# AIEDITOR

## NEURAL NETWORKS EDITOR

DDEV TECH

## Summary

Currently artificial intelligence is being used in major companies such as Amazon or Google and will soon be used to extend to our personal environment. Scientists from all over the world try to understand the functioning of our prodigious brains. Artificial intelligence is not easy to understand, and teaching people to use it is not a simple task. Therefore, in this project I have proposed to develop a simulator and at the same time editor of artificial intelligence with a graphical interface that facilitates to develop, to understand and to innovate in this technological sector.

**Keywords:**

Artificial intelligence, future, algorithms, neural network, autonomous learning, programming, AI, machine learning, deep learning, education

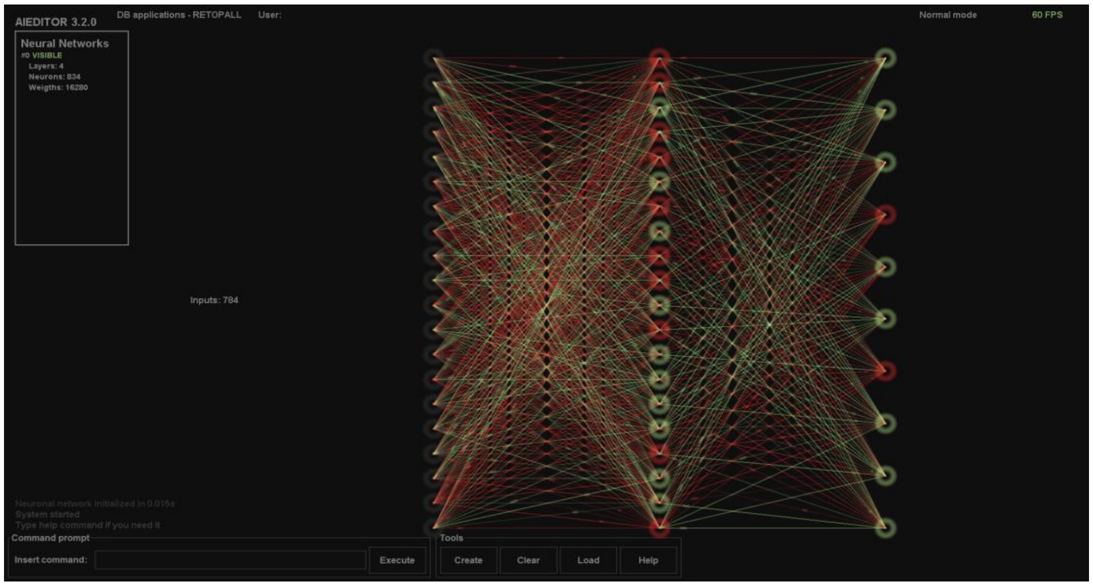
## Contents

# 1 Introduction

This project aims to design an artificial intelligence editor with which to create, edit, adjust and visualize different neural networks. The different algorithms, graphical interfaces and internal mechanisms will be created using Java[1] As the main programming language.

The program can be downloaded at the following Web address: https://www.retopall.com. In addition, you will see a trailer of the application on the link: https://www.youtube.com/watch?v=W1fH_xmx6Xg

The main concepts used of artificial intelligence in the program and the internal functioning of this will be explained. Basic mathematical and programming skills will be recommended.



```
Neuronal network initialized in 0.015s
System started
Type help command if you need it
```

Figure 1: Red neuronal 784x20x20x10

[1] Programming language widely used by companies and currently owned by the Oracle company

# 2 Methodology

The following tools are necessary for the elaboration of the program:

## 2.1 Computer Tools

I have worked with a computer with the following specifications for the elaboration of the program:

- Windows 10 operating system
- Intel Core i7 8700 processor-executing algorithms
- Nvidia GeForce 1070 graphics card-Network processing graphical environment
- 16GB RAM Memory DDR4-Storing dynamic program variables
- SSD disk-speed up the storage of neural networks on your computer and reading these.

The program does not use any external library because all the algorithms are written manually.

## 2.2 Mathematical tools

Neural networks use algorithms of great complexity, so I will use mathematical analysis tools (derivatives, partial derivatives, gradient descent, regression) and linear algebra (matrices, vectors, function analysis).

The tools mentioned above will be implemented in the program with their respective methods and functionalities.

## 2.3 Programming tools

An IDE will be used (*Integrated Development Environment*) to Program The algorithms and the graphical interface of the application. Use *Eclipse Photon*.[2] As stated above, the language used will be Java for algorithms and is widely used in large companies such as Google, Spotify, Netflix or Amazon.

Java, is the language most used by programmers, is being used for artificial intelligence and can integrate *Tensorflow*[3], a library developed by Google and more used to create algorithms and intelligent networks.

# 3 The Neural network editor

The Application object of this project can be downloaded for Windows, MacOSX or Linux. The program does not use external libraries, that is to say, everything is programmed manually without any added ease. The application has a total of 16280 lines of code and 81 classes, which makes it a large-scale project.

The application structure is as follows:

1. Connection to the Web server by means of HTTP requests
2. Functioning of a neural network
3. Artificial Intelligence algorithms
4. Graphical interface to adjust and manipulate the neural network
5. Graphic rendering process[4] and multitasking[5]
6. Storage and reading of neural networks

[2] Application developed by Eclipse Foundation. Downloadable at: https://www.eclipse.org/
[3] Open Source Library for automatic learning and used to train neural networks, detect patterns and other functions
[4] Render (render) is a function widely used by programmers to update on-screen graphics
[5] Programs that want to perform multiple runs at the same time must have a series of "threads" running sequentially

## 3.1 Connecting to the WEB server

The application is synchronized with the Web page mentioned above (www.retopall.com)

To log in you need to have a user license. To verify it requires running a request to the PHP Web server[6] For this to make another request to the MySQL database[7].

Passwords are stored securely in the database by using the MD5 encryption method.[8] The bytes of the password we want to encrypt are digested and that set of bytes becomes a string of numbers and letters.

To send the request to the Web server A URL connection is created and the request is carried out with the GET method[9].

The PHP code will produce a Boolean response[10] Depending on whether the data is correct. If the answer is true, the neural network editor will proceed to open. The request to the server is shown in the following code:

The graphical interface for logging into the application is shown in Figure 2

[6] Web programming language run on the server generally to run requests to a database securely
[7] Database management System
[8] 128-bit cryptographic reduction algorithm
[9] Request to a Web server in which the parameters that are given to this are visible
[10] False or True Value (0 or 1)

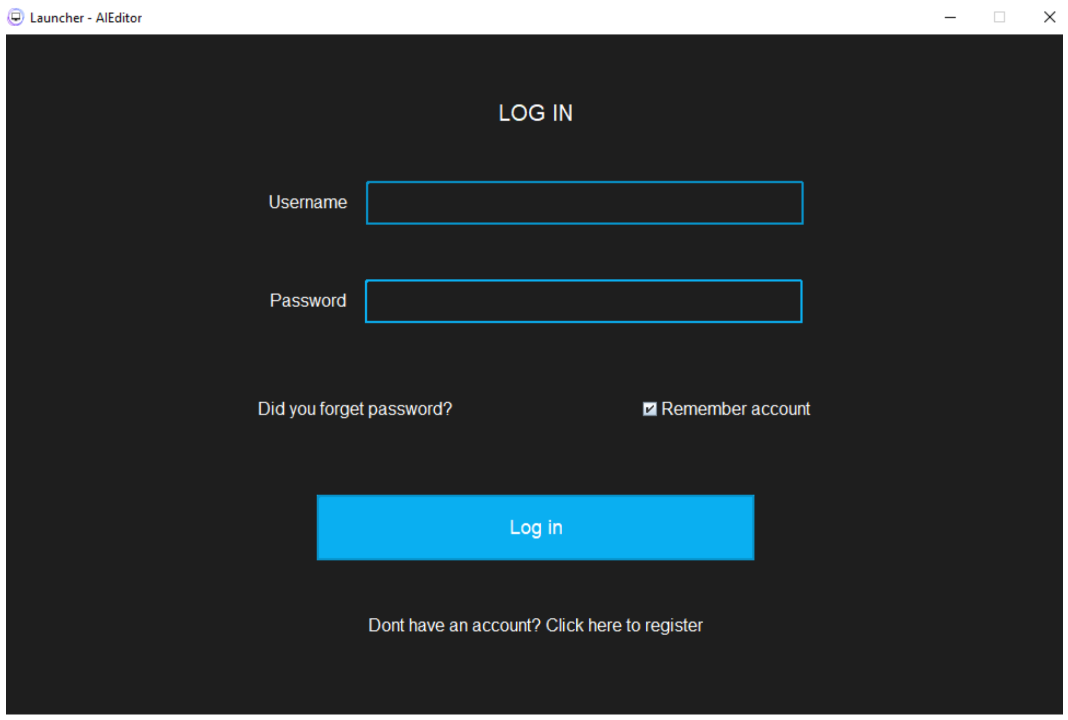


Figure 2: Logon window

## 3.2 Red neuronal

A neural network is a computational model used for learning By means of only numerical relationships.

These networks are formed by layers, neurons and weights mainly. The weights are the connections between neurons of a previous layer with neurons of the posterior one. Weights and neurons store a decimal value.

As for the layers, there is an input layer, another one of output, and a series of hidden layers.

The input layer adds the values that you want to leave. For example, pixels in an image to be scanned on the network would be input values. Depending on the color of the pixel, input neurons would be assigned input values in the range [0.1]. The neurons of the output layer are the values of the solutions and predictions of the neural network.

Finally, we would have the hidden layers that improve the intelligence of the neural network. For network training, the values of the weights will be edited to minimize the error of the output neurons. The values of the neurons will depend on the weights, except for the input that will depend on the given stimuli.

The structure of the neural network can be seen in the following graph. With a neural network that has 3 layers with an entry layer of four neurons, a hidden layer of 5 neurons and an output layer of a single neuron. Next-layer neurons are related to a series of weights.

To train the neural network we will need a set of data to learn from. In the proposed example of the input pixels, we need a set of images in which we will take the pixels to represent them as input value.

Next we will have to calculate the value of the following neurons based on the input values. This will be achieved with the algorithm *feed-forward* or forward propagation. This method will provide us with delimited output values between 0 and 1, which correspond to the certainty that the neural network will have in each output neuron.

We must bear in mind that the input values will always be the stimuli or information that we want the neural network to analyze to predict and produce a series of results in the output layer.

When the neural network is not trained, the output values will be virtually random. However, by carrying out the method *back-propagation* Or backward propagation, the neural network will calculate the error you have had on the output values compared to the ones you want and execute the gradient descent[11] To lessen the error and therefore learn.

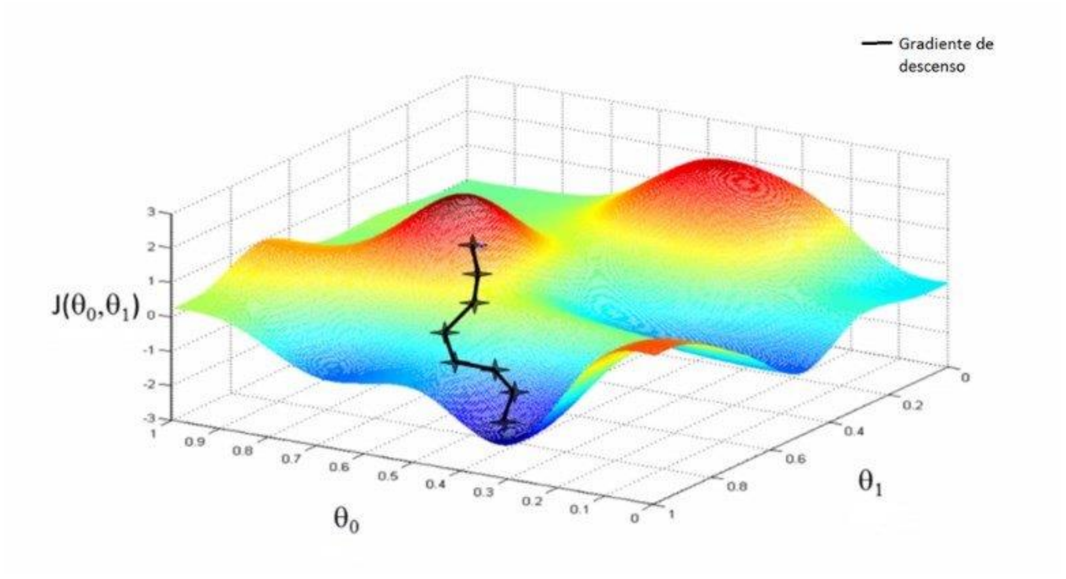


Figure 3: Minimization of error with gradient descent algorithm [10]

## 3.3 Artificial Intelligence Algorithms

The neural network will have two main functions, which will be the backpropagation and the feed-forward function.

The first one will be used to minimize the neural network error and the second to update the neural network. These two functions are synchronized by the learning function that will iterate over all the data sets. It will update the neural network, calculate the error,

[11] It is one of the most popular optimization algorithms in automatic learning, particularly for its extensive use in the field of neural networks. Gradient descent It is a general minimization method for any function f. The original version is considered slow but versatile, especially for case functions Multi-dimensional.

and eventually execute the back-propagation function.

### 3.3.1 Initialize

When creating the neural network, all weights and neurons will be initialized. The values of the weights will be created pseudo with a normal distribution generally in the interval [-1.1].

$$rand.nextGaussian() * 2 - 1;$$

### 3.3.2 Feed Forward

The feed-forward function is the one dedicated to updating the neural network. The input values will reach the output values. It will depend on the values assigned to all weights. The propagation is forward as it runs from the first layers to the last. Below are the terms used to run the feed-forward. The number of layers that the neural network has is the input, output, and hidden layer.

$a_n^l,\ a_m^l$: Are the values of a neuron in which $l$ is the layer and $m,n$ It's the positions on that layer.

$c$: These are the connections with the anterior layer to this neuron.

$w_{n,k}^l$: Weight connecting neuron $n$ of the layer $l$ With the neuron $m$ Of the next.

$\sigma(x)$: Activation function to be explained to After

$b_n^l$: Additional parameter also to adjust, which brings more intelligence to Neural network. In terms of programming it is called $bias$.

Each neuron will be updated in the following way:

$$a_m^{l+1} = \sigma(\sum_{n=0}^{c} w_{m,n}^l a_n^l + b_n^l)$$

The complete neural network update will be achieved in the following way. All layers will be traversed from the entrance to the exit where $l$ Be $0,1,2,\ldots$

$$\begin{bmatrix} a_0^{l+1} \\ a_1^{l+1} \\ \dots \\ a_m^{l+1} \end{bmatrix} = \sigma \left( \begin{bmatrix} w_{0,0}^l & w_{0,1}^l & w_{0,2}^l & \dots & w_{0,n}^l \\ w_{1,0}^l & w_{1,1}^l & w_{1,2}^l & \dots & w_{1,n}^l \\ \dots & \dots & \dots & \ddots & \dots \\ w_{m,0}^l & w_{m,1}^l & w_{m,2}^l & \dots & w_{m,n}^l \end{bmatrix} \begin{bmatrix} a_0^l \\ a_1^l \\ \dots \\ a_n^l \end{bmatrix} + \begin{bmatrix} b_0^l \\ b_1^l \\ \dots \\ b_n^l \end{bmatrix} \right)$$

Neurons can be delimited by activating functions. These functions, in this case denoted by the Greek symbol Sigma ($\sigma$), may vary depending on the purpose and use that you want to give to the neural network. There are many activation functions, but the main features I will use in my program are the Sigmoid and RELU functions.

- RELU function

This function is very useful because, if the activation is negative for a neuron, you will never get to activate that neuron, ie, it will not have a positive value. This function is shown below:

$$\sigma(x) = \begin{cases} 0 & si \quad x < 0 \\ x & si \quad x > 0 \end{cases}$$

- Sigmoid function

The sigmoid function is very sensitive to the values very close to $x = 0$. As we move away, the changes become indifferent as it has horizontal asymptotes in $y = 0$ And $y = 1$.

$$\lim_{x \to \infty} \sigma(x) = 1 \lim_{x \to -\infty} \sigma(x) = 0$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Graphical representation of sigmoid function:

There are also other activations like the Hipérbolica function (Tanh) $\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

### 3.3.3 Back Propagation

The main objective of the backward propagation is the modification of each and every one of the weights of the network so that the neural network learns.

An algorithm will be carried out in which, when knowing the output and the desired values, the error will be calculated.

By traversing the set of data to be learned, each datum will have its desired value from the output layer. Then, with the gradient descent method, the neural network error will decrease. Each and every one of the weights will be traversed, their variation will be calculated depending on the total error and these values will be modified in order to decrease the function of cost $E(x)$.

The following steps will be followed for the reduction of this cost:

**Calculation of output layer error**

To carry out the calculation of the neural network error The error of the output layer will be determined, in other words, the neurons of the last layer will be traversed and the error will be calculated by adding all the squares of the differences between the desired value and the real one obtained in the Neu Rona out.

$x_n$: The desired value that the neuron should have in the position $n$ of the output layer

$a_n$: The actual value obtained in the neuron in the position $n$ of the layer of exit

$k$: Number of data sets to train.

$$E_{total} = \frac{1}{k}\sum_{a=0} (x_n - a_n)^2$$

**B) Minimizing the error of the output layer**

To minimize the error we will have to calculate how much decreases this with respect to each weight of the neural network. This will be done by calculating the partial derivative of the error with respect to each weight.

The notation $out$ Represents the value of the neuron with the activation function applied. The notation $net$ It represents the value of the neuron without applying the activation function.

$$\frac{\partial E_{total}}{\partial w_{m,n}} = \frac{\partial E_{total}}{\partial out(a_n^l)}\frac{\partial out(a_n^l)}{\partial net(a_n^l)}\frac{\partial net(a_n^l)}{\partial w_{m,n}}$$

By solving the derivatives you get:

$$\frac{\partial E_{total}}{\partial w_{m,n}} = -(x_n^l - a_n^l)\sigma'(a_n^l)a_m^{l-1}$$

This formula can be used only in the weights that connect the output layer with the last hidden, as the other weights depend on the variation of the above updated.

The derivative of the activation function $\sigma'$ In the case of the sigmoid function it will be:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\sigma'(x) = \sigma(x)(1-\sigma(x))$$

In the calculation formula of partial error derivative it is observed that it depends on the anterior neuron. We can define a new variable $\delta$ That will be the sensitivity of a single neuron repecto to the total error.

Use $\delta$ To calculate the sensitivity of the error with respect to a weight that will depend only on a neuron $a_n^l$

$$\delta_n^l = -(x_n^l - a_n^l)\sigma'(a_n^l)$$

Also, if we want to change the *bias* The network, you should minimize the error with respect to each $\frac{\partial E_{total}}{\partial b_m^l}$

**C) minimizing hidden layer error**

Knowing the variation necessary to minimize the error of the weights of the output layer, you can find the necessary variations for the rest of weights. However these, depending on the previous weights, will have a different calculation. This is why the backward propagation is from the output layer to that of input and runs in the opposite direction of the feed-forward algorithm.

$k$: Run all of the weights of the layer before the weight to be adjusted.

$l$: The layer of neurons that is connected to the weight to be adjusted and the weights of the next layer (already up-to-date).

We will then proceed to the calculation of the other weights that are not connected to the output layer.

$$\frac{\partial E_{total}}{\partial w_{m,n}} = \left(\sum_{k=0} (\delta_k^l w_{m,k}^{l+1})\right)\sigma' a_n^{l-1}$$

**D) Modifying the weights and bias of the network**

By having the optimal variation of a weight, to minimize the error we'll update the value of the weight depending on its variation ($\frac{\partial E_{total}}{\partial w_{m,n}}$).

Call $\eta$ to the learning coefficient. It is used to not lower the error drastically because there is another type of data and the network needs to provide some tolerance to all of them. The error obtained only depends on a single datum.

There are several ways to update weights:

• Stochastic gradient descent: Updates each weight for each new data

$$w_{m,n}^{l} = w_{m,n}^{l} - \eta \frac{\partial E_{total}}{\partial w_{m,n}}$$

• Batch descent: In each round of the entire data set, the weights are adjusted. $t$: Data Set Size

$$w_{m,n}^{l} = w_{m,n}^{l} - \eta \frac{1}{t} \sum_{t=0}^{t} \frac{\partial E_{total}}{\partial w_{m,n}}$$

• Gradient decrease by mini-Batch descent:

It is the intermediate algorithm between the previous ones, in which the weights are updated after a certain number of iterations on the dataset. It is the optimal algorithm among those described, because it does not consume a lot of memory to the processor but at the same time it is fast and therefore efficient.

$s$ Mini size or *mini-batch* (20-100).

$$w_{m,n}^{l} = w_{m,n}^{l} - \eta \frac{1}{s} \sum_{s=0}^{s} \frac{\partial E_{total}}{\partial w_{m,n}}$$

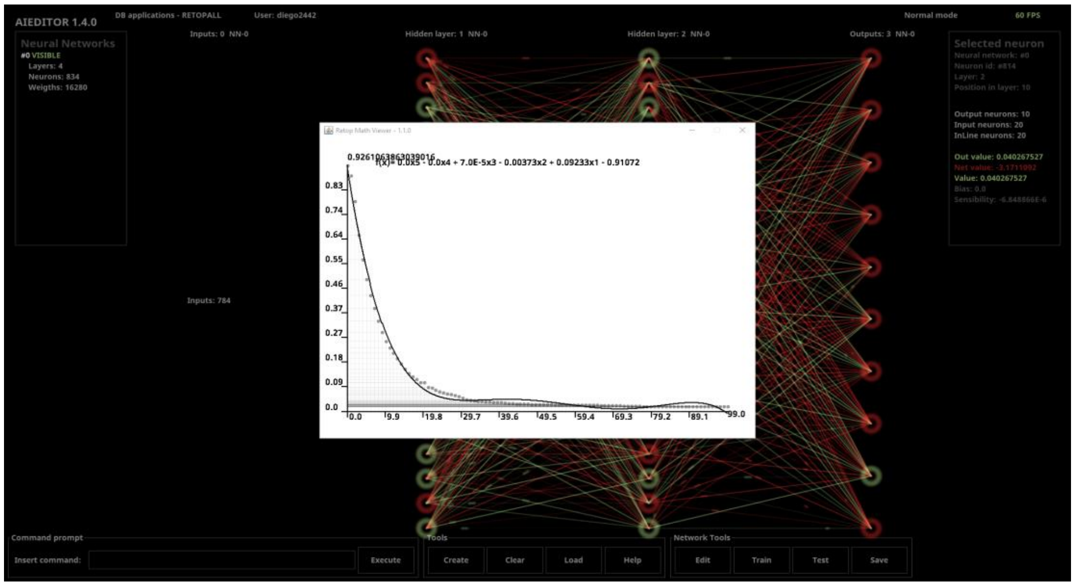
- Bias Modification

$\beta$: Bias learning coefficient.

As with weights the bias will also be modified as follows:

$$b_{m}^{l} = b_{m}^{l} - \beta \frac{\partial E_{total}}{\partial b_{m}^{l}}$$

These algorithms will be executed on each weight and so all of them will be adjusted and the total network error decreases. You can see this decrease in image 4


AIEDITOR 1.4.0

DB applications - RETOPALL

User: diego2442

Normal mode

60 FPS

Neural Networks
#0 VISIBLE
Layers: 4
Neurons: 834
Weigths: 16280

Inputs: 0 NN-0

Hidden layer: 1 NN-0

Hidden layer: 2 NN-0

Outputs: 3 NN-0

Inputs: 784

Selected neuron
Neural network: #0
Neuron id: #814
Layer: 2
Position in layer: 10

Output neurons: 10
Input neurons: 20
InLine neurons: 20

Out value: 0.040267527
Net value: -3.1711092
Value: 0.040267527
Bias: 0.0
Sensibility: -6.848866E-6

Retop Math Viewer - 1.1.0

0.9261063863039016

f(x)= 0.0x5 - 0.0x4 + 7.0E-5x3 - 0.00373x2 + 0.09233x1 - 0.91072

Figure 4: Graph of neural network error growth

### 3.3.4 Training

To carry out a complete training, the following variables will be required:

• Data set: It will be an array with the images and therefore the pixels of all the images.

Iterations $i$ The number of data set to learn from.

Times $epochs$ The number of times the data set will be iterated for learning.

They will go through every epoch. In each of these, it will go through the whole data set if the gradient descent is stochastic.

The pixel colors of the images will be scanned and mapped in the first layer as input values. Then it will run the function *feed-forward* And with the output values obtained the error will be calculated. Then it will take place the *back-propagation* To decrease the error or cost function $C(x)$ and update the weights of the neural network.

### 3.3.5 Optimization algorithms

The gradient decrease will seek to decrease the error, but will not get to the absolute minimum in which the learning of the neural network has been maximized with the data sets. In addition, it is not the fastest way to reach a minimum since the learning coefficient is constant so there will always be a margin of error in the decrease of the cost. Therefore, optimization algorithms will be used [Figure 5].

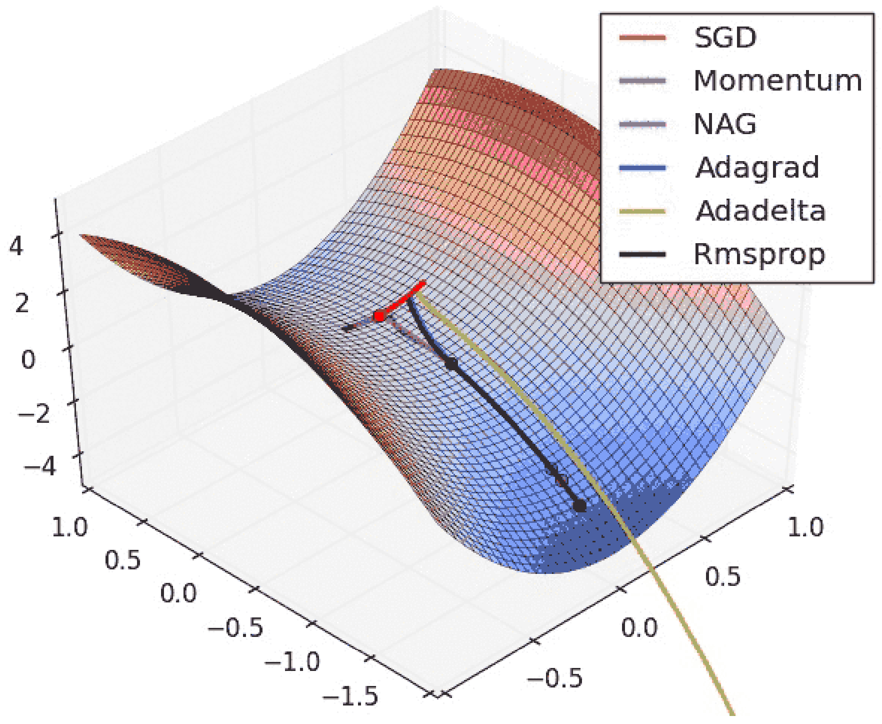


Figure 5: Optimization algorithms [11]

There are many optimization algorithms:

**Momentum**

It can be achieved to reduce the large oscillations that presents the descent of stochastic gradient in the search of a local minimum. Accelerates the descent in the right direction and reduces acceleration in all other directions [Figure 6].

$v(t)$: Speed in a specific time $t$

$\gamma$: Term of the moment. Depending on whether you want to advance more but with fewer oscillations or vice versa. It's usually 0.9.

$$v(t) = \gamma v(t-1) - \eta \frac{\partial E_{total}}{\partial w_{m,n}}$$

When we calculate $v(t)$ We will proceed, as explained above, to update the weights but with $v(t)$. We will be able to use the descent by mini-groups if we proceed to it.

$$w_{m,n}^{l} = w_{m,n}^{l} - v(t)$$

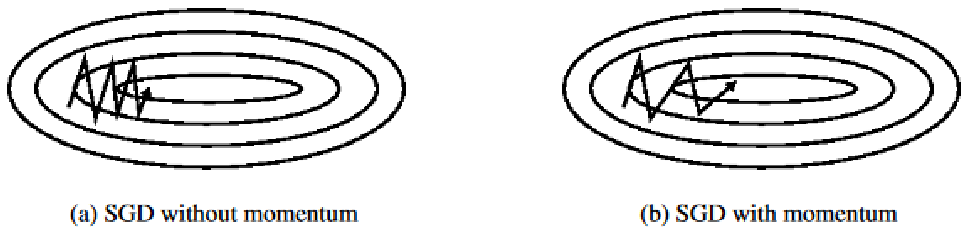


Figure 6: Momentum en contraposición a SGD [12]

**RMSProp**

It also tries to decrease the oscillations of the directions that do not point to the minimum. $\beta$: The value of the moment. is usually 0.9

$$v_{dw} = \beta v_{dw} + (1-\beta)dw^2$$

When updating weights, the following must be done to vary the learning coefficient:

$$w_{m,n}^{l} = w_{m,n}^{l} - \eta \frac{dw}{\sqrt{v_{db}\epsilon}}$$

To prevent you from getting a very high speed descent and passing the minimum, you use $\epsilon$, with positive values relatively close to 0.

**Adagram**

Gradient optimization algorithm, which adapts the learning coefficient with some parameters applying small variations to this.

$g_t$ — is the variation of a given weight to decrease the error. The term $t$ Represents when the weight has been updated. Therefore, with this notation the stochastic gradient descent would be:

$$g_t = \frac{\partial E_{total}}{\partial w_{m,n}} \Longrightarrow w_{t+1} = w_t - \eta g_t$$

The Adagram optimization algorithm will try to modify the learning coefficient depending on the previous gradients $g_t$ Where $t$ It represents the order of execution of those gradients.

$G_t$: It's a diagonal matrix where the position $i, i$ Is the sum of the squares of the weight gradients that we want to change to the last previous change $t$.

$\epsilon$: Term that prevents division by 0. is a very low number as $10^{-8}$.

You will have to calculate the vector product between the matrices $G_t$ And $g_t$ As shown below:

$$w_{m,n}^{l} = w_{m,n}^{l} - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

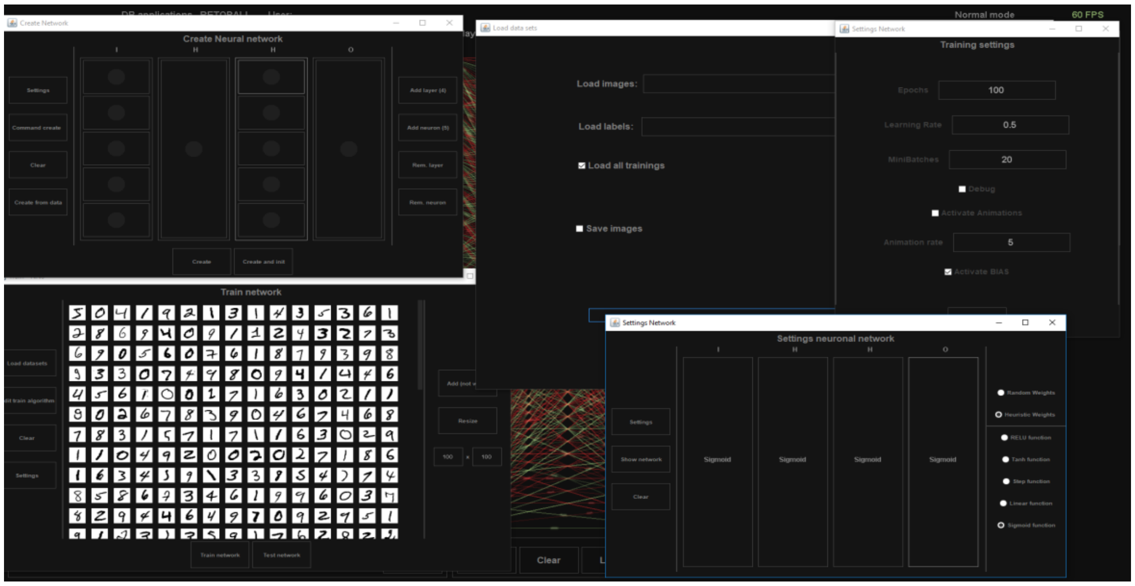
The main benefit of the ADAGRAM algorithm is the elimination of the need to manually change the learning coefficient.

*Adadelta, ADAM, Nadam y Nesterov Accelerated Gradient* They are other examples of gradient-lowering optimization algorithms. However there are many more and variants of those mentioned.

## 3.4 Graphic interface

The application has a sophisticated interface to simplify the program controls and functions described below. It also uses the integrated Java graphics library called *Swing*. Text fields, buttons, editors and menus have been created with a simple and useful design and always running smoothly.

The technical Manual of the operation of the application and the first steps can be obtained in the following link: https://docs.google.com/document/d/1RCckXaYssdckRxK-5rSRjsPZ1_A6wLahVYPceHjJPkk/edit?usp=sharing.



Settings Network

Training settings

Epochs: 100

Learning Rate: 0.5

MiniBatches: 20

☐ Debug

☐ Activate Animations

Animation rate: 5

☑ Activate BIAS

Figure 7: Windows of the Neural network editor

### 3.4.1 Creation window

In the upper left side you can see a neural network editor in which the user can visually configure a neural net. Offers the ability to create by command or from the data sets

### 3.4.2 Import window

In the upper central zone you can see a window in which you can quickly import data sets to train the network Neuronal. Neural networks can be modified by editing the learning rhythm, iterations and activations.

### 3.4.3 Learning window

In the lower left pane you can view the different sets of data, in this case to train a neural network with digits. This is the training panel. You can select a single specimen to train or all the specimens. It offers the possibility to change the size of the images and to configure them.

### 3.4.4 Simulations

There is also a window to test different simulations and see the results, accuracy and performance of the neural network. The numbering of each formula is composed of two numbers: that of the section, within which is the formula, and the number that occupies the formula within this section. (Can be numbered in other ways, of course).

## 3.5 rendering and updating process

In addition to the entire interface and all the algorithms it uses, it is also possible to visualize and understand the behavior of the neural network. To do this, it integrates a graphic engine programmed from scratch to render the neural network, effects, texts...

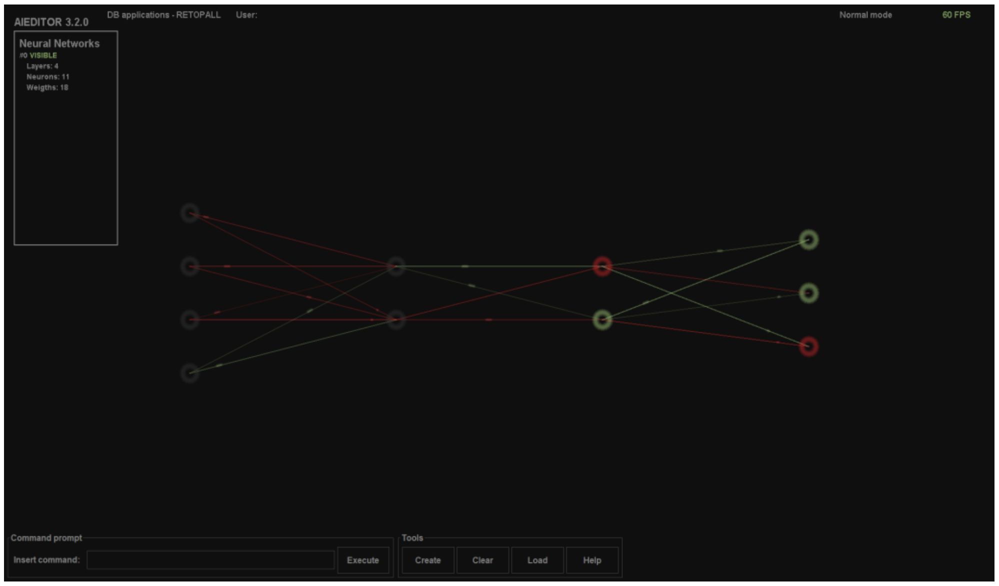
All the algorithms are executed in different threads of execution to prevent the decrease of the performance of the application and at the same time to manage several networks, trainings and other tasks that Requireren great computational performance.

This will be related to the number of objects to render. The larger the neural network, the more work it will cost the computer to maintain the fluidity of the program. Therefore, if the *FPS*[12] Decrease too much, the Program Soda rate. The application ceases to guarantee a continuous flow when the 10 million objects to be updated and the 100,000 to render are exceeded.

## 3.6 Neural network Storage

When training a neural network, you have the tool to save that neural network on a computer so that it can later be used by other people and/or by the user.

[12] *Frames For second* Or graphics that you can render a computer in a second



#0 VISIBLE
Layers: 4
Neurons: 11
Weigths: 18

Figure 8: Basic Neural network

All learning, adjustments, and all neurons, weights, layers, etc. will be saved. The neural network will be serializable to be stored in object form. In addition, to ensure data security, everything will be dynamically encrypted during program execution

## 4 Practices

In addition to programming the algorithms and designing the application I have been taking advantage of it creating different intelligent networks and evaluándo their performance. I've developed a neural network ready to recognize digits and letters written manually.

The images used to train the network are from 28x28 pixels passed to black and white. Therefore, the number of neurons in the input layer will be 784 neurons. The output neurons will be digits from 0 to 9. I've been varying the hidden layers and finally managed to get the best performance with a 784x20x20x10 network and it worked with a recognition accuracy of 99%. The network has 16280 pesos to modify in the training.

In order to analyze the performance of the network, I have implemented in the program a library that I have programmed in Java and that includes techniques of mathematical analysis, representation of functions, matrices and other tools. You can download the application and see the source code in Https://github.com/dDevTech/MathLibrary. The cost function of the neural network can be viewed in Figure 4. The polynomial function is the function of the error with the regression tool applied.

With what I have come to learn from artificial intelligence with this project I have created a simulator of autonomous cars using neural networks, but this time without back-propagation and with learning of effort and in particular with genetic algorithms. You can see a demo of the simulator in: https://www.youtube.com/watch?v=aC7Euu4s66A

The program is written in C# with the Unity 3d graphic engine [Figure 9].

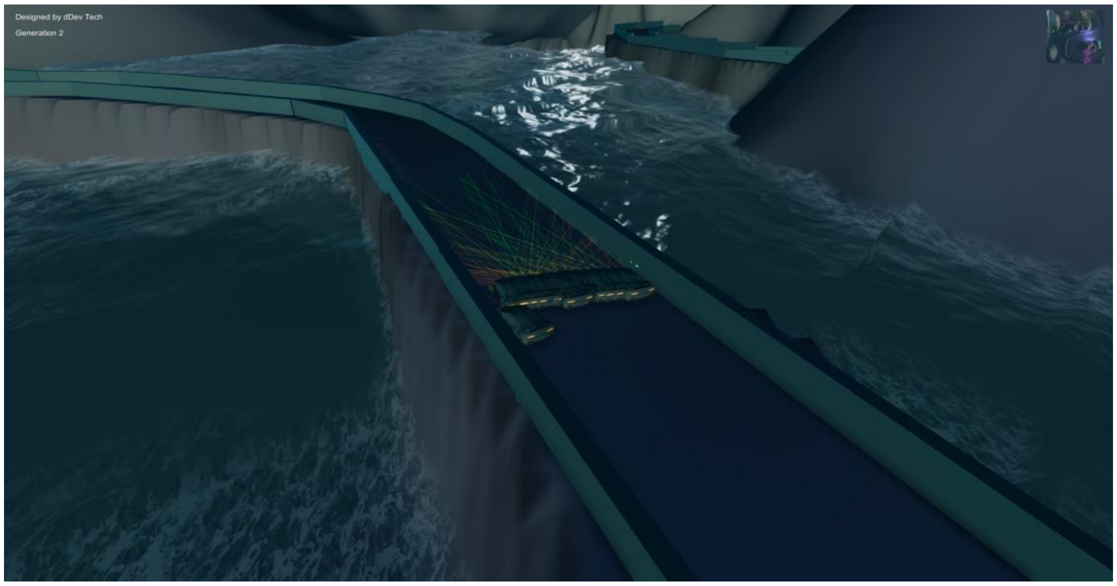


Figure 9:3d self-contained car simulator

# 5 Conclusion

Thanks to this application I have been able to know in depth the functioning of the neural networks, as well as being able to use it for other purposes and applications. For example to genetic algorithms. I have also observed that the fastest form of learning for numbers is through minibatches, with a low learning coefficient (0.1-0.2) and with the sigmoid activation function.

The application is available on my website so any interested person can download it. I am currently developing, thanks to the knowledge learned with this project, other applications of artificial intelligence using genetic algorithms and other models of neural networks as *Adagram* To make smarter applications. However, I continue to develop this application with new features so that anyone can take advantage *AIEditor* .

## Thanks

This project could not have been carried out without the support of some people who have helped me in my doubts and have worked hard to revise the project.

First of all to Maria Gaspar and Irene Gutierrez for reviewing the project several times and give me your opinion of it.

Juan Carlos Berrocal for listening to me when I explained the application without having knowledge of programming.

Javier Martín can not forget to help me in the doubts that have arisen during the elaboration of the project, along with Maria Gaspar.

## References

[1] Matt Mazur *Step by step propagation example* URL: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

[2] 3Blue1Brown *Explaining Neural Networks* URL: https://www.youtube.com/watch?v=aircAruvnKk
list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi

[3] deeplearning.ai *Mini-batch gradient descent* URL: https://www.coursera.org/lecture/deep-neural-network/mini-batch-gradient-descent-qcogH

[4] Michael Nielsen *Using Neural Nets to recognize hand written digits* URL: http://neuralnetworksanddeeplearning.com/chap1.html

[5] Data Science Group *Loss function and optimization algorithms* URL: https://medium.com/data-science-group-iitr/loss-functions-and-optimization-algorithms-demystified-bb92daff331c

[6] Toward Data Science *Optimization algorithms used in Neural Networks*

[7] Sebastian Ruder *An overview of gradient descent optimization algorithms* URL: http://ruder.io/optimizing-gradient-descent/

[8] Diego Berrocal *Artificial Intelligence Editor* Figures 1, 2, 4, 7, 8 self-

elaboration.

[9] Diego Berrocal *Autonomous car simulator in Unity 3d* Self-elaboration Figure 9.

[10] *Gradient Descent* Figure 3 Https://www.researchgate.net/figure/Optimizacion-gradiente-de-descenso4-Este-gradiente-es-hallado-despues-de-ejecutar-el_fig1_308783857Ir to image

[11] *Optimization algorithms* Figure 5 Http://ruder.io/optimizing-gradient-descent/Ir to image

[12] *Momentum* Figure 6 Https://www.sciencedirect.com/science/article/pii/S1319157818300636Ir to image

## Appendix

Next I will add links of interest related to the project, its download and platforms to view the code of applications and libraries used. In addition, I will include important project scheduling schedules.

# 1 Annex I: Links of interest

DDev Tech: YouTube channel where you can see the demos of the applications and the project that I developed.

https://www.youtube.com/channel/UCwsrCNLN4xQE9OVSP5Sz2dA

DDev Tech: Code of programming projects such as the Librerís of mathematics or connection to the server.

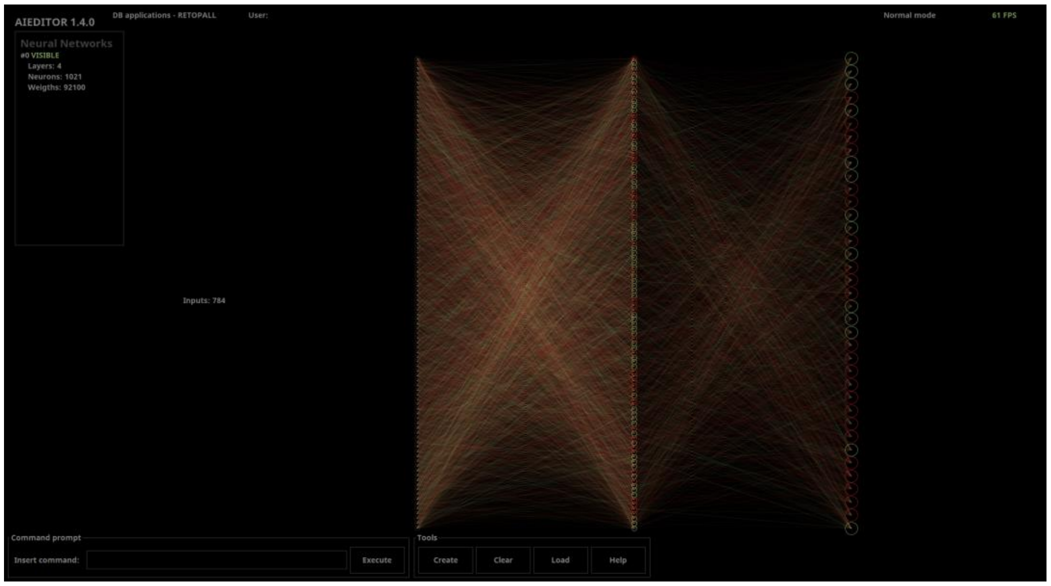
https://github.com/dDevTech

Retopall: website where I publish my projects and downloads of these.

https://retopall.com/

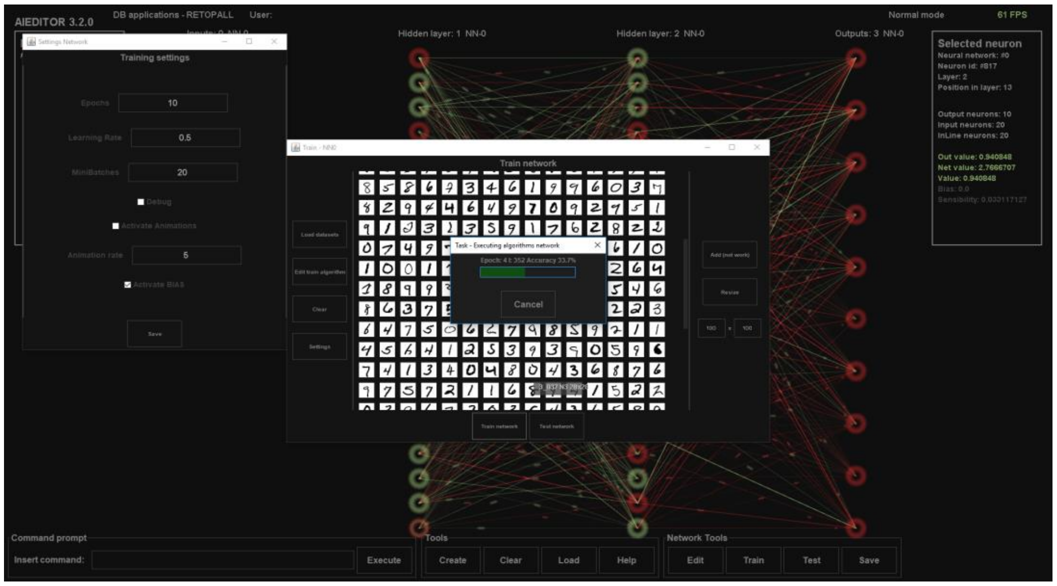
# 2 Annex II: Images of the network editor

The following images are screenshots of the AIEditor application:



Neural Networks
#0 VISIBLE
Layers: 4
Neurons: 1021
Weigths: 92100

Figure 10: Red neuronal 784x100x100x27



Training settings

Epochs: 10
Learning Rate: 0.5
MiniBatches: 20
Debug [ ]
Activate Animations [ ]
Animation rate: 5
Activate BIAS [✓]
Save

Epoch: 4 t: 352 Accuracy 33.7%

Selected neuron
Neural network: #0
Neuron id: #817
Layer: 2
Position in layer: 13

Output neurons: 10
Input neurons: 20
InLine neurons: 20

Out value: 0.940848
Net value: 2.7666707
Value: 0.940848
Bias: 0.0
Sensibility: 0.033117127

Figure 11: Neural Network Training

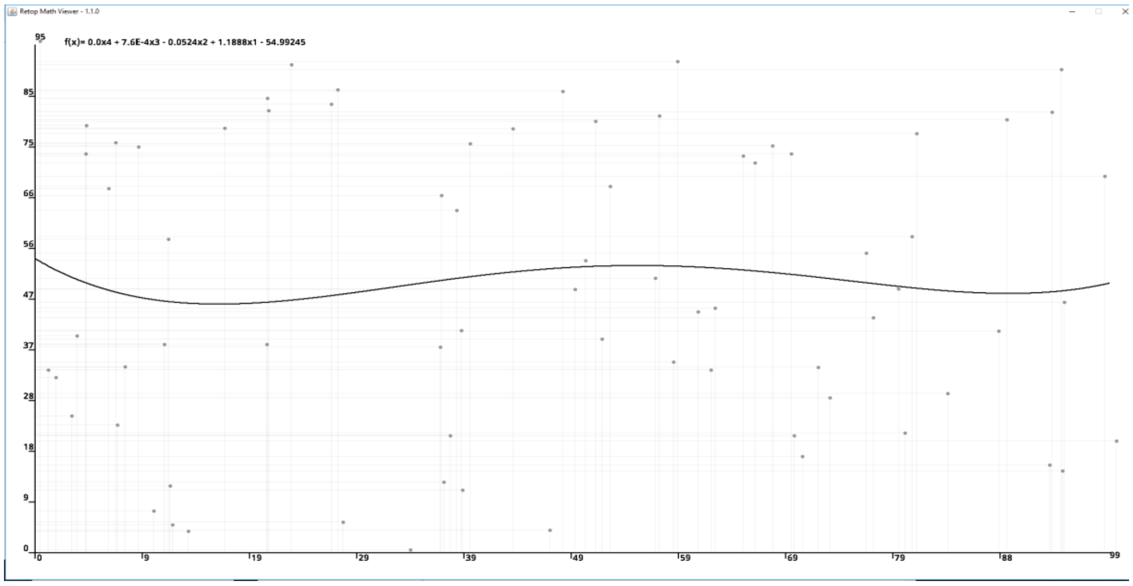
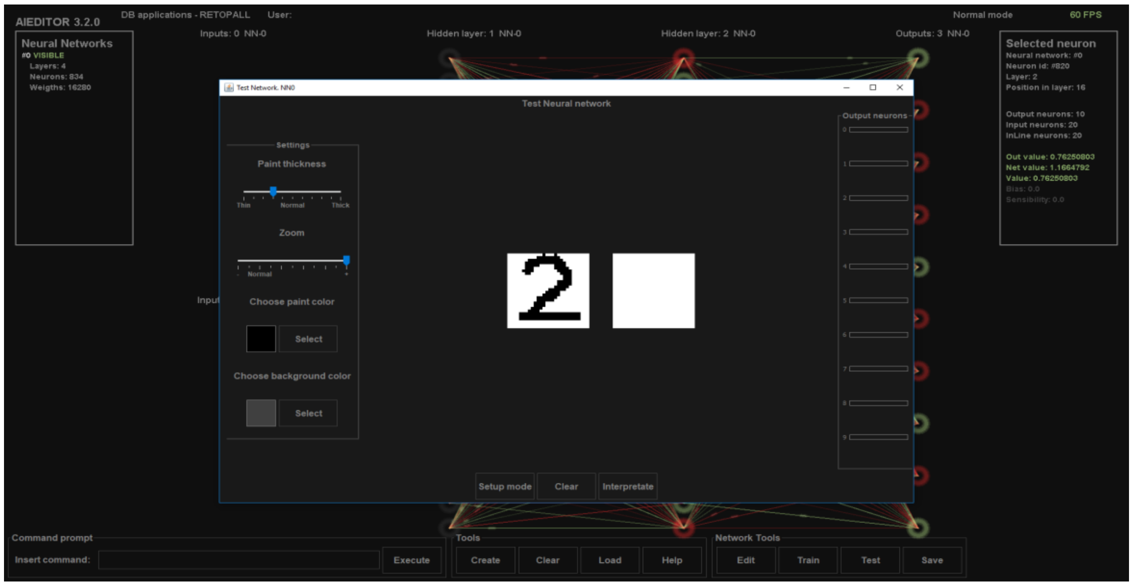


Figure 12: Regression of a data set



Neural Networks
#0 VISIBLE
Layers: 4
Neurons: 834
Weigths: 16280

Selected neuron
Neural network: #0
Neuron id: #820
Layer: 2
Position in layer: 16

Output neurons: 10
Input neurons: 20
InLine neurons: 20

Out value: 0.76250803
Net value: 1.1664792
Value: 0.76250803
Bias: 0.0
Sensibility: 0.0

Figure 13: Drawing numbers to test the functioning of the neural network