CNN

February 2, 2018

1 Convolutional Neural Network

1.1 Import Dependencies

In [1]:
```
%matplotlib inline
from imp import reload

import itertools
import numpy as np
import utils; reload(utils)

from utils import *
from __future__ import print_function
from sklearn.metrics import confusion_matrix, classification_report, f1_score
import matplotlib.pyplot as plt
```

In [2]:
```
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.layers import Embedding, SpatialDropout1D
from keras.layers import LSTM
from keras.layers import Conv1D, GlobalMaxPooling1D
from keras.layers import Flatten
from keras.datasets import imdb
from keras.utils import plot_model
from keras.utils.vis_utils import model_to_dot

from IPython.display import SVG
from IPython.display import Image
```

```
/home/konstantin/.local/lib/python3.5/site-packages/h5py/__init__.py:36: FutureWarning: Conversi
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

1.2 Configure Parameters

In [3]:
```
# Embedding
embedding_size = 50
```

```
max_features = 5000
maxlen = 400

# Convolution
kernel_size = 3
pool_size = 4
filters = 250

# Dense
hidden_dims = 250

# Training
batch_size = 64
epochs = 4
```

1.3 Data Preparation

```
In [4]: (x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
```

```
In [5]: # Pad sequences
        x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
        x_test = sequence.pad_sequences(x_test, maxlen=maxlen)

        print('Train data size:', x_train.shape)
        print('Test data size:', x_test.shape)
```

```
Train data size: (25000, 400)
Test data size: (25000, 400)
```

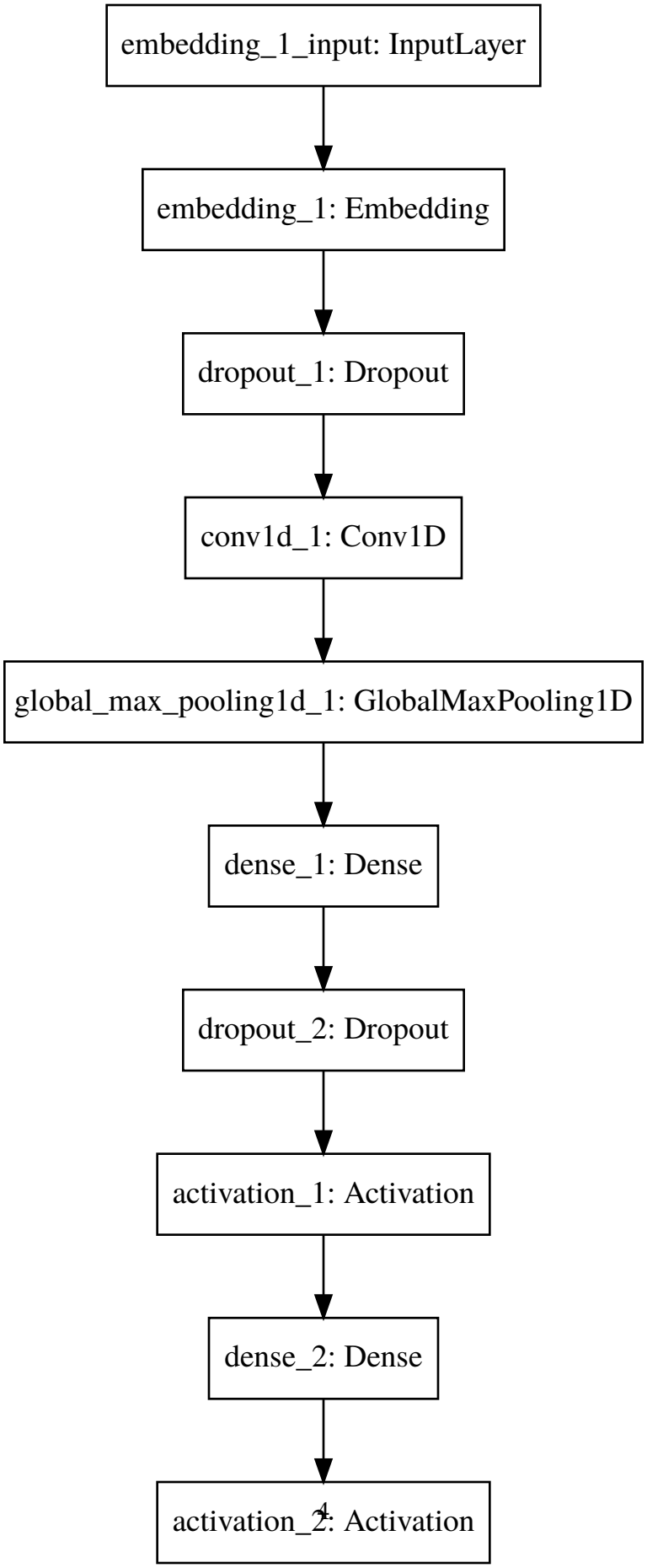
1.4 Modelling

```
In [6]: model = Sequential()

        # we start off with an efficient embedding layer which maps
        # our vocab indices into embedding_dims dimensions
        model.add(Embedding(max_features,
                            embedding_size,
                            input_length=maxlen))
        model.add(Dropout(0.2))


        model.add(Conv1D(filters,
                         kernel_size,
                         padding='valid',
                         activation='relu',
                         strides=1))
        model.add(GlobalMaxPooling1D())
```

```
        # We add a vanilla hidden layer:
        model.add(Dense(hidden_dims))
        model.add(Dropout(0.2))
        model.add(Activation('relu'))

        # We project onto a single unit output layer, and squash it with a sigmoid:
        model.add(Dense(1))
        model.add(Activation('sigmoid'))

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

        model.summary()

        plot_model(model, to_file='model_cnn.png', show_shapes=True)
        Image(filename = 'model_cnn.png')
        SVG(model_to_dot(model).create(prog='dot', format='svg'))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 400, 50)           250000
_________________________________________________________________
dropout_1 (Dropout)          (None, 400, 50)           0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 398, 250)          37750
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 250)               0
_________________________________________________________________
dense_1 (Dense)              (None, 250)               62750
_________________________________________________________________
dropout_2 (Dropout)          (None, 250)               0
_________________________________________________________________
activation_1 (Activation)    (None, 250)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 251
_________________________________________________________________
activation_2 (Activation)    (None, 1)                 0
=================================================================
Total params: 350,751
Trainable params: 350,751
Non-trainable params: 0
_________________________________________________________________
```

Out[6]:


embedding_1_input: InputLayer
embedding_1: Embedding
dropout_1: Dropout
conv1d_1: Conv1D
global_max_pooling1d_1: GlobalMaxPooling1D
dense_1: Dense
dropout_2: Dropout
activation_1: Activation
dense_2: Dense
activation_2: Activation

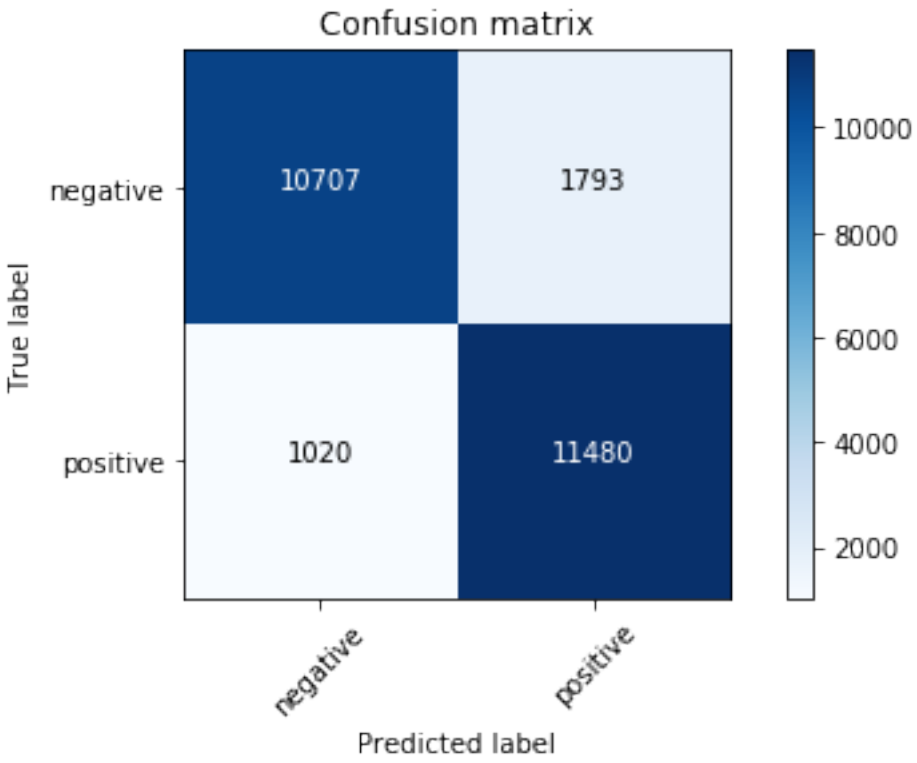
1.5 Evaluation

```
In [7]: # Train the model
        model.fit(x_train, y_train,
                  batch_size=batch_size,
                  epochs=epochs,
                  validation_data=(x_test, y_test),
                  verbose=1)

Train on 25000 samples, validate on 25000 samples
Epoch 1/4
25000/25000 [==============================] - 6s 247us/step - loss: 0.4395 - acc: 0.7752 - val_
Epoch 2/4
25000/25000 [==============================] - 5s 209us/step - loss: 0.2401 - acc: 0.9019 - val_
Epoch 3/4
25000/25000 [==============================] - 5s 205us/step - loss: 0.1675 - acc: 0.9370 - val_
Epoch 4/4
25000/25000 [==============================] - 5s 206us/step - loss: 0.1167 - acc: 0.9590 - val_


Out[7]: <keras.callbacks.History at 0x7fcdb5141588>

In [8]: # Evaluate model
        score, acc = model.evaluate(x_test, y_test, batch_size=batch_size)
        preds = model.predict_classes(x_test, batch_size=batch_size)

25000/25000 [==============================] - 1s 41us/step


In [9]: # Save the model weights
        model_path = 'models/'
        model.save(model_path + 'cnn_model.h5')
        model.save_weights(model_path + 'cnn_weights.h5')

In [10]: def plot_confusion_matrix(cm, classes,
                                   normalize=False,
                                   title='Confusion matrix',
                                   cmap=plt.cm.Blues):
             """
             This function prints and plots the confusion matrix.
             Normalization can be applied by setting `normalize=True`.
             """
             if normalize:
                 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
                 print("Normalized confusion matrix")
             else:
```

```
            print('Confusion matrix, without normalization')

        print(cm)

        plt.imshow(cm, interpolation='nearest', cmap=cmap)
        plt.title(title)
        plt.colorbar()
        tick_marks = np.arange(len(classes))
        plt.xticks(tick_marks, classes, rotation=45)
        plt.yticks(tick_marks, classes)

        fmt = '.2f' if normalize else 'd'
        thresh = cm.max() / 2.
        for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
            plt.text(j, i, format(cm[i, j], fmt),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")

        plt.tight_layout()
        plt.ylabel('True label')
        plt.xlabel('Predicted label')
```

```
In [11]: # Confusion Matrix
         cm = confusion_matrix(y_test, preds)
         plot_confusion_matrix(cm, {'negative': 0, 'positive': 1})
```

```
Confusion matrix, without normalization
[[10707  1793]
 [ 1020 11480]]
```




```
In [12]: # F1 score
         f1_macro = f1_score(y_test, preds, average='macro')
         f1_micro = f1_score(y_test, preds, average='micro')

         print('Test accuracy:', acc)
         print('Test score (loss):', score)
         print('')
         print('F1 Score (Macro):', f1_macro)
         print('F1 Score (Micro):', f1_micro)

Test accuracy: 0.887479999961853
Test score (loss): 0.30362622849464416

F1 Score (Macro): 0.8873723227145909
F1 Score (Micro): 0.88748
```