Implementation of an Artificial Neural Network

Project#1

Akiki Clara, Attieh Joseph

Abstract—Nowadays, Artificial Neural Networks (ANNs) are relatively new computing systems that are being extensively used for solving real world problems. They process information in a way similar to humans by simulating functions and activities from the biological brain. ANNs are parallel systems composed of neurons which are connected with each other. Over time, ANN grew from a single layer network to a more intelligent network that has multiple layers.

Index Terms— Artificial neural networks, backpropagation, input, neuron, output, neuron

CONTENTS

1 INTRODUCTION

THIS project aims to build a multilayer neural network from scratch that can take multiple inputs in parallel and produce their outputs. Once the network has been structured, it is ready to be trained using a data set. During training, the error between the output computed by the network and the desired output will propagate through the network causing the system to adjust its weights. Finally, the system can be tested by entering inputs and checking the outputs.

First, the user designs the network by choosing the number of inputs, number of outputs, number of hidden layers, number of neurons per hidden layer of the artificial neural network as well as the weights, the activation function and the threshold of each neuron in that network. Second, this multilayer network is trained by providing both inputs and desired outputs then the system will perform backpropagation and update the weights of each neuron. Third, the neural network can also be tested by entering the inputs and the desired outputs then the system will calculate the error between the output computed from the network and the desired output.

In practice, the multilayer network is tought using many examples. Then the network needs a data set containing the inputs and the desired outputs. Since the neural network has non-deterministic characteristics, validation methods are needed in order to test the intelligence of the network.

This whole project will be coded in JAVA and will be visualized graphically with a GUI interface so users can interact with the main functionalities. This friendly interface will be built using Javafx and CSS. It will mainly allow the user to build a multilayer neurons and choose all its characteristics, train it and test it.

2 BACKGROUND

2.1 Problem Analysis

In order to build a neural network, we need to start by building a single neuron/perceptron which has its own number of inputs, weights, activation function as well as threshold. Then we can build the entire neural network by connecting different neurons with each other as long as we know the number of inputs, outputs, layers and neurons per layer. Knowing that we structured our neuron, we move to the learning phase known as training. In this phase, inputs and desired outputs forming the data set are wisely chosen and are used to calculate the error between the output of the network and the desired output. This error is backpropagated through the network while updating the weights. Three validation methods are used to test the network: separate validation, K-fold cross-validation and Monte Carlo cross-validation. Finally, the user can test any input by providing him with the corresponding output.

2.2 Functionality

The main functionalities of this artificial neural network are:

1. Designing
2. Training and validating
3. Testing

This project uses JAVA as its core scripting language and javafx and other languages such as CSS for other add-ons designs.

The main functionalities offered by the system are the following:

1. Choosing the number of inputs, outputs, layers and neurons in each layer.
2. Choosing the weights, activation function and threshold for each neuron.
3. Choosing the error calculation method and the maximum number of iterations.
4. Editing the activation function and threshold for a group of neurons.
5. Importing a .txt file containing the inputs and their desired outputs for training.
6. Validating the network by choosing one of the three validation methods: separate validation, K-fold cross-validation and Monte Carlo cross-validation.
7. Allowing user to enter inputs and test their output.

3 PROCEDURE

3.1 Building the structure of the neural network

First, a single neuron (Perceptron.java) takes as parameter the inputs, weights, activation function and threshold, then it will produce one output. The inputs of a neuron depend on the entire design of the neural network. The weights are initialized randomly using equation (10). Then the user can either keep them or enter his/her own values. The activation function could be: unit function (1), bounded linear function (2), identity function (3), sigmoid function (4) or Gaussian function (5). As for the threshold, it is entered by the user. This neuron has one output which is calculated using the inputs, weights, activation function and threshold and using formula (6).

Second, multiple neurons connect together to form the neural network (ANN.java). The connections between neurons are done using two cascaded ArrayLists. Each of the inner ArrayLists contain all the neurons at a given layer. As for the outter ArrayList, it contains all the inner ArrayLists. For example, Perceptron 2 that is located in Layer 4 is the second element in the fourth ArrayList in the array in the neural network. Moreover, if the neuron is in the first layer, then its takes the number of the network's inputs as its number of inputs to be able later on get the input from the training sets. Otherwise, it takes the number of the neurons in the previous layer as the number of its input number. The neural network has many outputs as the number of neurons in its last layer which can be computed after the user enters the inputs. They are computed in a way that all the neurons of the first layer receive the inputs as an ArrayList (so that the inputs are entered once in the neuron). Then, each neuron computes the output based on the inputs using equation (6) [1] and produces one output that is inputted in an arraylist with the outputs of all the other neurons in the same layers. Then, this arraylist is feeded to the neurons of the next layer and so on until the last layer is reached and the output is computed. This process is known as the feedforward propagation.

3.2 Training

Now that the network is structured, we can move to the learning phase that requires an input and its desired output. First, from the input provided and the characteristics of the network entered by the user, the output is computed. Then a comparaison is done between the output just calculated and the desired output entered by the user. For the comparison, the user chooses between one of three methods: mean difference error (7), mean absolute error (8) and mean square error (9). If the error is less than an error threshold entered by the user, then training stops since the output and desired output

are really close. Otherwise, the error backpropagates through the network and updates all the weights. In the code, matrices are used for the backpropagation corresponding to the weights, the error and the input.
Furthermore, it is not practical to enter each input and output one by one to train the network. Multiple inputs and their corresponding desired outputs should be able to use the neuron in parallel. For this reason, a data set which is contained in a txt file can help in training the system faster. The user provides the application with a txt file containing the data set which allows the system to extract the training pairs (using parsing) and train them in parallel.
Finally, three validation methods help test the ANN built. The first one is the seperate validation in which the data set is divided into training sets (which get trained) and validation sets (which get tested). This division happens based on a percentage entered by the user. The second one is the K-fold cross-validation in which the data set is divided into k subsets and, by turn, each of the subset is used for validation and the others for training. This is done using multiple for loops. Ther third one is the Monte Carlo Cross-Validation in which the data set is repeatedely divided into 2 random sets, one is used for training and the other one for validation.
The learning phase we talked about earlier can be depicted using this float chart:

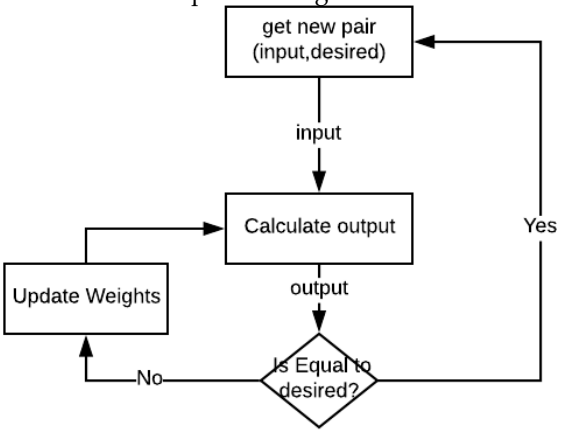


Fig. 1. Learning phase of the neuron

3.3 Testing

In order for us to proceed in making the Graphical interface of our application, we have to make sure that the Neural network that we are building is functioning as it is supposed to. So, we provided our artificial neural network with some examples in order to test it.
First, we used the example provided in class in order to test for backward propagation and front propagation.
We created the network made of 1 hidden layer, and three output neurons. Then, we set the weight of each neuron to its corresponding value such that:

$$W(t) = \begin{pmatrix} 1 & 2 \\ -1 & 1 \\ -1 & 3 \end{pmatrix} \qquad V(t) = \begin{pmatrix} 1 & 2 & 1 \\ -1 & 2 & 2 \end{pmatrix}$$

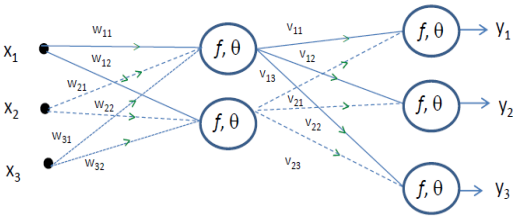


Fig. 2. Example of a neural network

- Neuron 1 of layer 1 has the weights equal to 1, -1 and 1
- Neuron 2 of layer 1 has the weights equal to 2, 1 and 3
- Neuron 1 of layer 2 has the weights equal to 1 and -1
- Neuron 2 of layer 2 has the weights equal to 2
- Neuron 3 of layer 2 has the weights equal to 1 and 2

Then, we set the activation threshold to 0 and the activation function to a unit step function with a=1. Also, we set the threshold error to be equal to 0.2 and the error function to be the Mean Absolute Error.

After that, we provide the training method with the inputs (1,1,1) with the desirable outputs (1,1,1). This method performs backward propagation until the error between the actual output and the desirable one is less or equal to the threshold. By calling this method, we remark that the new weights calculated by hand are the same as the one outputted on our console. This means that the neural network implemented seems to be working perfectly.

Second, we used Boolean networks to test for the front propagation (since no backward propagation was needed). By using the examples done in class, we made sure that our neural network worked properly.

Third, to test for the separate validation algorithm, k fold cross-validation and Monte Carlo cross validation, we proceeded by starting with small sets. We started with a set of data containing two input-output pairs that are identical. One pair was used to train and the other to verify which means that the ratio is 1.0 since the same pair that was learned by the network is now tested. And we started making the set of data bigger and observing how the ratio changes.

4 GRAPHICAL USER INTERFACE

After building our neural network and making sure that its training and testing functionalities are working perfectly, a GUI was created, allowing the user to enter all the parameters needed for the creation of the neuron, its training and its testing. First, the user runs the program and see a welcome screen.

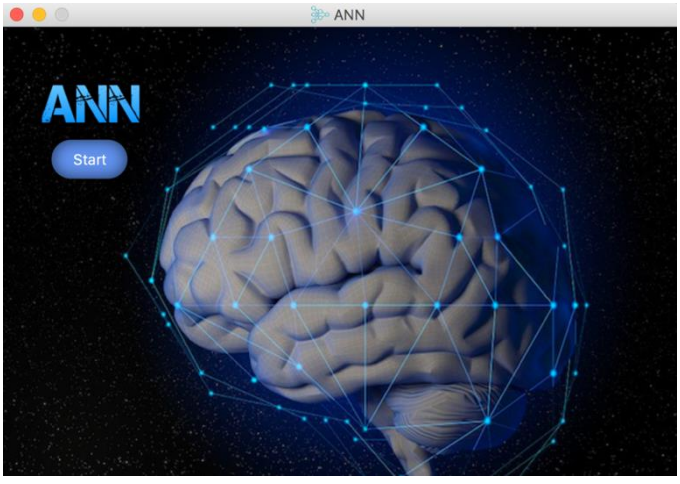


Fig. 3. Welcome screen

Second, the user enters all the parameters needed to build the neural network.

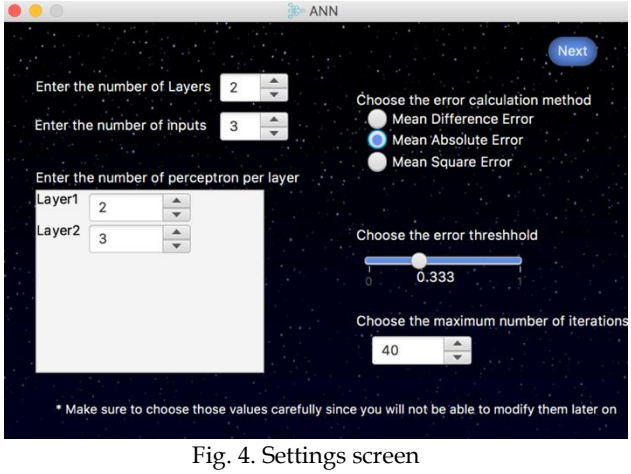


Fig. 4. Settings screen

Third, the user obtains a window where the neural network is drawn. In this screen, the user is able to press on any neuron that is drawn in order to change its parameters. Also, training, editing and testing can be accessed through this page. Fig. 5 shows the content of this page.

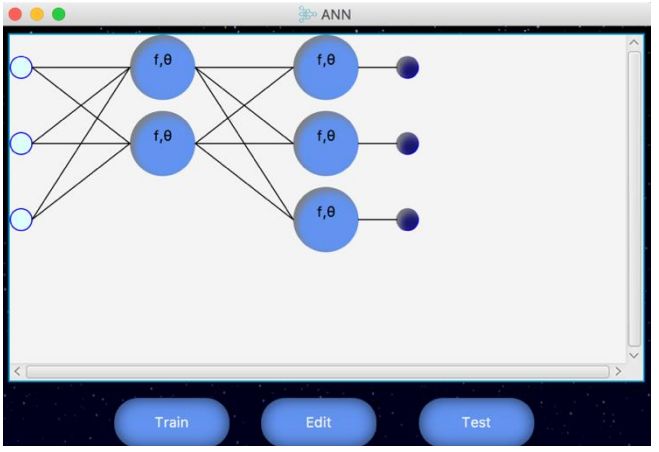
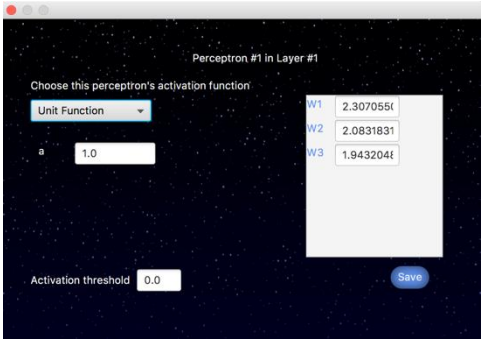


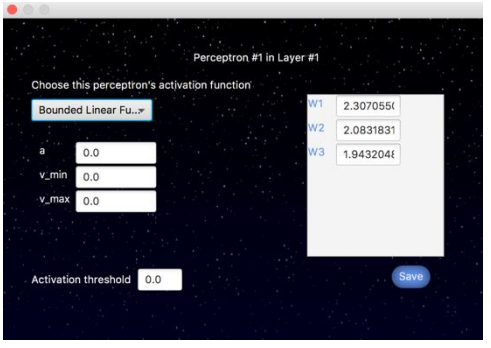
Fig. 5. Visualization of the neural network

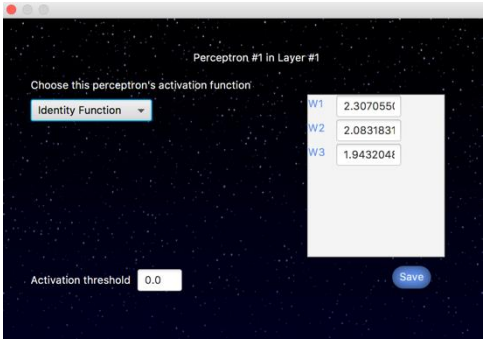
If the user chooses to press on a neuron, a popup will open. This popup will enable him/her to edit the settings of this neuron.

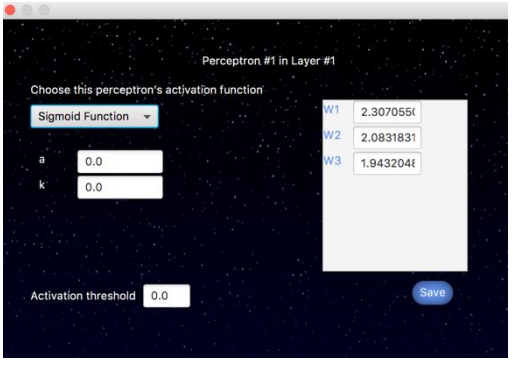
Weights are usually initialized using a random function (10) [1] which takes into consideration the size of the previous layer in assigning the random weights. This heuristic was found to be an efficient way to initialize the weights. However, those weights can be changed by the user. The activation threshold has a default value of 0 while the default activation function is the unit function with a equal to 1.

The multiple activation functions are provided in the pitctures below:









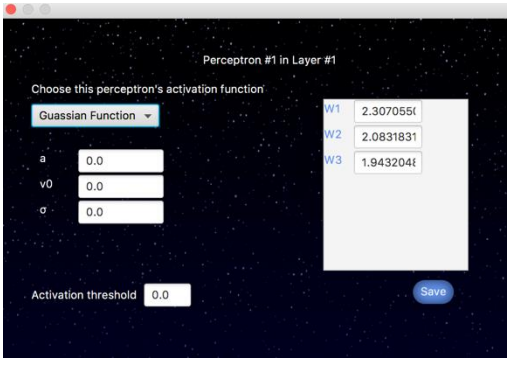


Fig. 6. Different activation functions per neuron

The user can set the parameters of multiple neurons in parallel by clicking on the edit button. This window appears:

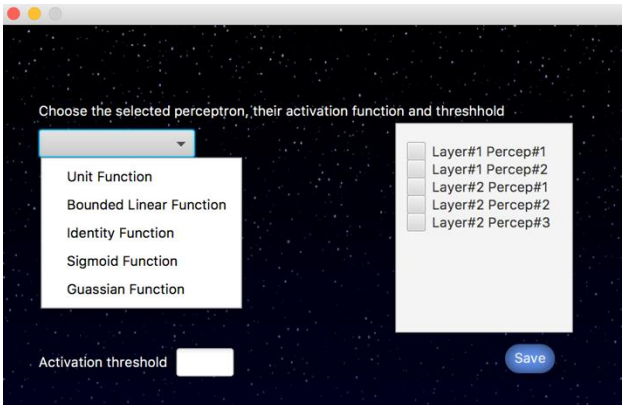


Fig. 7. Different activation functions for multiple neurons

After all the parameters of the entire neural network are entered, the learning can begin by pressing on the train button. A new window will popup which allows the user to enter the data set (txt file) containing multiple training pairs. The data could be trained or validated by one of the three validation methods. Those validating methods enable the user to efficientlty use the training set to make the learning faster and smarter. By using those methods, the user has the option to see three ratios and get a feel of the capacity of learning of the neural network. Furthermore, he/she would not have to worry a lot about the order of the data set and the memory loss issue. The screen enabling the user to train the network is showed below:

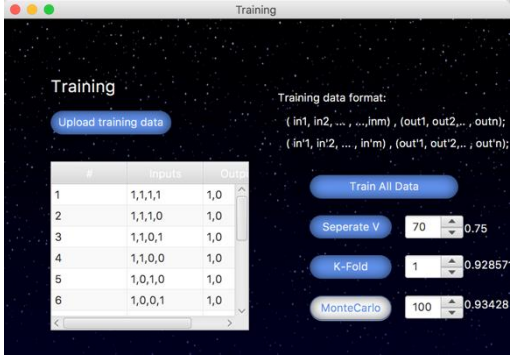


Fig. 8. Training Screen

If the training set does not match the architecture of the neural network (different numbers of inputs and outputs), a pop up window warning the user will appear:

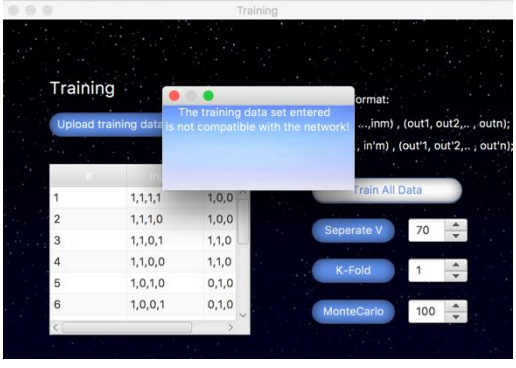


Fig. 9. Popup window

Finally, the last functionality implemented in this project is testing. When the user presses on the test button, a new window will pop up allowing the user to enter the input and providing him with the corresponding output.

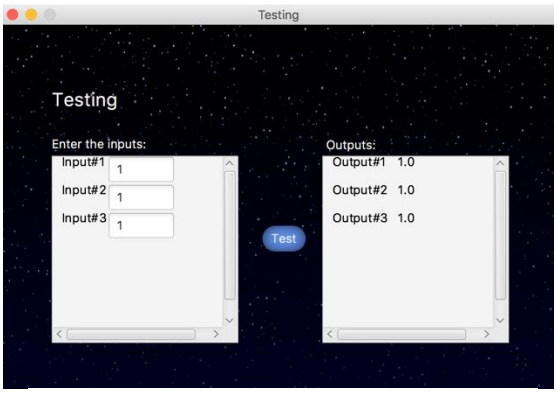


Fig. 10. Testing window

5 EXPERIMENTAL TESTING

To test the final version of our project, we made 4 sets of data.

5.1 Testing Set1

In this set, there are three outputs and two inputs.
The inputs represent the temperature of a room which goes from 1111 till 0000 (highest temperature to lowest temperature).
The first output is 1 if the temperature is hot and 0 if it is cold while the second output is 1 if the temperature is cold and 0 if it is hot.
The neural network that we used in this set was made of two layers: the hidden layer is made of 30 neurons while the second layer was made of only two. We used the Mean Average Error as the error function. All the neurons have an activation threshold equal to zero and an activation function equal to the unit impulse.
The training data was provided such that it follows the format *(input1, input2, input3, input4),(output1,output2);*
The training data is the following:

```
(1,1,1,1),(1,0);
(1,1,1,0),(1,0);
(1,1,0,1),(1,0);
(1,1,0,0),(1,0);
(1,0,1,0),(1,0);
(1,0,0,1),(1,0);
(1,0,0,0),(1,0);
(0,1,1,1),(0,1);
(0,1,1,0),(0,1);
(0,1,0,1),(0,1);
(0,1,0,0),(0,1);
(0,0,1,0),(0,1);
(0,0,0,1),(0,1);
(0,0,0,0),(0,1);
```

After training this set more than once (to get more accurate results) and after using our K Fold Cross-Validation, Monte Carlo Validation and the 70/30 separate validation, we remarked that not all the training data can be learned by the network. For example, by testing we remarked that (0,0,0,0) would produce (0,0) instead of (0,1) which means that the neural network was confused and concluded that the temperature is neither cold nor warm.

5.2 Testing Set2

In this set, the output is 0 if the input (represented as a binary number) is even, and 1 if the input is odd. The input was chosen to be made from 3 bits (3 inputs) while the output was made of one bit (one output).
The neural network that we used in this set was made of two layers: the hidden layer is made of 30 neurons while the second layer was made of only one. We used the Mean Average Error as the error function. All the neurons have an

activation threshold equal to zero and an activation function equal to the unit impulse.
The training data was provided such that it follows the format *(input1, input2, input3),(output);*
The training data is the following:

```
    (0,0,0),(0)
;
    (0,0,1),(1)
;
    (0,1,0),(0)
;
```

After training the network, we tested for each of those inputs. All the outputs matched the training data which means that the learning phase was successful. Thus, we can say that this neural network was able to learn all the data that we have provided correctly.

5.3 Testing Set3

In this set, there are two outputs and three inputs. The first output is equal to the second input while the second output is equal the third input.
The neural network that we used in this set was made of two layers: the hidden layer is made of 30 neurons while the second layer was made of two. We used the Mean Average Error as the error function. All the neurons have an activation threshold equal to zero and an activation function equal to the unit impulse.
The training data was provided such that it follows the format *(input1, input2, input3),(output1,output2);*

The training data is the following:

```
(0,0,0),(0,0);
(0,0,1),(0,1);
(0,1,0),(1,0);
(0,1,1),(1,1);
(1,0,0),(0,0);
(1,0,1),(0,1);
(1,1,0),(1,0);
(1,1,1),(1,1);
```

After training then testing, we remarked that with the following data set order, no memory loss occurred. Our network was able to learn all the data we have provided.

5.4 Testing Set4

In this set, there are two inputs and three outputs.
The inputs represent a binary number while the output represents the double of this binary number.
The neural network that we used in this set was made of two layers: the hidden layer is made of 30 neurons while the second layer was made of two. We used the Mean Average Error as the error function. All the neurons have an activation threshold equal to zero and an activation function equal to the unit impulse.
The training data was provided such that it follows the format *(input1, input2),(output1,output2,output3);*
The training data is the following:

```
(0,0),(0,0,0);
(0,1),(0,1,0);
(1,0),(1,0,0);
(1,1),(1,1,0);
```

After training then testing, we remarked that with the following data set order, no memory loss occurred. Our network was able to learn all the data we have provided.

6 HIGHLIGHTED FEATURES

6.1 Drop unused data

After testing our neural network, we realized that some values cannot be learned by this network since the design process requires a lot of fine tuning and a lot of decision making. Those "unlearnable" values will consume all the iterations (where the output is compared to the desirable output) and thus will ruin the weights of the connections. This will damage the ability of the network to give a correct output for the learned data. To fix this, our implementation checks if the input-desirable pair can be learned by the network. If it cannot be learned, the neural network recovers to the state just before trying to learn those data pairs. This approach made the neural network more efficient and enabled us to disregard a training pair that might ruin the knowledge of our network.

6.2 Edit a group of neurons

The user of the neural network might be dealing with a lot of neurons and a lot of layers. It becomes impractical for the user to change the parameters of each neuron. To solve this issue, the user was provided with the ability to select the neurons that he/she needs to edit and set the parameters with minimal effort.

6.3 Validations algorithms

As discussed before, three algorithms were implemented: the separation validation, k-fold cross validation and the Monte Carlo cross-validation. Those algorithms were implemented in a way that gives the user full control of the neural network. In the separate validation, the user could select the percentage of pairs that he/she need to use for training. In the K-fold cross-validation, the user has the choice to choose the number of subsets that need to hold the data pairs. For the Monte Carlo cross-validation, the user is provided with the choice to set the number of iterations in which the divison into random sets happen.

With this amount of flexibility, the user could observe if all the training data can be learned by the agents. This process can be done by uploading the file containing the training data and by running those three algorithms until the ratio visualized is maximized in the three of those algorithms. This ratio represents the ratio between the number of training pairs that were learned by the network over the number of the whole training pairs. This is an efficient way to train the neural network easily without having to worry that much about the order of data and memory losss.

6.4 Training Set

A user might by mistake, choose the wrong data set for the network. To solve this problem, the user is warned by a popup that using this data set with the current neural network is not allowed. This prevents the user from ruining his/her neural network.

7 EQUATIONS

Equations (1), (2), (3), (4) and (5) are activation functions corresponding to the unit function, bounded linear function, identity function, sigmoid function and Gaussian function. Equation (6) is the output vector a single neuron. Equations (7), (8) and (9) are error calculation functions corresponding to the difference mean error, absolute mean error and square error. Equation (10) was used to initialize the activation weights.

$$y = f(v) = \begin{cases} a & \text{if } v > 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$y = f(v) = \begin{cases} 0 & \text{if } v < v_{min} \\ a \times \dfrac{v - v_{min}}{v_{max} - v_{min}} & \text{if } v \in [v_{min}, v_{max}] \\ a & \text{otherwise} \end{cases} \tag{2}$$

$$y = f(v) = v \tag{3}$$

$$y = f(v) = a \times \frac{e^{kv} - 1}{e^{kv} + 1} \tag{4}$$

$$y = f(v) = a \times e^{\left(\frac{v-v_0}{\sigma}\right)^2} \tag{5}$$

$$y(t) = f_{w,\theta}(x) = f(w(t)\ x - \theta) = f\left(\left(\sum_{i=1\ldots4} w_i \times x_i\right) - \theta\right) \tag{6}$$

$$f_{MDE}(Y_m, D_m) = \frac{1}{J}\sum_{j=1\ldots J}(d_j^m - y_j^m) \tag{7}$$

$$f_{MAE}(Y_m, D_m) = \frac{1}{J}\sum_{j=1\ldots J}|d_j^m - y_j^m| \tag{8}$$

$$f_{MSE}(Y_m, D_m) = \frac{1}{J}\sum_{j=1\ldots J}(d_j^m - y_j^m)^2 \tag{9}$$

$$w = Random(size(layer[i]), size(layer[i-1]) \times \sqrt{\frac{2}{size(layer[i-1])}} \tag{10}$$

8 Conclusion

In conclusion, an artificial neural network provides a distinctive way to solve a problem. But it requires intuition in order to build the whole network. The decisions to be made in order to implement the intelligent neural network involves the number of inputs, outputs, layers and neuron per layer as well as the activation function, threshold and weights. As for the training and testing part, it also demands some decision making if we have a data set; we need to choose which validation method to be used and its corresponding parameters as well as the error calculation method. However, this computing system is considered a natural evolution and its intelligence stimulated during the backpropagation phase is allowing to discover hidden patterns.

References

[1] Ananthram, A. (2018, February 25). Random initialization for neural networks: a thing of the past. Retrieved from https://towardsdatascience.com/random-initialization-for-neural-networks-a-thing-of-the-past-bfcdd806bf9e