ISSN: 2769-545X

Research Article

Advances in Machine Learning & Artificial Intelligence

Straight Forward Constructive Deep Learning Neural Network (SFC-DLNN) Algorithm

Ndom Francis Rollin[1*], Mveh-Abia Chantal[3], Ayissi Raoul[2], Etoua Remy[1] and Emvudu Yves[2]

[1*]*Department of Mathematics and Physical Sciences, National Advanced School of Engineering of the University of Yaounde I, Melen, Yaoundé, 8390, Cameroon.*

[2]*Department of Mathematics, Faculty of Sciences of the University of Yaounde I, Ngoa-Ekelle, Yaoundé, 812, Cameroon.*

[3]*Department of Computer Engineering, National Advanced School of Engineering of the University of Yaounde I, Melen, Yaoundé, 8390, Cameroon.*

[*]Corresponding author
Ndom Francis Rollin, Department of Mathematics and Physical Sciences, National Advanced School of Engineering of the University of Yaounde I, Melen, Yaoundé, 8390, Cameroon

Submitted: 19 Aug 2022; **Accepted**: 24 Aug 2022; **Published**: 27 Aug 2022

Citation: *Rollin NF*, Chantal MA, Raoul A, Remy E and Yves E. (2022). Straight Forward Constructive Deep Learning Neural Network (Sfc-Dlnn) Algorithm. Adv Mach Lear Art Inte, 3(2): 80 -91.*

Abstract

Straight Forward Constructive Deep Learning Neural Network (SFC-DLNN) algorithm is a new architecture-based algorithm for artificial neural networks. Rather than simply adjusting the weights in a fixed topology network, SFC-DLNN starts with a minimal topology (perceptron), then builds up their network by gradually trains and adds new nodes one by one, creating multiple layers' network. Once a unit has been added to the network, the weights of the new architecture are generated. This unit then stands as a permanent detector of features in the network, and a more complex feature space is then created where the data is likely to be linearly separable. The SFC-DLNN algorithm has many advantages over existing ones: it has good learning speed, the network determines its topology size, and the structures it has built is retained after the training stage.

We obtain from our built model (SFC-DLNN) an accuracy and specificity of 83:5% from a simulated data set using the uniform distribution. This is not the best but is enough to approve the model prediction capacity.

Keywords: Neural Networks, Feed-Forward Algorithm, Cascade-Correlation Growing, Deep Learning, Neural Network Algorithm

Introduction

Deep learning is generally considered a breakthrough technology in the last decade. It is a growing trend in general data analysis and relies on artificial neural networks, which are computational models composed of several layers programmed "neuron" (single programmed "neuron" is known as Perceptron and multiply programmed "neuron" as Multi-Layer Perceptron (MLP)).

Indeed, MLP is a Deep Neural Network (DNN) that uses the feedforward mode of information now very popular in many applications because of its stable and strong architecture used in several available training algorithms.

These networks however encounter some problems when their depth rises with a lot of hidden layers and several links between them. Irregular initialization has an impact on the convergence process because it can slow down or even completely stall the convergence process. Therefore, network initialization is one of the major problems faced by DNNs [1-4]. The following problem is the difficulty of discovering a suitable network architecture that can give good accuracy and better success in generalization. Although many reasons can push for under fitting or overfitting neural network models, one of these is the unsuitable architecture of the network. Thus, network architecture sizing is required for better performance in generalization and fast convergence of the training algorithm.

The lack of knowledge on some problems bearing a huge amount of data has made a neural network with a fixed architecture less preferable than a constructive one that performs dynamically the neural network architecture. So, the network architecture is self-built during the training process although requiring some external parameters like the maximum number of neurons per layer.

By definition, a constructive algorithm starts a neural network with a limited number of hidden layers, neurons, and links, then gradually attaches hidden nodes and weight until an optimal structure is identified. There are usually used for classification problems and have been shown to result in good architecture shaping and faster learning [5].

In this class of constructive algorithms, we find Cascade-Correlation Neural Network (CCNN) known to be fast, as the new candidate neuron is learned just before connecting to the network, and the weight of the hidden input nodes is frozen during the learning. So, the weight is frequently trained at the output nodes after each new hidden neuron has been added.

Cascade-Correlation Growing Deep Learning Neural Network (CCG-DLNN) is a method proposed by, which uses the growing phase at the "Growing Pruning Deep Neural Network Algorithm" proposed by, however by training the latest hidden unit (a candidate unit) when connecting to an existing model, and after that, the weight of the hidden inputs are frozen [1, 6].

We proposed a method called "straight forward constructive deep learning neural network algorithm (SFC-DLNN)" which uses the growing phase or feed-forward at "growing pruning deep learning neural network algorithm (GP-DLNN)", however, by training the latest hidden unit (a candidate unit) when connecting to an existing model, and after that, the weights of the hidden inputs are not frozen. Consequently, the weights of the output neurons are constantly trained after adding each new hidden neuron as in the CCNN algorithm. However, the difference here with the CCG-DLNN is that firstly, the cascade architecture adds the hidden node to the neural network each time with a maximum number per layer and it is not changed after inserting it; secondly, it is a learning algorithm that creates and adds a new hidden neuron based on the cost error.

This work aims to find the best and most suitable architecture of a Neural Network (Multi-Layer Perceptron) for a given data set. To understand how interesting, it is, we should remind that the Support Vector Machine's main limit is finding "a better space" or the transformation $\emptyset(x)$ where the data is separable. As the MLP offers the possibility to find both this transformation $\emptyset(x)$ and the parameters for the regression or classification purpose. The stake of suitable architecture in NNs is to solve the problem of dimensionality and to approximate a large class of functions with regularity that the NNs can capture [7].

The remaining document is organized in the following format. Section 2 presents, on the one-hand side, the architecture of neural networks from simple perceptron to deep neural networks, and on the second-hand side, it gives the SFC-DLNN algorithm syntax and organigram. Moreover, section 2 also describes two theorems based on the convergence state of the SFC-DLNN algorithm and their proofs. Section 3 presents the data sets used in the experimentation phase, gives the simulation results and, the discussion of our results is the aim of section 4. Finally, section 5 deals with the conclusion and perspectives of the work.

Straight Forward Constructive Deep Learning Neural Network (SFC-DLNN) algorithm

Deep Neural Network Presentation

The graphs below are the respective representations of a basic architecture of a neural network with two layers (input and output layers) called "simple Perceptron" (figure 1) and the deep neural network with four layers (figure 2), namely: the input layer which takes large volumes of input data in the form of audio files, texts, numbers, image pixels, etc; the hidden layers liable to perform mathematical operations, pattern analysis, feature extraction, etc; the output layer is responsible for generating through mathematical operations the desired output.

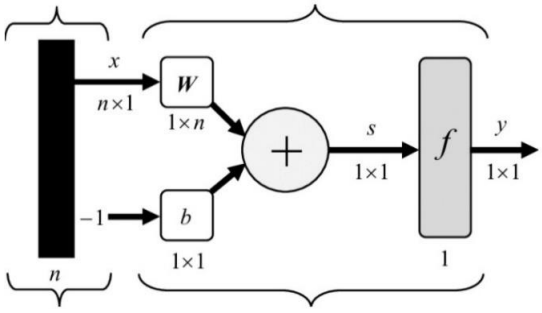


Figure 1: Basic Architecture of a Neural Network with a Single Node in the Output Layer

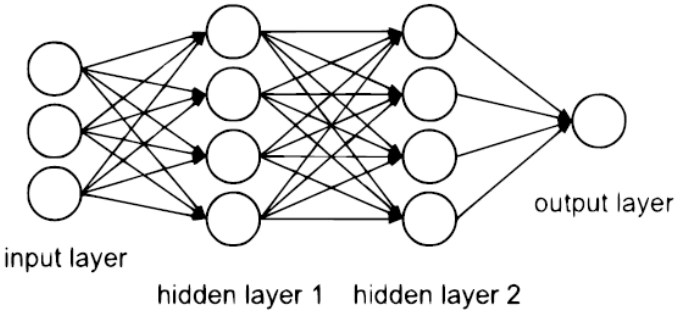


Figure 2: Architecture of a Multi-Layer Neural Network with Two Hidden Layers and One Output

Straight Forward Constructive Deep Learning Neural Network ALGORITHM

The SFC-DLNN algorithm takes as input are given data set and splits it into training and test sets. Then the training set is used to construct the NN by training and adding new hidden units one by one, starting with the single perceptron architecture and creating a multi-layer structure. The test set will be used for the validation (cross-validation) of the newly constructed model.

In the SFC-DLNN algorithm, the training and adding of new hidden units is based on the cost function. Generally, in machine learning, we use loss functions to evaluate how well the model is to the data. The cost function (mean of the lost function over the sample data) is optimized through the gradient method. According to [9], we can categorize loss functions into three: regression loss functions; binary classification loss functions, and multiclass classification loss functions. The entropy cost is a binary classification cost function and is used for the purpose. So, in detail we can see the following input variables:

X: training data set;
M: the training data set size;
Num_iter: iteration number for the NN training;
T: the index of training step;
L: currently hidden layer index;
Wl : Matrix of the links from layers l to (l -1);
Lt: The cost function at the t step;
Eps: The threshold Error;
lr: The training rate;
ME: maximum error (which is acceptable);
Tau: adding a new hidden layer Threshold;
C: maximum of neurons possible in a layer, after which a new layer is added.

Algorithm 1: SFC-DLNN (X; y; M; eps; ME; tau; lr)
Input: X,y, M, eps, ME, tau, lr
Output: Deep Neural Network

Begin

1. initlayer size < ------- initializeLayerSize (X; y; hidden layer = c(1)) // return a list with the training data and lazer sizes
2. init_params < ------ initializeP arameters(X; initlayer size)
3. fwd_prop < ------- fwdPropagation(X; init params; initlayer size)
4. cost_history = c() // declares an empty vector
5. cost_history[1]=0
6. cost < -------- computeCost(X; y; fwd prop; initlayer size)
7. back_prop < ------ backwdPropagation(X; y; fwd prop; init params; initlayer size; cost history; i)
8. update_params < ------ updateP aramet(X; y; fwd prop; back prop; init params; lr; initlayer size)
9. updatelayer _ize < ------ updateLayerSize(initlayer size; cost history; i)
10. trainModel < ------ function(X; y;num iter; hidden layer; lr) {
11. init_params < ------ update params
12. initlayer_size < ------ updatelayer size
13. for (i in 1:num iter) {
14. fwd_prop < ------ fwdPropagation(X; init params; initlayer size)
15. cost < ------ computeCost(X; y; fwd prop; initlayer size)
16. cost history[i + 1] cost
17. back_prop < ------ backwdPropagation(X; y; fwd prop; init params; initlayer size; cost history; i)
18. update_params < ------ updateP aramet(X; y; fwd prop; back prop; init params; lr; initlayer size)
19. 19: init_params < ------ update_params
20. updatelayer_size < ------ updateLayerSize(initlayer size; cost history; i)
21. initlayer_size < ------ updatelayer size }
22. classif < ------ classifun(X; y; updatelayer size; initlayer size; cost history; fwdPropagation; Init_params; back prop;num iter)
23. model_out < ------ list("updated params" = update params; "cost hist" = cost history; "layersize" = updatelayer size; "fwd prop" = fwd prop; "back prop" = back prop, "classif"=classif)
24. return (model out) }

End

Theorem 1
For neurons within the newly added neurons in the layer l (l > 1) and the output neurons in the Straight Forward Constructive Deep Learning Neural Network (SFC-DLNN) algorithm, there exists an adjustment of weight such that the cost error given by the network, after this addition of the N neurons to the l layer, is less than the one before (i.e., $L_t < L_t$-1).

Proof

Let assume l=2

At iteration t, the entropy cost is defined as:

$$L_t^l = \frac{-1}{n}\sum_{i=1}^{n} y_i ln(\sigma(w^{(l)} \cdot x_i + b^{(l)})) + (1-y_i) ln\left(1-\sigma(w^{(l)} \cdot x_i + b^{(l)})\right)$$

Knowing the sigmoid function σ is defined from R to σ(R)=[0,1], we can the following approximation:

$$ln\left(1-\sigma(w^{(l)} \cdot x_i + b^{(l)})\right) \sim -\sigma(w^{(l)} \cdot x_i + b^{(l)})$$

and $ln(\sigma(w^{(l)} \cdot x_i + b^{(l)})) = ln(1-\left(1-\sigma(w^{(l)} \cdot x_i + b^{(l)})\right) \sim -1+\sigma(w^{(l)} \cdot x_i + b^{(l)})$

also, if x is near 0, we have: $\sigma(x) = \frac{1}{1+e^{-x}} \sim \frac{1}{1+1-x} = \frac{1}{2}\frac{1}{1-x/2} \sim \frac{1}{2}(1-x/2) = \frac{1}{2}-\frac{x}{4}$

so,

$$L_t^l \sim \frac{-1}{n}\sum_{i=1}^{n}[y_i(-1+\sigma(w^{(l)} \cdot x_i + b^{(l)})) - (1-y_i)\sigma(w^{(l)} \cdot x_i + b^{(l)})]$$

and

$$L_t^l \sim \frac{-1}{n}\sum_{i=1}^{n}[-y_i + (-1+2y_i)\sigma(w^{(l)} \cdot x_i + b^{(l)})]$$

Then

$$\begin{aligned} L_t^l &\sim \frac{-1}{n}\sum_{i=1}^{n}[-y_i + (-1+2y_i)(\frac{1}{2}-\frac{1}{4}(w^{(l)} \cdot x_i + b^{(l)}))] \\ &= \frac{-1}{n}\sum_{i=1}^{n}[\frac{-1}{2} + (\frac{1}{4}-\frac{1}{2}y_i)(w^{(l)} \cdot x_i + b^{(l)})] \\ &= \frac{1}{2}-\frac{1}{2n}\sum_{i=1}^{n}(\frac{1}{2}-y_i)(w^{(l)} \cdot x_i + b^{(l)}) \\ &= \frac{1}{2}-\frac{1}{4n}\sum_{i=1}^{n}(w^{(l)} \cdot x_i + b^{(l)}) + \frac{1}{2n}\sum_{i=1}^{n} y_i(w^{(l)} \cdot x_i + b^{(l)}) \\ &= \frac{1}{2}-\frac{1}{4}\hat{\bar{y}}_n + \frac{1}{2n}\sum_{i=1}^{n} y_i(w^{(l)} \cdot x_i + b^{(l)}) \\ &= \frac{1}{2}-\frac{1}{4}\hat{\bar{y}}_n + \frac{1}{2}L_t^{l-1} \end{aligned}$$

As all the individuals are not yet classified at the l-1 layer, we have

$$L_t^l \sim \frac{1}{2}-\frac{1}{4}\hat{\bar{y}}_n + \frac{1}{2}L_t^{l-1} < L_t^{l-1}$$

Corollary

The first l-1 layer of the Straight Forward Constructive Deep Learning Neural Network (SFC-DLNN) algorithm help to find the suitable initial point of convergence.

Theorem 2

There exists a layer number l (l>1) where, from this layer the Straight Forward Constructive Deep Learning Neural Network (SFC-DLNN) algorithm converges after passing this layer.

Proof

Let the cost error Lt and the layer number l, the Lt Gradient can be defined as

$$\nabla_W L_t = \frac{\partial L_t}{\partial w_{ij}^l} \quad (1)$$

$$\nabla_b L_t = \frac{\partial L_t}{\partial b_{ij}^l} \quad (2)$$

In terminology, a single mathematical neuron is called perceptron. We can also refer to a Single Layer Perceptron as "unit".

For A Single Layer Perceptron

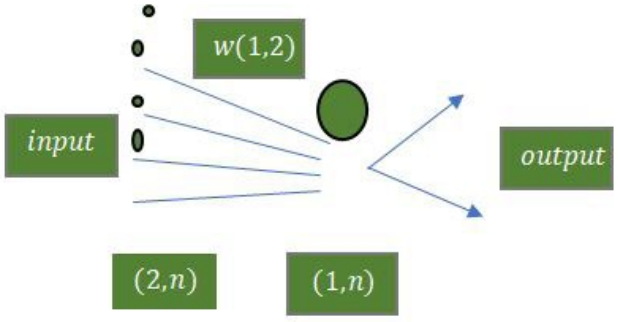


Figure Architecture of a Single Layer Perceptron

- A vector input X= (X1, X2……X_n) of length N which models the inputs of units;
- A vector of weight W= (W1, W2, ….,W_n) also of length N, which models all the synaptic weight of the unit. Each component Wi gives the weight of corresponding to the i'th entry of X;
- A bias b that correspond to the activation threshold of the unit.
- For classification purpose, the cost is evaluated in term of number of misclassified elements:

$$K = \text{which}((W\%*\%X + b)*y<0) \quad (3)$$

$$\nabla_{W_i^t} L_t = dW_i^t = t(y)[i]*t(X)[i,] \; ; \; i \in K \quad (4)$$

$$\nabla_{b_i} L_t = db_i = y[i] \; ; i \in K \quad (5)$$

$$W^{t+1} = W^t + \nabla_{W_i^t} L_t \; ; i \in K \text{ pick arbitrary} \quad (6)$$

ASSUMPTION 1

If there exists R∈R and γ∈R, such that for $i \in \{1,2,\ldots,M\}$

$$\| x^i \| \leq R \text{ and } t(y)[i] * t(X)[i,] > \gamma$$

Then the algorithm converges within this single neuron. This is known as the NOVIKOV Theorem.

For A New Neuron Added In the Layer

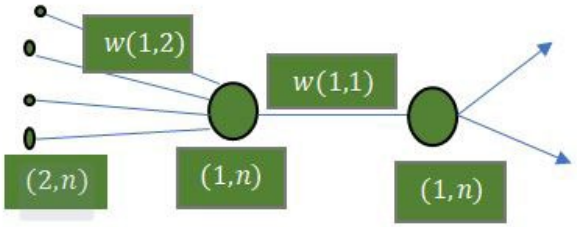


Figure Architecture of a Neural Network with One Hidden Neuron

In the case we add a neuron, there will be a hidden layer with a single neuron, the backpropagation algorithm, merely use for calculating the parameters gradient stated as dW and db, is challenging because its depends on the NNs architecture. In general, we use the chain rule if there is a hidden layer in the NNs architecture and update the weights using Equation (4):

$$w_{ij}^{l} - w_{ij}^{l-1} = \eta \nabla_W L_t \quad (4)$$

Where the learning rate is η and l >1.
The cost function of the first layer uses the input vector x and the activation functionσ.
At iteration t, we calculate the cost

$$L_t = \frac{1}{n}\sum_{i=1}^{n}[y_i ln(\sigma(w^t \cdot x_i + b)) + (1 - y_i) ln(1 - \sigma(w^t \cdot x_i + b))] \quad (5)$$

The gradient is calculated using the chain rule:

$$\nabla_W L_t = \frac{\partial L_t}{\partial w} = \frac{\partial L_t}{\partial \sigma}\frac{\partial \sigma}{\partial w} \quad (6)$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i}{\sigma(w^t \cdot x_i + b)} - \frac{1 - y_i}{1 - \sigma(w^t \cdot x_i + b)}\right)\frac{\partial \sigma}{\partial w}$$

$$= \frac{1}{n}\sum_{i=1}^{n} x_i(\sigma(w^t \cdot x_i + b^t) - y_i)$$

At iteration t+1, we update the parameters by

$$w^{t+1} = w^t + \frac{1}{n}\sum_{i=1}^{n} x_i(\sigma(w^t \cdot x_i + b^t) - y_i)$$

$$||w^{t+1}||^2 = ||w^t + \tfrac{1}{n}\textstyle\sum_{i=1}^{n} x_i(\sigma(w^t \cdot x_i + b^t) - y_i)||^2 \quad (5)$$

$$= ||w||^2 + \tfrac{1}{n^2}||\textstyle\sum_{i=1}^{n} x_i\big(-y_i + \sigma(w \cdot x_i + b)\big)||^2 + 2\tfrac{1}{n}\textstyle\sum x_i(\sigma(w^t \cdot x_i + b) - y_i)w^t \quad (6)$$

By setting $-y_i + \gamma(w^t \cdot x_i + b) = \hat{y}_i - y_i$ (7)

We have the following:

$$||w^{t+1}||^2 = \big||w^t|\big|^2 + \tfrac{1}{n^2}||\textstyle\sum_{i=1}^{n} x_i(\hat{y}_i - y_i)||^2 + \tfrac{2}{n}\textstyle\sum w^t \cdot x_i(\hat{y}_i - y_i) \quad (8)$$

Let $u = \sum_{i=1}^{n} x_i(\hat{y}_i - y_i)$ (9)

Then, we have

$$||w^{t+1}||^2 = ||w^t||^2 + \tfrac{1}{n^2}||\textstyle\sum_{i=1}^{n} x_i(\hat{y}_i - y_i)||^2 + \tfrac{2}{n}w^t \cdot \textstyle\sum_{i=1}^{n} x_i(\hat{y}_i - y_i) \quad (10)$$

$$= ||w^t||^2 + \frac{||u||^2}{n^2} + \frac{2w^t \cdot u}{n} \quad (11)$$

ASSUMPTION 2

Recalling assumption 1, we assume that there exists $R \in R$ and $\gamma \in$ R, such that for $i \in \{1,2,\ldots,M\}$ $\| x^i \| \leq R$. So, (9) gives by using the triangular inequality:

$$\| u \| = \| \textstyle\sum_{i=1}^{n} x_i(\hat{y}_i - y_i) \| \leq \textstyle\sum_{i=1}^{n} \| x_i(\hat{y}_i - y_i) \| \leq nR \quad (12)$$

ASSUMPTIONS 3

Assume that there exist some parented vectors w^* such that $||w^*||=1$, and also, $\gamma>0$ such that for all $t=1,\ldots..,n$

$$(\hat{y}_i - y_i)(x_i \cdot w^*) \geq \gamma \quad (13)$$

Then,

$$\textstyle\sum_{i=1}^{n}(\hat{y}_i - y_i)(x_i \cdot w^*) \geq n\gamma \quad (14)$$

Taking $u = \sum_{i=1}^{n} x_i(\hat{y}_i - y_i),\ we\ obtain:$

$$w^{t+1} = w^t + \frac{1}{n}u \quad (15)$$

$$w^{t+1} \cdot w^* = w^{t+1} \cdot w^* + \frac{1}{n} u \cdot w^* \quad (16)$$

From (14), we have, $u \cdot w^* > n\gamma$

By induction,
$$w^{t+1} \cdot w^* > t\gamma \quad (17)$$

Knowing that

Then

$$\left||w^{t+1}|\right| \cdot \left||w^*|\right| \geq w^{t+1} \cdot w^* > t\gamma \quad (18)$$

$$||w^{t+1}||^2 \, ||w^*||^2 > t^2\gamma^2 \Rightarrow ||w^{t+1}||^2 > t^2\gamma^2 \quad (19)$$

Because we assumed $||w^*||^2=1$
Furthermore, from (11) we have :

$$\| w^{t+1} \|^2 = \| w^t \|^2 + \frac{1}{n^2} \| u \|^2 + \frac{2}{n} w^t \cdot u \quad (20)$$

While $w^t \cdot u \leq 0$ *for iteration* t *, we have*

$$\| w^{t+1} \|^2 \leq \| w^t \|^2 + \frac{1}{n^2} \| u \|^2 \quad (21)$$

By induction, we get: $\| w^{t+1} \|^2 \leq \frac{t}{n^2} \| u \|^2$ (22)

By the asumption 2, we obtain $\| w^{t+1} \|^2 \leq \frac{tn^2R^2}{n^2} = tR^2$ (23)

(19) and (23) give $t^2\gamma^2 < \| w^{t+1} \|^2 < tR^2$ (24)

Finally gives $t^2\gamma^2 < tR^2 \Rightarrow t < \frac{R^2}{n^2}$ (25)

For the following architecture,

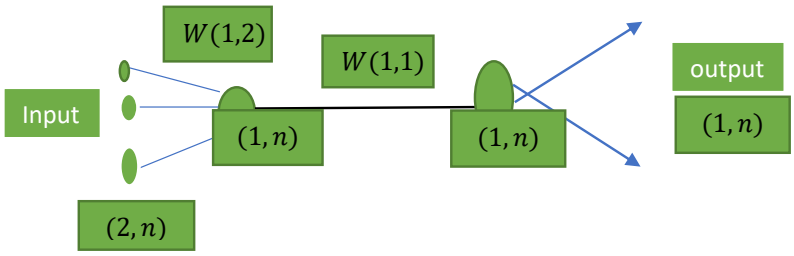


Figure Architecture of a Neural Network with One Hidden Neuron

The cost function of the second layer uses as input vector, the output of the previous layer which is the activated vector a(1).

$$L = \frac{1}{n}\sum_{i=1}^{n}[y_i ln\left(\sigma\left(wa_i^{(1)} + b\right)\right) + (1 - y_i) ln(1 - \sigma\left(wa_i^{(1)} + b\right))] \quad (26)$$

In fact, the first assumption remains valid because $a^{(1)} \in [0,1]$ and the gradient of the cost function $\partial L/\partial w$ can be calculated with the same previous formulae.

For A Two Neurons Added In the Hidden Layer

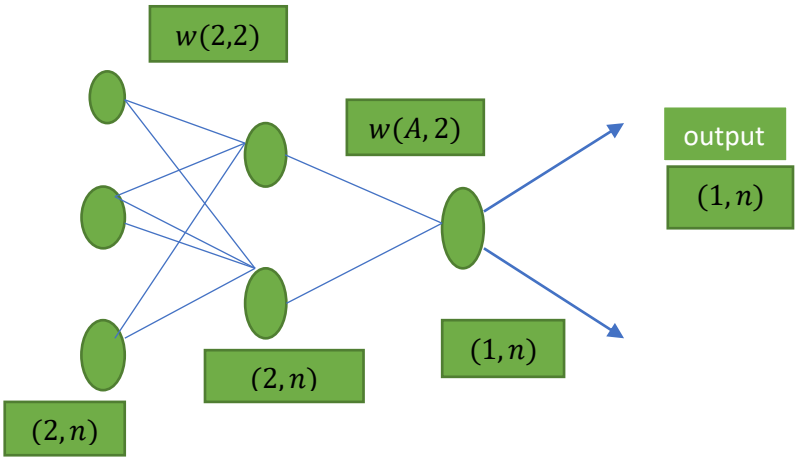


Figure Architecture of a Neural Network with One Hidden Layer of Two Neurons

As the final neuron output dimension is (1, n) and the activation elementsa_i□[o,1], the formulae of the cos function does not change, this is while the first assumption remains valid. Therefore, we have the result by maintaining the second assumption.

In general, for the straightforward growing algorithm the number of neurons in the current layer when changing, brings a change in the current weight parameters but the cost is the same.

For *l layer in the NNs*,

$$L = \frac{-1}{n}\sum_{i=1}^{n} y_i l_n(\sigma\left(w^{(l)} \cdot a_i^{(l-1)} + b^{(l)}\right) + (1-y_i)L_n\left(1-\sigma\left(w^{(l)} \cdot a_i^{(l-1)} + b^{(l)}\right)\right)]$$

We also show, that $w_k^{(l)}$ converge as in the first case by writing

$$\text{Where } u = \sum_{i=1}^{n} a_i^{(l-1)}(\hat{y}_i - y_i) \quad w_{k+1}^{(l)} = w_k^{(l)} + \frac{1}{n}u$$

Theorem 3

The SFC-DLNN construct a sequence of vector space $(H_i)_{l\leq i\leq n}$, such that $H_{i+1} \subset H_i$ and there exists $l \in \{1,\ldots,n\}$ in which SFC-DLNN gives the minimum risk or the best learning rate.

Proof

The Task Now Is to Calculate the VC-dimension of the SFC-DLNN Algorithm

We start from the Mirchandani and Cao (1989) who calculate the VC dimension of a PMC with one hidden layer and generalize. Knowing that, MLP's output can be expressed as product of matrices because the activation function does'nt change the weight matrices structure [10].

Let us consider $W_1 \in M_R\ (m_1,d), W_2 \in M_R\ (m_2,m_1), \ldots, W_k \in M_R\ (m_k,m_{k-1})$ where m_1 is the number of row and d the number of column, which is also the input vector space dimension. By taking the product respectful to the rule of matrices product:

Where k is the number of layer within the neural network.

$$W = W_1 * W_2 * \ldots * W_k \in M_{\mathbb{R}}(m_k, d)$$

Then we apply the Mirchandani and Cao (1989) [10] theorem on the W matrix and the result gives the VC-dimension of the SFC-DLNN algorithm:

$$R(m_1, m_2, \ldots, m_k, d) = \sum_{i=0}^{\min(d,\, m_k)} C_{m_k}^{i}$$

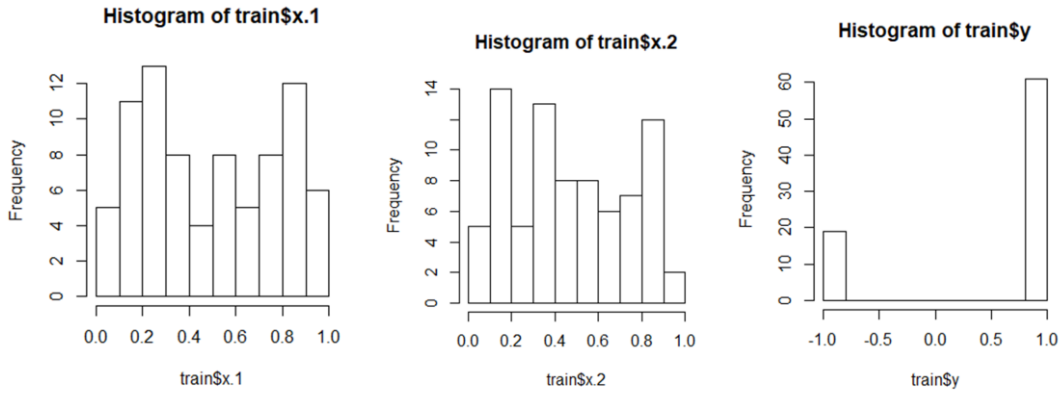
$R(m_1,m_2,\ldots,m_k,d)$ is the maximum number of linearly separable regions in R^d by the SFC-DLNN algorithm.

Experimental Results

In this part, we first present the data set used and the simulation results we obtain from the data set while using the Straight Forward Constructive Deep Learning Neural Network.

Data Presentation

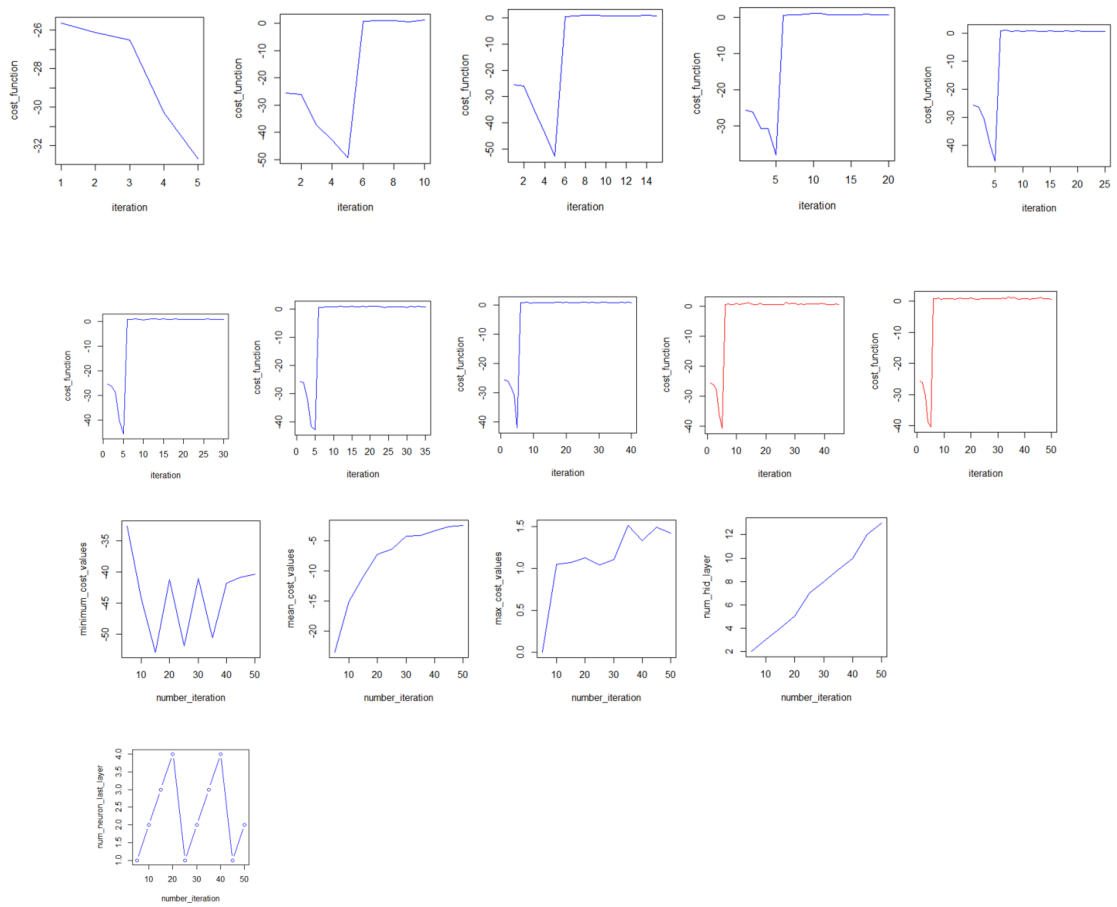
The data set used in this work is the sample of 1000 points (two dimension vectors) from the uniform distribution and to which we associate their label (1 or -1). In this set, we use 80 percent of the data to train the model and 20 percent as the test set.



Simulation Results

To train the deep neural network, we generate a sample points from uniform distribution probability and we use the following parameters: learning_rate = 0.09 and Eps = 10E-8

Number Iteration	Minimum cost	Mean cost	Maximum cost	Number of Hidden Layer	Number of Neuron in the last Layer
5	-32.67334	-23.5387	0	2	1
10	-44.1937	-15.08176	1.052235	3	2
15	-52.87325	-10.9766	1.07426	4	3
20	-41.2456	-7.211606	1.124532	5	4
25	-51.86505	-6.390297	1.04339	7	1
30	-41.01809	-4.254701	1.10818	8	2
35	-50.48194	-4.111634	1.51614	9	3
40	-41.7909	-3.332859	1.330655	10	4
45	-40.8602	-2.677842	1.494754	12	1
50	-40.35381	-2.434314	1.420828	13	2



Statistics Measures of Learning Model Prediction

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \times precision \times recall}{precision + recall}$$

$$accuracy = \frac{TP + TN}{TP + FN + TN + FP}$$

$$specificity = \frac{TN}{TN + FP}$$

TP : True positive

TN : True negative

FN : False Negative

FP : False positive

Table 3: Consensus Confusion Matrix

Predicted Accuracy	True	False
Positif	TP = 0	FP = 33
Negatif	FN = 0	TN = 167

The performance, over all the available models ignoring applicability domain considerations gives 0 True Positive (TP), 0 False Negative (FN), 33 False Positive (FP) and 167 True Negative (TN) individuals in the data set.

Table 4: Indicators Performance Matrix

Number of records		precision	recall	F1	Accuracy	specificity
1000	Training 800	0	0	0	0.835	0.835
	Test 200					

The AUC (Area under the Roc Curve) is the function giving the sensitivity (Y-axis) by using 1-specificity (X-axis).

Conclusion

The aim of this work is to find the best and suitable architecture of a Neural Network (Multi-Layer Perceptron) for a given data set. To understand how interesting, it is, we should remind that the Support Vector Machine main limit is finding "a better space" or the transformation $\emptyset(x)$ where the data is separable. As the MLP offers the possibility to find both this transformation $\emptyset(x)$ and the parameters for the regression or classification purpose. The stake of suitable architecture in NNs is to solve the problem of dimensionality and to approximate a large class of function with regularity that the NNs can capture [7-10].

We show in the first theorem that there exists an adjustment of weight such that the cost error given by the network, after this addition of the N neurons to the l layer, is less than the one before. In the second theorem, we show that there exists a layer number l (l>1) where, from this layer the Straight Forward Constructive Deep Learning Neural Network (SFC-DLNN) algorithm converges after passing this layer. We give Vapnik- Chervonenkys property of the SFC-DLNN by applying the [11-16].

We draw the cost function graphs (minimum cost, mean cost, maximum cost) using the number of iteration (5, 10, 15, 20, 25, 30, 35, 40, 45, 50) and observe that the all have the "square root form''. We also find that these curves decrease before the fifth iteration and increase after.

We obtain from our builded model (SFC-DLNN) an accuracy and precision of 83.5% from a simulated data set using the uniform distribution. This is not the best but is enough to approve the model prediction capacity.

However extension can be done on this work by replacing the sigmoid function with another activation because as we can see, this sigmoid activation function gives very closed value after a certain number of iteration. This has an influence on the precision, F1 and recall statistics, which are equal to 0.

References

1. Mohamed, S. A. E. M., Mohamed, M. H., & Farghally, M. F. (2021). A New Cascade-Correlation Growing Deep Learning Neural Network Algorithm. Algorithms, 14(5), 158.
2. Katanforoosh & Kunin,. (2019). "Initializing neural networks", deeplearning.ai
3. Ananthram, A. (2018). Random initialization for neural networks: a thing of the past. Towards Data Science.
4. Glorot, X., & Bengio, Y. (2010, March). Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 249-256). JMLR Workshop and Conference Proceedings.
5. Huemer, A., Elizondo, D., & Gongora, M. (2009). A Constructive Neural Network for Evolving a Machine Controller in Real-Time. In Constructive Neural Networks (pp. 225-242). Springer, Berlin, Heidelberg.
6. Zemouri, R., Omri, N., Fnaiech, F., Zerhouni, N., & Fnaiech, N. (2020). A new growing pruning deep learning neural network algorithm (GP-DLNN). Neural Computing and Applications, 32(24), 18143-18159.
7. J.E Campagne,. (2020). l'apprentissage face à la malédiction de la grande dimension, Notes et commentaires au sujet des conférences de S. Mallat du Collège de France.
8. Novikov A. On convergence proofs for perceptrons, Symposium on Mathematical Theory of Automata, Polytechnic Institute of Brooklyn, april 1962.
9. Khyati Mahendru, A Detailed Guide to 7 Loss Functions for Machine Learning Algorithms with Python Code.
10. Mirchandani, G., & Cao, W. (1989). On hidden nodes for neural nets. IEEE Transactions on Circuits and systems, 36(5), 661-664.
11. Nie, F., Hu, Z., & Li, X. (2018). An investigation for loss functions widely used in machine learning. Communications in Information and Systems, 18(1), 37-52.
12. Sheela, K. G., & Deepa, S. N. (2013). Review on methods to fix number of hidden neurons in neural networks. Mathematical problems in engineering, 2013.
13. Li, J. Y., Chow, T. W., & Yu, Y. L. (1995, December). The estimation theory and optimization algorithm for the number of hidden units in the higher-order feedforward neural network. In Proceedings of ICNN'95-International Conference on Neural Networks (Vol. 3, pp. 1229-1233). IEEE.
14. Xu, S., & Chen, L. (2008). A novel approach for determining the optimal number of hidden layer neurons for FNN's and its application in data mining.
15. Hunter, D., Yu, H., Pukish III, M. S., Kolbusz, J., & Wilamowski, B. M. (2012). Selection of proper neural network sizes and architectures—A comparative study. IEEE

Transactions on Industrial Informatics, 8(2), 228-240.
16. Kostadin Yotov, Emil Hadzhikolev, Stanka Hadzhikoleva. Determining the Number of Neurons in Arti_cial Neural Networks for Approximation, Trained with Algorithms Using the Jacobi Matrix. TEM Journal.