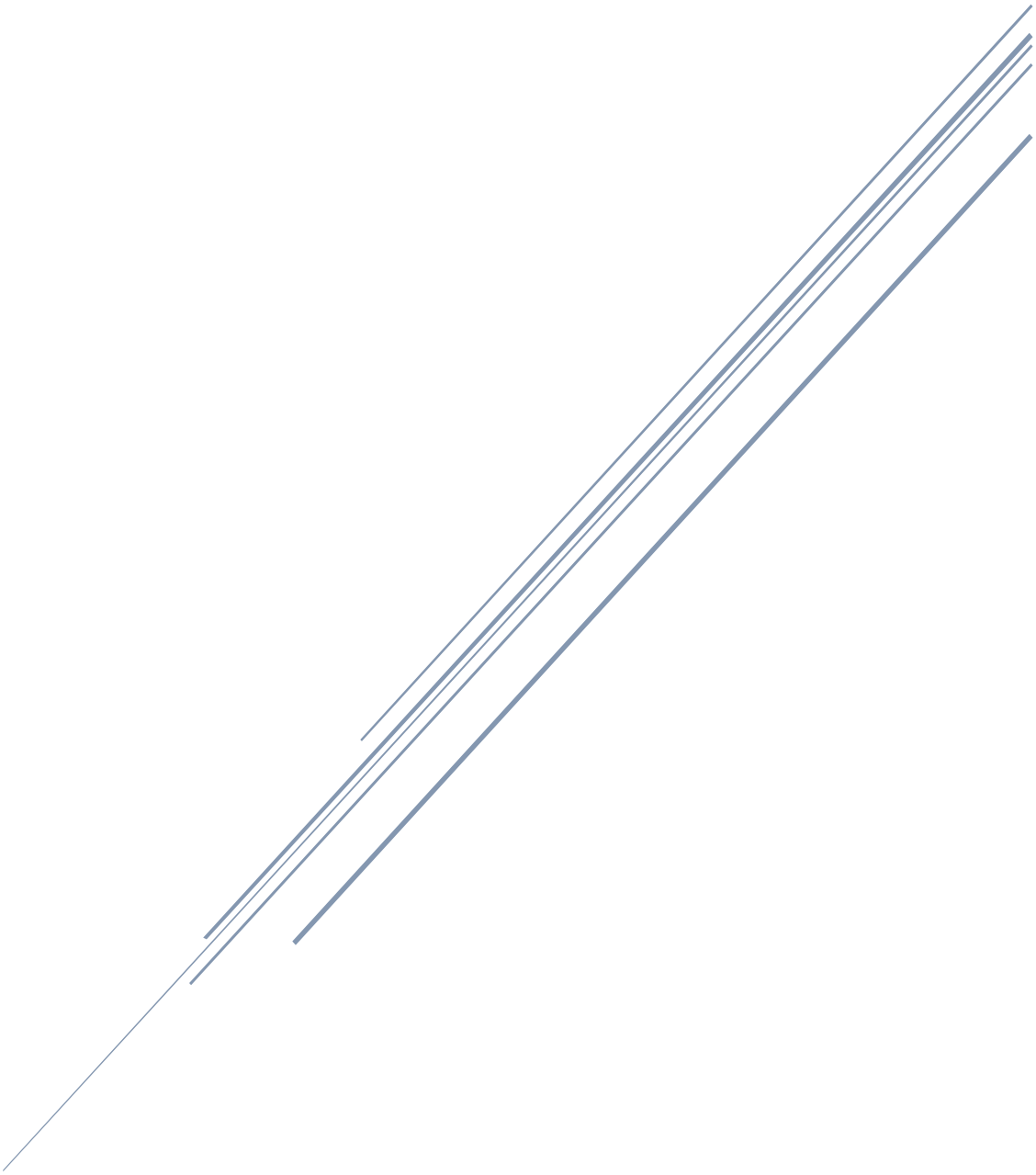
NEURAL NETWORKS & CI CONCEPTS

CSC-20023: Computational Intelligence 1



Student Number: W4S22
Student Name: Alex Farrell

Contents

Part 1: Pattern Classification of Digitised Letters

The best network(s) for each topology is highlighted in yellow. Any anomalies are highlighted in red. Please note, all anomalies have been discounted from the calculation of any averages.

2 Hidden Layers

Network Name	Training epochs	Validation Patterns Correct	Test Patterns Correct	Sum squared error
63_2_10 v1	2000	0	0	13.45468998
63_2_10 v2	2000	0	0	14.39789009
63_2_10 v3	1900	1	0	13.23003387
63_2_10 v4	2000	0	0	13.94284578
63_2_10 v5	1800	0	0	13.10644913
Average	**1940**	**0.2**	**0**	**13.62638177**

To train and a network with 2 hidden layers, I set a limit of 2000 training epochs, each interval being 100 epochs. Before starting, I did a test with a version of this neural network to see roughly how many epochs it would take before either the network could correctly identify all 10 validation patterns, or the generalisation rate started to fall. In this case, the generalisation rate began to fall around 2000 epochs, hence choosing this value as my upper limit.

4 Hidden Layers

Network Name	Training epochs	Validation Patterns Correct	Test Patterns Correct	Sum squared error
63_4_10 v1	1200	10	10	0.229450464
63_4_10 v2	1200	10	10	0.229450464
63_4_10 v3	1300	7	8	0.184671342
63_4_10 v4	1200	10	10	0.229450464
63_4_10 v5	400	10	8	0.657790244
Average	**1225**	**9.25**	**9.5**	**0.218255684**

To train a network with 4 hidden layers I set a limit of 1300 training epochs, each interval being 100 epochs. Before starting, I did a test with a version of this neural network to see roughly how many epochs it would take before either the network could correctly identify all 10 validation patterns, or the generalisation rate started to fall. In this case, by 1300 epochs, the network could correctly identify all 10 validation patterns, hence choosing this value as my upper limit.

There was however an anomaly (63_4_10 v5). I chose to stop training as soon as the network could correctly recognise all 10 validation patterns. On this occasion, after 400 training epochs, this case was reached. The shorter training regime was reflected in the results of both the test patterns and the sum squared error as the network was only able to recognise 8 test patterns as appose to v1, v2 and v4, which correctly identified all 10 test patterns. The sum squared error was also much higher with a value over 3 times the other networks.

6 Hidden Layers

Network Name	Training epochs	Validation Patterns Correct	Test Patterns Correct	Sum squared error
63_6_10 v1	150	10	6	1.297285557
63_6_10 v2	250	10	8	0.540525496
63_6_10 v3	250	10	8	0.540525496
63_6_10 v4	150	10	7	1.286847591
63_6_10 v5	150	10	7	1.286847591
Average	**190**	**10**	**7.2**	**0.990406346**

To train and a network with 6 hidden layers, I set a limit of 500 training epochs, each interval being 50 epochs. Before starting, I did a test with a version of this neural network to see roughly how many epochs it would take before either the network could correctly identify all 10 validation patterns, or the generalisation rate started to fall. In this case by 500 epochs, the network could correctly identify all 10 validation patterns, hence choosing this value as my upper limit.

8 Hidden Layers

Network Name	Training epochs	Validation Patterns Correct	Test Patterns Correct	Sum squared error
63_8_10 v1	125	10	7	1.411833763
63_8_10 v2	150	10	8	0.817087948
63_8_10 v3	75	10	7	1.761594296
63_8_10 v4	75	10	8	1.963321328
63_8_10 v5	75	10	7	1.62525034
Average	**100**	**10**	**7.4**	**1.515817535**

To train and a network with 8 hidden layers, I set a limit of 200 training epochs, each interval being 25 epochs. Before starting, I did a test with a version of this neural network to see roughly how many epochs it would take before either the network could correctly identify all 10 validation patterns, or the generalisation rate started to fall. In this case by 200 epochs, the network could correctly identify all 10 validation patterns, hence choosing this value as my upper limit.

10 Hidden Layers

Network Name	Training epochs	Validation Patterns Correct	Test Patterns Correct	Sum squared error
63_10_10 v1	100	10	8	0.849677622
63_10_10 v2	100	10	8	0.937438905
63_10_10 v3	125	10	9	0.575377703
63_10_10 v4	100	10	6	0.998154879
63_10_10 v5	100	10	8	1.629444361
Average	**105**	**10**	**7.8**	**0.998018694**

To train and a network with 10 hidden layers, I set a limit of 150 training epochs, each interval being 25 epochs. Before starting, I did a test with a version of this neural network to see roughly how many epochs it would take before either the network could correctly identify all 10 validation patterns, or the generalisation rate started to fall. In this case by 150 epochs, the network could correctly identify all 10 validation patterns, hence choosing this value as my upper limit.

Averages

Network Name	Training epochs	Validation Patterns Correct	Test Patterns Correct	Sum squared error
63_2_10	1940	0.2	0	13.62638177
63_4_10	1225	9.25	9.5	0.218255684
63_6_10	190	10	7.2	0.990406346
63_8_10	100	10	7.4	1.515817535
63_10_10	105	10	7.8	0.998018694

From the table of averages above, there are several conclusions that we can draw out. The first is that the results show that as the number of hidden layers increases, generally, the better the network becomes at being able to correctly identify the validation patters, and in less training epochs.

We can also say categorically, that the best network topology is one with 4 hidden layers. Although this network topology took over 1000 training epochs to be able to correctly identify all 10 validation patterns, the SSE (sum squared error) was subsequently very low. This network topology was also the best at being able to correctly identify all 10 test patterns with 3 of the 5 training cycles being able to do this.

Best Networks

Network Name	Training epochs	Validation Patterns Correct	Test Patterns Correct	Sum squared error
63_2_10 v3	1900	1	0	13.23003387
63_4_10 v1	1200	10	10	0.229450464
63_6_10 v2	250	10	8	0.540525496
63_8_10 v2	150	10	8	0.817087948
63_10_10 v2	100	10	8	0.937438905

From the table of best networks above, once again, the best network topology is one with 4 hidden layers. However, I believe there is a reason why this network is so well trained. I think that I over trained this network. Whilst training, the network managed identify 9 validation patterns after 700 training epochs. It took until 1200 epochs to correctly identify the final validation pattern. I therefore believe I went past the peak generalisation rate to achieve this.

Part 2: CI Problem Solving

2.1 Neural Networks Question

Part A

To effectively train this neural network, a supervised neural network architecture should be used. The conditions required to undertake supervised learning include having a pre-existing data set which can be used to train the neural network, as well as knowing what the correct output should look like for a given set of input values. In the scenario given, the input data that is available is 'a wealth of detailed house-sale data'. Attached to each set of house-sale data, is the actual selling price for that house. This will be the output that the system will present to the user.

The type of supervised learning strategy that should be used is a regression problem. This is where we are trying to predict results within a continuous output. As the price is a form of continuous output, this classifies as a regression problem.

Part B

For the system to be able to work with the data, it must be processed so that it can be presented to the system, in a form that it can understand. This means that first, we must decide on the type of data for each characteristic. The table below shows each characteristic as well as the data type it has been assigned and the number of units required to represent each one:

Characteristic	Data type	Input units required
Neighbourhood rating	Discrete	1
Condition of property	Discrete	1
Number of bedrooms	Discrete	1
Number of bathrooms	Discrete	1
Total living area	Continuous	1
Total plot-size	Continuous	1
Garage size	Discrete	1
Heating system	Categorical	4
Glazing	Discrete	1
	Total input units required	**12**

House price (Output)	**Continuous**	**1**

Now that we know the data type that will be used for each of the characteristics, we need to modify the data so that the system can understand it:

- Discrete data: Use the normalisation function to work out the magnitude for each of the possibilities, where d_{min} is the smallest value to be used (e.g. 1), and d_{max} is the largest value to be used (e.g. 4). Using this function will mean that any discrete data type can be represented as a number between 0 and 1 inclusive.

- Continuous data: Use the logarithmic normalisation function to work out the magnitude for each possibility in the range. Logarithmic normalisation is preferred to standard normalisation here, due to the large range of values which can be entered. If standard normalisation was used, the values at the upper end of the range, would require more units to represent them, whereas logarithmic normalisation scales values at both ends of the range so that there is a more equal distribution of the magnitude.
- Categorical data: This data type requires one input unit for each possibility due to this data not having any sense of magnitude. Each unit has 2 possibilities (0 or 1) and only 1 unit in each set can be active (i.e. only 1 unit can have a value of 1, meaning the rest of the units in the set will be 0).

To summarise, below is a table stating the size of both the input and output layer of the neural network:

Input Layer	12 units
Output Layer	1 unit

Part C

To translate the network's output to represent a value that can be recognized by the user as the valuation for the property, we need to do the reverse of the logarithmic normalisation function that we are using due to price being a continuous data type.

As we only have 1 output unit, the activation that will be produced will be a value between 0 and 1. This value will represent the magnitude. For the sake of this network, we are going to assume that the minimum valuation (d_{min}) is £10,000 and the maximum valuation (d_{max}) is £1,000,000. Putting both values through the logarithmic function $log_{10}(d)$ (where d is the valuation), gives us the following values:

- $log_{10}(d_{min}) = 4$
- $log_{10}(d_{max}) = 6$

Using these values in the normalisation function, we now know that a valuation of £10,000 will produce a magnitude of 0 and a valuation of £1,000,000 will produce a magnitude of 1.

Doing the reverse of this function for any given output activation that the neural network will produce, will allow us to determine a valuation by reversing the logarithmic normalisation function. Below is an example of how we would do this using a resulting activation of 0.5:

Logarithmic Normalisation function: $d' = (log_{10}(d) - d_{min}) / (d_{max} - d_{min})$

$0.5 = (log_{10}(d) - 4) / (6 - 4)$

$0.5 * (6-4) = log_{10}(d) - 4$

$5 = log_{10}(d)$

$10^5 = d$

$\underline{100{,}000 = d}$

Part D

The training regime that I would use is the '3 dataset MLP' training regime using back propagation. First, I would split the historical data that has been provided into 3 sets:

- New training dataset: this dataset will be used to train the network.
- Validation dataset: this dataset will be used to determine the generalisation rate.
- Testing dataset: this would be passed through at the end to determine a final generalisation rate.

The reason that 3 datasets are required is because if you give a network enough time with the training data, it would be able to memorise this data almost perfectly. However, if you were to then present the network with a completely different set of unfamiliar data, there is no guarantees that the performance level will be the same. This is known as over-training.

I would present the neural network with the training dataset and train for N epochs. I would then pause training and pass through the validation dataset to get an initial generalisation rate. Training would then resume with the training dataset for N epochs. Training would then be paused again, to get an updated generalisation rate. This process would continue until the generalisation rate begins to fall. This would indicate that the network has reached optimal performance. Once training has stopped completely, I would pass through the testing dataset to determine a final generalisation rate.

Determining the size of the hidden layer would begin with the lowest number of nodes, H, that seems likely to be a feasible solution. The 3 dataset MLP training regime as stated above, would then be used. This regime is then repeated with an extra hidden node each time. The entire process is only halted when the Test Dataset Generalisation Rate starts to fall. The best hidden layer size will have been determined as 'H - 1' nodes.

2.2 Evolutionary Algorithms Question

Part A

i) Optimization problems: can be described by several characteristics. The first of which is an objective function. This represents the quantity to be optimized and the quantity to me minimized and maximized. Some problems such as constraint-satisfaction problems, do not define an explicit function. Instead, the objective is to find a solution that satisfies a set of constraints. There is a set of unknown variables. These affect the value of the objective function.

We also have a set of constraints which restrict the values that can be assigned to the unknowns. Most problems define at least a set of boundary constraints which define the domain of values for each variable. Constraints can be more complex, excluding certain candidate solutions from being considered as solutions.

ii) Fitness functions: are used to determine the ability of an individual evolutionary algorithm to survive. It quantifies how good the solution represented by a chromosome is. The function maps chromosome representation into a scalar value.

Usually, a fitness function provides an absolute measure of fitness. However, for some problems, this is not possible. Instead, a relative fitness measure is used to quantify the performance of an individual in relation to that of other individuals in the population or a competing population. Relative fitness measures are used in co-evolutionary algorithms.

iii) A Genotype: describes the genetic composition of an individual, as inherited from its parents. It represents which allele the individual possesses. It is used to express characteristics of individuals in genetic algorithms when modelling genetic evolution.

iv) Representation: In evolutionary computation, each individual represents a candidate solution to an optimization problem. Organisms have certain characteristics that influence their ability to survive and reproduce. These are represented by long strings of information contained in the chromosome of an organism. These characteristics refer to variables of an optimization problem, for which optimal assignment is sought. Each variable that needs to be optimized is referred to as a gene, the smallest unit of information.

Part B

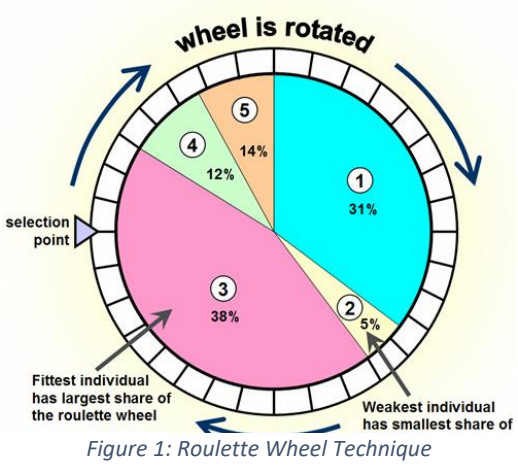


Figure 1: Roulette Wheel Technique

Roulette wheel selection is an example proportional selection operator where fitness values are normalised by dividing each fitness by the maximum fitness value. The size of each slice of the roulette wheel is proportional to the normalised selection probability of an individual. Selection can be represented by the spinning of a roulette wheel and recording which slice is landed on. The corresponding individual is then selected.

The wheel is then spun again and depending on which section is landed on, an individual from that

genome will be selected. The pair of individuals, also known as parents, will then reproduce, creating a child genome. It is important to note that parents get selected in proportion to their fitness rate and some genomes may be selected more than once. Not every one of the original genomes will necessarily survive this selection process.

Part C

Mutation

Mutation only requires 1 parent. This method of reproduction alters one or more gene values in a chromosome from its initial state. Due to this, the solution may change entirely from the previous solution. Mutation occurs according to user-definable mutation probability. This should be set low as if it is set too high, the method of reproduction will turn into a primitive random search and will take much longer to find the best solution possible.

Cross-over

Cross-over requires 2 or more parents. This method of reproduction takes the bit patterns of each of the parents and combines them. With predetermined probability, we can randomly pick a cut-point from amongst the genotype's array locations. This method of reproduction would produce a solution that is somewhere in the middle of the parents that have created it. Depending on the fitness of each parent, the new solution would be closer to that of the parent with the highest fitness rate.