In [1]:
```
cd /home/nv/Downloads
```

```
/home/nv/Downloads
```

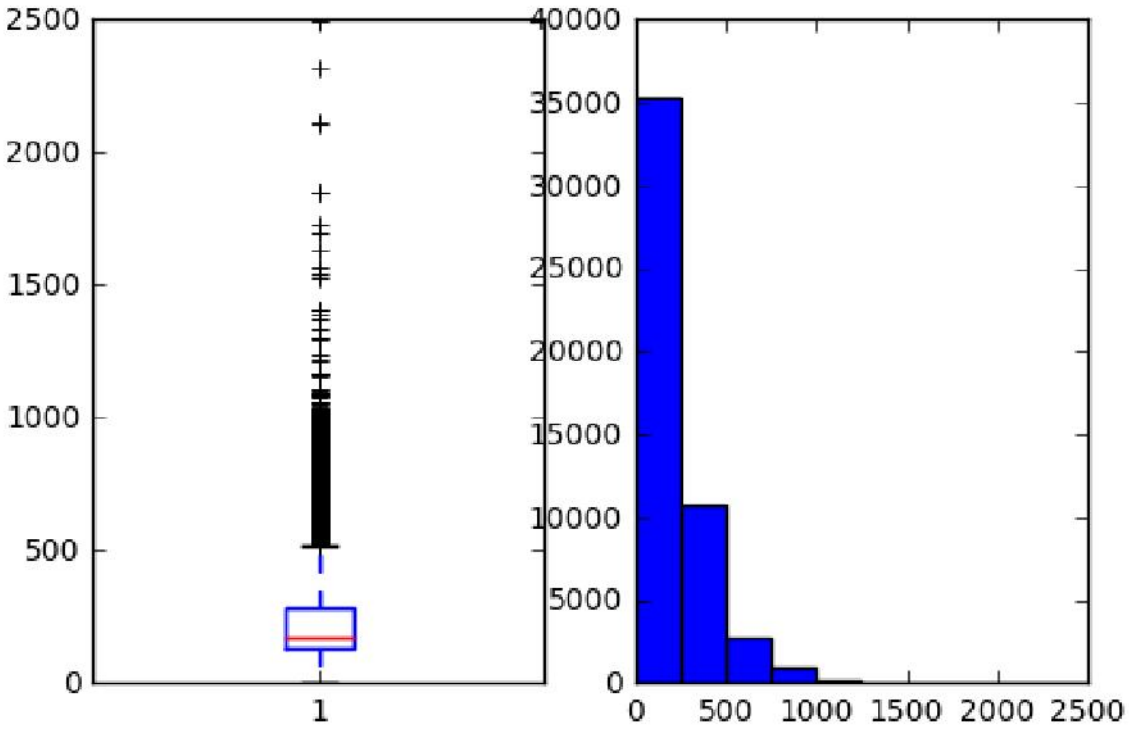
In [2]:
```
import keras
print keras.__version__
```

```
1.1.1
```

In [3]:
```
import theano
```

In [5]:
```
# Load and Plot the IMDB dataset
import numpy
from keras.datasets import imdb
from matplotlib import pyplot
# load the dataset
(X_train, y_train), (X_test, y_test) = imdb.load_data()
X = numpy.concatenate((X_train, X_test), axis=0)
y = numpy.concatenate((y_train, y_test), axis=0)
# summarize size
print("Training data: ")
print(X.shape)
print(y.shape)
# Summarize number of classes
print("Classes: ")
print(numpy.unique(y))
# Summarize number of words
print("Number of words: ")
print(len(numpy.unique(numpy.hstack(X))))
# Summarize review length
print("Review length: ")
result = map(len, X)
print("Mean %.2f words (%f)" % (numpy.mean(result), numpy.std(result)))
# plot review length as a boxplot and histogram
pyplot.subplot(121)
pyplot.boxplot(result)
pyplot.subplot(122)
pyplot.hist(result)
pyplot.show()
```

```
Training data:
(50000,)
(50000,)
Classes:
[0 1]
Number of words:
88585
Review length:
Mean 234.76 words (172.911495)
```



In [6]:
```
# MLP for the IMDB problem, optimizer='adam', nb_epoch =10, input_length(max_words)=500
import numpy
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
import matplotlib.pyplot as plt
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# load the dataset but only keep the top n words, zero the rest
top_words = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(nb_words=top_words)
max_words = 500
X_train = sequence.pad_sequences(X_train, maxlen=max_words)
X_test = sequence.pad_sequences(X_test, maxlen=max_words)
# create the model
model = Sequential()
model.add(Embedding(top_words, 32, input_length=max_words))
model.add(Flatten())
model.add(Dense(250, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), nb_epoch=10, batch_size=128,verbose=1)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
embedding_1 (Embedding)          (None, 500, 32)       160000      embedding_input_1[0][0]
____________________________________________________________________________________________________
flatten_1 (Flatten)              (None, 16000)         0           embedding_1[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 250)           4000250     flatten_1[0][0]
____________________________________________________________________________________________________
```

```
dense_2 (Dense)                  (None, 1)             251         dense_1[0][0]
==================================================================================================
Total params: 4160501
__________________________________________________________________________________________________
None
Train on 25000 samples, validate on 25000 samples
Epoch 1/10
25000/25000 [==============================] - 55s - loss: 0.5366 - acc: 0.6864 - val_loss: 0.2966 -
 val_acc: 0.8758
Epoch 2/10
25000/25000 [==============================] - 56s - loss: 0.2048 - acc: 0.9208 - val_loss: 0.2982 -
 val_acc: 0.8764
Epoch 3/10
25000/25000 [==============================] - 58s - loss: 0.0725 - acc: 0.9794 - val_loss: 0.3935 -
 val_acc: 0.8655
Epoch 4/10
25000/25000 [==============================] - 66s - loss: 0.0153 - acc: 0.9968 - val_loss: 0.4752 -
 val_acc: 0.8633
Epoch 5/10
25000/25000 [==============================] - 55s - loss: 0.0025 - acc: 0.9999 - val_loss: 0.5428 -
 val_acc: 0.8669
Epoch 6/10
25000/25000 [==============================] - 57s - loss: 7.3811e-04 - acc: 1.0000 - val_loss: 0.57
97 - val_acc: 0.8674
Epoch 7/10
25000/25000 [==============================] - 64s - loss: 4.1734e-04 - acc: 1.0000 - val_loss: 0.60
54 - val_acc: 0.8681
Epoch 8/10
25000/25000 [==============================] - 65s - loss: 2.7427e-04 - acc: 1.0000 - val_loss: 0.62
74 - val_acc: 0.8681
Epoch 9/10
25000/25000 [==============================] - 65s - loss: 1.9458e-04 - acc: 1.0000 - val_loss: 0.64
72 - val_acc: 0.8684
Epoch 10/10
25000/25000 [==============================] - 65s - loss: 1.4375e-04 - acc: 1.0000 - val_loss: 0.66
33 - val_acc: 0.8684
Accuracy: 86.84%
```

As we can see this model overfits very quicky so i will change it very few training epochs and sincce there is a lot of data so i will keep batch size same as 128

In [7]:
```
# MLP for the IMDB problem,optimizer='adam',input_length (max_words)=500, nb_epoch=5
import numpy
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
import matplotlib.pyplot as plt
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# load the dataset but only keep the top n words, zero the rest
top_words = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(nb_words=top_words)
max_words = 500
X_train = sequence.pad_sequences(X_train, maxlen=max_words)
X_test = sequence.pad_sequences(X_test, maxlen=max_words)
# create the model
model = Sequential()
model.add(Embedding(top_words, 32, input_length=max_words))
model.add(Flatten())
model.add(Dense(250, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), nb_epoch=5, batch_size=128,verbose=1)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================
embedding_2 (Embedding)      (None, 500, 32)           160000      embedding_input_2[0][0]
_________________________________________________________________
flatten_2 (Flatten)          (None, 16000)             0           embedding_2[0][0]
_________________________________________________________________
dense_3 (Dense)              (None, 250)               4000250     flatten_2[0][0]
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 251         dense_3[0][0]
=================================================================
Total params: 4160501
_________________________________________________________________
None
Train on 25000 samples, validate on 25000 samples
Epoch 1/5
25000/25000 [==============================] - 58s - loss: 0.5366 - acc: 0.6864 - val_loss: 0.2966 -
 val_acc: 0.8758
Epoch 2/5
25000/25000 [==============================] - 58s - loss: 0.2048 - acc: 0.9208 - val_loss: 0.2982 -
 val_acc: 0.8764
Epoch 3/5
25000/25000 [==============================] - 55s - loss: 0.0725 - acc: 0.9794 - val_loss: 0.3935 -
 val_acc: 0.8655
Epoch 4/5
25000/25000 [==============================] - 48s - loss: 0.0153 - acc: 0.9968 - val_loss: 0.4752 -
 val_acc: 0.8633
Epoch 5/5
25000/25000 [==============================] - 61s - loss: 0.0025 - acc: 0.9999 - val_loss: 0.5428 -
 val_acc: 0.8669
Accuracy: 86.69%
```

In [8]:
```
# MLP for the IMDB problem for input length 1000 (max_words),optimizer='adam',nb_epoch = 5
import numpy
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
import matplotlib.pyplot as plt
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# load the dataset but only keep the top n words, zero the rest
top_words = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(nb_words=top_words)
max_words = 1000
X_train = sequence.pad_sequences(X_train, maxlen=max_words)
X_test = sequence.pad_sequences(X_test, maxlen=max_words)
# create the model
model = Sequential()
model.add(Embedding(top_words, 32, input_length=max_words))
model.add(Flatten())
model.add(Dense(250, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), nb_epoch=5, batch_size=128,verbose=1)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================
embedding_3 (Embedding)      (None, 1000, 32)          160000      embedding_input_3[0][0]
_________________________________________________________________
flatten_3 (Flatten)          (None, 32000)             0           embedding_3[0][0]
_________________________________________________________________
dense_5 (Dense)              (None, 250)               8000250     flatten_3[0][0]
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 251         dense_5[0][0]
=================================================================
Total params: 8160501
_________________________________________________________________
None
Train on 25000 samples, validate on 25000 samples
Epoch 1/5
25000/25000 [==============================] - 109s - loss: 0.4993 - acc: 0.7350 - val_loss: 0.3011
- val_acc: 0.8716
Epoch 2/5
25000/25000 [==============================] - 105s - loss: 0.1888 - acc: 0.9294 - val_loss: 0.2982
- val_acc: 0.8744
Epoch 3/5
25000/25000 [==============================] - 103s - loss: 0.0619 - acc: 0.9826 - val_loss: 0.3901
- val_acc: 0.8668
Epoch 4/5
25000/25000 [==============================] - 130s - loss: 0.0099 - acc: 0.9985 - val_loss: 0.4957
- val_acc: 0.8660
Epoch 5/5
25000/25000 [==============================] - 109s - loss: 0.0019 - acc: 0.9999 - val_loss: 0.5418
- val_acc: 0.8678
Accuracy: 86.78%
```

In [14]:
```
# MLP for the IMDB problem for input length 500 (max_words),optimizer='rmsprop',nb_epoch = 5
import numpy
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
import matplotlib.pyplot as plt
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# load the dataset but only keep the top n words, zero the rest
top_words = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(nb_words=top_words)
max_words = 500
X_train = sequence.pad_sequences(X_train, maxlen=max_words)
X_test = sequence.pad_sequences(X_test, maxlen=max_words)
# create the model
model = Sequential()
model.add(Embedding(top_words, 32, input_length=max_words))
model.add(Flatten())
model.add(Dense(250, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
print(model.summary())
# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), nb_epoch=5, batch_size=128,verbose=1)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================
embedding_9 (Embedding)      (None, 500, 32)           160000      embedding_input_9[0][0]
_________________________________________________________________
flatten_9 (Flatten)          (None, 16000)             0           embedding_9[0][0]
_________________________________________________________________
dense_17 (Dense)             (None, 250)               4000250     flatten_9[0][0]
_________________________________________________________________
dense_18 (Dense)             (None, 1)                 251         dense_17[0][0]
=================================================================
Total params: 4160501
_________________________________________________________________
None
Train on 25000 samples, validate on 25000 samples
Epoch 1/5
25000/25000 [==============================] - 60s - loss: 0.4978 - acc: 0.7324 - val_loss: 0.3045 -
 val_acc: 0.8711
Epoch 2/5
25000/25000 [==============================] - 45s - loss: 0.1985 - acc: 0.9220 - val_loss: 0.3105 -
 val_acc: 0.8723
Epoch 3/5
25000/25000 [==============================] - 45s - loss: 0.0791 - acc: 0.9730 - val_loss: 0.4628 -
 val_acc: 0.8523
Epoch 4/5
25000/25000 [==============================] - 44s - loss: 0.0233 - acc: 0.9928 - val_loss: 0.5901 -
 val_acc: 0.8502
Epoch 5/5
25000/25000 [==============================] - 44s - loss: 0.0063 - acc: 0.9983 - val_loss: 0.7891 -
 val_acc: 0.8426
Accuracy: 84.26%
```

In [15]:
```
# MLP for the IMDB problem for input length 1000 (max_words),optimizer='rmsprop',nb_epoch = 5
import numpy
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
import matplotlib.pyplot as plt
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# load the dataset but only keep the top n words, zero the rest
top_words = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(nb_words=top_words)
max_words = 1000
X_train = sequence.pad_sequences(X_train, maxlen=max_words)
X_test = sequence.pad_sequences(X_test, maxlen=max_words)
# create the model
model = Sequential()
model.add(Embedding(top_words, 32, input_length=max_words))
model.add(Flatten())
model.add(Dense(250, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
print(model.summary())
# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), nb_epoch=5, batch_size=128,verbose=1)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_10 (Embedding)     (None, 1000, 32)          160000
_________________________________________________________________
flatten_10 (Flatten)         (None, 32000)             0
_________________________________________________________________
dense_19 (Dense)             (None, 250)               8000250
_________________________________________________________________
dense_20 (Dense)             (None, 1)                 251
=================================================================
Total params: 8160501
_________________________________________________________________
None
Train on 25000 samples, validate on 25000 samples
Epoch 1/5
25000/25000 [==============================] - 102s - loss: 0.5048 - acc: 0.7457 - val_loss: 0.3070
- val_acc: 0.8697
Epoch 2/5
25000/25000 [==============================] - 102s - loss: 0.2030 - acc: 0.9189 - val_loss: 0.3002
- val_acc: 0.8754
Epoch 3/5
25000/25000 [==============================] - 97s - loss: 0.0939 - acc: 0.9656 - val_loss: 0.3962 -
 val_acc: 0.8658
Epoch 4/5
25000/25000 [==============================] - 103s - loss: 0.0325 - acc: 0.9892 - val_loss: 0.5644
- val_acc: 0.8560
Epoch 5/5
25000/25000 [==============================] - 98s - loss: 0.0087 - acc: 0.9971 - val_loss: 0.6809 -
 val_acc: 0.8546
Accuracy: 85.46%
```

In [9]:
```
# CNN for the IMDB problem with input size =max_words 500,optimizer='adam', nb_epoch=5
import numpy
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Convolution1D
from keras.layers.convolutional import MaxPooling1D
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# load the dataset but only keep the top n words, zero the rest
top_words = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(nb_words=top_words)
# pad dataset to a maximum review length in words
max_words = 500
X_train = sequence.pad_sequences(X_train, maxlen=max_words)
X_test = sequence.pad_sequences(X_test, maxlen=max_words)
# create the model
model = Sequential()
model.add(Embedding(top_words, 32, input_length=max_words))
model.add(Convolution1D(nb_filter=32, filter_length=3, border_mode='same',
activation='relu'))
model.add(MaxPooling1D(pool_length=2))
model.add(Flatten())
model.add(Dense(250, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), nb_epoch=5, batch_size=128,
verbose=1)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
/home/nv/anaconda2/lib/python2.7/site-packages/keras/backend/theano_backend.py:1500: UserWarning: DE
PRECATION: the 'ds' parameter is not going to exist anymore as it is going to be replaced by the par
ameter 'ws'.
  mode='max')
/home/nv/anaconda2/lib/python2.7/site-packages/keras/backend/theano_backend.py:1500: UserWarning: DE
PRECATION: the 'st' parameter is not going to exist anymore as it is going to be replaced by the par
ameter 'stride'.
  mode='max')
/home/nv/anaconda2/lib/python2.7/site-packages/keras/backend/theano_backend.py:1500: UserWarning: DE
PRECATION: the 'padding' parameter is not going to exist anymore as it is going to be replaced by th
e parameter 'pad'.
  mode='max')
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================
embedding_4 (Embedding)      (None, 500, 32)           160000      embedding_input_4[0][0]
_________________________________________________________________
convolution1d_1 (Convolution1D)  (None, 500, 32)       3104        embedding_4[0][0]
_________________________________________________________________
maxpooling1d_1 (MaxPooling1D)    (None, 250, 32)       0           convolution1d_1[0][0]
_________________________________________________________________
flatten_4 (Flatten)          (None, 8000)              0           maxpooling1d_1[0][0]
_________________________________________________________________
dense_7 (Dense)              (None, 250)               2000250     flatten_4[0][0]
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 251         dense_7[0][0]
=================================================================
Total params: 2163605
_________________________________________________________________
None
Train on 25000 samples, validate on 25000 samples
Epoch 1/5
25000/25000 [==============================] - 74s - loss: 0.4857 - acc: 0.7164 - val_loss: 0.2764 -
 val_acc: 0.8864
Epoch 2/5
25000/25000 [==============================] - 74s - loss: 0.2249 - acc: 0.9123 - val_loss: 0.2675 -
 val_acc: 0.8890
Epoch 3/5
25000/25000 [==============================] - 74s - loss: 0.1680 - acc: 0.9359 - val_loss: 0.2855 -
 val_acc: 0.8841
Epoch 4/5
25000/25000 [==============================] - 120s - loss: 0.1242 - acc: 0.9554 - val_loss: 0.3293
- val_acc: 0.8810
Epoch 5/5
25000/25000 [==============================] - 125s - loss: 0.0825 - acc: 0.9729 - val_loss: 0.3956
- val_acc: 0.8749
Accuracy: 87.49%
```

In [13]:
```
# CNN for the IMDB problem with input size =max_words 1000,optimizer='adam', nb_epoch = 5
import numpy
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Convolution1D
from keras.layers.convolutional import MaxPooling1D
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# load the dataset but only keep the top n words, zero the rest
top_words = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(nb_words=top_words)
# pad dataset to a maximum review length in words
max_words = 1000
X_train = sequence.pad_sequences(X_train, maxlen=max_words)
X_test = sequence.pad_sequences(X_test, maxlen=max_words)
# create the model
model = Sequential()
model.add(Embedding(top_words, 32, input_length=max_words))
model.add(Convolution1D(nb_filter=32, filter_length=3, border_mode='same',
activation='relu'))
model.add(MaxPooling1D(pool_length=2))
model.add(Flatten())
model.add(Dense(250, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), nb_epoch=5, batch_size=128,
verbose=1)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

Layer (type)	Output Shape	Param #	Connected to
embedding_8 (Embedding)	(None, 1000, 32)	160000	embedding_input_8[0][0]
convolution1d_5 (Convolution1D)	(None, 1000, 32)	3104	embedding_8[0][0]
maxpooling1d_5 (MaxPooling1D)	(None, 500, 32)	0	convolution1d_5[0][0]
flatten_8 (Flatten)	(None, 16000)	0	maxpooling1d_5[0][0]
dense_15 (Dense)	(None, 250)	4000250	flatten_8[0][0]
dense_16 (Dense)	(None, 1)	251	dense_15[0][0]

Total params: 4163605

None
Train on 25000 samples, validate on 25000 samples
Epoch 1/5
25000/25000 [==============================] - 156s - loss: 0.4529 - acc: 0.7555 - val_loss: 0.3227 - val_acc: 0.8612
Epoch 2/5
25000/25000 [==============================] - 186s - loss: 0.2360 - acc: 0.9060 - val_loss: 0.2768 - val_acc: 0.8850
Epoch 3/5
25000/25000 [==============================] - 170s - loss: 0.1769 - acc: 0.9329 - val_loss: 0.3257 - val_acc: 0.8713
Epoch 4/5
25000/25000 [==============================] - 212s - loss: 0.1306 - acc: 0.9521 - val_loss: 0.3263 - val_acc: 0.8785
Epoch 5/5
25000/25000 [==============================] - 209s - loss: 0.0845 - acc: 0.9724 - val_loss: 0.3977 - val_acc: 0.8707
Accuracy: 87.07%

In [11]:
```
# CNN for the IMDB problem with input size =max_words 500 , optimizer='rmsprop',nb_epoch =5
import numpy
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Convolution1D
from keras.layers.convolutional import MaxPooling1D
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# load the dataset but only keep the top n words, zero the rest
top_words = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(nb_words=top_words)
# pad dataset to a maximum review length in words
max_words = 500
X_train = sequence.pad_sequences(X_train, maxlen=max_words)
X_test = sequence.pad_sequences(X_test, maxlen=max_words)
# create the model
model = Sequential()
model.add(Embedding(top_words, 32, input_length=max_words))
model.add(Convolution1D(nb_filter=32, filter_length=3, border_mode='same',
activation='relu'))
model.add(MaxPooling1D(pool_length=2))
model.add(Flatten())
model.add(Dense(250, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
print(model.summary())
# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), nb_epoch=5, batch_size=128,
verbose=1)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================
embedding_6 (Embedding)      (None, 500, 32)           160000      embedding_input_6[0][0]
_________________________________________________________________
convolution1d_3 (Convolution1D) (None, 500, 32)        3104        embedding_6[0][0]
_________________________________________________________________
maxpooling1d_3 (MaxPooling1D) (None, 250, 32)          0           convolution1d_3[0][0]
_________________________________________________________________
flatten_6 (Flatten)          (None, 8000)              0           maxpooling1d_3[0][0]
_________________________________________________________________
dense_11 (Dense)             (None, 250)               2000250     flatten_6[0][0]
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 251         dense_11[0][0]
=================================================================
Total params: 2163605
_________________________________________________________________
None
Train on 25000 samples, validate on 25000 samples
Epoch 1/5
25000/25000 [==============================] - 100s - loss: 0.4547 - acc: 0.7606 - val_loss: 0.3823
- val_acc: 0.8306
Epoch 2/5
25000/25000 [==============================] - 98s - loss: 0.2494 - acc: 0.8985 - val_loss: 0.2613 -
 val_acc: 0.8920
Epoch 3/5
25000/25000 [==============================] - 98s - loss: 0.2039 - acc: 0.9195 - val_loss: 0.2836 -
 val_acc: 0.8806
Epoch 4/5
25000/25000 [==============================] - 158s - loss: 0.1755 - acc: 0.9320 - val_loss: 0.2996
- val_acc: 0.8814
Epoch 5/5
25000/25000 [==============================] - 109s - loss: 0.1482 - acc: 0.9443 - val_loss: 0.3039
- val_acc: 0.8852
Accuracy: 88.52%
```

In [12]:
```
# CNN for the IMDB problem with input size =max_words 1000,optimizer='rmsprop'
import numpy
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Convolution1D
from keras.layers.convolutional import MaxPooling1D
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# load the dataset but only keep the top n words, zero the rest
top_words = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(nb_words=top_words)
# pad dataset to a maximum review length in words
max_words = 1000
X_train = sequence.pad_sequences(X_train, maxlen=max_words)
X_test = sequence.pad_sequences(X_test, maxlen=max_words)
# create the model
model = Sequential()
model.add(Embedding(top_words, 32, input_length=max_words))
model.add(Convolution1D(nb_filter=32, filter_length=3, border_mode='same',
activation='relu'))
model.add(MaxPooling1D(pool_length=2))
model.add(Flatten())
model.add(Dense(250, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
print(model.summary())
# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), nb_epoch=5, batch_size=128,
verbose=1)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
embedding_7 (Embedding)          (None, 1000, 32)      160000      embedding_input_7[0][0]
____________________________________________________________________________________________________
convolution1d_4 (Convolution1D)  (None, 1000, 32)      3104        embedding_7[0][0]
____________________________________________________________________________________________________
maxpooling1d_4 (MaxPooling1D)    (None, 500, 32)       0           convolution1d_4[0][0]
____________________________________________________________________________________________________
flatten_7 (Flatten)              (None, 16000)         0           maxpooling1d_4[0][0]
____________________________________________________________________________________________________
dense_13 (Dense)                 (None, 250)           4000250     flatten_7[0][0]
____________________________________________________________________________________________________
dense_14 (Dense)                 (None, 1)             251         dense_13[0][0]
====================================================================================================
Total params: 4163605
____________________________________________________________________________________________________
None
Train on 25000 samples, validate on 25000 samples
Epoch 1/5
25000/25000 [==============================] - 194s - loss: 0.4642 - acc: 0.7551 - val_loss: 0.4727
- val_acc: 0.7914
Epoch 2/5
25000/25000 [==============================] - 197s - loss: 0.2475 - acc: 0.9001 - val_loss: 0.2583
- val_acc: 0.8916
Epoch 3/5
25000/25000 [==============================] - 186s - loss: 0.1992 - acc: 0.9233 - val_loss: 0.3620
- val_acc: 0.8576
Epoch 4/5
25000/25000 [==============================] - 193s - loss: 0.1725 - acc: 0.9344 - val_loss: 0.2779
- val_acc: 0.8884
Epoch 5/5
25000/25000 [==============================] - 221s - loss: 0.1475 - acc: 0.9448 - val_loss: 0.3049
- val_acc: 0.8840
Accuracy: 88.40%
```

In [ ]: