Optical Character Recognition: Handwritten Digits

Rémi Wong

March 2020

Contents

1 Overview

1.1 Introduction

In this coursework, a machine learning system has been devised to enable the categorizing of an Optical Handwritten digit dataset obtained from the UCI dataset resource library. Before implementation, the different requirements of the coursework have been investigated, leading to the methodology to be followed for the design, implementation, testing, and evaluation of the system. The methodology has taken in consideration the instructions for the successful completion of this coursework and the associated marking scheme.

1.2 Problem Analysis

The preprocessing of handwritten digits have already been carried in the obtained datasets. The conversion from digital image to pixel generated 8 x 8 matrix representations of the digits with each matrix element's integer value ranging from 0 to 16. The training set and test sets provided both have 64 input elements and an additional 65^{th} attribute which represents the expected class of the inputted. The two datasets each have 2810 digit representation samples which need to be utilized for the training testing of a suitable machine learning algorithm to solve for handwritten digits recognition problem. According to the requirements of the coursework, the algorithm needs to surpass the base line published on UCI which was 98.00% accuracy which using K-Nearest Neighbour algorithm with a value of k=1.

2 Multi-Layer Perceptron Neural Network

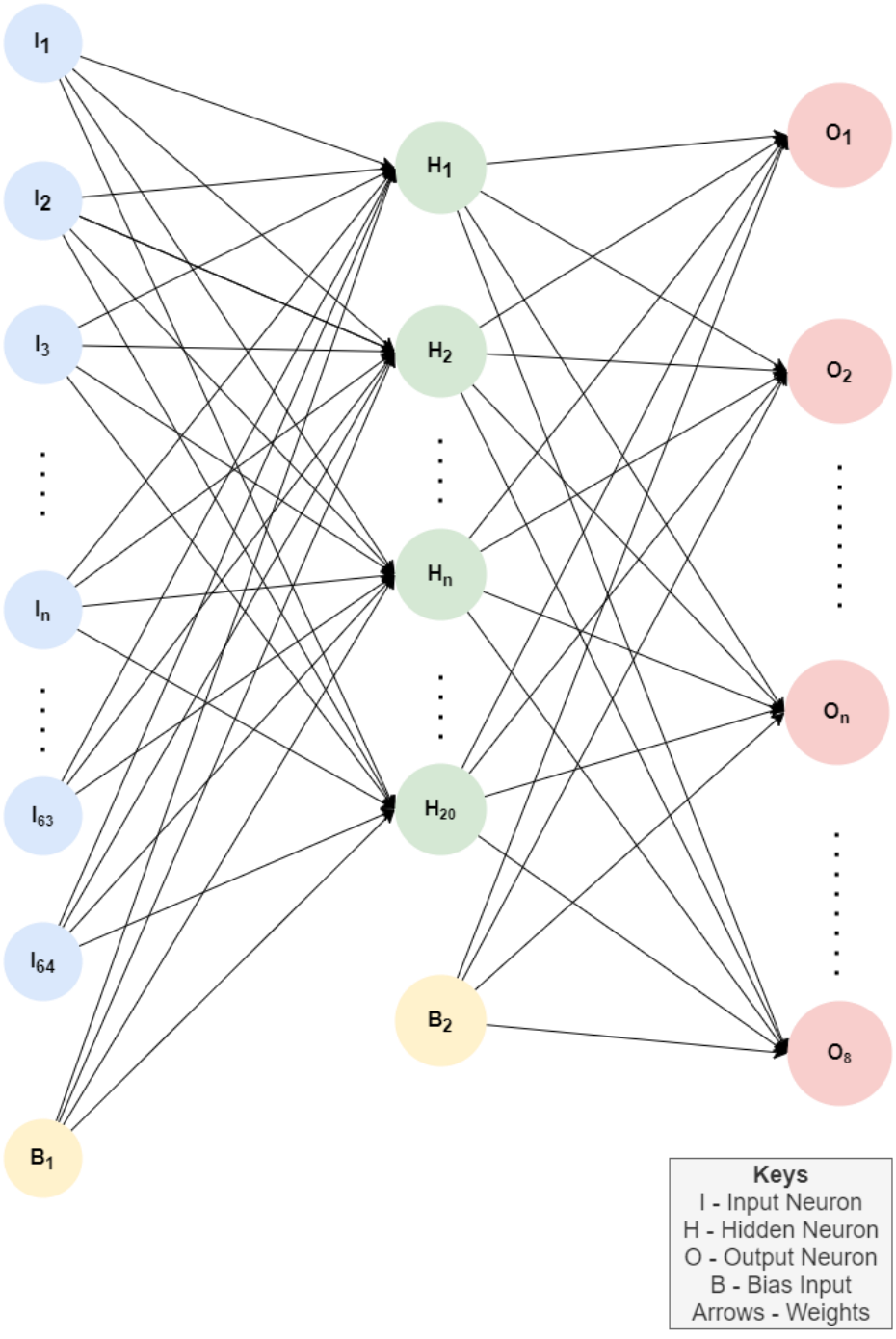
2.1 Concept

In order to solve the digit recognition problem, the Multi-Layer Perceptron (MLP) was chosen as the machine learning algorithm because of its perfect suitability in a supervised learning setting such as this scenario where there is a training dataset and a test dataset. Moreover it is an algorithm which can be used in multiple types of applications. Hence the machine learning system has relied on a neural network for its learning and classification process. The system relies on three main components which are:

- Network Initialization
- Forward Propagation
- Backward Propagation

The MLP network designed consists of three layers, namely; Input Layer (64 inputs), Hidden Layer (20 neurons), and Output Layer (8 output neurons).

A bias input of default value 1 has been used to stabilize both the input and hidden layers.



The network initialization deals with the creation of the nodes representing the neural network and the linking of the nodes from the input layer to the hidden layer, then from the latter to the output layer using activation weights. The weights are initially randomized to hold a value between 0 and 1 but through several trials, it was found that randomizing between 0 and 0.5 tend to produce more accurate results.

The forward propagation is used to generate the different outputs of the network using a set of input values and the weight values set. The input values have been scaled between 0 and 1 by dividing the different input integers by 16. Every neuron in the network is fired using the formula;

$$v_k = \sum_{j=1}^{m} w_{kj} x_j + b_k$$

Then the neuron is activated using the sigmoid activation function which prorates the fired neuron's output between 0 and 1.

$$f(x) = \frac{1}{1 + e^{-x}}$$

In the backward propagation phase, the different weights of the neural network are calibrated after determining the output error using the targeted activated output and the generated output obtained in forward propagation so that the correct outputs are generated. The simplified calculations were based on the work carried out by Matt Mazur (Mazur, 2015). The amount of change of weight represents the cost function of the error correction.

$$Output\ Error = obtained\ activated\ output - targeted\ output$$

The hidden neuron weights are calibrated using the formula;

$$\text{new weight value} = \text{previous weight value} - \left(\text{learning rate} * \frac{\partial(\text{total Error caused by weight})}{\text{d(previous weight value)}}\right)$$

where,

$$\frac{\text{d(total Error caused by weight)}}{\text{d(previous weight value)}} = (Output - Targeted\ Output) * Output(1 - Output) * Activated\ Hidden\ Neuron\ value$$

The input neuron weights are calibrated using;

$$w_n = w_n - n * \left(\frac{\partial E_{total}}{\partial w_n}\right)$$

$$\frac{\partial E_{total}}{\partial w_n} = \left(\sum_{o=0}^{n} \frac{\partial E_{total}}{\partial out_o} * \frac{\partial out_o}{\partial net_{hn}} * \frac{\partial net_{hn}}{\partial w_n}\right)$$

$$\frac{\partial E_{total}}{\partial w_n} = \left(\sum_{o=0}^{n} \delta_o * w_{ho}\right) * out_{hn}(1 - out_{hn}) * i_n$$

2.2 Running the Program

The codes for the multi-layer perceptron algorithm trial is found in the OCRremi folder and the different neural network parameters have to be preset before running the application. The epoch parameter has to be preset in the main class OCRremi and the number of neurons and learning rate parameters are adjusted in the class Network. The datasets provided in the coursework guidelines has already been incorporated in the java project folder so as to facilitate the experimentation process.

```
11
12    public static void main(String[] args) {
13        int epoch = 100;
14
```

```
1  class Network {
2      //set number of neurons
3      int numHiddenNeurons=32;
4      //set learning rate value
5      double learningRate=0.1;
6
```

2.3 Experimental Results

Twelve different combinations of neurons, learning rate and epoch values have been experimented to see how well they fared to detect the expected digits. The whole 2810 test and 2810 training sets have been used to find the accuracy of the multi-layered perceptron system devised.

Number of Neurons	Learning Rate	Number of Epoch	Digits Detected	Accuracy (%)
10	0.1	50	1661	59.11
10	0.3	50	1676	59.64
10	0.1	100	1685	59.96
10	0.3	100	1674	59.57
20	0.1	50	1716	61.07
20	0.3	50	1791	63.74
20	0.1	100	1779	63.31
20	0.3	100	1766	62.85
32	0.1	50	1714	61.00
32	0.3	50	1762	62.70
32	0.1	100	1837	65.37
32	0.3	100	1802	64.13

From the different trials, it was found that the combination below produced the best result out of the solution designed:

- Epochs: 100
- Number of Neurons: 32
- Learning Rate: 0.1

```
[WELCOME TO DIGIT RECOGNITION PROBLEM]
In Progres ....
Digit-0 training: 285 test: 173
Digit-1 training: 281 test: 187
Digit-2 training: 277 test: 159
Digit-3 training: 297 test: 191
Digit-4 training: 275 test: 198
Digit-5 training: 273 test: 212
Digit-6 training: 284 test: 215
Digit-7 training: 274 test: 196
Digit-8 training: 288 test: 120
Digit-9 training: 276 test: 186
Total Classified: 1837.0 Total Expected : 2810.0 Overall Accuracy: 65.37366548042705 %
```

2.4 MLP – Two Fold Test

The two fold test was carried out by adjusting the amount of training samples and test samples amount that are used in the application by varying the two variables trainingFold and testFold.

```
double trainingFold = 2810;
double testFold = 2810;
```

In the table below, three different combinations of two-fold of sample sets have been processed with the total number of samples being 2810 sets. It was found that the accuracy of the multi-layer perceptron increased upon the use of a higher variety of training digit samples over the amount of test samples as it generated an accuracy result of 71.06 %.

% Training Set	% Test Set	Digits Detected	% Accuracy
50	50	932	66.33
70	30	559	71.06
30	70	857	61.00

2.5 Verdict

Based on the results obtained from the Multi-layer Perceptron solution where a maximum accuracy of 65.37% was achieved. A few possibilities were considered as means to improve the accuracy but due to uncertain in the ability to surpass the UCI benchmark, such as increasing the number of hidden layers in the network, pre-filtering of the dataset before training and testing, and using a complex activation functions such as reLU and softmax. According to some researches, the algorithm can potentially achieve more than 98% accuracy if improved drastically but the complexity of the algorithm requires much more time and analysis than was available for completion of this coursework. Hence these improvements were not entertained due to limitations of time. Another algorithm had to be chosen in turn to solve this problem.

3 Weighted K-Nearest Neighbour

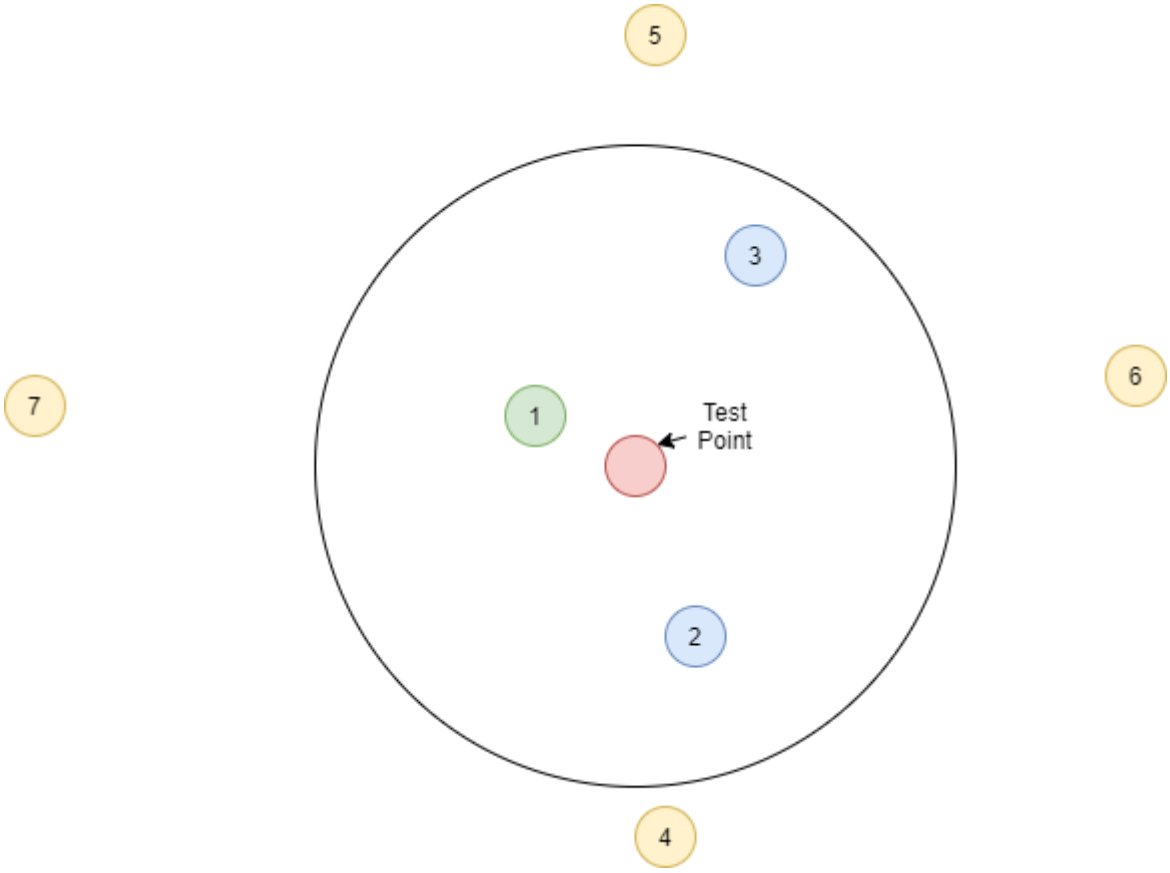
3.1 Concept

K-Nearest Neighbour was chosen as the appropriate algorithm to improve substantially the categorizing of the test dataset since it has historically been proven to produce very high detection of digits despite being much simpler than multi-layer perceptron machine learning system. In order to surpass the accuracy benchmark obtained from UCI Machine Learning database, which was of 98.00 %.

The algorithm considers the nearest points on a 2-Dimensional plane to the test data point through the use of distance metrics. The K parameter is used to determine the number of nearest neighbours which are considered for the classification of the test data. The output digit is selected out of the closest nearest neighbours based on the number of occurences of a possible outcome. The distance between the tests points and the training points have been calculated using Euclidean distance since it was found to be simple but yet very effective. The figure below shows a scenario where 7 nearest neighbours were considered.

$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$



Once the UCI benchmark was obtained, a simple means to improve the algorithm was utilized in the form of Weighted K-Nearest Neighbour. It was seen that the most accuract recognition occurred when k=1, which shows that the closest neighbour is the most decisive one in terms of producing the real class output. As show in the figure above, the orange points, being more frequent when k=7, were chosen as the classified result despite that they may not be the expected class output. The purpose of the weighted reasoning is to compare the most frequent possible class with the most nearest neighbor (first after sorting). If they do not match, the weight of the first neighbours are calculated against the each other to determine the ratio of distance between the test point and the training points by calculate the summation of the inverse of displacement between each possible digit output. The digit with the highest weight value is chosen as the final output of the algorithm, thus superseding the frequency output generated.

$$Weight, w_i = \sum \frac{1}{d_i}$$

3.2 Running the Program

The Weighted K-Nearest Neighbour application is found in the OCRremiKNN project folder. Same as the previous algorithm, the datasets have already been pre-loaded in the folder. The value of k and number of weights have to be preset before running the application.

```
public static void main(String args[]) {
    //preset number of neighbours and number of weights
    int k = 7;
    int numWeights = 3;
```

The sorting and occurrence selection parts of the K-NN code was inspired from Dr Noureddin Sadawi K-NN since it was found to be the most efficient selection means (Sadawi, 2015).

3.3 Experimental Results

The K-NN algorithm was tested with different parametric values of K and number of Weights to observe the classification accuracy of the algorithm. Two weight values of 3 and 5 were selected since they were found to work efficiently with the algorithm.

Number of nearest neighbours, K	Number of Weights, w	Digits Classified	W-KNN Accuracy (%)	UCI Accuracy (%)	Improvement (%)
1	3	2755	98.04	98.00	0.04
2	3	2756	98.08	97.38	0.70
3	3	2754	98.01	97.83	0.18
4	3	2755	98.04	97.61	0.43
5	3	2756	98.08	97.89	0.19
6	3	2756	98.08	97.77	0.31
7	3	2758	98.15	97.66	0.49

8	3	2758	98.15	97.66	0.49
9	3	2758	98.15	97.72	0.43
10	3	2757	98.11	97.55	0.56
11	3	2757	98.11	97.89	0.22

Number of nearest neighbours, K	Number of Weights, w	Digits Classified	W-KNN Accuracy (%)	UCI Accuracy (%)	Improvement (%)
1	5	2755	98.04	98.00	0.04
2	5	2758	98.15	97.38	0.77
3	5	2757	98.11	97.83	0.28
4	5	2756	98.08	97.61	0.47
5	5	2751	97.90	97.89	0.01
6	5	2752	97.94	97.77	0.17
7	5	2757	98.11	97.66	0.45
8	5	2757	98.11	97.66	0.45
9	5	2758	98.15	97.72	0.43
10	5	2755	98.04	97.55	0.49
11	5	2756	98.08	97.89	0.19

From the tables above, the best parameter values below generated the best results for the K-Nearest Neighbour algorithm with a correct detection of 2758 digits out of 2810 test sets which represents an accuracy of 98.15 %:

- Number of Nearest Neighbours: 7
- Number of Weights: 3

```
Digit-0 training: 285 test: 285
Digit-1 training: 281 test: 279
Digit-2 training: 277 test: 276
Digit-3 training: 297 test: 289
Digit-4 training: 275 test: 271
Digit-5 training: 273 test: 270
Digit-6 training: 284 test: 284
Digit-7 training: 274 test: 268
Digit-8 training: 288 test: 269
Digit-9 training: 276 test: 267


Detected: 2758.0
Accuracy: 98.14946619217082 %
```

Hence the algorithm has successfully been able to surpass the UCI benchmark top score of 98.00 %.

3.4 Two-Fold Testing

A told-fold test has again been carried out with the K-Nearest Neighbour using the same fold ratios as used for multi-layer perceptron.

% Training Set	% Test Set	Digits Detected	% Accuracy
50	50	1366	97.22
70	30	1945	98.10
30	70	808	95.88

From the test it was found that the accuracy of the algorithm depends again on the amount of training set. This is shown when a ratio of training to test of 70 : 30 is used which results in a further better accuracy of 98.10 % for 843 training sets and 1967 test sets. Hence increasing the number of training sets is a way to improve the accuracy of the K-NN algorithm

3.5 Verdict

From the second algorithm devised, it can be deduced that K-Nearest Neighbour is very effective to recognize handwritten digits despite relying on simple concepts of distance and occurrences. Moreover there are existing ways to improve its mechanism, thus increasing the accuracy. The use of weightage has also helped to increase slightly it's accuracy by looking at the contribution of the nearest neighbours but this depends on the first nearest neighbours and the immediate occurrences.

4 Conclusion

4.1 Achievements

In this coursework, two popular algorithms which are used in many classification applications have been devised to tackle the handwritten digits recognition. Unfortunately due to time and complexity limitations, the multi-layer perceptron algorithm built could not better the UCI scores. However the improved weighted K-Nearest Neighbour algorithm successful managed to slightly surpass the UCI benchmark by 0.15 %.

5 References

Mazur, M., 2015. *A Step by Step Backpropagation Example.* [Online]
Available at: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/
[Accessed 10 February 2020].

Sadawi, N., 2015. *KNN.* [Online]
Available at: https://github.com/nsadawi/KNN
[Accessed 03 March 2020].

6 Bibliography

Abdar, M., Yen, N.Y. and Hung, J.C.S., 2018. Improving the diagnosis of liver disease using multilayer perceptron neural network and boosted decision trees. *Journal of Medical and Biological Engineering*, *38*(6), pp.953-965.

Agostinelli, F., Hoffman, M., Sadowski, P. and Baldi, P., 2014. Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*.

Cordella, L.P., De Stefano, C., Tortorella, F. and Vento, M., 1995. A method for improving classification reliability of multilayer perceptrons. *IEEE Transactions on Neural Networks*, *6*(5), pp.1140-1147.

Syaliman, K.U., Nababan, E.B. and Sitompul, O.S., 2018, March. Improving the accuracy of k-nearest neighbor using local mean based and distance weight. In *Journal of Physics: Conference Series* (Vol. 978, No. 1, p. 012047). IOP Publishing.