University of St Andrews

Assessment 1

Artificial Neural Networks

Student ID: 120022067

University of St Andrews

CS5011 Artificial Intelligence Practice

Contents

List of Figures

List of Tables

Listings

1 Introduction

1.1 Overview

"Guess Who?" game is a two-player character guessing game where the players take turns to ask various yes/no questions to eliminate the candidates. The goal is to be the first to guess the character correctly.

The aim of this assignment is to create and train an artificial neural network, and use it to play a single player "Guess Who?" game. In this case, the user has to think of a character and an agent has to guess the character that the user thought of by asking a set of yes/no questions to the user.

1.2 Parts Implemented

Part 1: Implemented every feature stated in the requirements. The neural network was designed and trained using backpropagation algorithm. Full details of the implementation can be found in Section 3.

Part 2: For part 2, a new character and a new feature were added, and the network was retrained. An extended version of "Guess Who?" game was implemented with an option to use early guess feature. The network was also tested with patterns that do not exist in the training set as well. An in-depth discussion of part 2's implementation can be found in Section 4

Part 3: Resilient propagation and Manhattan propagation was used to train the network. Due to time constraints, it is not possible to implement other extensions stated in the requirements. The comparison and discussion of different training approaches can be found in Section 5.

1.3 Libraries

This section covers the libraries used in this assignment.

1.3.1 Encog

Encog is a machine learning framework developed by Heaton Research (Research 2017). It was used to construct and train the artificial neural network to play "Guess Who?" game.

1.3.2 Joinery

Joinery is a data frame implementation in Java developed by Bryan Cardillo (Cardillo 2017). This library was used to read and manipulate csv dataset.

2 Literature Review

2.1 Artificial Neural Networks

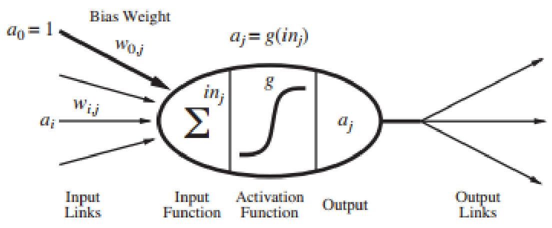


Figure 1: A simple mathematical model for a neuron (Russell & Norvig 2009).

Artificial neural network (ANN) is inspired by the functionality of human brains with numerous interconnected neurons (Russell & Norvig 2009). Russell & Norvig (2009) explains that an ANN is consisted of neurons (see Figure 1) connected by directed links. A link from neuron i to neuron j is used to propagate the activation function a_i from i to i (Russell & Norvig 2009). Each connection also contains an associated numeric weight w_{ij}. The weight indicates the strength and sign of the link (Russell & Norvig 2009). Similar to linear regression model, each neuron has a bias input $a_0 = 1$ with a corresponding weight w_{0j} (Russell & Norvig 2009). Russell & Norvig (2009) states that each neuron j has to first calculate a weighted sum of its input with the equation:

$$in_j = \sum_{i=0}^{n} w_{i,j} a_i \tag{1}$$

After that, an activation function g needs to be applied to the weighted sum in order to derive the output (Russell & Norvig 2009):

$$a_j = g(in_j) = g(\sum_{i=0}^{n} w_{i,j} a_i) \tag{2}$$

One of the most basic ANN architecture is the multilayer feed forward neural network. A typical multilayer feed forward neural network usually consists of an input layer, one or more hidden layers and an output layer (Russell & Norvig 2009). The input layer consists of neurons that represents the features or input data. Russell & Norvig (2009) explains that the input data passes through the neurons in the hidden layer and the weights and biases are used to these input values in order to produce a predicted output at the output layer. An illustration of a multilayer feed forward neural network is shown in Figure 2

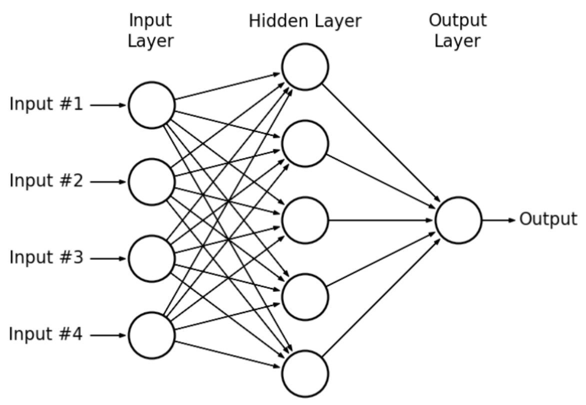
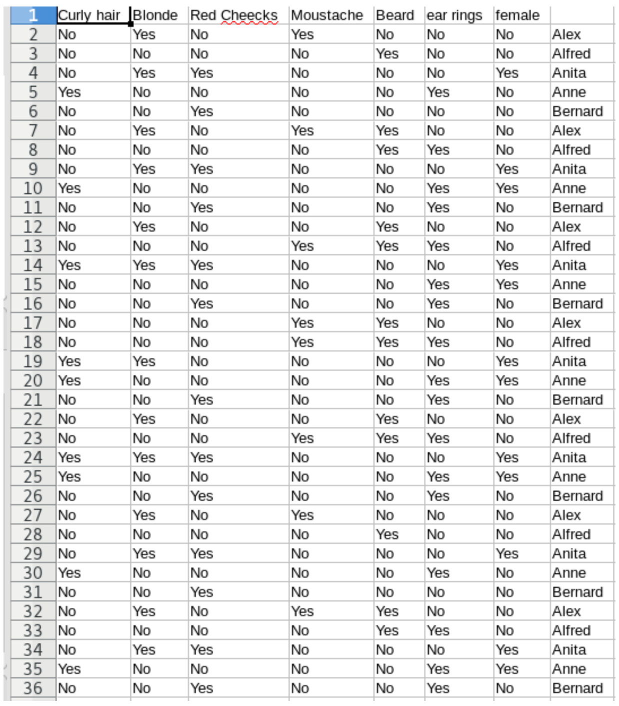


Figure 2: Multilayer feed forward neural network illustration (Vanderplas 2012).

3 Part 1

3.1 Design

3.1.1 Dataset



1	Curly hair	Blonde	Red Cheecks	Moustache	Beard	ear rings	female	
2	No	Yes	No	Yes	No	No	No	Alex
3	No	No	No	No	Yes	No	No	Alfred
4	No	Yes	Yes	No	No	No	Yes	Anita
5	Yes	No	No	No	No	Yes	No	Anne
6	No	No	Yes	No	No	No	No	Bernard
7	No	Yes	No	Yes	Yes	No	No	Alex
8	No	No	No	No	Yes	Yes	No	Alfred
9	No	Yes	Yes	No	No	No	Yes	Anita
10	Yes	No	No	No	No	Yes	Yes	Anne
11	No	No	Yes	No	No	Yes	No	Bernard
12	No	Yes	No	No	Yes	No	No	Alex
13	No	No	No	Yes	Yes	Yes	No	Alfred
14	Yes	Yes	Yes	No	No	No	Yes	Anita
15	No	No	No	No	No	Yes	Yes	Anne
16	No	No	Yes	No	No	Yes	No	Bernard
17	No	No	No	Yes	Yes	No	No	Alex
18	No	No	No	Yes	Yes	Yes	No	Alfred
19	Yes	Yes	No	No	No	No	Yes	Anita
20	Yes	No	No	No	No	Yes	Yes	Anne
21	No	No	Yes	No	No	Yes	No	Bernard
22	No	Yes	No	No	Yes	No	No	Alex
23	No	No	No	Yes	Yes	Yes	No	Alfred
24	Yes	Yes	Yes	No	No	No	Yes	Anita
25	Yes	No	No	No	No	Yes	Yes	Anne
26	No	No	Yes	No	No	Yes	No	Bernard
27	No	Yes	No	Yes	No	No	No	Alex
28	No	No	No	No	Yes	No	No	Alfred
29	No	Yes	Yes	No	No	No	Yes	Anita
30	Yes	No	No	No	No	Yes	No	Anne
31	No	No	Yes	No	No	No	No	Bernard
32	No	Yes	No	Yes	Yes	No	No	Alex
33	No	No	No	No	Yes	Yes	No	Alfred
34	No	Yes	Yes	No	No	No	Yes	Anita
35	Yes	No	No	No	No	Yes	Yes	Anne
36	No	No	Yes	No	No	Yes	No	Bernard

Figure 3: Dataset used to train the neural network.

The dataset was provided in csv format. As illustrated in Figure 3, each row represents a run of the game i.e. each row is a set of answers given by a user for a specific character. For this part, there are 5 characters to be identified, these characters are: Alex, Alfred, Anita, Anne, and Bernard. The main objective of this part is to create a neural network that is able to learn from previous game runs. The dataset does not have random answers, but there are some mistakes. The neural network should be able to guess the character despite the fact that the user made some mistakes when answering questions.

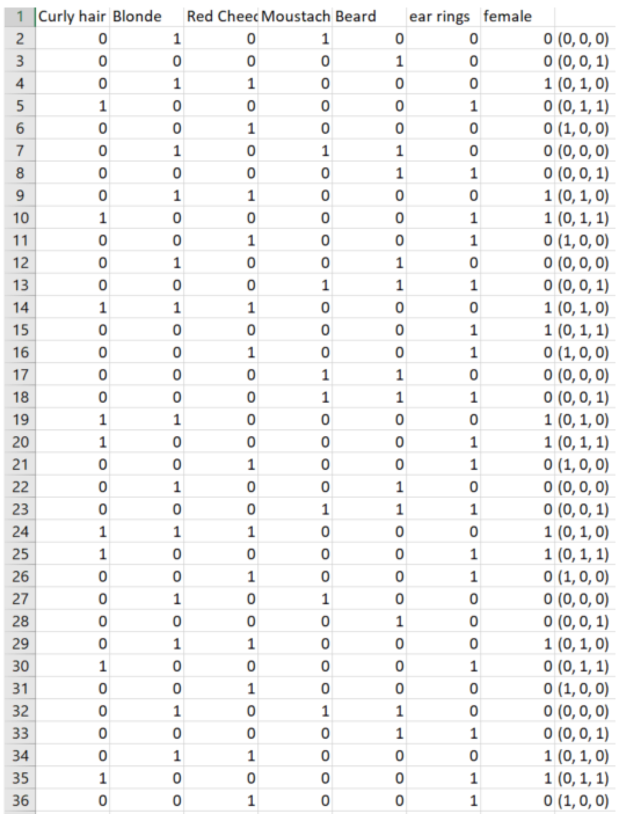
3.1.2 Input/Output Encoding

Raw input	Encoded input
Yes	1
No	0

Table 1: Encoding of input from the dataset

Raw output	Encoded output
Alex	(0, 0, 0)
Alfred	(0, 0, 1)
Anita	(0, 1, 0)
Anne	(0, 1, 1)
Bernard	(1, 0, 0)

Table 2: Encoding of output from the dataset



1	Curly hair	Blonde	Red Cheec	Moustach	Beard	ear rings	female	
2	0	1	0	1	0	0	0	(0, 0, 0)
3	0	0	0	0	1	0	0	(0, 0, 1)
4	0	1	1	0	0	0	1	(0, 1, 0)
5	1	0	0	0	0	1	0	(0, 1, 1)
6	0	0	1	0	0	0	0	(1, 0, 0)
7	0	1	0	1	1	0	0	(0, 0, 0)
8	0	0	0	0	1	1	0	(0, 0, 1)
9	0	1	1	0	0	0	1	(0, 1, 0)
10	1	0	0	0	0	1	1	(0, 1, 1)
11	0	0	1	0	0	1	0	(1, 0, 0)
12	0	1	0	0	1	0	0	(0, 0, 0)
13	0	0	0	1	1	1	0	(0, 0, 1)
14	1	1	1	0	0	0	1	(0, 1, 0)
15	0	0	0	0	0	1	1	(0, 1, 1)
16	0	0	1	0	0	1	0	(1, 0, 0)
17	0	0	0	1	1	0	0	(0, 0, 0)
18	0	0	0	1	1	1	0	(0, 0, 1)
19	1	1	0	0	0	0	1	(0, 1, 0)
20	1	0	0	0	0	1	1	(0, 1, 1)
21	0	0	1	0	0	1	0	(1, 0, 0)
22	0	1	0	0	1	0	0	(0, 0, 0)
23	0	0	0	1	1	1	0	(0, 0, 1)
24	1	1	1	0	0	0	1	(0, 1, 0)
25	1	0	0	0	0	1	1	(0, 1, 1)
26	0	0	1	0	0	1	0	(1, 0, 0)
27	0	1	0	1	0	0	0	(0, 0, 0)
28	0	0	0	0	1	0	0	(0, 0, 1)
29	0	1	1	0	0	0	1	(0, 1, 0)
30	1	0	0	0	0	1	0	(0, 1, 1)
31	0	0	1	0	0	0	0	(1, 0, 0)
32	0	1	0	1	1	0	0	(0, 0, 0)
33	0	0	0	0	1	1	0	(0, 0, 1)
34	0	1	1	0	0	0	1	(0, 1, 0)
35	1	0	0	0	0	1	1	(0, 1, 1)
36	0	0	1	0	0	1	0	(1, 0, 0)

Figure 4: Final training table.

As shown in Table 1, the input fields are encoded with One-hot encoding. This is because there are only two possible answers (”Yes” or ”No”), the One-hot encoding is sufficient.

Table 2 illustrates that the output fields are encoded with binary encoding approach. This method was chosen so that the number of output units of the neural network can be minimized. In this case, the neural network requires 3 output units. The complete training table can be seen in Figure 4.

3.1.3 Artificial Neural Network Construction

The network used in this assignment is a multilayer feedforward neural network, with one hidden layer, and sigmoid activation function. In the dataset, there are 7 features. Therefore, the network will contain 7 input units. As mentioned in Section 3.1.2, 3 output units are required because the binary encoding was used.

Hidden units	Number of epochs (avg)
2	132.3
3	60.6
4	37.3
5	36.3
6	32.0
7	28.3
8	29.3
9	25.3
10	28.7
11	26
12	30
13	35
14	35.66
15	28.33

Table 3: The number of hidden units and the average number of epochs required to train the network.

```
input units = 7 + 1 bias
hidden units = [2,15] (1 bias)
output units = 3
learning rate = 0.5
momentum = 0
```

Listing 1: Parameters used to experiment with number of hidden units

To complete the construction of the network, the number of hidden units needs to be decided. The method used to find a suitable number of hidden units was to train the network 10 times on each number of hidden units and record the average number of epochs required to train the network (see Table 3). The parameters used in this test are shown in Listing 1.

The learning of 0.5 was chosen because it is not too big or too small, which means that the network should not take too long to finish training. The number of hidden units starts from 2 in this test because the network never converge when 1 hidden unit was used. The number of hidden units that requires minimum number of epochs was chosen. In this case, the chosen number of hidden units is 9. It is also worth noting that error was not taken into account when experimenting with different network structures and training parameters. This is because the training will terminate only when the error is lower than or equal to 0.01 (1%), which means that the network is considered acceptable as long as the error is less than or equal to 1%.

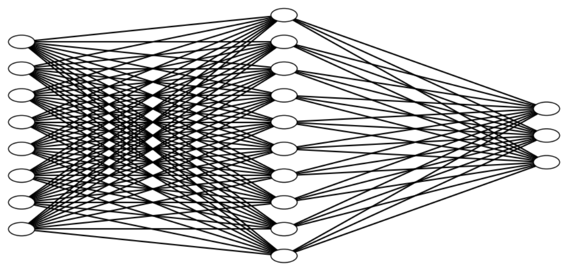


Figure 5: Final neural network structure for part 1.

The final neural network structure for part 1 is shown in Figure 5. It consists of 7 input units, 9 hidden units, and 3 output units (with 1 bias at input layer and hidden layer).

3.1.4 Training Parameters

Learning rate	Number of epochs (avg)
0.2	64.4
0.3	44.5
0.4	33.4
0.5	27.1
0.6	24.6
0.7	22.3
0.8	29.1
0.9	87.5
1.0	372.3

Table 4: Learning rates and the average number of epochs required to train the network.

```
input units = 7 + 1 bias
hidden units = 9 + 1 bias
output units = 3
learning rate = [0.2,1.0]
momentum = 0
```

Listing 2: Parameters used to experiment with learning rate

Before the network can be trained using backpropagation algorithm, the appropriate parameters like learning rate and momentum would need to be set. Similar to the method used to select the appropriate hidden units, the parameters used in this test are shown in Listing 2.

The network was trained 10 times on each learning rate. The average number of epochs required to train the network are recorded for each learning rate (see Table 4). Learning rate in this test starts from 0.2 because lower learning rate took too long to converge, which mean that they are not suitable for this system. The learning rate that requires minimum number of epochs was chosen. In this case, the chosen number of learning rate is 0.7.

Momentum	Number of epochs (avg)
0.1	69.5
0.2	42
0.3	48.8
0.4	433.4
0.5	41.9

Table 5: Momentums and the average number of epochs required to train the network.

```
input units = 7 + 1 bias
hidden units = 9 + 1 bias
output units = 3
learning rate = 0.7
momentum = [0.1,0.5]
```

Listing 3: Parameters used to experiment with momentum

The last parameter is momentum. The parameters used in the momentum test are shown in Listing 3. The network was trained 10 times on each momentum value. The average number of epochs required to train the network are recorded for each momentum value. As shown in Table 5, using momentum generally results in longer training time. From the experiment, It is clear that momentum should not be used with the current network. The summary of network structure and training parameters can be found in Table 6.

Input units	7 + 1 bias
Hidden units	9 + 1 bias
Output units	3
Learning rate	0.7
Momentum	0

Table 6: Final network structure and training parameters for part 1

3.1.5 Neural Network Training

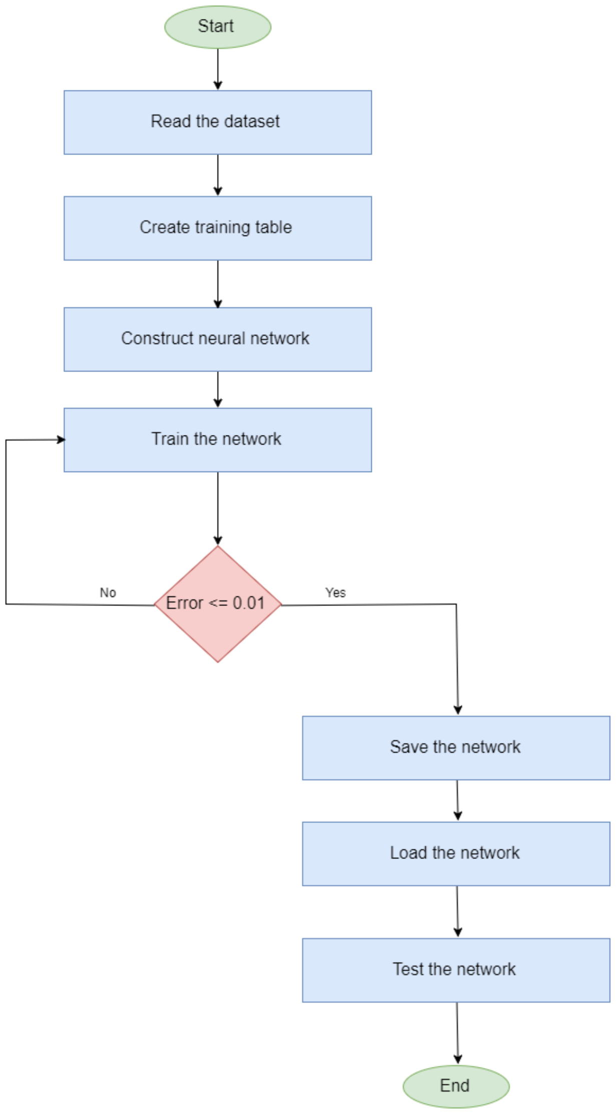


Figure 6: Flowchart showing the execution cycle of part 1.

As illustrated in Figure 6, the program reads in the csv dataset and construct the training table. Then, it creates the neural network, which has the structure that was discussed in Section 3.1.3. After that, the network is trained with the chosen parameters (see Section 3.1.4) until the error is less than or equal to 0.01 (1%). Once the training stops, the network is saved to a file. Finally, the program loads the trained network and tests it against training data.

3.1.6 Network Output Interpretation

```
private static Map<String, double[]> createCharMap(){
  // maps characters' name to the binary arrays that represent them
  Map<String, double[]> charMap = new HashMap<String, double[]>();
  double[] Alex = {0, 0, 0};
  double[] Alfred = {0, 0, 1};
  double[] Anita = {0, 1, 0};
  double[] Anne = {0, 1, 1};
  double[] Bernard = {1, 0, 0};
  double[] Cercei = {1, 0, 1};

    charMap.put("Alex", Alex);
    charMap.put("Alfred", Alfred);
    charMap.put("Anita", Anita);
    charMap.put("Anne", Anne);
    charMap.put("Bernard", Bernard);
    charMap.put("Cercei", Cercei);

    return charMap;
}
```

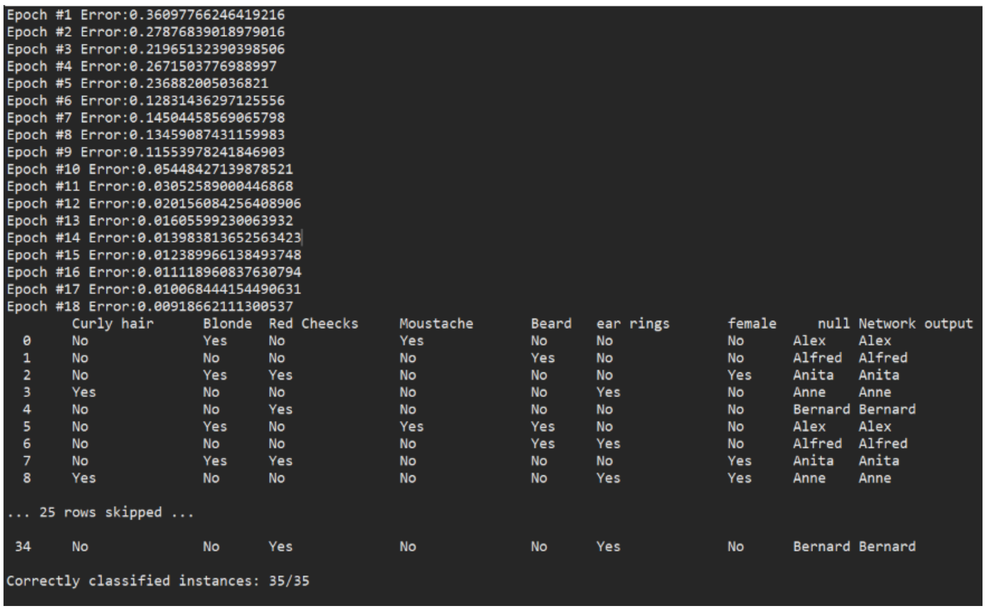
Listing 4: Mapping of characters to the corresponding binary encoding

```
output: [0.2, 0.1, 0.9] -> [0, 0, 1] -> Alfred
```

Listing 5: An example of network output interpretation

With current network structure, the output will be in the form of 3 double values. The threshold of 0.5 is used to decide if the output should equal to 0 or 1. If the output is grater than of equal to 0.5, then the it is assigned a value 1. Otherwise, it gets the value 0. Once the values are assigned, Java's HashMap is used to map the network output to a character (see Listing 4). An example of network output interpretation can be found in Listing 5.

3.2 Examples and Testing



```
Epoch #1 Error:0.36097766246419216
Epoch #2 Error:0.27876839018979016
Epoch #3 Error:0.21965132390398506
Epoch #4 Error:0.2671503776988997
Epoch #5 Error:0.236882005036821
Epoch #6 Error:0.12831436297125556
Epoch #7 Error:0.14504458569065798
Epoch #8 Error:0.13459087431159983
Epoch #9 Error:0.11553978241846903
Epoch #10 Error:0.05448427139878521
Epoch #11 Error:0.03052589000446868
Epoch #12 Error:0.020156084256408906
Epoch #13 Error:0.01605599230063932
Epoch #14 Error:0.013983813652563423
Epoch #15 Error:0.012389966138493748
Epoch #16 Error:0.011118960837630794
Epoch #17 Error:0.010068444154490631
Epoch #18 Error:0.00918662111300537
       Curly hair      Blonde  Red Cheecks     Moustache       Beard   ear rings       female     null Network output
  0    No              Yes     No              Yes             No      No              No      Alex    Alex
  1    No              No      No              No              Yes     No              No      Alfred  Alfred
  2    No              Yes     Yes             No              No      No              Yes     Anita   Anita
  3    Yes             No      No              No              No      Yes             No      Anne    Anne
  4    No              No      Yes             No              No      No              No      Bernard Bernard
  5    No              Yes     No              Yes             Yes     No              No      Alex    Alex
  6    No              No      No              No              Yes     Yes             No      Alfred  Alfred
  7    No              Yes     Yes             No              No      No              Yes     Anita   Anita
  8    Yes             No      No              No              No      Yes             Yes     Anne    Anne

... 25 rows skipped ...

 34    No              No      Yes             No              No      Yes             No      Bernard Bernard

Correctly classified instances: 35/35
```

Figure 7: Printout from the execution of part 1 program.

An example of printout from the execution of part 1 program is shown in Figure 7. The program displays the error at each iteration, so that the training progress can be monitored. Once the training stops, the network is tested against the training data, and the classification results are printed out as well.

Epochs (avg)	25.9
Error (avg)	0.009517758
Correctly classified characters (avg)	100%

Table 7: Summary of network performance after training the network 10 times

The summary of network performance after training the network 10 times can be found in Table 7. The results indicates that the network is performing at satisfactory level because it can be trained in a short period of time, while having small error. The network can also guess the characters correctly 100% of the time as well.

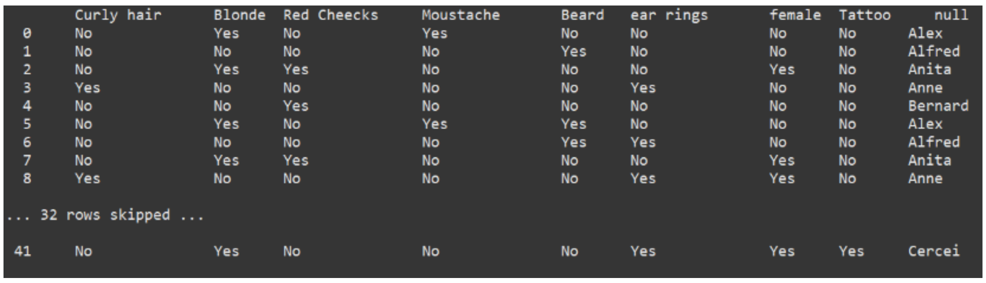
3.3 Evaluation

Overall, the network is proven to be highly accurate at guessing the characters in the current version of "Guess Who?" game. When tested against the training data, the network was able to guess every character correctly. Due to time constraints, it is not possible to test the network with more configurations. There might be different number of hidden units, learning rate, or momentum that can further improve the performance of the network. One possible extension is to modify the training algorithm to adjust the learning rate during training e.g. use simulated annealing.

4 Part 2

4.1 Design

4.1.1 Dataset



```
      Curly hair      Blonde  Red Cheecks     Moustache       Beard   ear rings       female  Tattoo     null
 0    No              Yes     No              Yes             No      No              No      No      Alex
 1    No              No      No              No              Yes     No              No      No      Alfred
 2    No              Yes     Yes             No              No      No              Yes     No      Anita
 3    Yes             No      No              No              No      Yes             No      No      Anne
 4    No              No      Yes             No              No      No              No      No      Bernard
 5    No              Yes     No              Yes             Yes     No              No      No      Alex
 6    No              No      No              No              Yes     Yes             No      No      Alfred
 7    No              Yes     Yes             No              No      No              Yes     No      Anita
 8    Yes             No      No              No              No      Yes             Yes     No      Anne

... 32 rows skipped ...

41    No              Yes     No              No              No      Yes             Yes     Yes     Cercei
```

Figure 8: Updated training table for extended "Guess Who?" game.

Character	Character encoding
Alex	(0, 0, 0)
Alfred	(0, 0, 1)
Anita	(0, 1, 0)
Anne	(0, 1, 1)
Bernard	(1, 0, 0)
Cercei	(1, 0, 1)

Table 8: Character encoding for extended "Guess Who?" game

For this part, the training data has been updated to accommodate a new feature "Tattoo" and a new character "Cercei". An updated character mapping can be found in Table 8. In the updated training table, the only character that has answer "Yes" in the Tattoo column is the new character Cercei. The printout of the updated dataset is shown in Figure 8.

4.1.2 Neural Network

Epochs (avg)	25.9
Error (avg)	0.009391446
Correctly classified characters (avg)	100%

Table 9: Summary of network performance after training the network against the updated training set 10 times

For extended "Guess Who?" game, the network was trained with the same configurations and parameters because the overall performance of the network is still at satisfactory level. As shown in Table 9, the performance is not very different from when it was trained with the initial dataset.

4.1.3 Extended "Guess Who?" Game

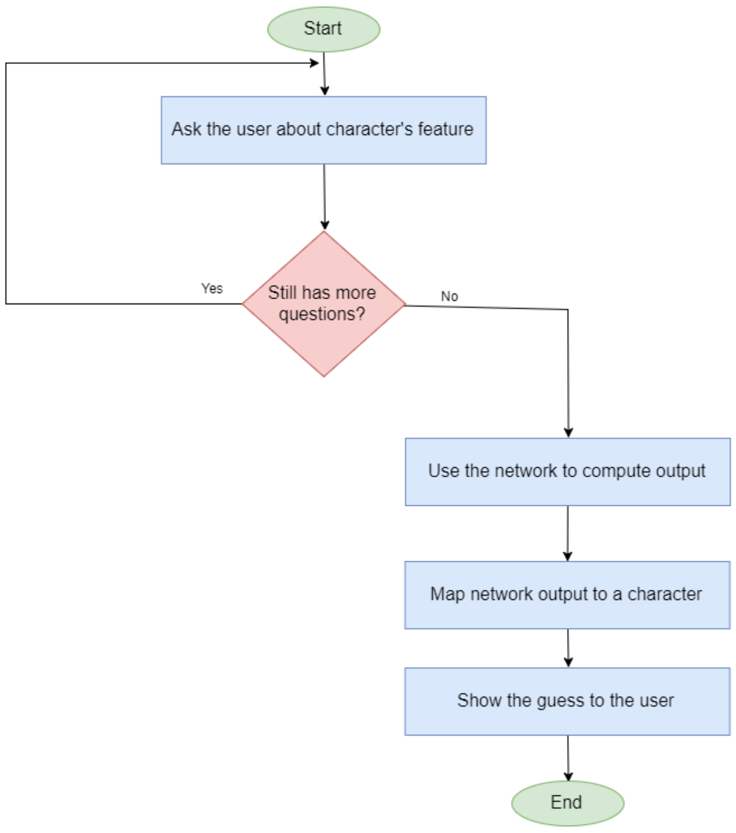

Start
Ask the user about character's feature
Still has more questions?
Yes
No
Use the network to compute output
Map network output to a character
Show the guess to the user
End

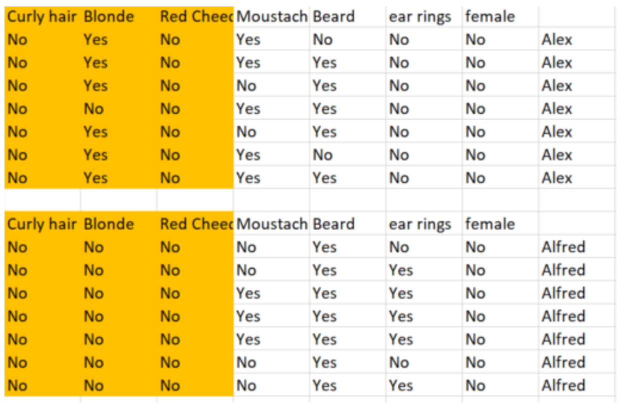
Figure 9: Flowchart showing a run of extended "Guess Who?" game.

```java
private static Map<String, String> createFeatureMap(){
    // maps feature to corresponding question
    Map<String, String> FEATURE_MAP = new LinkedHashMap<String, String>();
    FEATURE_MAP.put("Curly hair", "Does your character have curly hair?");
    FEATURE_MAP.put("Blond", "Is your character blond?");
    FEATURE_MAP.put("Red Cheecks", "Does your character have red cheeks?");
    FEATURE_MAP.put("Moustache", "Does your character have a moustache?");
    FEATURE_MAP.put("Beard", "Does your character have a beard?");
    FEATURE_MAP.put("ear rings", "Does your character wear ear rings?");
    FEATURE_MAP.put("female", "Is your character female?");
    FEATURE_MAP.put("Tattoo", "Does your character have a tattoo?");

    return FEATURE_MAP;
  }
```

Listing 6: Mapping of features to the corresponding questions

Figure 9 demonstrates a run of extended "Guess Who?" game. First, the agent asks a set of questions corresponding to the features of the dataset. The mapping of features and questions is shown in Listing 6. Once the agent asked all of the questions, it uses the network from part 1 to compute the output and maps the output to a character. Finally, the agent shows the guess to the user.

4.1.4 Extended "Guess Who?" Game with Early Guess



Curly hair	Blonde	Red Cheec	Moustach	Beard	ear rings	female	
No	Yes	No	Yes	No	No	No	Alex
No	Yes	No	Yes	Yes	No	No	Alex
No	Yes	No	No	Yes	No	No	Alex
No	No	No	Yes	Yes	No	No	Alex
No	Yes	No	No	Yes	No	No	Alex
No	Yes	No	Yes	No	No	No	Alex
No	Yes	No	Yes	Yes	No	No	Alex

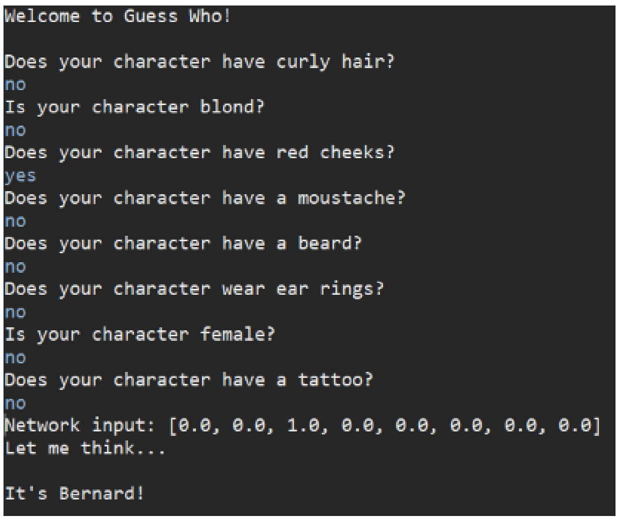
Curly hair	Blonde	Red Cheec	Moustach	Beard	ear rings	female	
No	No	No	No	Yes	No	No	Alfred
No	No	No	No	Yes	Yes	No	Alfred
No	No	No	Yes	Yes	Yes	No	Alfred
No	No	No	Yes	Yes	Yes	No	Alfred
No	No	No	Yes	Yes	Yes	No	Alfred
No	No	No	No	Yes	No	No	Alfred
No	No	No	No	Yes	Yes	No	Alfred

Figure 10: Patterns in the first three features.

Early guess feature is also implemented. When this feature is enabled, the agent will ask the user 3 questions, then assume that the answers to the rest of the questions are "No" and attempt to make an early guess. Based on the information from Figure 10, knowing the first three features should be enough for the agent to guess the character, since they contain the same answers for each character. Due to time constraints, it is not possible to implement a more complex early guess feature.

4.2 Examples and Testing

4.2.1 Extended "Guess Who?" Game

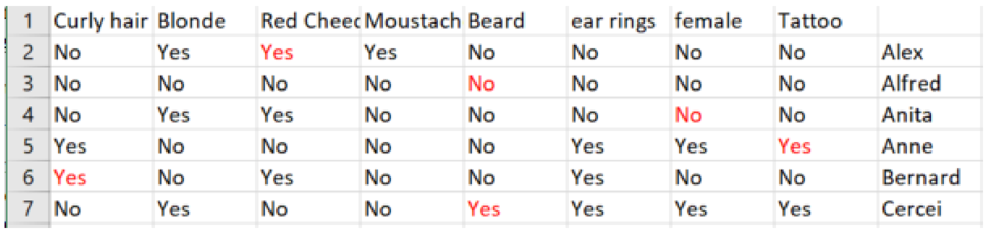


```
Welcome to Guess Who!

Does your character have curly hair?
no
Is your character blond?
no
Does your character have red cheeks?
yes
Does your character have a moustache?
no
Does your character have a beard?
no
Does your character wear ear rings?
no
Is your character female?
no
Does your character have a tattoo?
no
Network input: [0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Let me think...

It's Bernard!
```

Figure 11: Printout from a run of "Guess Who?" game.

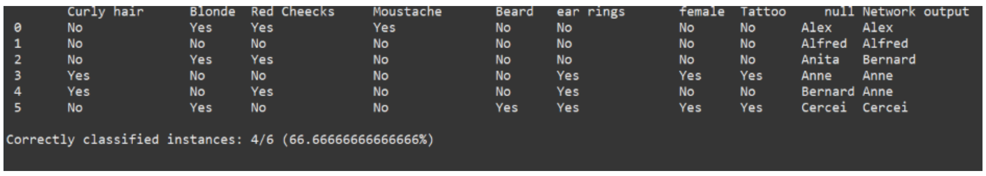
The extended version of the game itself is performing according to the requirement. As shown in Figure 11, the agent asks questions and the user responses with "yes" or "no" (not case-sensitive). At the end, the agent shows the guess to the user as stated in the requirement.

4.2.2 Unseen Pattern



1	Curly hair	Blonde	Red Cheec	Moustach	Beard	ear rings	female	Tattoo	
2	No	Yes	Yes	Yes	No	No	No	No	Alex
3	No	No	No	No	No	No	No	No	Alfred
4	No	Yes	Yes	No	No	No	No	No	Anita
5	Yes	No	No	No	No	Yes	Yes	Yes	Anne
6	Yes	No	Yes	No	No	Yes	No	No	Bernard
7	No	Yes	No	No	Yes	Yes	Yes	Yes	Cercei

Figure 12: Minor changes to create pattern that does not exist in the training data.

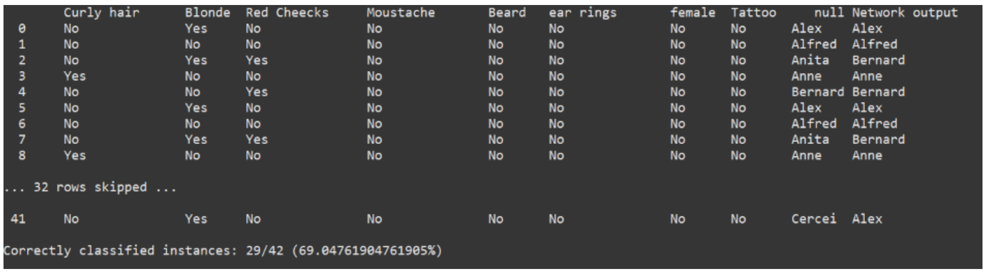


```
       Curly hair      Blonde  Red Cheecks     Moustache       Beard   ear rings       female  Tattoo     null Network output
0      No              Yes     Yes             Yes             No      No              No      No      Alex    Alex
1      No              No      No              No              No      No              No      No      Alfred  Alfred
2      No              Yes     Yes             No              No      No              No      No      Anita   Bernard
3      Yes             No      No              No              No      Yes             Yes     Yes     Anne    Anne
4      Yes             No      Yes             No              No      Yes             No      No      Bernard Anne
5      No              Yes     No              No              Yes     Yes             Yes     Yes     Cercei  Cercei

Correctly classified instances: 4/6 (66.66666666666666%)
```

Figure 13: Network's classification with pattern that does not exist in the training set.

The network was tested against pattern that does not exist in the training set as well. The first six entries from the original dataset was copied over to a separate dataset, and minor changed has been made in order to create ”unseen pattern”. The changes are highlighted in red as shown in Figure 12. As demonstrated in Figure 13, the network managed to classify 4 out of 6 characters (66.6%) correctly. Due to time constraints, it is not possible to test the network against all possible combinations of unseen pattern.

4.2.3 Early Guess Feature



```
       Curly hair      Blonde  Red Cheecks     Moustache       Beard   ear rings       female  Tattoo     null Network output
 0     No              Yes     No              No              No      No              No      No      Alex    Alex
 1     No              No      No              No              No      No              No      No      Alfred  Alfred
 2     No              Yes     Yes             No              No      No              No      No      Anita   Bernard
 3     Yes             No      No              No              No      No              No      No      Anne    Anne
 4     No              No      Yes             No              No      No              No      No      Bernard Bernard
 5     No              Yes     No              No              No      No              No      No      Alex    Alex
 6     No              No      No              No              No      No              No      No      Alfred  Alfred
 7     No              Yes     Yes             No              No      No              No      No      Anita   Bernard
 8     Yes             No      No              No              No      No              No      No      Anne    Anne

... 32 rows skipped ...

 41    No              Yes     No              No              No      No              No      No      Cercei  Alex

Correctly classified instances: 29/42 (69.04761904761905%)
```

Figure 14: Network’s classification with early guess enabled.

The early guess feature was tested by copying the content of the original over to a separate dataset and changing the answer from the fourth column onward to ”No”. This allows the simulation of the situation discussed in Section 4.1.4. As shown in Figure 14, the network managed to classify 29 out of 42 characters (69%) correctly.

4.3 Evaluation

In general, the extended ”Guess Who?” performs according to the requirements. The agent asks a number of questions and shows the guess to the user at the end as expected. Based on the test results from Section 4.2.2, it is clear that the network can handle unseen pattern better than expected. With larger training data, it should be possible for the network to improve the handling of unseen pattern even more. The network also has good performance when it comes to making early guesses. This also exceeded expectation because the implementation of early guess feature is very simple due to time constraints. It should be possible to improve the early guess feature by identifying the features that can eliminate for candidates, and have the agent ask those questions first in order to narrow down possible characters.

5 Part 3

5.1 Design

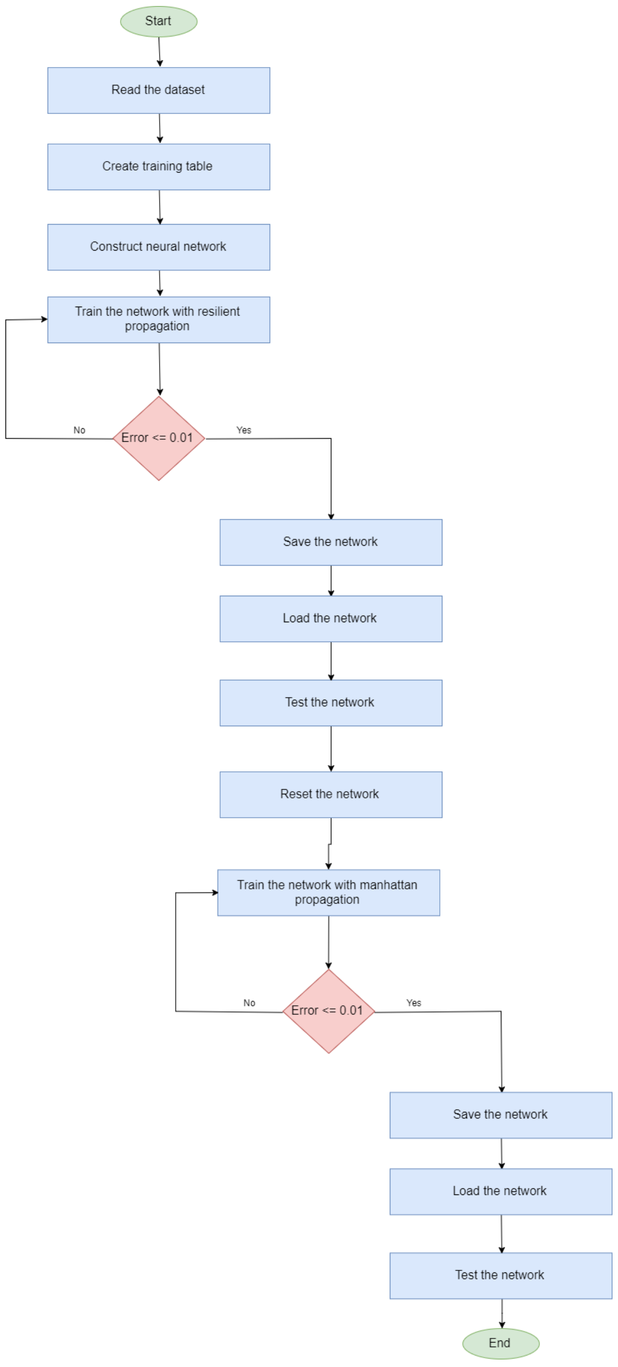


Figure 15: Flowchart showing neural network training in part 3.

As shown in Figure 15, the network was trained using resilient propagation and manhattan propagation algorithms. Similar to part 2, the network structure and training parameters are left unchanged because there is no big different in network performance when changing the parameters.

5.2 Examples and Testing

	Backpropagation	Resilient propagation	Manhattan propagation
Epochs (avg)	25.9	15.9	173.5
Error (avg)	0.009391446	0.007833748	0.008258585
Correctly classified characters (avg)	100%	100%	90.71428571%

Table 10: Summary of network performance after training the network with different training algorithms against the training set 10 times

The summary of network performance after training the network with different training algorithms against the training set 10 times can be found in Table 10.

5.3 Evaluation

Overall, the network has similar performance when it was trained with different algorithms. As illustrated in Table 10, resilient algorithm seems to slightly better the network performance than backpropagation (less epochs and error). On the other hand, Manhattan requires more epochs to train the network, but still has similar error. However, it cannot guess all of the characters correctly, which indicates that Manhattan is not suitable for this dataset. Due to time constraints, it is not possible to do more extensive tests on the two algorithms. There might be minor changes in parameters that can result in better network performance.

6 Running

6.1 Learning 1

Learning1.jar can be executed with the command:

```
java -jar Learning1.jar dataset_file
```

Listing 7: The command used to run Learning1.jar

To run Learning1.jar from AI-Practice-A1 folder, the command in Listing 8 should be used.

```
java -jar Learning1.jar char.csv
```

Listing 8: The command used to run Learning1.jar from AI-Practice-A1 folder

6.2 Learning 2

Learning2.jar can be executed with the command:

```
java -jar Learning2.jar <early_guess>
```

Listing 9: The command used to run Learning2.jar

For this part, early guess argument is optional. To run Learning2.jar from AI-Practice-A1 folder, the commands in Listing 10 should be used.

```
Running the game in default mode: java -jar Learning2.jar
Running the game with early guess enabled: java -jar Learning2.jar E
```

Listing 10: The command used to run Learning1.jar from AI-Practice-A1 folder

6.3 Learning 3

Learning3.jar can be executed with the command:

```
java -jar Learning3.jar dataset_file
```

Listing 11: The command used to run Learning3.jar

To run Learning3.jar from AI-Practice-A1 folder, the command in Listing 12 should be used.

```
java -jar Learning3.jar char.csv
```

Listing 12: The command used to run Learning3.jar from AI-Practice-A1 folder

Bibliography

Cardillo, B. (2017), 'Joinery — data frames for java', https://cardillo.github.io/joinery/v1.8/api/reference/joinery/DataFrame.html. Accessed: 2017-10-07.

Research, H. (2017), 'Encog machine learning framework', http://www.heatonresearch.com/encog/. Accessed: 2017-10-07.

Russell, S. J. & Norvig, P. (2009), *Artificial Intelligence: A Modern Approach*, 3 edn, Pearson Education.

Vanderplas, J. (2012), 'Neural network diagram', http://www.astroml.org/book_figures/appendix/fig_neural_network.html. Accessed: 2017-10-09.