An Introduction to Deep Learning With Python

[6.2] Using word embeddings

Prof. Yuzo Iano

pgs: 184 - 195

Instantiating an Embedding layer

In [1]:
```
from keras.layers import Embedding

embedding_layer = Embedding(1000, 64)
```

```
Using TensorFlow backend.
```

Loading the IMDB data for use with an Embedding layer

In [2]:
```
from keras.datasets import imdb
from keras import preprocessing

max_features = 10000
maxlen = 20

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)
```

Using an Embedding layer and classifier on the IMDB data

In [3]:
```
from keras.models import Sequential
from keras.layers import Flatten, Dense, Embedding

model = Sequential()
model.add(Embedding(10000, 8, input_length=maxlen))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
WARNING:tensorflow:From C:\Users\pablo\AppData\Roaming\Python\Python36\site-packages\tensorflow\py
thon\framework\op_def_library.py:263: colocate_with (from tensorflow.python.framework.ops) is depr
ecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 20, 8)             80000
_________________________________________________________________
flatten_1 (Flatten)          (None, 160)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 161
=================================================================
Total params: 80,161
Trainable params: 80,161
Non-trainable params: 0
_________________________________________________________________
```

In [4]:
```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])

history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
```

```
WARNING:tensorflow:From C:\Users\pablo\AppData\Roaming\Python\Python36\site-packages\tensorflow\py
thon\ops\math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will b
e removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 20000 samples, validate on 5000 samples
Epoch 1/10
20000/20000 [==============================] - 1s 42us/step - loss: 0.6759 - acc: 0.6043 - val_los
s: 0.6398 - val_acc: 0.6810
Epoch 2/10
20000/20000 [==============================] - 1s 35us/step - loss: 0.5657 - acc: 0.7428 - val_los
s: 0.5467 - val_acc: 0.7206
Epoch 3/10
20000/20000 [==============================] - 1s 30us/step - loss: 0.4752 - acc: 0.7808 - val_los
s: 0.5113 - val_acc: 0.7384
Epoch 4/10
20000/20000 [==============================] - 1s 32us/step - loss: 0.4263 - acc: 0.8079 - val_los
s: 0.5008 - val_acc: 0.7454
Epoch 5/10
20000/20000 [==============================] - 1s 32us/step - loss: 0.3930 - acc: 0.8257 - val_los
s: 0.4981 - val_acc: 0.7540
Epoch 6/10
20000/20000 [==============================] - 1s 32us/step - loss: 0.3668 - acc: 0.8395 - val_los
s: 0.5013 - val_acc: 0.7534
Epoch 7/10
20000/20000 [==============================] - 1s 34us/step - loss: 0.3435 - acc: 0.8534 - val_los
s: 0.5051 - val_acc: 0.7518
Epoch 8/10
20000/20000 [==============================] - 1s 33us/step - loss: 0.3223 - acc: 0.8658 - val_los
s: 0.5132 - val_acc: 0.7486
Epoch 9/10
20000/20000 [==============================] - 1s 31us/step - loss: 0.3022 - acc: 0.8765 - val_los
s: 0.5213 - val_acc: 0.7492
Epoch 10/10
20000/20000 [==============================] - 1s 31us/step - loss: 0.2839 - acc: 0.8860 - val_los
s: 0.5302 - val_acc: 0.7466
```

Download the raw IMDB dataset

From the following link http://ai.stanford.edu/~amaas/data/sentiment/ (http://ai.stanford.edu/~amaas/data/sentiment/) and uncompress it.

In [5]:
```
import os

imdb_dir = 'aclImdb/aclImdb'
train_dir = os.path.join(imdb_dir, 'train')

labels = []
texts = []

for label_type in ['neg', 'pos']:
    dir_name = os.path.join(train_dir, label_type)
    for fname in os.listdir(dir_name):
        if fname[-4:] == '.txt':
            f = open(os.path.join(dir_name, fname), encoding="utf-8")
            texts.append(f.read())
            f.close()
            if label_type == 'neg':
                labels.append(0)
            else:
                labels.append(1)
```

In [6]:
```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import numpy as np

maxlen = 100
training_samples = 200
validation_samples = 10000
max_words = 10000

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

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

data = pad_sequences(sequences, maxlen=maxlen)

labels = np.asarray(labels)
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)

indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]

x_train = data[:training_samples]
y_train = labels[:training_samples]
x_val = data[training_samples: training_samples + validation_samples]
y_val = labels[training_samples: training_samples + validation_samples]
```

```
Found 88582 unique tokens.
Shape of data tensor: (25000, 100)
Shape of label tensor: (25000,)
```

Download the Glove word embeddings

From the following link https://nlp.stanford.edu/projects/glove (https://nlp.stanford.edu/projects/glove), download the precomputed embeddings from 2014 English Wikipedia. It's an 822 MB zip file called glove.6B.zip and unzip it.

In [7]:
```
glove_dir = 'glove.6B'

embeddings_index = {}
f = open(os.path.join(glove_dir, 'glove.6B.100d.txt'), encoding="utf-8")

for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print('Found %s word vectors. ' % len(embeddings_index))
```

```
Found 400000 word vectors.
```

In [8]:
```
embeddings_dim = 100

embeddings_matrix = np.zeros((max_words, embeddings_dim))
for word, i in word_index.items():
    embeddings_vector = embeddings_index.get(word)
    if i < max_words:
        if embeddings_vector is not None:
            embeddings_matrix[i] = embeddings_vector
```

Model definition

In [9]:
```
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense

model = Sequential()
model.add(Embedding(max_words, embeddings_dim, input_length=maxlen))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 100, 100)          1000000
_________________________________________________________________
flatten_2 (Flatten)          (None, 10000)             0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                320032
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 33
=================================================================
Total params: 1,320,065
Trainable params: 1,320,065
Non-trainable params: 0
_________________________________________________________________
```

Loading pretrained word embeddings into the Embedding layer

In [10]:
```
model.layers[0].set_weights([embeddings_matrix])
model.layers[0].trainable = False
```

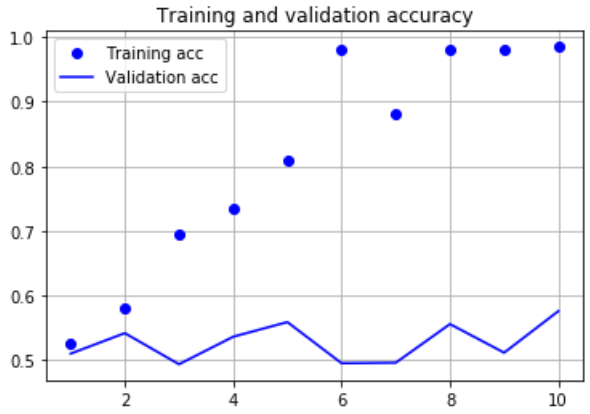
Training and evaluation

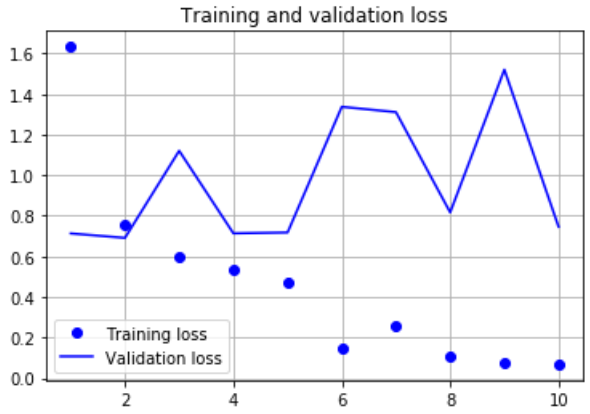
In [11]:
```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])

history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_data=(x_val, y_val))

model.save_weights('pre_trained_glove_model.h5')
```

```
Train on 200 samples, validate on 10000 samples
Epoch 1/10
200/200 [==============================] - 1s 3ms/step - loss: 1.6331 - acc: 0.5250 - val_loss: 0.
7130 - val_acc: 0.5100
Epoch 2/10
200/200 [==============================] - 0s 2ms/step - loss: 0.7565 - acc: 0.5800 - val_loss: 0.
6910 - val_acc: 0.5418
Epoch 3/10
200/200 [==============================] - 0s 2ms/step - loss: 0.5956 - acc: 0.6950 - val_loss: 1.
1205 - val_acc: 0.4936
Epoch 4/10
200/200 [==============================] - 0s 2ms/step - loss: 0.5335 - acc: 0.7350 - val_loss: 0.
7134 - val_acc: 0.5362
Epoch 5/10
200/200 [==============================] - 0s 2ms/step - loss: 0.4713 - acc: 0.8100 - val_loss: 0.
7177 - val_acc: 0.5589
Epoch 6/10
200/200 [==============================] - 0s 2ms/step - loss: 0.1448 - acc: 0.9800 - val_loss: 1.
3373 - val_acc: 0.4952
Epoch 7/10
200/200 [==============================] - 0s 2ms/step - loss: 0.2545 - acc: 0.8800 - val_loss: 1.
3110 - val_acc: 0.4960
Epoch 8/10
200/200 [==============================] - 0s 2ms/step - loss: 0.1102 - acc: 0.9800 - val_loss: 0.
8168 - val_acc: 0.5558
Epoch 9/10
200/200 [==============================] - 0s 2ms/step - loss: 0.0760 - acc: 0.9800 - val_loss: 1.
5204 - val_acc: 0.5115
Epoch 10/10
200/200 [==============================] - 0s 2ms/step - loss: 0.0680 - acc: 0.9850 - val_loss: 0.
7458 - val_acc: 0.5759
```

Ploting the results

In [13]:
```
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

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

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.grid()
plt.show()
```





Training the same model without pretrained word embeddings

In [14]:
```
model = Sequential()
model.add(Embedding(max_words, embeddings_dim, input_length=maxlen))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

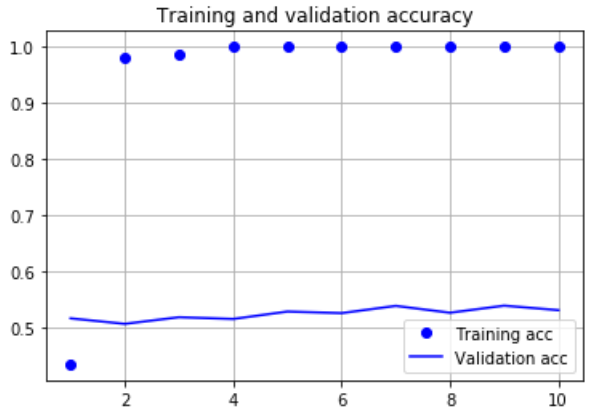
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 100, 100)          1000000
_________________________________________________________________
flatten_3 (Flatten)          (None, 10000)             0
_________________________________________________________________
dense_4 (Dense)              (None, 32)                320032
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 33
=================================================================
Total params: 1,320,065
Trainable params: 1,320,065
Non-trainable params: 0
_________________________________________________________________
```

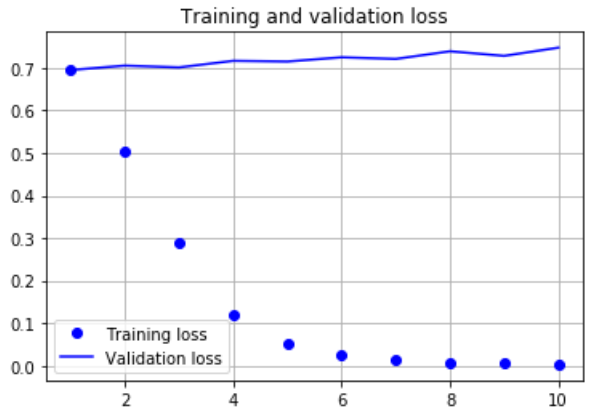
In [15]:
```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])

history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_data=(x_val, y_val))
```

```
Train on 200 samples, validate on 10000 samples
Epoch 1/10
200/200 [==============================] - 1s 5ms/step - loss: 0.6951 - acc: 0.4350 - val_loss: 0.
6950 - val_acc: 0.5167
Epoch 2/10
200/200 [==============================] - 1s 3ms/step - loss: 0.5028 - acc: 0.9800 - val_loss: 0.
7054 - val_acc: 0.5069
Epoch 3/10
200/200 [==============================] - 1s 3ms/step - loss: 0.2898 - acc: 0.9850 - val_loss: 0.
7012 - val_acc: 0.5187
Epoch 4/10
200/200 [==============================] - 1s 3ms/step - loss: 0.1183 - acc: 1.0000 - val_loss: 0.
7166 - val_acc: 0.5156
Epoch 5/10
200/200 [==============================] - 1s 3ms/step - loss: 0.0524 - acc: 1.0000 - val_loss: 0.
7150 - val_acc: 0.5288
Epoch 6/10
200/200 [==============================] - 1s 3ms/step - loss: 0.0261 - acc: 1.0000 - val_loss: 0.
7249 - val_acc: 0.5260
Epoch 7/10
200/200 [==============================] - 1s 3ms/step - loss: 0.0141 - acc: 1.0000 - val_loss: 0.
7211 - val_acc: 0.5389
Epoch 8/10
200/200 [==============================] - 1s 3ms/step - loss: 0.0082 - acc: 1.0000 - val_loss: 0.
7390 - val_acc: 0.5267
Epoch 9/10
200/200 [==============================] - 1s 3ms/step - loss: 0.0049 - acc: 1.0000 - val_loss: 0.
7283 - val_acc: 0.5393
Epoch 10/10
200/200 [==============================] - 1s 3ms/step - loss: 0.0030 - acc: 1.0000 - val_loss: 0.
7476 - val_acc: 0.5313
```

Ploting the results

In [16]:
```
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

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

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.grid()
plt.show()
```





Tokenizing the data of the test set

In [17]:
```
test_dir = os.path.join(imdb_dir, 'test')

labels = []
texts = []

for label_type in ['neg', 'pos']:
    dir_name = os.path.join(test_dir, label_type)
    for fname in sorted(os.listdir(dir_name)):
        if fname[-4:] == '.txt':
            f = open(os.path.join(dir_name, fname), encoding="utf-8")
            texts.append(f.read())
            f.close()
            if label_type == 'neg':
                labels.append(0)
            else:
                labels.append(1)

sequences = tokenizer.texts_to_sequences(texts)
x_test = pad_sequences(sequences, maxlen=maxlen)
y_test = np.asarray(labels)
```

Evaluating the model on the test set

In [18]:
```
model.load_weights('pre_trained_glove_model.h5')
model.evaluate(x_test, y_test)
```

```
25000/25000 [==============================] - 1s 44us/step
```

Out[18]:
```
[0.7448734223175049, 0.57604]
```

Pablo Minango

- pablodavid218@gmail.com