# Optical Digital Recognition Data set:

The main challenge in designing a neural network is to tune the parameter in a way that can achieve higher accuracy with the minimal number hidden layers and internal nodes. This will decrease the learning time and the complexity for the overall model.

To achieve this, the design for the NN will be splitted up in two parts. First, feature engineering. Second, tune the actual parameters.

**Feature Engineering:**

From the previous analysis, the dataset has 64 feature. Based on our knowledge of the dataset, each column has values ranging from 0 to 16. In addition to that, some columns might have only 2 or 3 values from the range [0-16] while other might have more. To simplify the resulted neural network, a scaler will be used to transform each column to values from -1,1 with 0 mean and 1 standard deviation. This transformation will make the NN converge faster for example the table below show the result for the same neural network one trained without normalizing while the other trained after normalizing:

|  |  |  |
| --- | --- | --- |
| Input Type | Accuracy Over Training Set | Time (seconds) |
| Without Normalizing | 10% | 0.719 |
| With Normalizing | 61.2% | 0.659 |

Table 1 In both cases NN has 200 iterations with RELU activation function and 1e-5 learning rate

**Parameter Tuning:**

The parameter selected to tune are:

1. Activation function
2. The weight optimizer
3. Number of hidden layer
4. Number of nodes in each layer

**Activation Function:**

There are many different activation functions that could be used and converge. To select one a comparison between ‘identity’, ‘logistic’, ‘tanh’ and ‘relu’ functions conducted in terms of accuracy and time.

|  |  |  |  |
| --- | --- | --- | --- |
| Activation Function | Normalized Input | Accuracy on Train Set | Time (seconds) |
| Identity | No | 73.2% | 0.876 |
|  | Yes | 80.3% | 0.626 |
| Logistic | No | 29.3% | 0.762 |
|  | Yes | 40.4% | 0.619 |
| Tanh | No | 20.9% | 0.8 |
|  | Yes | 51.8% | 0.775 |
| Relu | No | 10.2% | 0.355 |
|  | Yes | 60.7% | 0.706 |

Table 2: All results generated based on NN with 2 hidden layers and 2 nodes in each layer, 200 iterations and 1e-5 learning rate

As shown above the identity function works better compared to the others. While in terms of time Relu is much faster. Since, the dataset is not huge the identity function will be used for later analysis.

**The weight optimizer**Like activation function weight optimizer is also data dependent and should be selected beforehand. ‘lbfgs’, ‘sgd’ and ‘adam’ are different optimizer methods. ‘lbfgs’ is one of the quasi-newton. ‘sgd’ is the stochastic gradient decent and ‘adam’ is a stochastic gradient-based optimizer. Practically, ‘lbfgs’ converges faster for small dataset, ‘sgd’ needs lots of effort for tuning and ‘adam’ is very robust to large dataset. The table below show a comparison between the three.

|  |  |  |
| --- | --- | --- |
| Weight Optimizer | Accuracy on Training Set | Time(seconds) |
| Sgd | 66.3% | 1.764 |
| Lbfgs | 80.4% | 0.589 |
| Adam | 75.6% | 1.847 |

Table 3 NN with 200 iteration, Identity function, 1e-5 learning rate

With 200 iterations ‘lbfgs’ converges faster and gives a higher accuracy. So, it will be used. The results agree with the theory that lbfgs works better for relatively smaller dataset.

**Number of Hidden Layer & Number of Nodes**

The previous analysis conducted with the accuracy on the same data. The reason behind this, is just to get a flavor of how the NN behaves on different parameters given same number of hidden layers and nodes which (2,2). Since the number of hidden layers and nodes is the main component of NN design, cross-validation with 5 foldes will be used over the training set to make sure the NN is generalizable. For the cross validation 70% used for training and 30% used as validation set.

Since, it’s advisable to favor more simple Neural Networks over complicated ones. The first test will be conducted using 1 hidden layer and change the number of nodes in the hidden layer. The results were as shown in the table below.

|  |  |  |
| --- | --- | --- |
| Number of Nodes | Accuracy | Time(sec) |
| 1 | 45.93% | 2.34 |
| 2 | 77.59% | 2.21 |
| 3 | 88.74% | 2.13 |
| 4 | 90.34% | 2.10 |
| 5 | 92.01% | 2.15 |
| 6 | 93.50% | 0.71 |
| 7 | 94.98% | 0.61 |
| 8 | 94.94% | 0.52 |
| 9 | 94.98% | 0.52 |
| 10 | 95.66% | 0.51 |
| 11 | 95.17% | 0.53 |
| 12 | 95.47% | 0.52 |
| 13 | 95.40% | 0.50 |
| 14 | 95.22% | 0.51 |
| 15 | 95.62% | 0.50 |
| 16 | 95.66% | 0.51 |
| 17 | 95.38% | 0.56 |
| 18 | 95.66% | 0.51 |
| 19 | 95.38% | 0.52 |

Table 4: Accuracy and time for 1-hidden layer

Figure 1: A graph for the accuracy over different number of nodes on a one single hidden layer NN with 5 cross validations

As shown from the figure above, after 6 nodes for the hidden layer the accuracy change is small. And it’s almost the same after 9 nodes while it’s maximum value at 10 nodes. This result means that, each hidden node among the 10 will contribute to estimate the function representing one of the discrete input [0-9].

Figure 2 Training time for different node time

In terms of time, there’s a big drop in time after adding the sixth node. This result is expected because the convergence for one node given multiple input will take longer time compared to having more nodes that can represent the function.

For further analysis, same study conducted using 2 hidden layer with different combinations nodes in each layer i.e. first and second hidden layers. The study conducted first by assigning one node for the first hidden layer and test the accuracy over 5 cross validation with number of nodes from 1 to 10 in the second layer. Then the first layer is set to 2 nodes and same analysis conducted on the second layer and so on.

Figure 3 Different combination for number of nodes for two hidden layers

The graph shows that, adding a layer doesn’t boost the accuracy to higher values compared to the previous graph of one hidden layer. Also, it shows that there is a cyclic behavior when adding nodes on each layer. In terms of performance, the one hidden layer performs better that two layer.

**Final Design**

Based on the previous analysis, the selected neural network parameter will be as follows:

|  |  |
| --- | --- |
| Number of Iterations | 200 |
| Number of hidden layer | 1 |
| Number of nodes in hidden layer | 10 |
| Learning rate | 1e-5 |
| Weight optimizer | Lbfgs |
| Activation function | identity |

The result on the test set was: 93.3%

# Amazon Baby Product Review Dataset

Same approach will be followed with this dataset. First, feature engineering. Then, tuning parameters.

**Feature Engineering**

Since the data set has two text columns i.e product title and product review. The two columns processed as follows:

1. Combine both columns together
2. Drop stop words
3. Take the stem for every word
4. Count the number of word occurrence and use it as feature for further processing

Counting process results in more than 5500 columns. To reduce the dimension a threshold of minium number of word count should be determined. The table below shows the number of columns (features) resulted with different threshold value.

|  |  |  |
| --- | --- | --- |
| Minimum Number of Words | Number of Features | Difference from previous cutoff |
| 50 | 5328 |  |
| 100 | 3856 | 1472 |
| 150 | 3149 | 707 |
| 200 | 2727 | 422 |
| 250 | 2443 | 284 |
| 300 | 2216 | 227 |
| 350 | 2050 | 166 |
| 400 | 1908 | 142 |
| 450 | 1781 | 127 |
| 500 | 1662 | 119 |

Table 5 Number of features for differnt min word count

Based on the table above, the most dimension reduction happens with changing the cutoff from 50 to 100. So, 100 word will be used as a minimum word count for further analysis.

The normalizing process is very data dependable on the dataset. Hence, a small experiment conducted to decide wither continue with normalized columns or raw columns. As follows,

|  |  |  |
| --- | --- | --- |
| Input Type | Accuracy Over Training Set | Time (seconds) |
| Without Normalizing | 67.5% | 33 |
| With Normalizing | 67.4% | 26 |

Table 6: IN BOTH CASES NN HAS 200 ITERATIONS WITH RELU ACTIVATION FUNCTION AND 1E-5 LEARNING RATE

**Parameter Tuning:**

The parameter selected to tune are:

1. Activation function
2. The weight optimizer
3. Number of hidden layer
4. Number of nodes in each layer

**Activation Function:**

There are many different activation functions that could be used and converge. To select one a comparison between ‘identity’, ‘logistic’, ‘tanh’ and ‘relu’ functions conducted in terms of accuracy and time.

|  |  |  |
| --- | --- | --- |
| Activation Function | Accuracy on Train Set | Time (seconds) |
| Identity | 67.6% | 21 |
| Logistic | 64.4% | 29 |
| Tanh | 67.7% | 31 |
| Relu | 67.3% | 28 |

Table 7: ALL RESULTS GENERATED BASED ON NN WITH 2 HIDDEN LAYERS AND 2 NODES IN EACH LAYER, 200 ITERATIONS AND 1E-5 LEARNING RATE

Based on the table above, most activation function produce almost similar results. Looking at training time ‘identity’ was the best so it will be chosen for further analysis.

**The weight optimizer**Similar to activation function weight optimizer is also data dependent and should be selected beforehand. ‘lbfgs’, ‘sgd’ and ‘adam’ are different optimizer methods. ‘lbfgs’ is one of the quasi-newton. ‘sgd’ is the stochastic gradient decent and ‘adam’ is a stochastic gradient-based optimizer. Practically, ‘lbfgs’ converges faster for small dataset, ‘sgd’ needs lots of effort for tuning and ‘adam’ is very robust to large dataset. The table below show a comparison between the three.

|  |  |  |
| --- | --- | --- |
| Weight Optimizer | Accuracy on Training Set | Time(seconds) |
| Sgd | 67.5% | 24.8 |
| Lbfgs | 67.6% | 24.0 |
| Adam | 67.5% | 50 |

Table 8 NN with 200 iteration, Identity function, 1e-5 learning rate

Again, with 200 iterations ‘lbfgs’ converges faster and gives a higher accuracy. So, it will be used. The results agree with the theory that lbfgs works better for relatively smaller dataset.

**Number of Hidden Layer & Number of Nodes**

The previous analysis conducted with the accuracy on the same data. The reason behind this, is just to get a flavor of how the NN behaves on different parameters given same number of hidden layers and nodes which (2,2). Since the number of hidden layers and nodes is the main component of NN design, cross-validation with 5 foldes will be used over the training set to make sure the NN is generalizable. For the cross validation 70% used for training and 30% used as validation set.

Since, it’s advisable to favor more simple Neural Networks over complicated ones. The first test will be conducted using 1 hidden layer and change the number of nodes in the hidden layer. The results were as shown in the table below.

|  |  |  |
| --- | --- | --- |
| Number of Nodes | Accuracy | Time(sec) |
| 1 | 62.99% | 15.92 |
| 2 | 66.62% | 41.56 |
| 3 | 66.91% | 63.93 |
| 4 | 66.81% | 96.45 |
| 5 | 66.82% | 111.34 |
| 6 | 66.82% | 90.71 |
| 7 | 66.82% | 106.03 |

Table 9: ACCURACY AND TIME FOR 1-HIDDEN LAYER

Figure 4: A graph for the accuracy over different number of nodes on a one single hidden layer NN with 5 cross validations

As shown above, the highest accuracy achieved on 3 nodes while the execution time is acceptable (63 seconds).

In terms of time, 3 nodes (highest accuracy) gives a reasonable time. There is a drop-in time for 6 nodes and this because each function is presented by one neuron so the convergence is faster but still the 3 nodes works better.

For further analysis, same study conducted using 2 hidden layer with different combinations nodes in each layer i.e. first and second hidden layers. The study conducted first by assigning one node for the first hidden layer and test the accuracy over 5 cross validation with number of nodes from 1 to 7 in the second layer. Then the first layer is set to 2 nodes and same analysis conducted on the second layer and so on.

|  |  |  |  |
| --- | --- | --- | --- |
| 1st Layer | 2nd Layer | Avg. Accuracy | Time(sec) |
| 1 | 1 | 62.99% | 16.86 |
| 1 | 2 | 62.99% | 30.50 |
| 1 | 3 | 62.99% | 24.10 |
| 1 | 4 | 62.99% | 22.66 |
| 1 | 5 | 62.99% | 27.34 |
| 1 | 6 | 62.99% | 34.45 |
| 1 | 7 | 62.99% | 37.96 |
| 2 | 1 | 62.98% | 32.24 |
| 2 | 2 | 66.62% | 77.30 |
| 2 | 3 | 66.62% | 72.61 |
| 2 | 4 | 66.62% | 78.98 |
| 2 | 5 | 66.63% | 56.93 |
| 2 | 6 | 66.62% | 52.59 |
| 2 | 7 | 66.62% | 59.60 |
| 3 | 1 | 62.99% | 38.26 |
| 3 | 2 | 66.62% | 77.42 |
| 3 | 3 | 66.89% | 101.29 |
| 3 | 4 | 66.91% | 77.06 |
| 3 | 5 | 66.90% | 91.62 |

Table 10: two layer nn acuraccy with execution time

As shown from the table above there is no improvement compared to the 1 hidden layer with 3 neurons. So the later will be used as the selected model.

**Final Design**

Based on the previous analysis, the selected neural network parameter will be as follows:

|  |  |
| --- | --- |
| Number of Iterations | 200 |
| Number of hidden layer | 1 |
| Number of nodes in hidden layer | 3 |
| Learning rate | 1e-5 |
| Weight optimizer | Lbfgs |
| Activation function | identity |

The result on the test set was: 67%