(https://cognitiveclass.ai)

Build a Regression Model in Keras

Introduction

Import data from https://cocl.us/concrete_data (https://cocl.us/concrete_data)

In [ ]:

```
#!wget https://cocl.us/concrete_data
```

In [ ]:

```
pd.read_csv(r'https://cocl.us/concrete_data.csv')
```

Import Keras, Sklearn and Packages

Let's start by importing Keras and some of its modules.

In [1]:

```
import keras
import sklearn

from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
```

```
Using Theano backend.
WARNING (theano.configdefaults): g++ not available, if using conda: `conda
install m2w64-toolchain`
C:\Users\Pat\anaconda3\lib\site-packages\theano\configdefaults.py:560: Use
rWarning: DeprecationWarning: there is no c++ compiler.This is deprecated
and with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable
to execute optimized C-implementations (for both CPU and GPU) and will def
ault to Python implementations. Performance will be severely degraded. To
remove this warning, set Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for B
LAS functions.
```

Since we are dealing we images, let's also import the Matplotlib scripting layer in order to view the images.

In [2]:

```
import matplotlib.pyplot as plt
```

The Keras library conveniently includes the MNIST dataset as part of its API. You can check other datasets within the Keras library here (https://keras.io/datasets/).

So, let's load the MNIST dataset from the Keras library. The dataset is readily divided into a training set and a test set.

In [3]:

```
# import the data
from keras.datasets import mnist

# read the data
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 4s 0us/step
```

A. Build a baseline model (5 marks)

Use the Keras library to build a neural network with the following:

- One hidden layer of **10 nodes**, and a ReLU activation function
- Use the adam optimizer and the **mean squared error** as the loss function.

1. Randomly split the data into a training and test sets by holding 30% of the data for testing. You can use the train_test_splithelper function from Scikit-learn.
2. Train the model on the training data using 50 epochs.
3. Evaluate the model on the test data and compute the mean squared error between the predicted concrete strength and the actual concrete strength. You can use the mean_squared_error function from Scikit-learn.
4. Repeat steps 1 - 3, 50 times, i.e., create a list of 50 mean squared errors.
5. Report the mean and the standard deviation of the mean squared errors. Submit your Jupyter Notebook with your code and comments. Upload File

Build a Neural Network

- One hidden layer of **10 nodes**, and a ReLU activation function
- Use the adam optimizer and the **mean squared error** as the loss function.

In [9]:

```
# define classification model
def classification_model():
    # create model
    model = Sequential()
    model.add(Dense(num_pixels, activation='relu', input_shape=(num_pixels,)))
    model.add(Dense(10, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))


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

Split the Data

- Randomly split the data into a training and test sets by holding 30% of the data for testing. You can use the train_test_split helper function from Scikit-learn.

In [ ]:

```

```

Train the model

- Train the model on the training data using 50 epochs.

Evaluate the model

- Evaluate the model on the test data and compute the mean squared error between the predicted concrete strength and the actual concrete strength. You can use the mean_squared_error function from Scikit-learn.

In [ ]:

```
from sklearn.metrics import mean_squared_error
```

In [ ]:

```

```

Create a list of mean squared errors

- Repeat steps 1 - 3, 50 times, i.e., create a list of 50 mean squared errors.

Train and Test the Network

In [ ]:

```
```

Report statistical measures

- Report the mean and the standard deviation of the mean squared errors.

In [ ]:

```
```

In [ ]:

```
# build the model
model = classification_model()

# fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50, verbose=2)

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
```

Let's print the accuracy and the corresponding error.

In [ ]:

```
print('Accuracy: {}% \n Error: {}'.format(scores[1], 1 - scores[1]))
```

Just running 10 epochs could actually take over 2 minutes. But enjoy the results as they are getting generated.

Sometimes, you cannot afford to retrain your model everytime you want to use it, especially if you are limited on computational resources and training your model can take a long time. Therefore, with the Keras library, you can save your model after training. To do that, we use the save method.

In [ ]:

```
model.save('classification_model.h5')
```

Since our model contains multidimensional arrays of data, then models are usually saved as .h5 files.

When you are ready to use your model again, you use the load_model function from **keras.models**.

In [ ]:

```
from keras.models import load_model
```

In [ ]:

```
pretrained_model = load_model('classification_model.h5')
```

Other Assignment Parts

B. Normalize the data (5 marks) Repeat Part A but use a normalized version of the data. Recall that one way to normalize the data is by subtracting the mean from the individual predictors and dividing by the standard deviation. How does the mean of the mean squared errors compare to that from Step A?

In [ ]:

```

```

C. Increate the number of epochs (5 marks) Repeat Part B but use 100 epochs this time for training. How does the mean of the mean squared errors compare to that from Step B?

In [ ]:

```

```

D. Increase the number of hidden layers (5 marks) Repeat part B but use a neural network with the following instead:

- Three hidden layers, each of 10 nodes and ReLU activation function. How does the mean of the mean squared errors compare to that from Step B?

In [ ]:

```

```