```
import numpy as np
from keras.datasets import reuters
```

```
vocabulary=7500 # we will only use the 7500 most frequently used words
```

Next block of code block has been commented out because it does not work anymore

```
# 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=vocab

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

```
np.load.__defaults__=(None, True, True, 'ASCII')
(train_data, train_labels), (test_data, test_labels) = reuters.load_data(num_words=voc
np.load.__defaults__=(None, False, True, 'ASCII')
```

In the next line of code we will print the lists that contain sequences of words represented by a word index. If indices we would need to add one pre-processing step using Tokenizer

```
print(train_data[1]) # train_data is a list of word sequences
```

Now we will vectorize the training and test data. Basically we will create a matrix where the rows are the revie (7500 columns). We will set a 1 in the correct column if the word of the review matches a word of the vocabu

```
def vectorize_sequences(sequences, dimension=vocabulary):
    results=np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence]=1
    return results
```

Now we apply the function to our training and test data as well as the labels.

```
x_train=vectorize_sequences(train_data)
x_test=vectorize_sequences(test_data)

from keras.utils.np_utils import to_categorical
one_hot_train_labels=to_categorical(train_labels)
one_hot_test_labels=to_categorical(test_labels)
```

```
from keras import models
from keras import layers
from keras import optimizers
from keras import losses
from keras import metrics

model=models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(vocabulary,)))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(46, activation='softmax'))
```

Validation Set

```
x_val=x_train[:1000]
partial_x_train=x_train[1000:]

y_val=one_hot_train_labels[:1000]
partial_y_train=one_hot_train_labels[1000:]
```

```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['acc'])
```

```
history = model.fit(partial_x_train, partial_y_train, epochs=20, batch_size=512, valid
```

```
import matplotlib.pyplot as plt
history_dict=history.history
loss_values=history_dict['loss']
val_loss_values=history_dict['val_loss']
epochs=range(1, 21)
```

```
plt.plot(epochs, loss_values, 'bo', label='Training Loss') #bo is for blue dotted line
plt.plot(epochs, val_loss_values, 'b', label='Validation Loss') #b is for blue line
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show
```

```
model=models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(vocabulary,)))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(46, activation='softmax'))

model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['acc'])
model.fit(partial_x_train, partial_y_train, epochs=12, batch_size=512, validation_data=
results=model.evaluate(x_test, one_hot_test_labels)
```

```
print(results)
```