Course3_Assign1_Building keras models

VERSION AUTHOR	LAST UPDATED	LANGUAGE
Henry Udeogu	20 Sep 2018, 8:59 AM	Python 3.5 with Spark 2.1

Keras exercise

In this exercise you will be creating a Keras model by loading a data set, preprocessing input data, building a Sequential Keras model and compiling the model with a training configuration. Afterwards, you train your model on the training data and evaluate it on the test set. To finish this exercise, you will then save your model in HDF5 format.

Data

For this exercise we will use the Reuters newswire dataset. This dataset consists of 11,228 newswires from the Reuters news agency. Each wire is encoded as a sequence of word indexes, just as in the IMDB data we encountered in lecture 5 of this series. Moreover, each wire is categorised into one of 46 topics, which will serve as our label. This dataset is available through the Keras API.

Goal

We want to create a Multi-layer perceptron (MLP) using Keras which we can train to classify news items into the specified 46 topics.

Instructions

We start by importing everything we need for this exercise:

In [4]:
```
import pip

try:
    __import__('keras')
except ImportError:
    pip.main(['install', 'keras'])

try:
    __import__('h5py')
except ImportError:
    pip.main(['install', 'h5py'])

import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.utils import to_categorical

seed = 1337
np.random.seed(seed)
```

IMPORTANT! => In case h5py has been installed, please restart the kernel by clicking on "Kernel"->"Restart and Clear Outout" and wait until all output disapears. Then your changes are beeing picked up

As you can see, we use Keras' Sequential model with only two types of layers: Dense and Dropout. We also specify a random seed to make our results reproducible. Next, we load the Reuters data set:

In [5]:
```
from keras.datasets import reuters

max_words = 1000
```

```
(x_train, y_train), (x_test, y_test) = reuters.load_data(num_words=max_
words,
                                                         test_split=0.2
,
                                                         seed=seed)
num_classes = np.max(y_train) + 1  # 46 topics
```

Note that we cap the maximum number of words in a news item to 1000 by specifying the *num_words* key word. Also, 20% of the data will be test data and we ensure reproducibility by setting our random seed.

Our training features are still simply sequences of indexes and we need to further preprocess them, so that we can plug them into a *Dense* layer. For this we use a *Tokenizer* from Keras' text preprocessing module. This tokenizer will take an index sequence and map it to a vector of length *max_words=1000*. Each of the 1000 vector positions corresponds to one of the words in our newswire corpus. The output of the tokenizer has a 1 at the i-th position of the vector, if the word corresponding to i is in the description of the newswire, and 0 otherwise. Even if this word appears multiple times, we still just put a 1 into our vector, i.e. our tokenizer is binary. We use this tokenizer to transform both train and test features:

In [6]:
```
from keras.preprocessing.text import Tokenizer

tokenizer = Tokenizer(num_words=max_words)
x_train = tokenizer.sequences_to_matrix(x_train, mode='binary')
x_test = tokenizer.sequences_to_matrix(x_test, mode='binary')
```

1. Exercise part: label encoding

Use to_categorical, as we did in the lectures, to transform both *y_train* and *y_test* into one-hot encoded vectors of length *num_classes*:

In [7]:
```
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)
```

2. Exercise part: model definition

Next, initialise a Keras *Sequential* model and add three layers to it:

```
Layer: Add a *Dense* layer with in input_shape=(max_words,), 512 output unit
s and "relu" activation.
Layer: Add a *Dropout* layer with dropout rate of 50%.
Layer: Add a *Dense* layer with num_classes output units and "softmax" activ
ation.
```

In [8]:
```
model = Sequential()  # Instantiate sequential model
model.add(Dense(512, activation = 'relu', input_shape = (max_words,)))
# Add first layer. Make sure to specify input shape
model.add(Dropout(0.5))
model.add(Dense(512, activation = 'relu')) # Add second layer
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation = 'softmax')) # Add third layer
```

```
#model.summary()
```

3. Exercise part: model compilation

As the next step, we need to compile our Keras model with a training configuration. Compile your model with "categorical_crossentropy" as loss function, "adam" as optimizer and specify "accuracy" as evaluation metric. NOTE: In case you get an error regarding h5py, just restart the kernel and start from scratch

In [9]:
```
model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', me
trics = ['accuracy'])
```

In [10]:
```
#some learners constantly reported 502 errors in Watson Studio.
#This is due to the limited resources in the free tier and the heavy re
source consumption of Keras.
#This is a workaround to limit resource consumption

from keras import backend as K

K.set_session(K.tf.Session(config=K.tf.ConfigProto(intra_op_parallelism
_threads=1, inter_op_parallelism_threads=1)))
```

4. Exercise part: model training and evaluation

Next, define the batch_size for training as 32 and train the model for 5 epochs on *x_train* and *y_train* by using the *fit* method of your model. Then calculate the score for your trained model by running *evaluate* on *x_test* and *y_test* with the same batch size as used in *fit*.

In [11]:
```
batch_size = 32
model.fit(x_train, y_train, batch_size = batch_size, epochs = 5, valida
tion_data = (x_test, y_test))

score = model.evaluate(x_test, y_test, verbose = 0)
```

```
Train on 8982 samples, validate on 2246 samples
Epoch 1/5
8982/8982 [==============================] - 5s - loss: 1.5364 - acc:
0.6438 - val_loss: 1.1113 - val_acc: 0.7422
Epoch 2/5
8982/8982 [==============================] - 5s - loss: 1.0260 - acc:
0.7593 - val_loss: 0.9273 - val_acc: 0.7841
Epoch 3/5
8982/8982 [==============================] - 5s - loss: 0.7815 - acc:
0.8083 - val_loss: 0.8552 - val_acc: 0.7943
Epoch 4/5
8982/8982 [==============================] - 5s - loss: 0.6229 - acc:
0.8426 - val_loss: 0.8786 - val_acc: 0.7947
Epoch 5/5
8982/8982 [==============================] - 5s - loss: 0.5371 - acc:
0.8582 - val_loss: 0.8733 - val_acc: 0.8077
```

If you have done everything as specified, in particular set the random seed as we did above, your test accuracy should be around 80%

In [12]:
```
score[1]
```

Out[12]:
```
0.80765805871807239
```

5. Exercise part: model serialisation

As the final step in this exercise, save your model as "model.h5" and upload this file for evaluation:

In [13]:
```
model.save("model.h5")  # upload this file to the grader in the next co
de block
```

To upload the exported model to the grader we first need to encode it based64, we are doing this using a shell command:

In [14]:
```
!base64 model.h5 > model.h5.base64
```

Then we have to install a little library in order to submit to coursera

In [15]:
```
!rm -f rklib.py
!wget https://raw.githubusercontent.com/romeokienzler/developerWorks/ma
ster/coursera/ai/rklib.py
```

```
--2018-09-20 01:51:30--  https://raw.githubusercontent.com/romeokienzle
r/developerWorks/master/coursera/ai/rklib.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.
```