SEMEION HANDWRITTEN DIGIT DATA SET APPLIED TO MULTILAYER PERCEPTRON NEURAL NETWORK

Student Number: 26016931

ABSTRACT

Due to the overwhelming benefits of storing written information digitally as opposed to physically (on paper), software that can accurately recognise handwritten characters is needed. The purpose of this study is to train a multi-layered perceptron (MLP) neural network to correctly identify handwritten input data as its corresponding value. Using a large dataset of handwritten digits as input and validation data, the MLP was trained to solve the proposed problem (for the given dataset of digits). This study definitively shows that MLPs can and should be incorporated when developing handwritten character recognition software.

Index terms: Neural Network, Multi-layered Perceptron, Machine Learning, Handwritten Character Recognition Software.

1. INTRODUCTION

The following paper will give background information on the specific MLP used, the dataset that is applied to the MLP, and an evaluation of the results achieved.

1.1. Background on artificial neural networks

Artificial neural networks (ANN) apply the basic functionality of human neurons to a network of machine learning algorithms. Each neuron in the network takes in an input, and will fire depending on whether the threshold for the given weight is exceeded. In an MLP, neurons are organised in layers, which allows for different kinds of transformations on the neurons inputs. Signals in a MLP travel from the input layer, to the hidden layers, to finally the output layer.

1.2. MLP application used

The MLP used in this study is a modified version of 'RJMs Perceptron Neural Network Program[1]', which is given on the University of Reading's blackboard. This program uses object oriented techniques in java to construct an ANN through instances of different neuron classes (Sigmoidal/ Linear). In order to construct a MLP, a class called MultiLayerNetwork inherits from the SigmoidalLayerNetwork class, including the next layer. In effect, this defines the multi-layer network.

The program takes in user input data through three files, usertrain.txt, uservalid.txt and userunseen.txt which represent the training, validation and unseen data respectively. The first four lines of each input file contains the parameters, titles and formatting of the following data. The program bases the initial weights randomly of the Random Seed variable (that can be adjusted).

The programs GUI allows for the Learn Rate, Momentum, (amount of) Hidden Neurons, Max Epochs and Random Seed variables to be modified. It also allows for two graphical representations of information in the form of a tadpole plot diagram and SSE plot diagram.

2. USER APPLICATION

2.1. Dataset requirements

In order for the MLP to be trained both accurately and in a reasonable amount of time, I will isolate the possible character set to only the real numbers between 0 and 9. This would theoretically allow my trained MLP to read any handwritten real number. The same concept and formatting could be applied to any/all characters allowing the MLP to read any hand written script, however, this would require an exponential amount of data.

2.2. Chosen dataset background/explanation

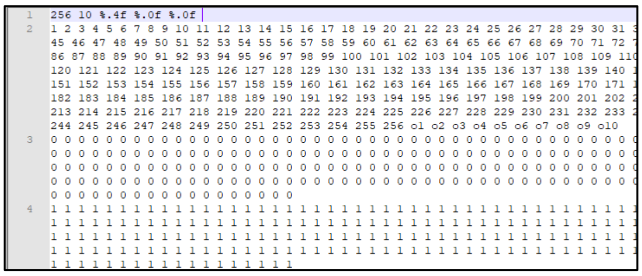
This study uses the 'Semeion Handwritten Digit Data Set[2]' found on the UCI Machine Learning Repository, as it meets my data set requirements, highlighted above.

The experiment taken place to obtain the data consisted of approximately 80 persons, who were instructed to write the digits from 0 to 9 on paper, twice. The initial set of digits had the subjects write in a 'normal way' focusing on accuracy, with the final set of digits being written in a 'fast way' with little accuracy. This is important as handwriting can differ massively depending on the speed at which it is written[3].

1593 total handwritten digits were gathered, scanned and then scaled to fit a 16x16 greyscale box. The imaged was then scaled into a set of Boolean values (0 being greyscale values under and including 127, 1 being any greyscale value above this threshold) giving us 256 pixels for each digit. This resolution gives more than enough inputs to produce an accurately trained MLP[3].

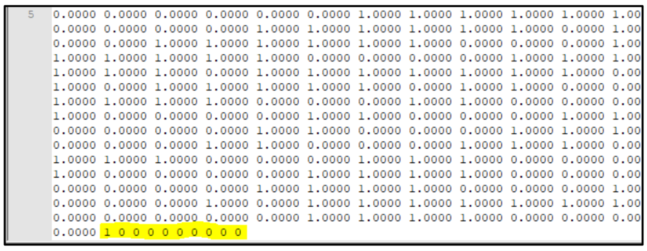
2.3. Dataset formatting/implementation

The MLP reads input data in a certain way, specified by the first four lines of each input file.



```
256 10 %.4f %.0f %.0f
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109
120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140
151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171
182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202
213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233
244 245 246 247 248 249 250 251 252 253 254 255 256 o1 o2 o3 o4 o5 o6 o7 o8 o9 o10
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Figure 1. Dataset formatting



```
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.00
0.0000 0.0000 0.0000 0.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 0.0000 1.00
0.0000 0.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 0.0000 0.0000 0.0000 1.00
1.0000 1.0000 1.0000 1.0000 1.0000 0.0000 0.0000 0.0000 0.0000 1.0000 1.0000 1.00
1.0000 1.0000 1.0000 0.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 0.00
1.0000 0.0000 1.0000 1.0000 1.0000 1.0000 1.0000 0.0000 1.0000 1.0000 0.0000 0.00
1.0000 1.0000 1.0000 1.0000 0.0000 0.0000 0.0000 1.0000 1.0000 0.0000 0.0000 0.00
1.0000 0.0000 0.0000 0.0000 0.0000 1.0000 1.0000 0.0000 0.0000 0.0000 1.0000 1.00
0.0000 0.0000 0.0000 0.0000 1.0000 1.0000 0.0000 0.0000 0.0000 1.0000 1.0000 1.00
0.0000 0.0000 0.0000 1.0000 1.0000 0.0000 0.0000 0.0000 1.0000 1.0000 1.0000 0.00
1.0000 1.0000 1.0000 0.0000 0.0000 0.0000 1.0000 1.0000 1.0000 0.0000 0.0000 0.00
1.0000 0.0000 0.0000 0.0000 0.0000 1.0000 1.0000 1.0000 1.0000 0.0000 0.0000 0.00
0.0000 0.0000 0.0000 0.0000 1.0000 1.0000 1.0000 1.0000 0.0000 0.0000 0.0000 1.00
0.0000 0.0000 0.0000 1.0000 0.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.00
0.0000 0.0000 0.0000 0.0000 0.0000 1.0000 1.0000 1.0000 1.0000 0.0000 0.0000 0.00
0.0000 1 0 0 0 0 0 0 0 0 0
```

Figure 2. Example block of data

2.3.1. Parameters and format

Section 1, of figure 1, defines the input parameters and formatting. Each input value is separated by a space.

256 10 %.4f %.0f %.0f

The first and second values, 256 and 10 denote the input and output amounts respectively. For this dataset each pixel is used as an input, with 10 possible outputs (character set of digits from 0-9). The next three values of format: %.$\boldsymbol{x}$f, where $\boldsymbol{x}$ is the amount of decimal places, represent the format of the inputs, outputs and raw neural network outputs respectively.

2.3.2. Input/output names

Section 2, of figure 1, defines the name for each input and output.

Each input is labelled by its corresponding position in the block, from 1-256. Each output is labelled as o1-o10, for each of the possible 9 characters (digits 0-9).

2.3.3. Minimum values

Each value in section 3 of figure 1 represents the minimum value of each corresponding input and output. As all values are Boolean, each input/output has a minimum of 0 (266 total values, 256 inputs + 10 outputs).

2.3.4. Maximum values

Each value in section 4 of figure 1 represents the maximum value of each corresponding input and output. As all values are Boolean, each input/output has a maximum of 1 (266 total values, 256 inputs + 10 outputs).

2.3.5. Data blocks

Figure 2 shows an example block of data, representing one handwritten character. First are the 256 input values for each pixel, followed by 10 binary values (highlighted in yellow) representing which digit the block denotes, all separated by spaces. The conversion between the binary output string and the digit set is as follows:

1	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9

Figure 4. Data block example digit conversion

As you can see from figure 4, the example data block from figure 2 represents the digit 0.

2.4. Dividing dataset

The dataset must be split into three sections for each of the three input files, usertrain.txt, uservalid.txt and userunseen.txt.

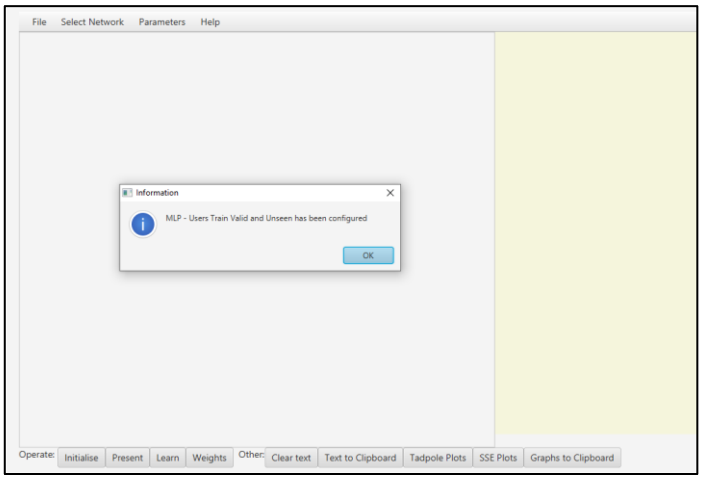
2.4.1. Training data

The dataset description[3] recommends a 50% split between the training and validation datasets. In order to make sure the MLP is accurate, I used this recommended amount for training data. This means approximately 797 handwritten digits are used to train the network.

2.4.2. Validation/unseen data

Even though it is recommended[3] to use a 50% split for validation data as well, I decided the validation set isn't as important as the training set, so have set the remaining split to 40% validation (approx. 637 digits), 10% unseen (approx. 159 digits). Since the unseen dataset is only used in assessing the performance of the network, it gets a minimal split.

3. RESULTS OF APPLICATION



MLP - Users Train Valid and Unseen has been configured

Figure 3. Overview of application GUI

As shown in figure 3, the three data sets are successfully configured for the network. I can edit all the parameters using the corresponding options, and can initialise, train and display information using the options along the bottom.

3.1. Default parameters

3.1.1. Initial results/ results explanation

The initial experiment will have the network trained on the default parameters: Learning rate = 0.20, Momentum = 0.00, Hidden neurons = 10, Max epochs = 1000, Random seed = 100.

Before 'learning' the network, the results are as follows:

Train:
SSE: 0.2078 0.0627 0.1635 0.1980 0.1986 0.5529 0.1659 0.0797 0.4388 0.1004
%Correct: 45 89 58 43 48 9 57 85 11 80

Unseen:
SSE: 0.1927 0.0563 0.1655 0.2020 0.2183 0.5546 0.1711 0.0764 0.4374 0.1074
%Correct: 46 89 56 42 48 10 56 85 11 76
Valid:
SSE: 0.2048 0.0589 0.1636 0.1962 0.2017 0.5571 0.1647 0.0786 0.4348 0.1155
%Correct: 44 89 58 47 47 9 59 84 13 77

In general, if an output is above 50% correct, that handwritten digit can be recognised by the network. However, I ideally want each output to be 100% for the highest accuracy, but above 90% is satisfactory. The unseen dataset results can be used to evaluate the networks effectiveness as it shows what 'new' data looks like on the network after training (without any given outputs).

As you can see by the results above, even though certain digits such as 1 (89%) and 7 (85%) seem to have high accuracy, they are mostly inaccurate. This is expected as the current initial weights is random.

3.1.2. Default settings training

The results after training the network on the default parameters are as follows:

Stopped learning after 40 epochs.
Train:
SSE: 0.0053 0.0134 0.0091 0.0091 0.0096 0.0117 0.0064 0.0069 0.0096 0.0090
%Correct: 99 98 98 99 99 98 100 100 99 98
Unseen:
SSE: 0.0163 0.0282 0.0283 0.0199 0.0280 0.0215 0.0223 0.0302 0.0346 0.0272
%Correct: 97 93 93 96 94 97 95 94 94 93
Valid:
SSE: 0.0114 0.0322 0.0323 0.0276 0.0257 0.0307 0.0180 0.0281 0.0346 0.0333
%Correct: 98 94 94 95 95 94 97 95 93 92

After 40 epochs, the network finished learning. Since all digits in the unseen dataset are above 90% correct, I can conclude that using the default parameters allows the network to be trained with a high degree of accuracy. However, the network has not reached its highest potential accuracy as only digits 6 and 7 are 100% correctly trained.

3.2. Adjusted parameters

3.2.1. Initial adjustment

The initial adjustment will have the network trained on the following parameters: Learning rate = 0.10, Momentum = 0.00, Hidden neurons = 25, Max epochs = 500, Random seed = 100.

The learning rate is decreased to 0.10 as the program takes a reasonably small amount of time to learn using the datasets anyway, so lowering this value (lowers the amount the weights change each epoch) wont effect the execution time that much, and might result in more accurate results. Hidden neurons is also increased to 25 in an attempt to increase accuracy. Max epochs is decreased to 500, as the previous attempt only took 40 epochs to learn. All other parameters are kept consistent with the previous experiment as control variables.

Stopped learning after 40 epochs.
Train:
SSE: 0.0063 0.0109 0.0080 0.0075 0.0064 0.0073 0.0061 0.0090 0.0083 0.0090
%Correct: 99 98 99 99 100 99 100 99 99 99
Unseen:
SSE: 0.0173 0.0227 0.0235 0.0277 0.0302 0.0266 0.0244 0.0278 0.0338 0.0304
%Correct: 97 95 95 93 94 94 96 94 93 94
Valid:
SSE: 0.0137 0.0353 0.0324 0.0360 0.0252 0.0317 0.0207 0.0388 0.0329 0.0339
%Correct: 98 93 93 93 95 94 97 92 94 94

The results are similar to the previous experiments, with certain digits such as 1 (95%) and 2 (95%) being more accurate whilst other digits such as 7 (94%) and 8 (93%) being less accurate. However, the training dataset improved slightly in accuracy (on average) so I can conclude this experiment brings the network closer to its highest potential accuracy.

3.2.2. Further adjustments

The only adjustment during this experiment will be: Hidden neurons = 50. All other parameters are kept consistent with the previous experiment as control variables.

Epoch 50:
SSE: 0.0039 0.0054 0.0048 0.0063 0.0043 0.0047 0.0046 0.0045 0.0048 0.0060
%Correct: 100 100 100 99 100 100 100 100 100 99
Epoch 100:
SSE: 0.0030 0.0042 0.0036 0.0042 0.0033 0.0037 0.0032 0.0032 0.0032 0.0038
%Correct: 100 100 100 100 100 100 100 100 100 100
Stopped learning after 110 epochs.
Train:
SSE 0.0029 0.0040 0.0036 0.0041 0.0031 0.0035 0.0030 0.0031 0.0030 0.0034
%Correct: 100 100 100 100 100 100 100 100 100 100
Unseen:
SSE: 0.0201 0.0416 0.0294 0.0368 0.0274 0.0308 0.0267 0.0385 0.0529 0.0399
%Correct: 96 91 93 92 95 92 95 94 90 91
Valid:
SSE: 0.0143 0.0377 0.0366 0.0327 0.0309 0.0340 0.0278 0.0373 0.0487 0.0361
%Correct: 98 93 93 93 94 93 95 93 91 93

After 50 epochs, all digits except from 3 and 9 are fully learnt from the training dataset, and by 100 epochs all digits are fully learnt from the training dataset.

Even though the accuracy observed from the unseen dataset is similar but varied compared to previous experiments, the network is trained to its highest potential accuracy, as shown by the training results.

I can conclude this experiment to be successful. No further experimentation is required as the network has reached its full potential. Further adjustments could involve fine tuning the amount of hidden neurons to improve execution time, however this is time consuming and rather irrelevant since even with 50 hidden neurons, the execution time is reasonable (under one minute).

4. DISCUSSION

Looking over each experiment's unseen and validation datasets (and figure 5, figure 6), I can conclude that this specific neural network is at least 90% accurate when trying to identify a handwritten digit (regardless of how accurately drawn the digits are).

To improve the accuracy even further, a larger sample size of digits from a larger amount of people could be used when gathering data. This would give a more varied set of data as everyone's handwriting is slightly different.

Possible applications for this study could include: increasing the set of gathered characters to include all mathematical operational symbols, and using the resulting MLP to develop software that can read handwritten mathematical notes. The software could even be developed to identify mathematical errors and to solve unsolved equations. However, I believe this would be difficult to achieve as there are many complexly drawn symbols used in mathematics that would most definitely decrease the MLP's accuracy

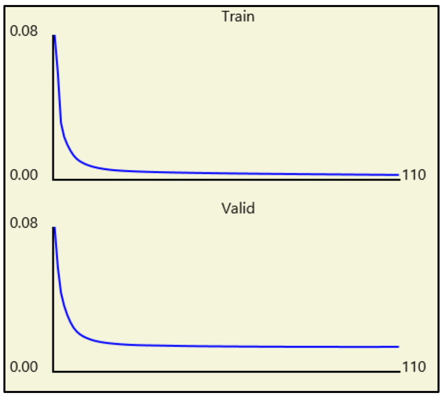


Figure 5. SSE plots graph

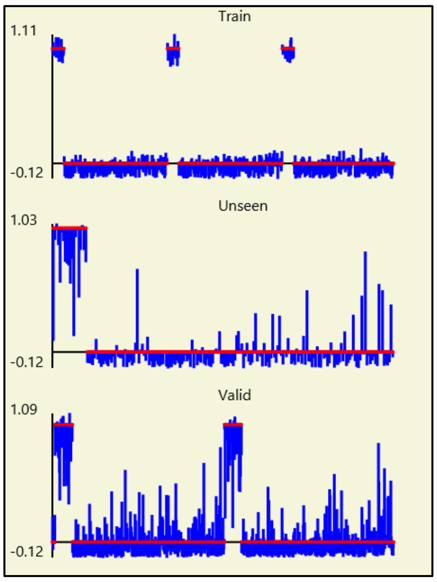


Figure 6. Tadpole plots graph

5. CONCLUSION

The results from this study definitively shows that a MLP can indeed be used with the 'Semeion Handwritten Digit Data Set[2]' to create a neural network that can identify handwritten digits to a reasonably high degree of accuracy.

6. REFERENCES

[1] R. J. Mitchel, Cybernetics Intelligence Research Group, School of Systems Engineering, University of Reading, "RJMs Working Neural Net". https://www.bb.reading.ac.uk/bbcswebdav/pid-3862702-dt-content-rid-12138483_2/xid-12138483_2 (2019)

[2] Tactile Srl, Brescia, Italy (http://www.tattile.it), Semeion Research Center of Sciences of Communication, Rome, Italy (http://www.semeion.it), "Semeion Handwritten Digit Dataset". https://archive.ics.uci.edu/ml/machine-learning-databases/semeion/semeion.data (1994)

[3] Tactile Srl, Brescia, Italy (http://www.tattile.it), Semeion Research Center of Sciences of Communication, Rome, Italy (http://www.semeion.it), "Semeion Handwritten Digit Dataset Description". https://archive.ics.uci.edu/ml/machine-learning-databases/semeion/semeion.names (1994)