1 Fully Connected Networks

Keras is a powerful and easy-to-use Python library for developing and evaluating deep learning models. It wraps the efficient numerical computation libraries Theano and TensorFlow and allows you to define and train neural network models in a few short lines of code. In this tutorial you will discover how to create your first neural network model in Python using Keras. After completing this lesson you will know:

- How to load a dataset for use with Keras.
- How to define and compile a Multilayer Perceptron model in Keras.
- How to evaluate a Keras model on a validation dataset.

1.1 Load Libraries

First step is to load the required libraries and setting the random seed.

```
import numpy

import matplotlib.pyplot as plt

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import Dropout

from keras.utils import np_utils

seed = 7

numpy.random.seed(seed)

#
```

1.2 Load Dataset in Numpy Format

MNIST is a simple computer vision dataset. It consists of images of handwritten digits as shown in figure 1. It also includes labels for each image, telling us which digit it is. For example, the labels for the above images are 5, 0, 4, and 1.

The MNIST data is split into three parts: $55,000$ data points of training data (mnist.train), $10,000$ points of test data (mnist.test), and $5,000$ points of validation data (mnist.validation). Each image is 28 pixels by 28 pixels.

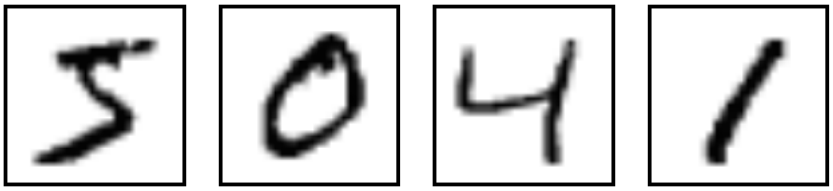

Figure 1: Sample images of MNIST dataset

```
X_train = numpy.load('./Data/mnist/x_train.npy')

X_test  =  numpy.load('./Data/mnist/x_test.npy')

y_train = numpy.load('./Data/mnist/y_train.npy')

y_test  = numpy.load('./Data/mnist/y_test.npy')

#
```

1.3 Formatting Data and Labels for Keras

We can flatten this array into a vector of $28 \times 28 = 784$ numbers. It doesn't matter how we flatten the array, as long as we're consistent between images. From this perspective, the MNIST images are just a bunch of points in a 784-dimensional vector space. The data should always be of the format (Number of data points, data point dimension). In this case the training data will be of format $55,000 \times 784$.

```
num_pixels = X_train.shape[1] * X_train.shape[2]

X_train = X_train.reshape(X_train.shape[0], num_pixels).astype('float32')

X_test = X_test.reshape(X_test.shape[0], num_pixels).astype('float32')

X_train = X_train / 255

X_test = X_test / 255

y_train = np_utils.to_categorical(y_train)

y_test = np_utils.to_categorical(y_test)
```

```
num_classes = y_test.shape[1]

#
```

1.4 Defining a single layer neural network model

Here we will define a single layer neural network. It will have a input layer of 784 neurons, i.e. the input dimension and output layer of 10 neurons, i.e. number of classes. The activation function used will be softmax activation.

```
# create model

model = Sequential()

model.add(Dense(num_classes, input_dim=num_pixels, activation='softmax'))

#
```

1.5 Compiling the model

Once the model is defined, we have to compile it. While compiling we provide the loss function to be used, the optimizer and any metric. Here we will use crossentropy loss with Adam optimizer and accuracy as a metric.

```
# Compile model

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

#
```

1.6 Training/Fitting the model

Now the model is ready to be trained. We will provide training data to the network. Also we will specify the validation data, over which the model will only be validated.

```
# Training model

model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10,
↪  batch_size=200, verbose=2)

#
```

1.7 Evaluating the model

Finally we will evaluate the model on the testing dataset.

```
# Final evaluation of the model

scores = model.evaluate(X_test, y_test, verbose=0)

print("Baseline Error: %.2f%%" % (100-scores[1]*100))

#
```

1.8 Defining a multi-layer model

Now we will define a multi layer neural network in which we will add 2 hidden layers having 500 and 100 neurons.

```
model = Sequential()

model.add(Dense(500, input_dim=num_pixels, activation='relu'))

model.add(Dense(100, activation='relu'))

model.add(Dense(num_classes, activation='softmax'))

#
```

1.9 Programming task

Now to put it all together to form a 3 layer neural network for 10 class digit classification.

```
import numpy
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.utils import np_utils
# fix random seed for reproducibility
```

```
#we always initialize the random number generator to a constant seed #value for
↪  reproducibility of results.
seed = 7
numpy.random.seed(seed)

# load data from the path specified by the user
X_train = numpy.load('./Data/mnist/x_train.npy')
X_test  =  numpy.load('./Data/mnist/x_test.npy')
y_train = numpy.load('./Data/mnist/y_train.npy')
y_test  = numpy.load('./Data/mnist/y_test.npy')

# flatten 28*28 images to a 784 vector for each image
num_pixels = X_train.shape[1] * X_train.shape[2]
# forcing the precision of the pixel values to be 32 bit
X_train = X_train.reshape(X_train.shape[0], num_pixels).astype('float32')
X_test = X_test.reshape(X_test.shape[0], num_pixels).astype('float32')
# normalize inputs from 0-255 to 0-1
X_train = X_train / 255
X_test = X_test / 255
# one hot encode outputs using np_utils.to_categorical inbuilt function
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
# define baseline model
#The model is a simple neural network with one hidden layer with the same number
↪  of neurons as there are inputs (784)
def baseline_model():
        # create model
        model = Sequential()
        model.add(Dense(500,kernel_initializer='normal', input_dim=num_pixels,
        ↪  activation='relu'))
        model.add(Dense(100,kernel_initializer='normal', activation='relu'))
        #A softmax activation function is used on the output
        #to turn the outputs into probability-like values and
        #allow one class of the 10 to be selected as the model's output
        ↪  #prediction.
        model.add(Dense(num_classes, kernel_initializer='normal',
        ↪  activation='softmax'))
        # Compile model
```

```
        model.compile(loss='categorical_crossentropy', optimizer='adam',
        ↪  metrics=['accuracy'])
        return model
# build the model
model = baseline_model()
# Fit the model
#The model is fit over 10 epochs with updates every 200 images. The test data is
↪  used as the validation dataset
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10,
↪  batch_size=200, verbose=2)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Baseline Error: %.2f%%" % (100-scores[1]*100))
```