Neural Network - Keras

July 20, 2018

This is a project to create a basic neural network in Keras, and then use this to generate predictions. The dataset consists of data to indicate whether or not one of the pima indians has diabetes. It is already cleaned data, so this is focused on understanding and creating a neural network, rather than data wrangling. The last column contains either a 1 to represent has diabetes or a 0. This binary output is what we want to predict using a neural network. Using Keras in Python is the simplest way to do this. We can add as many layers to our Sequential model as we like, but in the first layer we will specify the number of inputs to ensure that all 8 columns of data are being considered.

```
In [4]: import keras
        from keras.models import Sequential
        from keras.layers import Dense
        import numpy
        import sklearn
        from sklearn.model_selection import train_test_split
        # fix random seed for reproducibility
        numpy.random.seed(7)
```

```
Using TensorFlow backend.
```

```
In [5]: #load pima indians dataset as a numpy array
        dataset = numpy.loadtxt("pima-indians-diabetes.data.csv", delimiter=",")
```

Here is the dataset and the description of what is in it:
https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.names

```
In [6]: dataset.shape #look at the dataset to see its size and how many columns there are
```

```
Out[6]: (768, 9)
```

```
In [7]: #split into input (X) and output (Y) variables
        X = dataset[:,0:8] #columns 0-7 are X
        y = dataset[:,8] #column 8 is Y, looked at the dataset that was downloaded to see that
```

```
In [8]: #split the data into training and testing data
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

In [9]:
```
#create a sequential model in keras, that is, a model defined as a sequence of layers
model = Sequential()
#fully connected layers are defined using the Dense class
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu')) #rectifier/relu for the first two layers
model.add(Dense(1, activation='sigmoid')) #sigmoid for the output layer
```

In [10]:
```
#compile model using logarithmic loss (binary_crossentropy) and efficient gradient de
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [17]:
```
#fit the model, set the validation data as the testing data, set the number of iterat
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=150, batch_size=
```

In [12]:
```
#evaluate the model using the training data to see its accuracy
scores = model.evaluate(X_train, y_train)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
576/576 [==============================] - 0s 29us/step

acc: 75.69%
```

In [13]:
```
#evaluate the model using the testing data to see its accuracy
scores = model.evaluate(X_test, y_test)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
192/192 [==============================] - 0s 43us/step

acc: 75.00%
```

In [14]:
```
#use the model to make predictions
predictions = model.predict(X_train)
#round the predictions
rounded = [round(x[0]) for x in predictions]
```

In [16]:
```
#print out just the first 10 predictions
rounded[:10]
```

Out[16]:
```
[1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```