
  0.  1.  0.  0.  0.  0.  0.  0.  0.  1.  1.  0.  0.  0.  0.  0.  1.  0.
  0.  0.  1.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  1.  0.  0.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.  1.
  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.
  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  1.  1.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  1.  1.  0.  1.  0.
  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.
  0.  1.  0.  0.  0.  0.  0.  0.  0.  1.  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.  1.  0.  0.  0.  0.  0.  0.  1.  0.  0.  1.
```

```
 0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.
 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.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  1.  0.  0.  1.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  1.  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.  1.  0.  0.  0.  0.  0.  0.  1.
 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.  1.  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.  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.  1.  0.
 0.  0.  0.  1.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  1.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  1.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  1.  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.
 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.  1.  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.  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.]
```

And we'll also one-hot encode the output.

```
In [5]: # One-hot encoding the output
        num_classes = 2
        y_train = keras.utils.to_categorical(y_train, num_classes)
        y_test = keras.utils.to_categorical(y_test, num_classes)
        print(y_train.shape)
        print(y_test.shape)
```

```
(25000, 2)
(25000, 2)
```

1.4 4. Building the model architecture

Build a model here using sequential. Feel free to experiment with different layers and sizes! Also, experiment adding dropout to reduce overfitting.

In [10]:
```
from keras.models import Sequential
# TODO: Build the model architecture
model = Sequential()

from keras.layers.core import Dense, Activation

model.add(Dense(32, activation ='relu', input_shape=(1000,)))
model.add(Dropout(0.5))
model.add(Dense(2, activation ='sigmoid'))


# TODO: Compile the model using a loss function and an optimizer.
model.compile(loss = 'binary_crossentropy', optimizer = 'sgd', metrics = ['accuracy'])
model.summary()


# TODO: Compile the model using a loss function and an optimizer.
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 32)                32032
_________________________________________________________________
dropout_2 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 66
=================================================================
Total params: 32,098
Trainable params: 32,098
Non-trainable params: 0
_________________________________________________________________
```

1.5 5. Training the model

Run the model here. Experiment with different batch_size, and number of epochs!

In [12]:
```
# TODO: Run the model. Feel free to experiment with different batch sizes and number of
model.fit(x_train, y_train, epochs = 15, batch_size = 32, verbose = 0, validation_data
```

Out[12]: <keras.callbacks.History at 0x7fe494aee358>

1.6 6. Evaluating the model

This will give you the accuracy of the model, as evaluated on the testing set. Can you get something over 85%?

In [13]:
```
score = model.evaluate(x_test, y_test, verbose=0)
print("Accuracy: ", score[1])
```

```
Accuracy:  0.8576
```

In [14]:
```
model.fit(x_train, y_train, epochs = 30, batch_size = 32, verbose = 0, validation_data
score = model.evaluate(x_test, y_test, verbose=0)
print("Accuracy: ", score[1])
```

```
Accuracy:  0.85492
```

In [ ]: