CS5011: A4- Artificial Neural Networks- A ticketing routing agent - 160020220

Introduction

In this practical, the aim was to construct and use an artificial neural network applied in a help desk ticketing system. This task would develop my understanding of artificial neural network systems for multiclass classification problems.

3 main sections compose this practical.

The basic task revolves around creating a neural network using the Encog library in java. Then train this network using backpropagation to finally evaluate and validate the network.

The intermediate task is a user text interface where the correct team for as specific ticket would be retrieved by the network above. A user could also load a new ticket to the training set that would be then used to train the network again.

The Extension attempted here is a test between the backpropagation algorithm and the resilient propagation algorithm. The purpose of this extension is to understand how resilient propagation works (report) and to compare it with backpropagation (code).

Design and Implementation

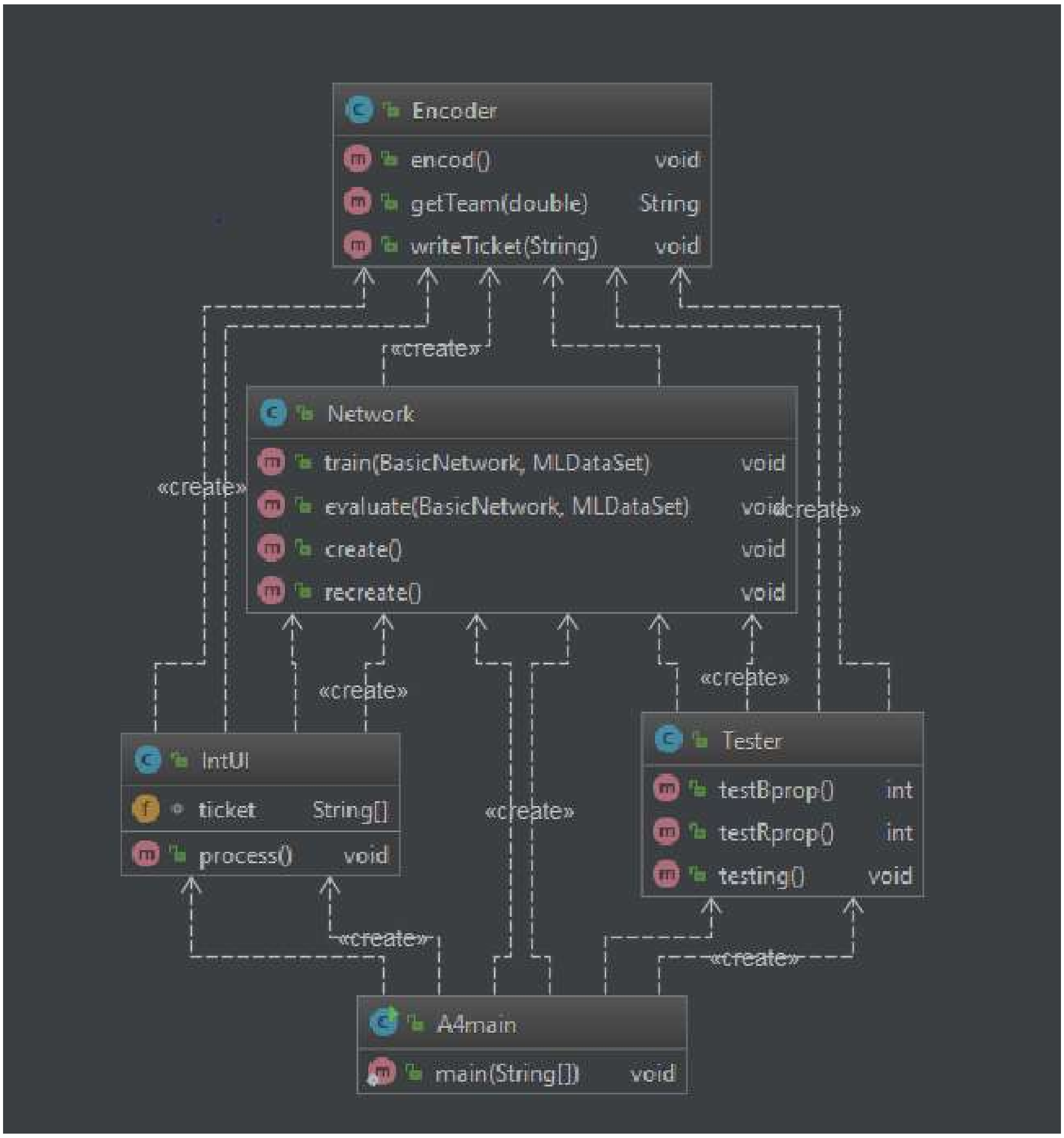
Running

To run the code, go to the A4src folder in the submission and run in the terminal:

```
Javac Main.java             to compile

java Main <Bas|Int|Adv>          to run
```

Code structure

The code structure of the submission is as follows:



Basic

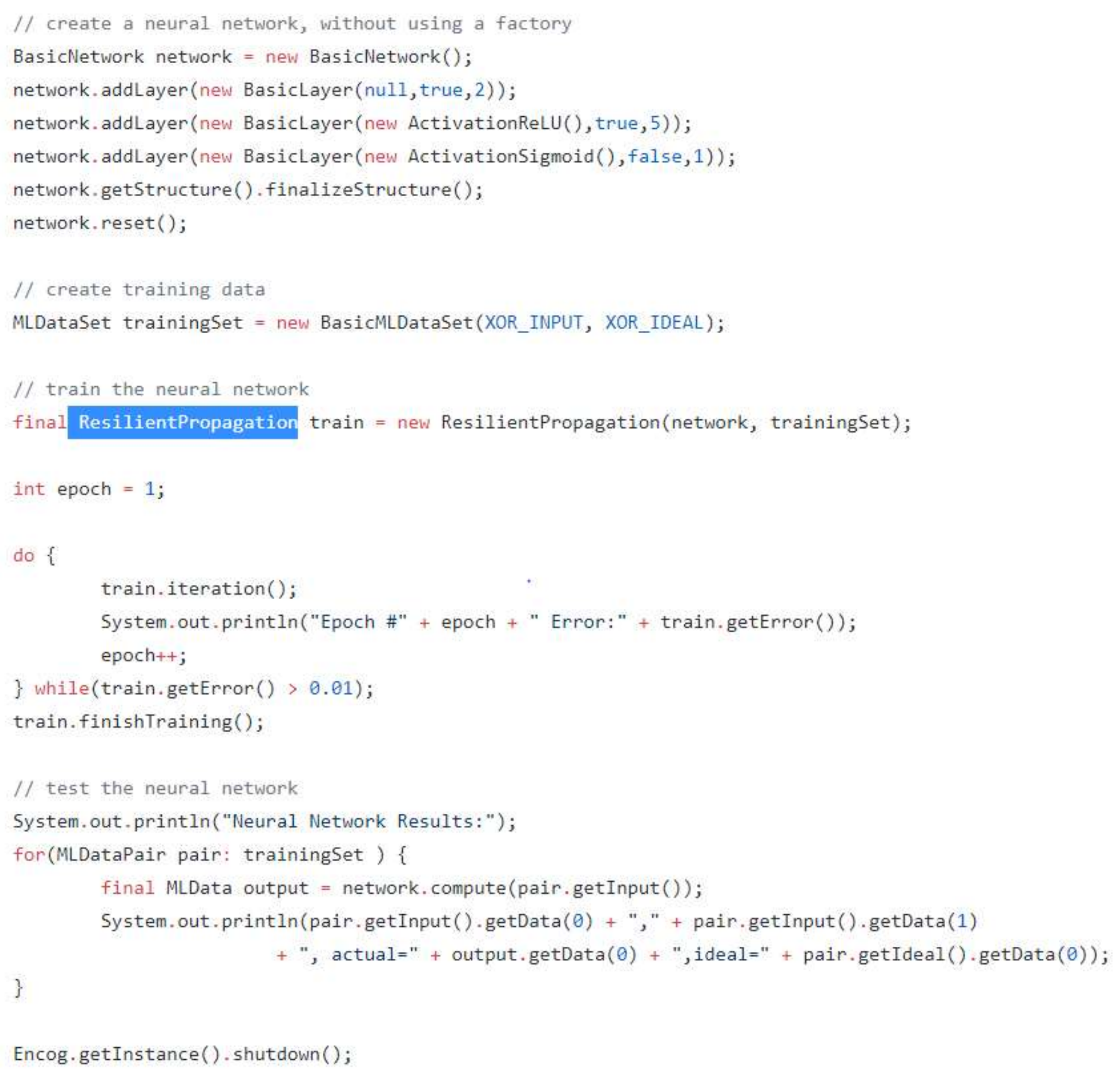
Encoding

The first thing we must do is encode inputs and outputs for the given dataset. To do this, I use the Encoder class which would take the csv ticket file and creates the encoded file with the new values.

Yes become 1, No becomes 0 and each team gets assigned a number between 0 and 1.

Network

To create the network, I inspired myself from the many example available in the Encog library, I started with the following piece of code:



```
// create a neural network, without using a factory
BasicNetwork network = new BasicNetwork();
network.addLayer(new BasicLayer(null,true,2));
network.addLayer(new BasicLayer(new ActivationReLU(),true,5));
network.addLayer(new BasicLayer(new ActivationSigmoid(),false,1));
network.getStructure().finalizeStructure();
network.reset();

// create training data
MLDataSet trainingSet = new BasicMLDataSet(XOR_INPUT, XOR_IDEAL);

// train the neural network
final ResilientPropagation train = new ResilientPropagation(network, trainingSet);

int epoch = 1;

do {
        train.iteration();
        System.out.println("Epoch #" + epoch + " Error:" + train.getError());
        epoch++;
} while(train.getError() > 0.01);
train.finishTraining();

// test the neural network
System.out.println("Neural Network Results:");
for(MLDataPair pair: trainingSet ) {
        final MLData output = network.compute(pair.getInput());
        System.out.println(pair.getInput().getData(0) + "," + pair.getInput().getData(1)
                        + ", actual=" + output.getData(0) + ",ideal=" + pair.getIdeal().getData(0));
}

Encog.getInstance().shutdown();
```

Reference: https://github.com/jeffheaton/encog-java-examples/blob/master/src/main/java/org/encog/examples/neural/xor/XORHelloWorld.java

Starting from this basis, I set my network input size to 9 as we have 9 tags. Then, I have to choose how many neurons I want.

However, as we know from the lectures, the neuron amount would probably be in this case

Input neurons + 1 bias neuron = total of 10 neurons.

I tested the network for 9 – 10 – 11 – … - 17 neurons and found that 10 neurons were indeed the best amount as I got the best results.

Furthermore, I would use the backpropagation algorithm to train the network. Train it by using iteration as in the example above. Then test the network by computing the data and comparing the output we got with the ideal output

Back propagation

The backpropagation algorithm consists of four steps:

1. Feed-forward computation

2. Backpropagation to output layer

3. Backpropagation to hidden layer(s)

4. Weight updating

In the first step, the algorithm is processed in a straightforward manner, with the output from one node being used as the input to the next node.

Backpropagation retraces through the network in reverse. Since the network is being run backwards, we evaluate using the left side of the node (the derivative). Instead of outputs being used as the inputs to the next node, outputs from a node are multiplied by the output of previous nodes.

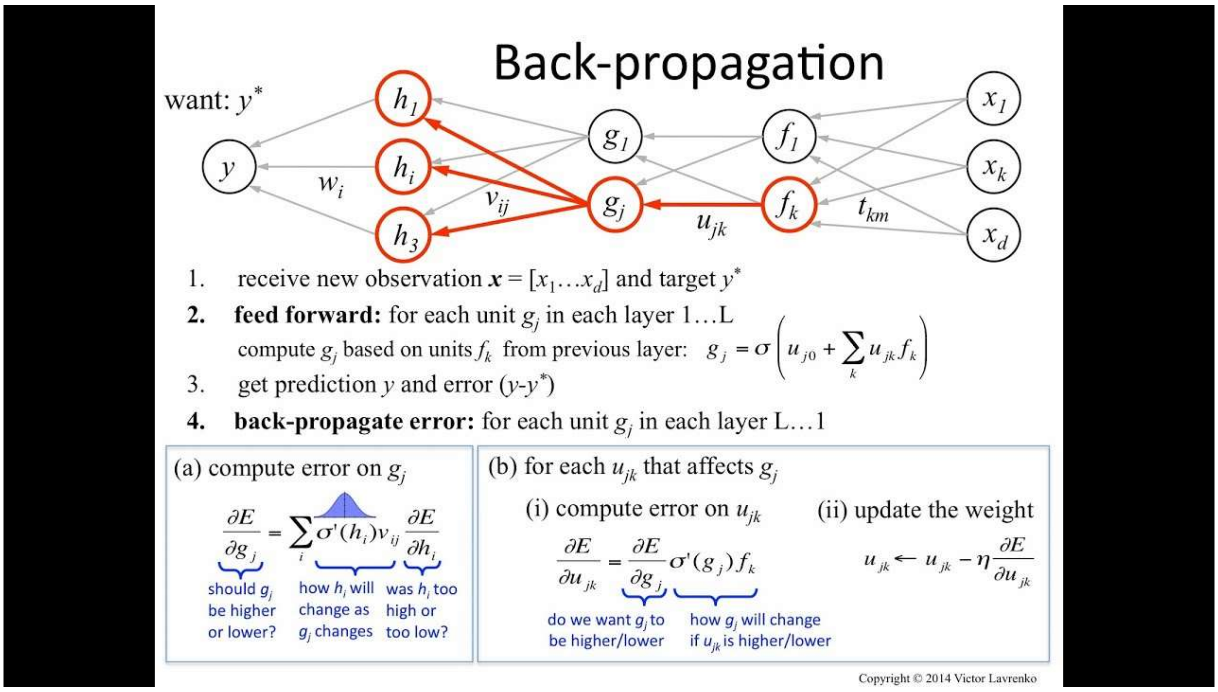
We extended the network to calculate the error function, so for the output layer we use that derivative as an input.

Backpropagation for the hidden layer(s) acts in the same way, using the values from the output layer as its input. The final step is weight updating.

The formula for updating the weight wij (the weight between node i and node j) is **Δwij = −γoiδj** , where γ is the learning rate, oi is the output from node i, and δj is the error from node j.

A possible variation is the inclusion of a momentum variable η. This can help make the learning rate more stable:

Δwij (t) = −γoiδj + ηΔwij (t − 1)



Back-propagation

1. receive new observation $\boldsymbol{x} = [x_1 \ldots x_d]$ and target y^*
2. **feed forward:** for each unit g_j in each layer 1…L
   compute g_j based on units f_k from previous layer: $g_j = \sigma\left(u_{j0} + \sum_k u_{jk} f_k\right)$
3. get prediction y and error $(y-y^*)$
4. **back-propagate error:** for each unit g_j in each layer L…1

(a) compute error on g_j

$$\underbrace{\frac{\partial E}{\partial g_j}}_{\text{should } g_j \text{ be higher or lower?}} = \sum_i \underbrace{\sigma'(h_i) v_{ij}}_{\text{how } h_i \text{ will change as } g_j \text{ changes}} \underbrace{\frac{\partial E}{\partial h_i}}_{\text{was } h_i \text{ too high or too low?}}$$

(b) for each u_{jk} that affects g_j

(i) compute error on u_{jk}

$$\frac{\partial E}{\partial u_{jk}} = \underbrace{\frac{\partial E}{\partial g_j}}_{\text{do we want } g_j \text{ to be higher/lower}} \underbrace{\sigma'(g_j) f_k}_{\text{how } g_j \text{ will change if } u_{jk} \text{ is higher/lower}}$$

(ii) update the weight

$$u_{jk} \leftarrow u_{jk} - \eta \frac{\partial E}{\partial u_{jk}}$$

Reference: https://www.youtube.com/watch?v=An5z8lR8asY

is the number of hidden layers appropriate?

A single hidden layer neural networks is capable of universal approximation: https://en.wikipedia.org/wiki/Universal_approximation_theorem.

The universal approximation theorem states that a feed-forward network, with a single hidden layer, containing a finite number of neurons, can approximate continuous functions with mild assumptions on the activation function.

While most "usefull" functions can be approximated with a neural network that has a single hidden layer, the number of neurons in that layer can grow exponentially (it's proven for certain functions). On the extreme case, you can use a very large single hidden layer as a lookup table for all different valid inputs of size n bits - there are at most 2^n such inputs.

A deeper network can use different layers to abstract increasingly complex features of the input, i.e a first layer to detect edges, a second layer to detect simple shapes, a third layer to detect objects composed of these shapes, etc. This often leads to better generalization ability.

For this reason, In our specific case. We only need 1 hidden layer to achieve a comfortable result. It is also important to note that in a Neural Network the number hidden layers and their neurons values should be balanced.

Intermediate

Core Process

The core process of this part is in the IntUI class. Here, the program runs in a while loop that is broken when the user enter 'exit'.

In this while loop we have a ticketing for loop running for 9 iterations as we have 9 input tags. At each iteration, the program outputs the specific tag as it is stored in a string array (thus using incrementation as array index). It then takes the user input and sets and the new inputData in a MLData structure.

The network computes the team allocated to the ticket using this MLData variable.

From the 5th tag, the network starts to make assumptions about the appropriate team for allocation.

Retrain the network with new entry

If the user is not satisfied with the allocated team, he just chooses what team would be best fit for the input ticket. This stores a MLData that is then appended to the csv encoded file. We then retrain the network with the new updated training set.

Extension

Resilient propagation

A promising alternative to backpropagation is resilient propagation where Instead of updating the weights based on how large the partial derivative of the error function is, the weights are updated based on whether the partial derivative is positive or negative.

First, the change for each weight is updated based on if the derivative has changed signs. If such a change has occurred, that means the last update was too large, and the algorithm has passed over a local minimum. To counter this, the update value will be decreased. If the sign stays the same, then the update value is increased.

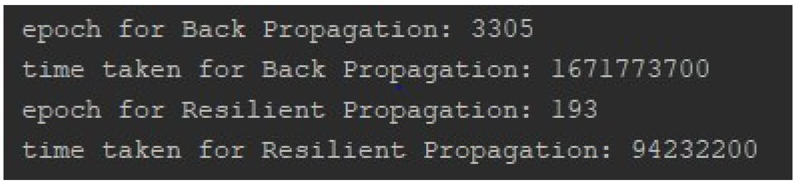
Once the update value is determined, the sign of the current partial derivative is considered. In order to bring the error closer to 0, the weight is decreased if the partial derivative is positive and increased if it is negative.

At the end of each epoch, all the weights are updated:

w (t+1) ij = w (t) ij + Δw (t) ij

The exception to this rule is if the partial derivative has changed signs, then the previous weight change is reversed

Evaluation



```
epoch for Back Propagation: 3305
time taken for Back Propagation: 1671773700
epoch for Resilient Propagation: 193
time taken for Resilient Propagation: 94232200
```

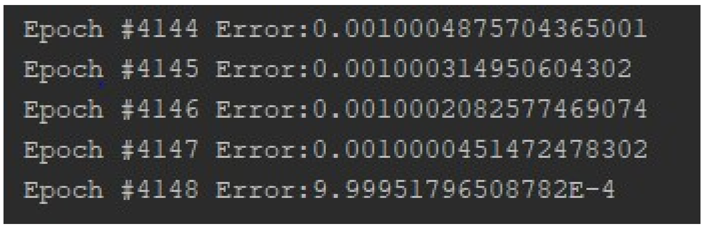
As we can see, the maths above are confirmed by those results, the resilient propagation method is faster than backpropagation.

This test is done in the Tester class where we use the 2 training algorithms and time their execution as well as getting the number of iterations needed.

Testing

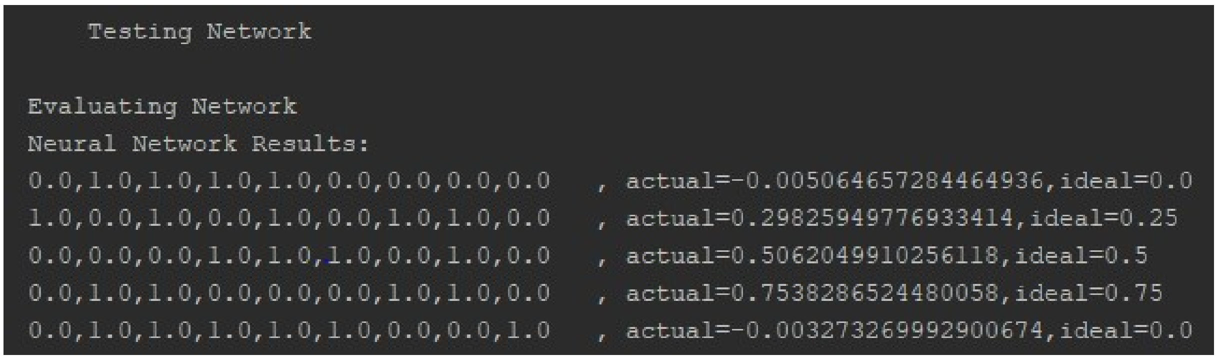
Basic

We test the network's creation and training:



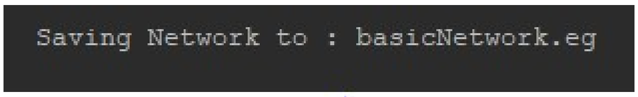
```
Epoch #4144 Error:0.0010004875704365001
Epoch #4145 Error:0.001000314950604302
Epoch #4146 Error:0.0010002082577469074
Epoch #4147 Error:0.0010000451472478302
Epoch #4148 Error:9.99951796508782E-4
```

We can see that the training algorithm iterates until the error rate threshold is reached.



```
    Testing Network

Evaluating Network
Neural Network Results:
0.0,1.0,1.0,1.0,1.0,0.0,0.0,0.0,0.0   , actual=-0.005064657284464936,ideal=0.0
1.0,0.0,1.0,0.0,1.0,0.0,1.0,1.0,0.0   , actual=0.29825949776933414,ideal=0.25
0.0,0.0,0.0,1.0,1.0,1.0,0.0,1.0,0.0   , actual=0.5062049910256118,ideal=0.5
0.0,1.0,1.0,0.0,0.0,0.0,1.0,1.0,0.0   , actual=0.7538286524480058,ideal=0.75
0.0,1.0,1.0,1.0,1.0,1.0,0.0,0.0,1.0   , actual=-0.003273269992900674,ideal=0.0
```

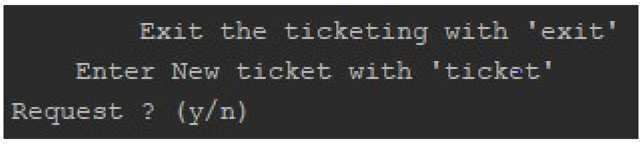
Then comparing the actual output with the ideal output, we can see that the neural net is performing well! The network is then saved as shown bellow:



```
Saving Network to : basicNetwork.eg
```

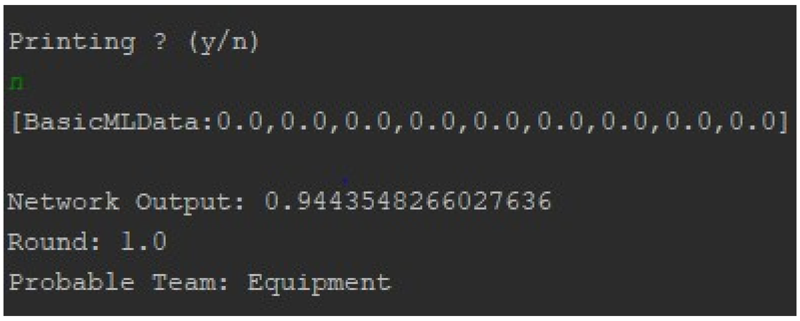
Intermediate

When running the user interface, the ticketing begins:



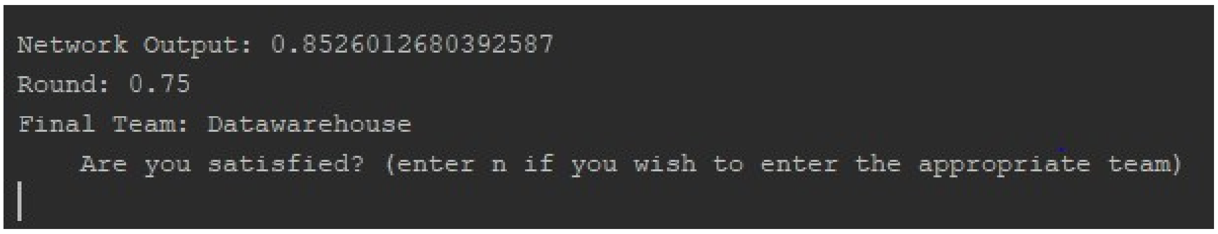
```
        Exit the ticketing with 'exit'
    Enter New ticket with 'ticket'
Request ? (y/n)
```

After a 4 tags requests, the network starts to guess the appropriate team allocation:



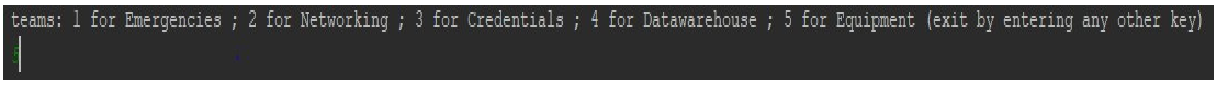
```
Printing ? (y/n)
n
[BasicMLData:0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]

Network Output: 0.9443548266027636
Round: 1.0
Probable Team: Equipment
```

After all the tags are entered, the team is given and the used is asked if satisfied.



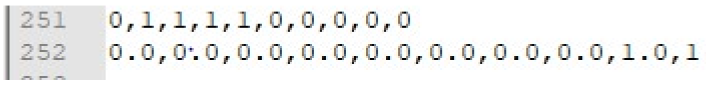
```
Network Output: 0.8526012680392587
Round: 0.75
Final Team: Datawarehouse
    Are you satisfied? (enter n if you wish to enter the appropriate team)
```

The user chooses the appropriate team from the list.



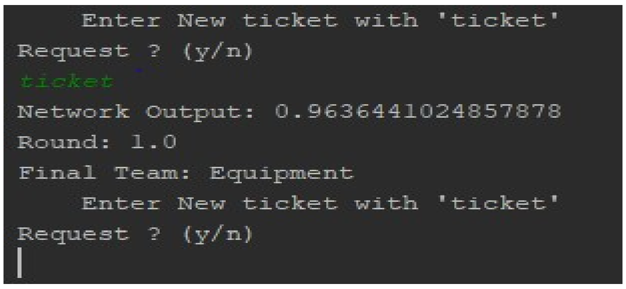
```
teams: 1 for Emergencies ; 2 for Networking ; 3 for Credentials ; 4 for Datawarehouse ; 5 for Equipment (exit by entering any other key)
5
```

The input is saved to the csv training set and the network retrains with this set. Here, we can see that the data is correctly added to the set.

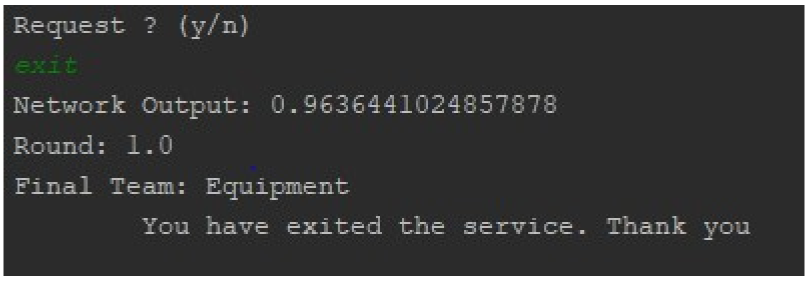


```
0,1,1,1,1,0,0,0,0,0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,1
```

The using can issue a new ticket by entering ‘ticket’ or exit with ‘exit’.



```
    Enter New ticket with 'ticket'
Request ? (y/n)
ticket
Network Output: 0.9636441024857878
Round: 1.0
Final Team: Equipment
    Enter New ticket with 'ticket'
Request ? (y/n)
```



```
Request ? (y/n)
exit
Network Output: 0.9636441024857878
Round: 1.0
Final Team: Equipment
        You have exited the service. Thank you
```

Conclusion

I am very happy with this submission.

References

https://www.heatonresearch.com/2017/06/01/hidden-layers.html

https://homepage.cs.uri.edu/faculty/hamel/pubs/theses/ms-thesis-winrich.pdf

Practical Checklist

BAS-1	Decide how to encode inputs and outputs for the given dataset.	DONE
2	Construct a training table with the chosen encoding for input/output of the dataset provided. This step can be done as part of your code or with an external tool (e.g., a spreadsheet editor).	DONE
3	Construct a multilayer feedforward neural network, with one hidden layer, and appropriate activation function according to the encoding used.	DONE
4	Train the network using backpropagation, setting appropriate parameters such as learning rate, momentum, etc and evaluate its output.	DONE
5	Save the trained network on a file.	DONE
6	Now consider the current neural network, is the number of hidden units appropriate?	DONE
INT-7	Develop a basic text-based interface for the user to log a new ticket.	DONE
8	The agent should ask the user whether a tag applies to the new ticket, repeating the question for all the tags of the dataset (e.g. Incident? Yes/No, Request? Yes/No, etc)	DONE
9	The agent should accept yes/no answers to the questions from the user	DONE
10	The agent should use the saved network from the basic agent to compute the output of the given input pattern.	DONE
11	The agent should communicate where this new ticket will be directed to. Please note that there is no need to ask or log the content of the ticket.	DONE
12	The system should then be ready to handle a new ticket.	DONE
13	The user might get tired of having to give an answer for each of the tags. Design a method for the agent to make an early prediction.	DONE
14	the agent should log the new entry in the training table, retrain the network and be ready to accept new tickets.	DONE
ADV-15	Use, compare and evaluate different training algorithms. Please ensure that a brief explanation of the algorithms used is given in the report.	DONE