Automatic Classification of Handwritten Digits:

Neural Networks for Optical Character Recognition

Tristen Bristow

04/07/2017

Abstract

This study concerns the development of an automatic classification of handwritten digits by Multi-Layered Neural Network. The network used for this study has a single hidden layer. Overall performance is established empirically for a variable number of hidden layer nodes. The best network architecture is chosen by minimal network complexity a maximum classification performance.

Import Libraries

```
from sklearn.preprocessing import Imputer
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier
import matplotlib.pyplot as plt
import pandas as pd
```

Load Data

Each instance has 64 attributes, each of which can have value of 0 through 16. The last entry on the data-frame row is the class label, which is value of 0 through 9 that represents handwritten digits of numbers 0 though 9.

```
test_data = np.loadtxt('optdigits.tes', delimiter = ',')
train_data = np.loadtxt('optdigits.tra', delimiter = ',')

np.random.seed(1)

test_class = test_data[: , 64]
train_class = train_data[: , 64]

test_data = test_data[: , 0 : 64]
train_data = train_data[: , 0 : 64]
```

Pre-Processing Step

We consider each of the 64 attributes of the character-pattern, a feature vector x, where x_i is a component of x corresponding to a feature (where i is the index of component features, 0 through 64). For each we process the transformation and store the result in x_i'. This process is called Standardization, and provides input formatting for optimal neural network performance. Standardization is performed on all data before neural network test and training. Parameters necessary for standardization are calculated from the training-set, and are used in training and test set Standardization.

Standardization Formula:

$x_i' = (x_i - u_i) / s_i$

Where u_i is the feature mean value, s_i is the feature standard variation, and x_i is a input vector component.

```
u_i  =  train_data.mean(axis = 0)
s_i  =  train_data.std(axis = 0)

train_data = (train_data - u_i) / s_i
test_data = (test_data - u_i) / s_i

imp = Imputer(missing_values='NaN', strategy='mean', axis=0)

imp = imp.fit(train_data)
train_data = imp.transform(train_data)

imp = imp.fit(test_data)
test_data = imp.transform(test_data)
```

Training the Classifier and Running Tests

The classifier is a neural network with a single hidden layer of variable width. Out-of-sample classification accuracy is predicted against various widths of the hidden layer (See fig A).

```
acc = []
tot = len(test_class)
print("Width of Hidden Layer vs. Classification Accuracy(%)")

for width in range(1, 15):

        mlp = MLPClassifier(hidden_layer_sizes = (width))
```

```
        mlp.fit(train_data, train_class)
        predictions = mlp.predict(test_data)

        te = 0

        for i in range(tot):
                if (predictions[i] == test_class[i]):
                        te += 1

        ca = (te / tot) * 100
        acc.append(ca)
        print(width, ca)

best_fit = max(acc)
best_i = np.argmax(acc)
print
print("Optimum number of hidden nodes: ", best_i)
print("With classification accuracy:", best_fit)
print
plt.plot(acc)
plt.title("Number of Hidden Nodes Vs. Test Classification Accuracy")
plt.xlabel("Number of Neurons")
plt.ylabel("Recognition Accuracy")
plt.show()

tot = len(test_class)

print("Network performance on test set")
```

```
print(best_i, " nodes chosen:")
mlp = MLPClassifier(hidden_layer_sizes = (best_i))
mlp.fit(train_data,train_class)
predictions = mlp.predict(test_data)
te = 0
for i in range(tot):
    if (predictions[i] == test_class[i]):
        te += 1
ca = (te / tot)*100
print(ca, "% accuracy")
```

Output:
Width of Hidden Layer vs. Classification Accuracy(%)
1 28.223907925712794
2 65.75987444415381
3 84.77635364896679
4 92.80669631179703
5 94.48077426105152
6 97.59351294794664
7 98.40439445461679
8 98.66596913418782
9 98.95370128171594
10 99.31990583311536
11 99.81689772430029
12 99.76458278838608
13 99.81689772430029
14 99.89537012817159
Optimum Number of Hidden Nodes: 14
With classification accuracy: 99.89537012817159

Network performance on test set
14 nodes chosen:
95.15859766277129 % accuracy

Caption I: Fitting for a maximum of 14 hidden-layer nodes.

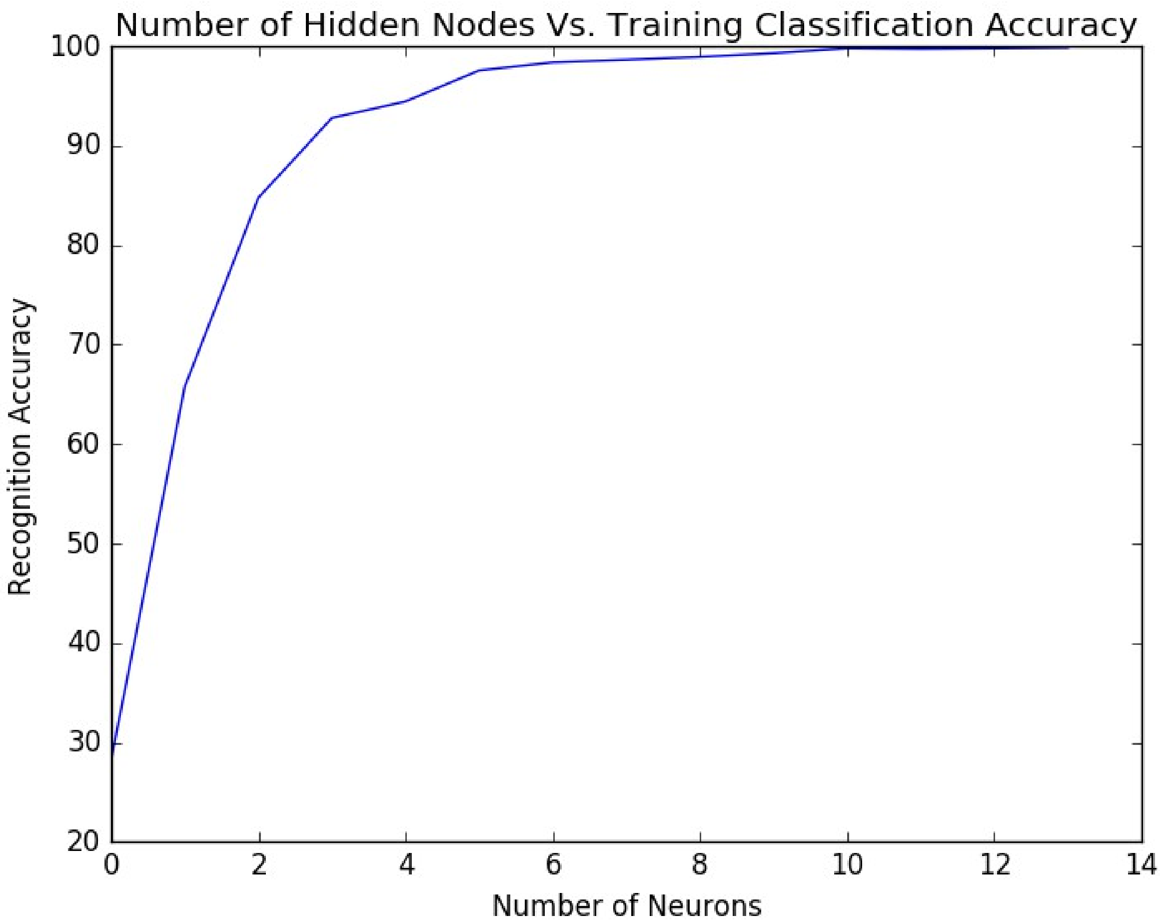

Fig A: Training accuracy curve over hidden-layer size

Output:
Width of Hidden Layer vs. Classification Accuracy(%)
1 28.223907925712794
2 65.75987444415381
3 84.77635364896679
4 92.80669631179703
5 94.48077426105152
6 97.59351294794664
7 98.40439445461679
Optimum Number of Hidden Nodes: 7
With classification accuracy: 98.40439445461679
Network performance on test set
7 nodes chosen:
93.65609348914859 % accuracy

Caption II: Fitting for a maximum of 7 hidden-layer nodes.

Results:

The neural network performance appears to not have issues with over-fitting, as it appears that continuously adding nodes to the hidden layer does not cause hindered performance on the test set (with 95% test accuracy recorded for a network with the maximum number of hidden nodes available (see caption I)). There is a point at which rapidly diminishing returns in training accuracy is observed in networks containing more than 4 hidden nodes. Were computational efficiency a direct concern, it would appear that a high degree of performance could be achieved with approximately one-half the nodes selected for the 14 hidden node model. It is the case that network architecture of 7 hidden nodes, for 64 input and output nodes is sufficient for almost-perfect training accuracy and a satisfactory test accuracy of 94% (see caption II).

Conclusion

This study indicates that state-of-the-art results can be obtained with optical character recognition of numeric digits using a multi-layered neural network model. While this learning approach cannot avoid adoption of a hidden layer (owing to linear inseparability of the classes), the strongest result indicates that a hidden layer of limited size, comparable to the input and output layer-width, is sufficient for the high-performance classification of high-dimensional data.