NEURAL NETWORKS PROJECT

Lucía Corpas
Javier Jordán
Illya Rozumovskyy
Jorge Velázquez

INTRODUCTION

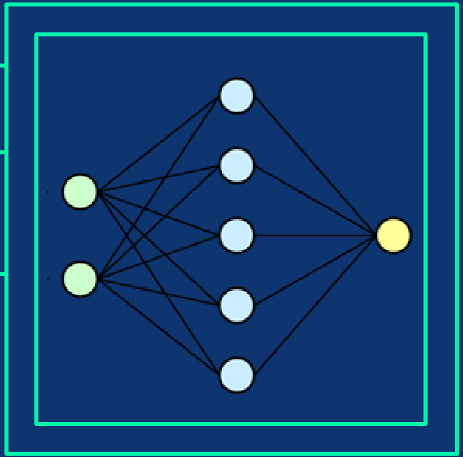
In this project, we have learned a function from several training patterns by using a multilayer perceptron neural network. The function that we have used has two real numbers (x,y) as inputs and a single real number as output:

$$\boxed{F(x,y) = \sin(x) * \cos(y)}$$

We assume that both inputs belong to the interval [-pi,pi]. The output is in the interval [-1,1].

We use the Encog library as the implementation of the multilayer perceptron:

https://www.heatonresearch.com/encog/



ENCOG LIBRARY

The Encog Library is a pure-Java/C# machine learning framework that was created back in 2008 to support genetic programming and other neural network technologies.

The source code for Encog can be much simpler to adapt for cases where you want to implement the neural network yourself from scratch. Some of the less mainstream technologies supported by Encog include NEAT, HyperNEAT, and Genetic Programming. Encog has minimal support for computer vision.

Encog supports a variety of advanced algorithms, as well as support classes to normalize and process data. Machine learning algorithms such as Support Vector Machines, Neural Networks, Bayesian Networks, Hidden Markov Models, Genetic Programming and Genetic Algorithms are supported. Most Encog training algorithms are multi-threaded and scale well to multicore hardware.

PROJECT STEPS

01

GENERATE AT RANDOM 1000 TRAINING SAMPLES OF THE FUNCTION

Generate the samples of (x,y) uniformly at random on the square [-pi,pi] x [-pi,pi]. Then compute the value of the function F(x,y) at those points.

02

GENERATE AT RANDOM 1000 VALIDATION SAMPLES OF THE FUNCTION

Generate the samples of (x,y) uniformly at random on the square [-pi,pi] x [-pi,pi]. Compute the value of the function F(x,y) at those points.

03

TRAIN A MULTILAYER PERCEPTRON WITH THE 1000 TRAINING SAMPLES OF THE FUNCTION

Choose the number of hidden neurons, the learning parameters, and the number of epochs. The training error and the validation error must be printed out for each training epoch.

PROJECT STEPS

04

GENERATE TEST SAMPLES OF THE FUNCTION

Generate the test samples of (x,y) at equally spaced locations in a grid of 100x100 points on the square [-pi,pi] x [-pi,pi]. Then compute the value of the function F(x,y) at those points.

05

SIMULATE THE MULTILAYER PERCEPTRON ON THE TEST SAMPLES

Compute the mean squared error for the test samples, and print it out on the console

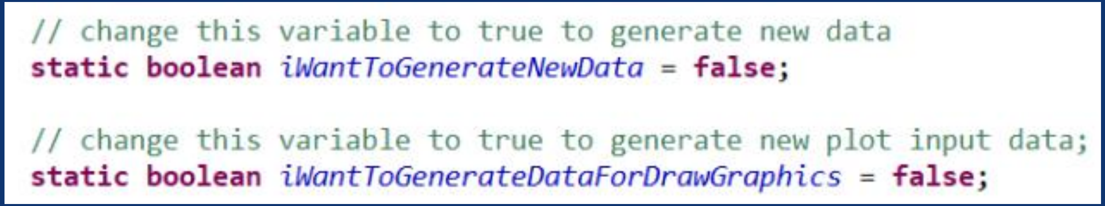
06

GENERATE ERROR PLOTS & 3D GRAPH

Generate a plot to show the evolution of the training error and the validation error at each training epoch

Generate a 3D figure with the true function and the learned function. Both must be plotted on a grid of 100x100 points on the square [-pi,pi] x [-pi,pi]

PROGRAM CODE



```java
// change this variable to true to generate new data
static boolean iWantToGenerateNewData = false;

// change this variable to true to generate new plot input data;
static boolean iWantToGenerateDataForDrawGraphics = false;
```

When you activate the first function new data sets will be generated and stored in the file.

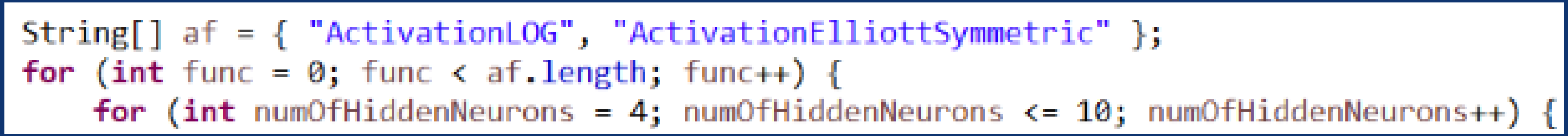
If you already have the data and the variable iWantToGenerateNewData is set to false, the data will be read from the files. It was made to simulate, to some extent, the real data input.

Data is processed using methods of the class DataProcessing.java.

When you activate the variable iWantToGenerateDataForDrawGraphics the NN will be created using different settings. For each model and each epoch the training error and validation error will be stored in the corresponding file.

It is done this way to provide the user different options when choosing the right settings based on the generated data. We can visualize this generated data in the form of graphics or diagrams.

PROGRAM CODE



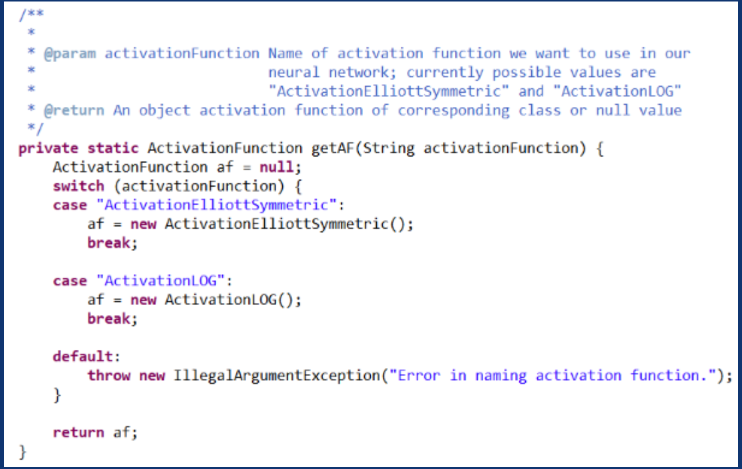
```
String[] af = { "ActivationLOG", "ActivationElliottSymmetric" };
for (int func = 0; func < af.length; func++) {
    for (int numOfHiddenNeurons = 4; numOfHiddenNeurons <= 10; numOfHiddenNeurons++) {
```

You can add to the string array the different activation function in order to test the NN with different settings.

The testing range of the hidden layer can be adjusted changing the for-loop range.

Currently it only supports two listed functions, but if you want to add more you need to modify the ActivationFuncion method in “method” package, adding a new corresponding case.

PROGRAM CODE



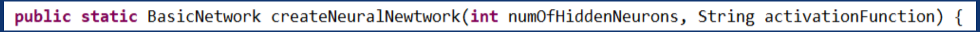
```java
/**
 *
 * @param activationFunction Name of activation function we want to use in our
 *                           neural network; currently possible values are
 *                           "ActivationElliottSymmetric" and "ActivationLOG"
 * @return An object activation function of corresponding class or null value
 */
private static ActivationFunction getAF(String activationFunction) {
    ActivationFunction af = null;
    switch (activationFunction) {
    case "ActivationElliottSymmetric":
        af = new ActivationElliottSymmetric();
        break;

    case "ActivationLOG":
        af = new ActivationLOG();
        break;

    default:
        throw new IllegalArgumentException("Error in naming activation function.");
    }

    return af;
}
```

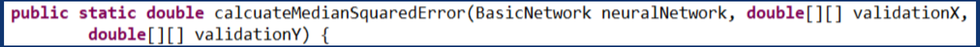
The creation of NN is done in the next method from NeuralNetwork.java class in “method” package:



```java
public static BasicNetwork createNeuralNewtwork(int numOfHiddenNeurons, String activationFunction) {
```

After analyzing the generated data in the training step, it was observed that the number of epochs have a positive impact on the NN prediction capabilities. The generated curves show that the very large number of epochs do not give substantial improvement, but at the same time creates unnecessary load to the hardware of the machine. It was decided to aim for 2% accuracy and then, about 750 epochs were needed for the NN. The NN has 2 hidden layers with 10 neurons on each level with a Logarithmic activation function.

To calculate the error we use the next method from NeuralNetwork.java class in “method” package:



```java
public static double calcuateMedianSquaredError(BasicNetwork neuralNetwork, double[][] validationX,
        double[][] validationY) {
```

We used a straightforward application of the MSE formula $\frac{1}{N} * \sum_{0}^{N-1}(y' - y)^2$ implemented in java language. Where ‘y’ is the real output of the formula provided in validationY dataset, ‘y’’ is the calculated by NN output.

PROGRAM CODE

If you deactivate the variable *iWantToGenerateDataForDrawGraphics* the program will jump directly to the analysis of the test set.

It was decided to use manual setted parameters for the final model of the neural network, in order to provide consistent output.

The network being trained using given parameters and the test set data is given for the analysis.

The results are documented and used to build the 3-dimensional plot.

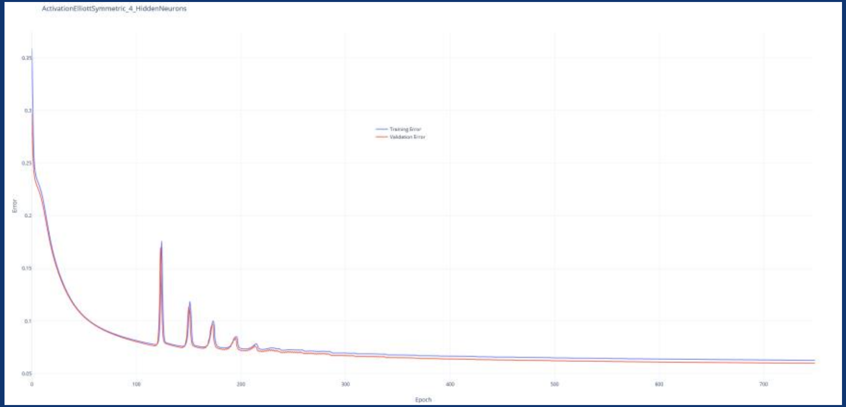
ERROR PLOTS

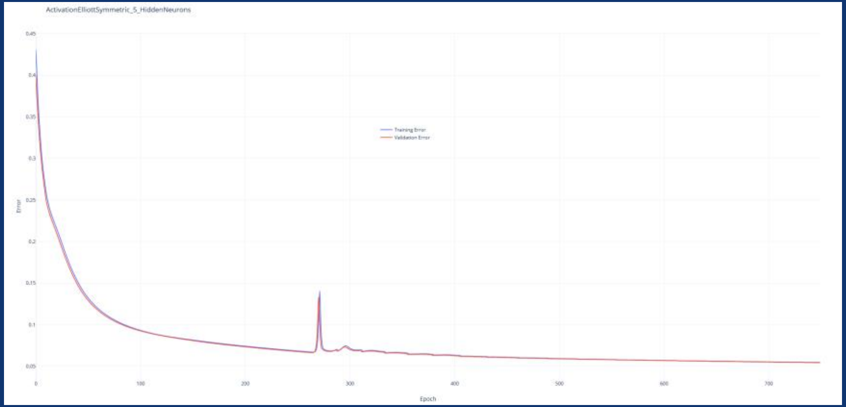
To check which is the best setting for the neural network, we have done several error plots with different activation functions. We used two hidden layers with different number of neurons. The number of neurons in the hidden layers go from 4 up to 10. We have chosen between two different activation functions:

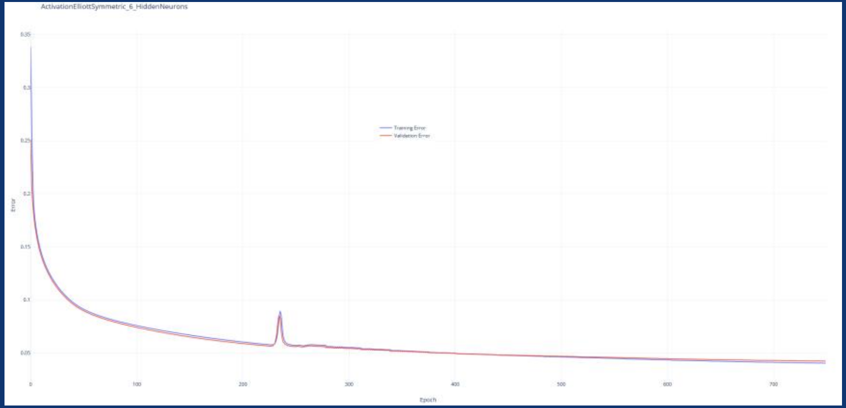
- **ACTIVATION ELLIOTT SYMMETRIC**
- **ACTIVATION LOGARITHM**

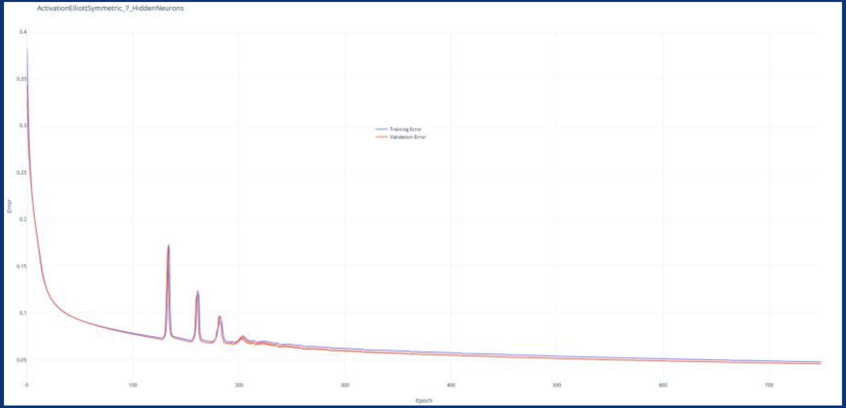
After seeing all possible plots, we choose the one which adapts better to our requirements.

ACTIVATION ELLIOTT SYMMETRIC

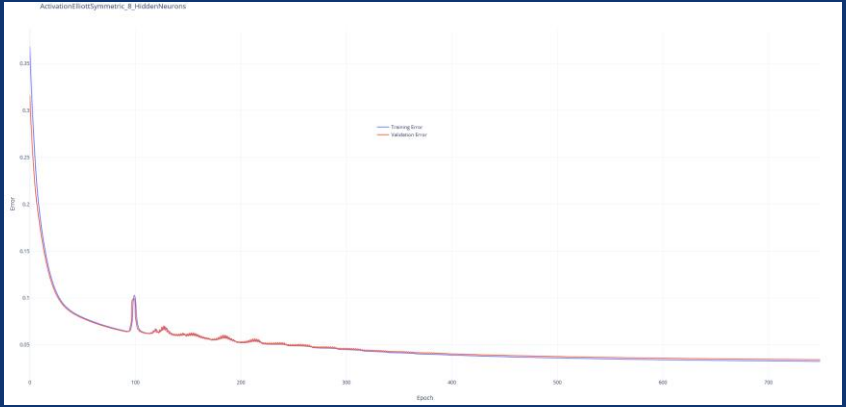


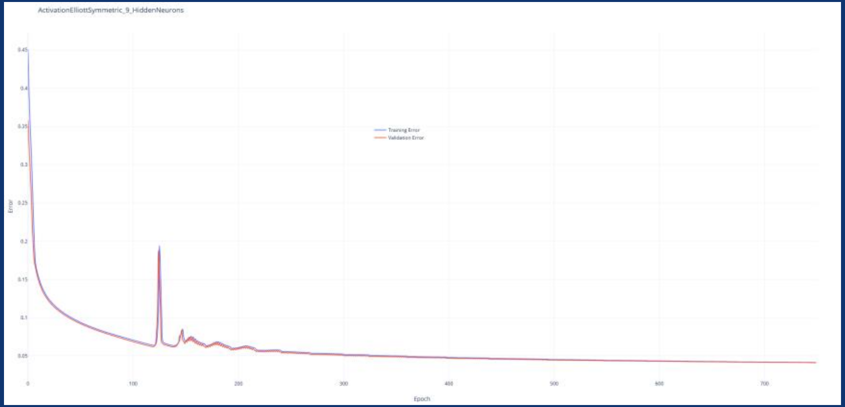


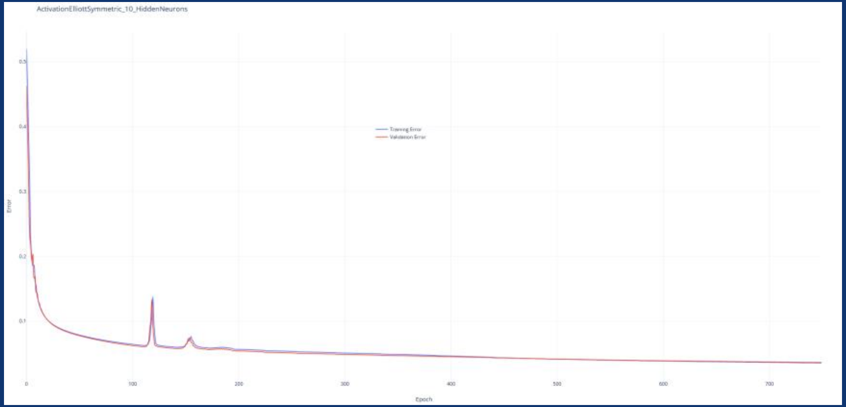




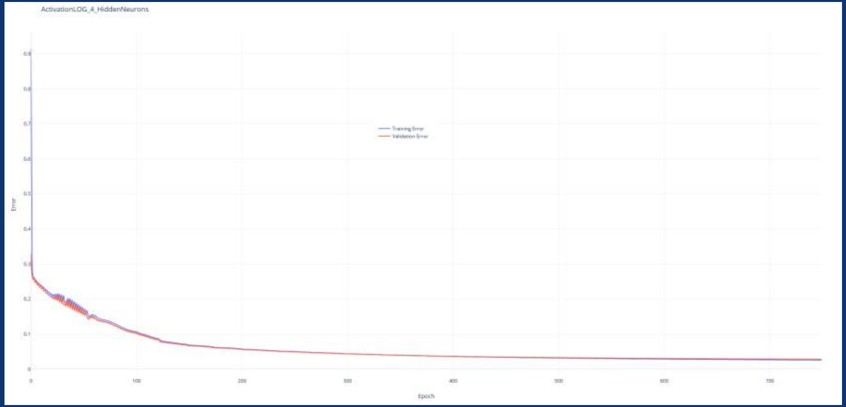
ACTIVATION ELLIOTT SYMMETRIC

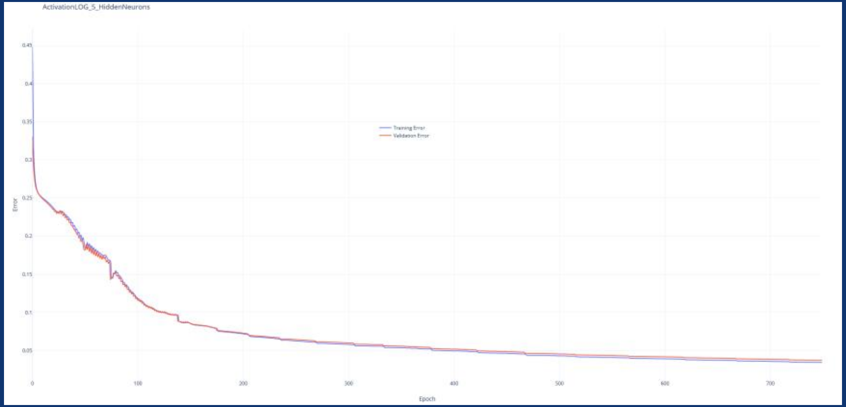


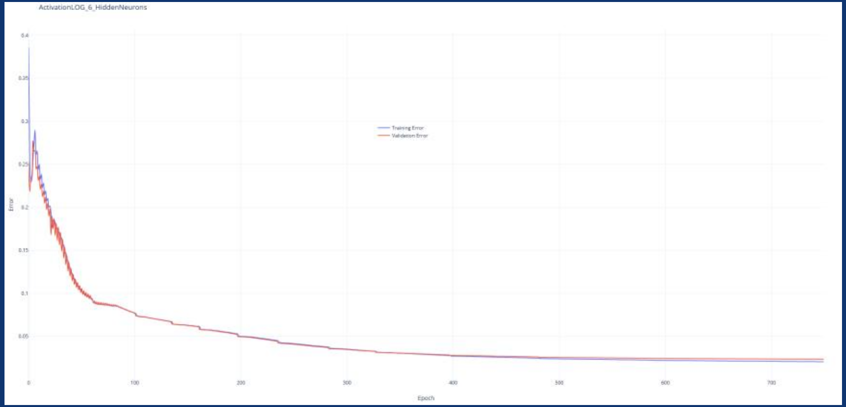


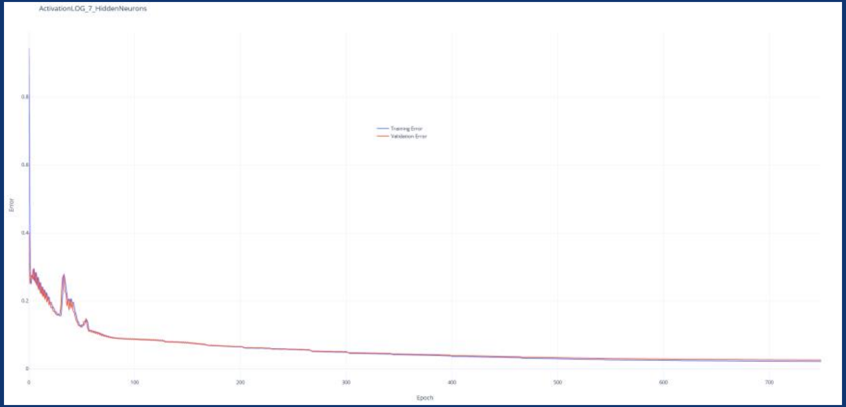


ACTIVATION LOGARITHM

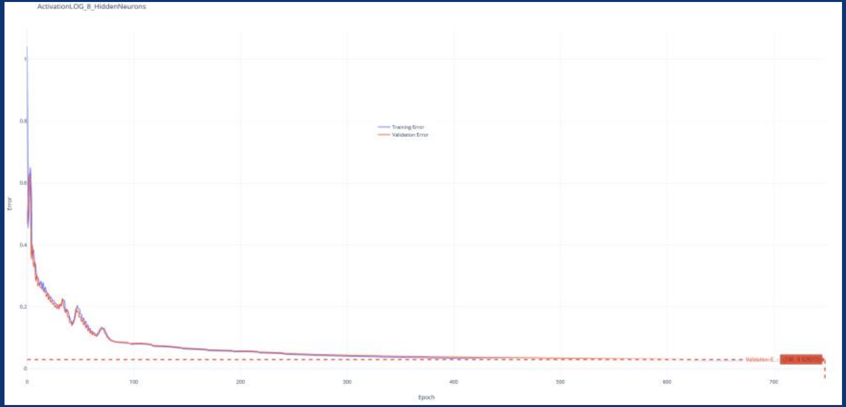


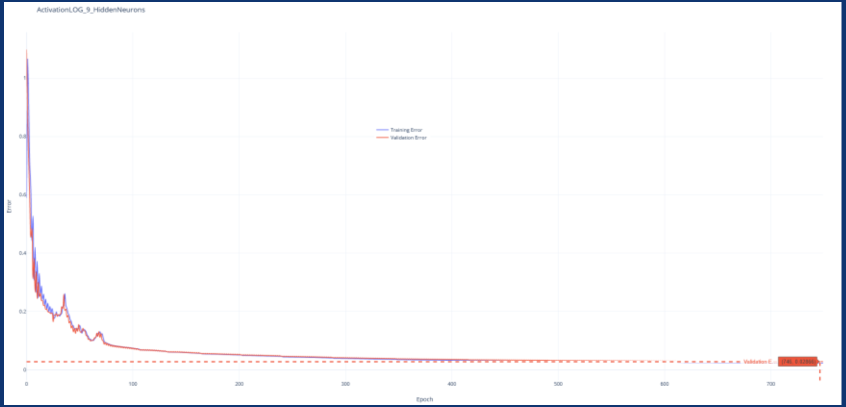


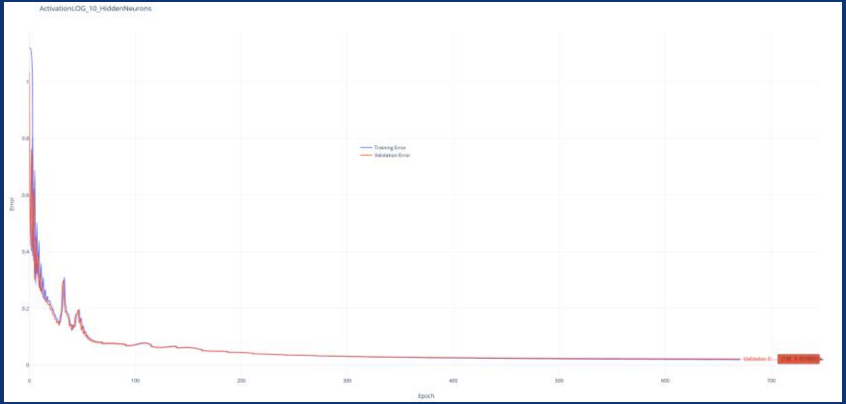




ACTIVATION LOGARITHM

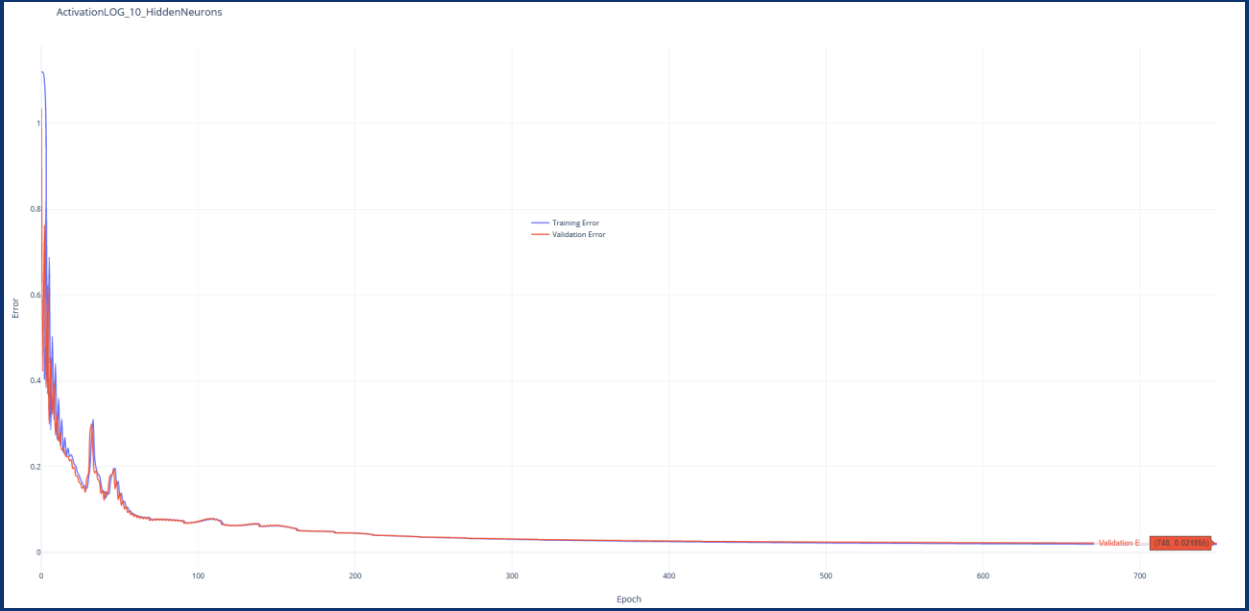






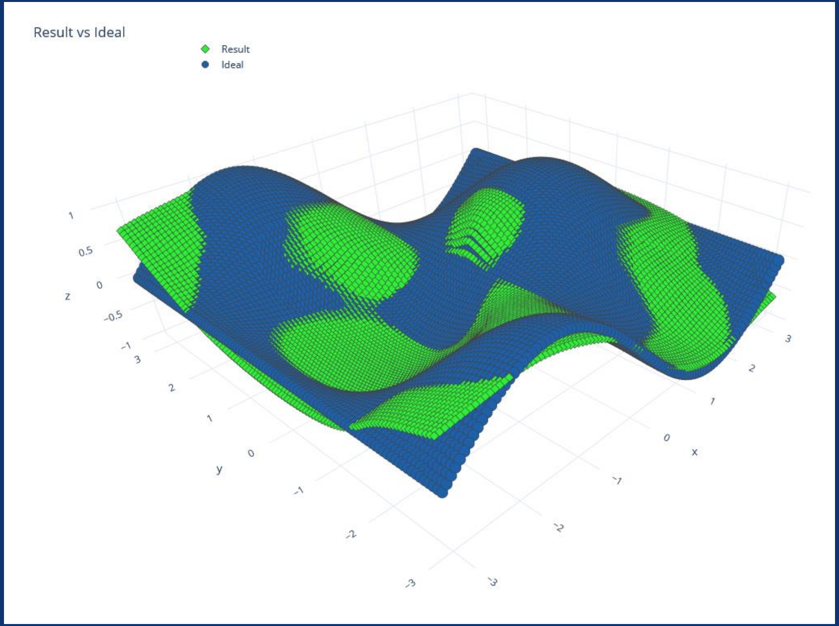
SELECTED PLOT

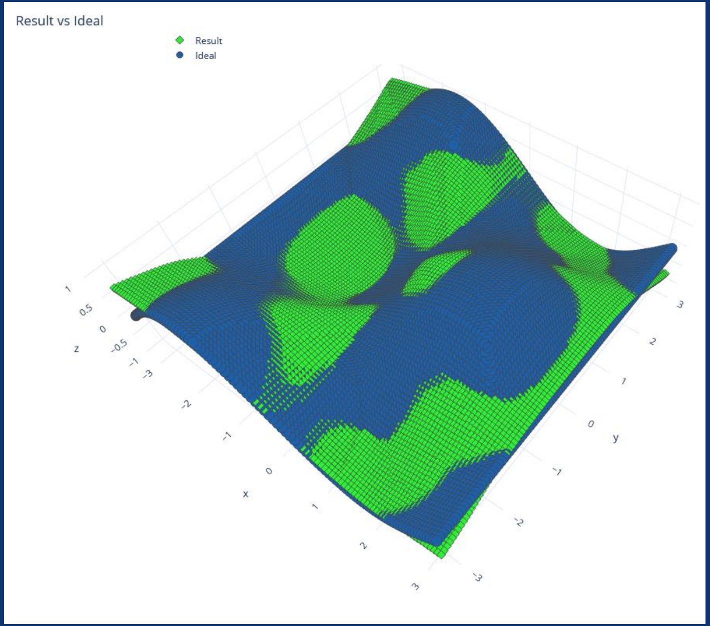
The selected plot is the logarithm plot with 10 hidden neurons, because it is the one that show us the minimum validation error with the minimum number of iterations:



3D GRAPH

We generated a 3D graph with the true function and the learned function to compare our solution with the ideal one.


Result vs Ideal
Result
Ideal
z
y
x


Result vs Ideal
Result
Ideal
z
x
y