Project: Sentiment Analysis Using Convnets

Implementing a 1D convnet

We would use a 1D convnet via the Conv1D layer, which has a very similar interface to Conv2D. It takes as input 3D tensors with shape and also returns similarly-shaped 3D tensors. The convolution window is a 1D window on the temporal axis, axis 1 in the input tensor.

In [1]:
```
import keras
keras.__version__
```

```
/anaconda/envs/py35/lib/python3.5/site-packages/h5py/__init__.py:36: FutureWa
rning: Conversion of the second argument of issubdtype from `float` to `np.fl
oating` is deprecated. In future, it will be treated as `np.float64 == np.dty
pe(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

Out[1]:
```
'2.2.4'
```

Processing the labels of the raw movie data

In [2]:
```
import os

movies_dir = '/data/home/dlvmadmin/notebooks/Deep_learning_python/data/aclImd
b/'
movies_train_dir = os.path.join(movies_dir, 'train')

movie_labels = []
movie_texts = []

for label_type in ['neg', 'pos']:
    movie_dir_name = os.path.join(movies_train_dir, label_type)
    for fname in os.listdir(movie_dir_name):
        if fname[-4:] == '.txt':
            movie_file = open(os.path.join(movie_dir_name, fname))
            movie_texts.append(movie_file.read())
            movie_file.close()
            if label_type == 'neg':
                movie_labels.append(0)
            else:
                movie_labels.append(1)
```

Tokenizing the text of the raw movie data

In [3]:
```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing import sequence
import numpy as np

# We'll cut reviews after 200 words
maxlen = 200

# We'll be training on 12500 samples
movie_training_samples = 12500

# We'll be validating on 15000 samples
movie_validation_samples = 15000

# We'll only consider the top 15000 words in the dataset
max_words = 15000

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(movie_texts)
sequences = tokenizer.texts_to_sequences(movie_texts)

word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))

movie_data = pad_sequences(sequences, maxlen=maxlen)

movie_labels = np.asarray(movie_labels)
print('Tensor shape for data:', movie_data.shape)
print('Tensor shape for label:', movie_labels.shape)

# Split the data into a training set and a validation set
# But first, shuffle the data, since we started from data
# where sample are ordered (all negative first, then all positive).
indices = np.arange(movie_data.shape[0])
np.random.shuffle(indices)
movie_data = movie_data[indices]
movie_labels = movie_labels[indices]

movie_data_train = movie_data[:movie_training_samples]

movie_labels_train = movie_labels[:movie_training_samples]

movie_data_val = movie_data[movie_training_samples: movie_training_samples +
                            movie_validation_samples]
movie_labels_val = movie_labels[movie_training_samples: movie_training_samples
+
                                movie_validation_samples]

print(len(movie_data_train), 'train sequences')
print(len(movie_data_val), 'test sequences')

print('Pad sequences (samples x time)')
movie_data_train = sequence.pad_sequences(movie_data_train, maxlen=maxlen)
movie_data_val = sequence.pad_sequences(movie_data_val, maxlen=maxlen)
print('movie_data_train shape:', movie_data_train.shape)
print('movie_data_val shape:', movie_data_val.shape)
```

```
Found 88582 unique tokens.
Tensor shape for data: (25000, 200)
Tensor shape for label: (25000,)
12500 train sequences
12500 test sequences
Pad sequences (samples x time)
movie_data_train shape: (12500, 200)
movie_data_val shape: (12500, 200)
```

1D convnets are structured in the same way as the 2D convnets. They consist of a stack of Conv1D and MaxPooling1D layers, which ending in either a global pooling layer or a Flatten layer, turning the 3D outputs into 2D outputs, allowing to add more Dense layers to the model.

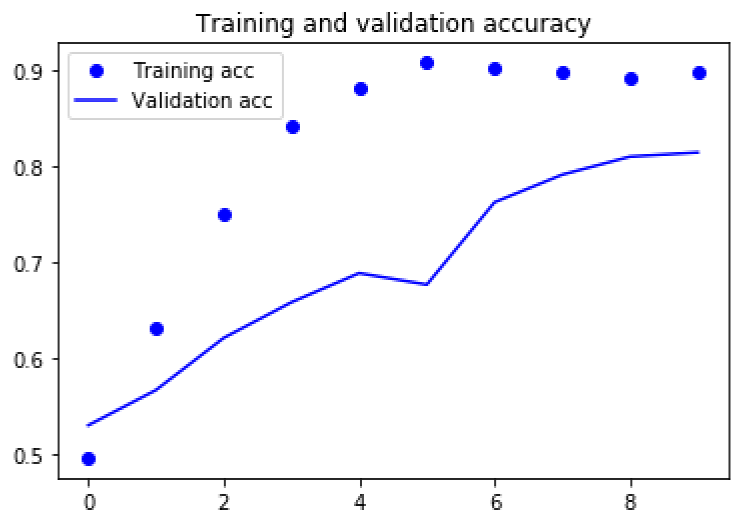
Define a model

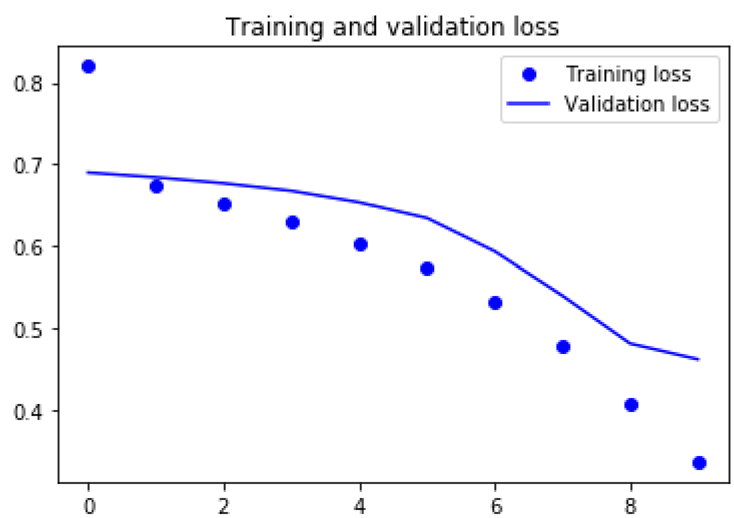
In [4]:
```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

conv1D_model = Sequential()
conv1D_model.add(layers.Embedding(max_words, 128, input_length=maxlen))
conv1D_model.add(layers.Conv1D(32, 7, activation='relu'))
conv1D_model.add(layers.MaxPooling1D(5))
conv1D_model.add(layers.Conv1D(32, 7, activation='relu'))
conv1D_model.add(layers.GlobalMaxPooling1D())
conv1D_model.add(layers.Dense(1))

conv1D_model.summary()

conv1D_model.compile(optimizer=RMSprop(lr=1e-4),
              loss='binary_crossentropy',
              metrics=['acc'])
history = conv1D_model.fit(movie_data_train, movie_labels_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
conv1D_model.save_weights('conv1D_model.h5')
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 200, 128)          1920000   
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 194, 32)           28704     
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 38, 32)            0         
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 32, 32)            7200      
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 32)                0         
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33        
=================================================================
Total params: 1,955,937
Trainable params: 1,955,937
Non-trainable params: 0
_________________________________________________________________
Train on 10000 samples, validate on 2500 samples
Epoch 1/10
10000/10000 [==============================] - 5s 521us/step - loss: 0.8203 -
acc: 0.4961 - val_loss: 0.6898 - val_acc: 0.5300
Epoch 2/10
10000/10000 [==============================] - 5s 461us/step - loss: 0.6736 -
acc: 0.6321 - val_loss: 0.6840 - val_acc: 0.5668
Epoch 3/10
10000/10000 [==============================] - 5s 458us/step - loss: 0.6512 -
acc: 0.7515 - val_loss: 0.6767 - val_acc: 0.6212
Epoch 4/10
10000/10000 [==============================] - 5s 463us/step - loss: 0.6288 -
acc: 0.8422 - val_loss: 0.6674 - val_acc: 0.6584
Epoch 5/10
10000/10000 [==============================] - 5s 461us/step - loss: 0.6040 -
acc: 0.8819 - val_loss: 0.6534 - val_acc: 0.6888
Epoch 6/10
10000/10000 [==============================] - 5s 460us/step - loss: 0.5737 -
acc: 0.9086 - val_loss: 0.6342 - val_acc: 0.6768
Epoch 7/10
10000/10000 [==============================] - 5s 462us/step - loss: 0.5331 -
acc: 0.9026 - val_loss: 0.5936 - val_acc: 0.7628
Epoch 8/10
10000/10000 [==============================] - 5s 464us/step - loss: 0.4770 -
acc: 0.8982 - val_loss: 0.5395 - val_acc: 0.7916
Epoch 9/10
10000/10000 [==============================] - 5s 462us/step - loss: 0.4058 -
acc: 0.8932 - val_loss: 0.4810 - val_acc: 0.8104
Epoch 10/10
10000/10000 [==============================] - 5s 460us/step - loss: 0.3362 -
acc: 0.8996 - val_loss: 0.4618 - val_acc: 0.8148
```

Here are our training and validation accuracy is somewhat lower than that of the LSTM. This is a convincing demonstration that a 1D convnet can offer a fast and less expensive alternative to a recurrent network on a word-level sentiment classification task.

In [5]:
```
%matplotlib inline

import matplotlib.pyplot as plt

accuracy = history.history['acc']
validation_accuracy = history.history['val_acc']
loss = history.history['loss']
validation_loss = history.history['val_loss']

epochs = range(len(accuracy))

plt.plot(epochs, accuracy, 'bo', label='Training acc')
plt.plot(epochs, validation_accuracy, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, validation_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





Because 1D convnets models process input patches independently, they are less sensitive to the order of the timesteps, unlike RNNs. Of course, in order to be able to recognize longer-term patterns, we could stack many convolution layers and pooling layers, resulting in upper layers that would "see" long chunks of the original inputs.

In [7]:
```
movie_test_dir = os.path.join(movies_dir, 'test')

movie_labels = []
movie_texts = []

for label_type in ['neg', 'pos']:
    movie_dir_name = os.path.join(movie_test_dir, label_type)
    for fname in sorted(os.listdir(movie_dir_name)):
        if fname[-4:] == '.txt':
            movie_file = open(os.path.join(movie_dir_name, fname))
            movie_texts.append(movie_file.read())
            movie_file.close()
            if label_type == 'neg':
                movie_labels.append(0)
            else:
                movie_labels.append(1)

sequences = tokenizer.texts_to_sequences(movie_texts)
movie_data_test = pad_sequences(sequences, maxlen=maxlen)
movie_labels_test = np.asarray(movie_labels)
```

In [8]:
```
conv1D_model.load_weights('conv1D_model.h5')
conv1D_model.evaluate(movie_data_test, movie_labels_test)
```

```
25000/25000 [==============================] - 3s 110us/step
```

Out[8]:
```
[0.4499091308307648, 0.81056]
```