Table of Contents

Keras Model Import into DeepLearning4J

Tom Hanlon

tom@skymind.io

Getting Help

- Join us on Gitter Chat
    - https://gitter.im/deeplearning4j/deeplearning4j
- Email Tom@skymind.io

Training Information

- https://skymind.ai/academy

LABS

- Questions:
    - tom@skymind.io

Lab Prep

Introduction

In this class you will do some work in Python and some work in Java.

For ease of downloading everything has been setup as a github repo for a java project. A virtual Machine is also available, ask your instructor

The Python code is stored in the resources folder in a folder named Python.

Setting up Keras

Version

Keras recently released version 2. DeepLearning4J support for Version 2 models is not yet complete. If you wish to use the model import features you must install keras version 1.

Dependencies

Keras uses the following dependencies:

- numpy, scipy
- yaml
- HDF5 and h5py (optional, required if you use model saving/loading functions)

When using the TensorFlow backend:

TensorFlow See installation instructions.

https://www.tensorflow.org/install/

When using the Theano backend:

Theano See installation instructions.
http://deeplearning.net/software/theano/install.html#install

To install Keras:

sudo pip install keras==1.2

Switching from TensorFlow to Theano

By default, Keras will use TensorFlow as its tensor manipulation library. Follow these instructions to configure the Keras backend.

1. Step One

```
git clone https://github.com/tomthetrainer/KerasWorkshop.git
```

1. Step two

Import the project into Intellij by following instructions at https://deeplearning4j.org/quickstart

1. Additional Resources

Two saved Keras Model for VGG-16 and the DL4J version are too large to put into the main github repo. I have included them on github by creating a release for the project.

Although there is no specific Lab to load those models, there are examples in the Solution project that show loading these large useful models for image recognition.

Get the saved models from the following url

https://github.com/tomthetrainer/KerasWorkshop/releases

vgg16.zip

vgg_combined_save.h5

The Labs expect those to be in the /tmp directory. Either place them there or modify the java code that refers to the file path.

1. Structure of the Lab Repo.

Navigate to src/main/java/ai.skymind.training and you will find 3 projects

- demos
- labs
- solutions

labs

You will work through the Lab detailed later in this document in the labs project. A stub file is prepared for you and the instructions will walk you through completing the code.

demos

Some demos are available in the demo project. These are complete code examples.

1. ImageDemo demonstrates simple image into INDArray using NativeImageLoder
2. IrisNoImport is a working example of classification of iris flower data
3. Simplest network is a very simple network that takes a single value input and trains towards a single value output. It also shows the Web based User Interface.
4. VGG16SparkJavaWebApp, takes network that was imported from Keras into DL4J and builds web app to run inference onuser selected image.

solutions

Working solutions to the labs and a few other included examples.

Building Keras Models

Note

Copying and pasting from a pdf can sometimes introduce non-printing characters, typically for indents or newlines. You should be aware of this if your code looks correct but fails. The solution is to delete the white space and replace the white space manually.

A Keras Model

In this Lab you will build a Keras Model to analyze the iris flower dataset for classification of Iris' based on the measurements of petals and sepals

The file you are going to us

- resources/Keras/iris.csv

The Data looks like this.

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
```

Step One

Action:

Create a text file called iris.py in your favorite text editor.

Add the following imports to the file

```
import numpy
import pandas
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import np_utils
from sklearn.cross_validation import cross_val_score, KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
```

Step 2

Background:

The intitial weights of the Neural Network are set Randomly. In order to have reproducable results you can set a Random Seed manually. This way multiple runs will be consistent

Action:

Add the following lines to your iris.py file

```
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
```

Step 3

Background:

To load the date we rely on pandas dataframe. The common pattern is to specify the Parameters as X and the Labels as Y. The data is in a CSV file. Use the padas.read_csv method to load the data.

The Print statements are there so you can visualize the data

Action:

Add the following lines to your iris.py file.

```
# load dataset
dataframe = pandas.read_csv("iris.csv", header=None)
dataset = dataframe.values
X = dataset[:,0:4].astype(float)
Y = dataset[:,4]
print(X)
print(Y)
```

Step 4: Convert text Labels to numeric

Background:

Neural Nets ingest multi dimensional arrays of numeric values, all input has to be converted to numeric.

Action:

Add the following code to your iris.py file.

```
#encode class values as integers
encoder = LabelEncoder()
encoder.fit(Y)
encoded_Y = encoder.transform(Y)
```

Step 5 convert integer class labels to one_hot, or dummy encoding.

Background:

The three classes are currently encoded 0,1,2. We need to encode them as.

```
1,0,0
0,1,0
0,0,1
```

Action: Add the following lines to your iris.py file

```
# convert integers to dummy variables (hot encoded)
dummy_y = np_utils.to_categorical(encoded_Y)
print(dummy_y)
```

Step 6 Create a model

Action

Add the following lines to your iris.py file

```
# define baseline model
#def baseline_model():
# create model
model = Sequential()
model.add(Dense(4, input_dim=4, activation='relu'))
model.add(Dense(3,activation='sigmoid'))
```

Step 7 Compile the model

Background:

Step 6 just defines the model to use the model it must be compiled.

Action:

Add the following lines to your iris.py file.

```
# Compile model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accurac
y'])
```

Step 8 Train the model and get prediction

Background:

The model is ready to train, Calling the fit method on the model class trains the model. When training you specify the NDarray for the data, the NDarray for the labels, the number of epochs(total passes through the data) and the batchsize(home many records between weight updates.

Action:

Add the following lines to your iris.py file

```
# Train model and get prediction
#model.fit
model.fit(X, dummy_y, nb_epoch=200, batch_size=5)
prediction = model.predict(numpy.array([[4.6,3.6,1.0,0.2]]));
print(prediction);
```

Step 9 Save the model

Background:

To move the model to production in Java you need to save some combination of the configuration the weights or both.

Action:

Add the following lines to your iris.py

```
# To save just the weights
model.save_weights('/tmp/iris_model_weights')

# To save the weights and the config
# Note this is what is used for this demo
model.save('/tmp/full_iris_model')

# To save the Json config to a file
json_string = model.to_json()
text_file = open("/tmp/iris_model_json", "w")
text_file.write(json_string)
text_file.close()
```

Keras Model Import Lab

Introduction

- Keras
  - Popular Python tool for Deep Learning
- DeepLearning4J
  - Production worthy Java based Deep Learning
- Keras Model Import
  - Best of both worlds

A Keras Model

- resources/Keras/iris.py

```
import numpy
import pandas
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import np_utils
from sklearn.cross_validation import cross_val_score, KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline

# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)

# load dataset
dataframe = pandas.read_csv("iris.csv", header=None)
dataset = dataframe.values
X = dataset[:,0:4].astype(float)
Y = dataset[:,4]
print(X)
print(Y)

#encode class values as integers
encoder = LabelEncoder()
encoder.fit(Y)
encoded_Y = encoder.transform(Y)


# convert integers to dummy variables (hot encoded)
dummy_y = np_utils.to_categorical(encoded_Y)
print(dummy_y)

# define baseline model
#def baseline_model():
# create model
model = Sequential()
model.add(Dense(4, input_dim=4, activation='relu'))
model.add(Dense(3,activation='sigmoid'))

# Compile model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
#    return model
#model.fit
model.fit(X, dummy_y, nb_epoch=200, batch_size=5)
prediction = model.predict(numpy.array([[4.6,3.6,1.0,0.2]]));
print(prediction);



# To casve just the weights
model.save_weights('/tmp/iris_model_weights')

# To save the weights and the config
# Note this is what is used for this demo
model.save('/tmp/full_iris_model')

# To save the Json config to a file
json_string = model.to_json()
text_file = open("/tmp/iris_model_json", "w")
text_file.write(json_string)
text_file.close()
```

Iris DataSet

- Flower data
    - resources/Keras/iris.csv

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
6.1,2.8,4.0,1.3,Iris-versicolor
6.3,2.5,4.9,1.5,Iris-versicolor
7.1,3.0,5.9,2.1,Iris-virginica
6.3,2.9,5.6,1.8,Iris-virginica
```

Saving a Keras Model

```
model.save('/tmp/full_iris_model')
``



-------------------
<div style="page-break-after: always;"></div>

# Lab Step 1

In this lab we have run the python Keras code and the saved model and weights are
stored in ``` resources/Keras/full_iris_model
```

Create a String for the filepath to the full_iris_model file.

We use DataVec's ClassPathResource here to get information on files in our Classpath by providing a name of the file. The String for the classpath could be provided directly, we use ClassPathResource so we can drop a file in the resources directory and have a portable demonstration environment.

```
String kerasModelfromKerasExport = new ClassPathResource("Keras/full_iris_model").
getFile().getPath();
```

Lab Step 2

Create a MultiLayerNetwork by using `KerasModelImport.importKerasSequentialModelAndWeights` and providing a String for the FilePath to the saved Keras Model.

```
 MultiLayerNetwork model = KerasModelImport.importKerasSequentialModelAndWeights(k
erasModelfromKerasExport);
```

The required import for this step is:

```
import org.deeplearning4j.nn.modelimport.keras.KerasModelImport;
```

Lab Step 3

Add some code to verify the model import was successfull. In this step you create the Input Array to test with.

The Keras model was trained and then tested with a single input.

```
prediction = model.predict(numpy.array([[4.6,3.6,1.0,0.2]]))
```

The output showed high probability for Class 1, less for class 2, and even less for class 3.

```
[[ 0.92084521  0.13397516  0.03294737]]
```

Add this code to the class.

```
INDArray myArray = Nd4j.zeros(1, 4); // one row 4 column array
        myArray.putScalar(0,0, 4.6); // first value sepal length
        myArray.putScalar(0,1, 3.6); // second value sepal width
        myArray.putScalar(0,2, 1.0); // third value petal length
        myArray.putScalar(0,3, 0.2); // fourth value petal width
```

Lab Step 4

Validate the model is working and results match the Keras model.

Add this to the class.

```
INDArray output = model.output(myArray);
        System.out.println("First Model Output");
        System.out.println(myArray);
        System.out.println(output);
```

Lab Step 5

Run your code and verify the output.

You should see on the console output the following values.

```
[4.60, 3.60, 1.00, 0.20]
[0.92, 0.13, 0.03]
```

The first line is our input INDArray The second line is our output.

Does it match the prediction of the model when run in Keras?

Congratulations you have imported a Keras Model into DeepLearning4J.