Article

# A New Cascade-Correlation Growing Deep Learning Neural Network Algorithm

Soha Abd El-Moamen Mohamed [1,*], Marghany Hassan Mohamed [2] and Mohammed F. Farghally [1]

[1] Information System Department, Faculty of Computer and Information, Assiut University, Assiut 71515, Egypt; mfseddik@aun.edu.eg
[2] Computer Science Department, Faculty of Computer and Information, Assiut University, Assiut 71515, Egypt; marghny@aun.edu.eg
* Correspondence: sohamoamen@aun.edu.eg

**Abstract:** In this paper, a proposed algorithm that dynamically changes the neural network structure is presented. The structure is changed based on some features in the cascade correlation algorithm. Cascade correlation is an important algorithm that is used to solve the actual problem by artificial neural networks as a new architecture and supervised learning algorithm. This process optimizes the architectures of the network which intends to accelerate the learning process and produce better performance in generalization. Many researchers have to date proposed several growing algorithms to optimize the feedforward neural network architectures. The proposed algorithm has been tested on various medical data sets. The results prove that the proposed algorithm is a better method to evaluate the accuracy and flexibility resulting from it.

**Keywords:** constructive neural networks; deep learning; neural networks; cascade correlation

**Citation:** Mohamed, S.A.E.-M.; Mohamed, M.H.; Farghally, M.F. A New Cascade-Correlation Growing Deep Learning Neural Network Algorithm. *Algorithms* **2021**, *14*, 158. https://doi.org/10.3390/a14050158

Academic Editor: Mircea-Bogdan Radac

Received: 6 May 2021
Accepted: 17 May 2021
Published: 19 May 2021

## 1. Introduction

Several types of neural network models have been proposed, such as pattern classification and regression problems. The feedforward neural networks (FNNs) class is common because of their structural stability, strong representation, and large number of available training algorithms. The growth in available data requires greater complexity in the structure of the neural networks to define large data sizes. Such special (FNN) architectures are called deep neural networks (DNNs) and are now very popular in many applications such as cancer diagnosis and medical imaging. Deep learning involves several layers of the nonlinear processing of information. In order to learn abstraction layers with a stronger capacity for generalization and representation, learning architectures are performed as repeated sequences of smaller functions [1].

Although the training process is more complex because of a lot of hidden layers and several links between them, the random connections initialization is also difficult because it creates the non-convergence of the training algorithm. If the number of neural parameters increases, the possibilities of the state space of all parameters' values will be unlimited. The probability of beginning the network from an optimal initial step is very little and this is the first problem.

The second problem is the difficulty of discovering the best neural network that can learn training data accurately and with great success in generalization. When the two parameters, namely the number of layers and the number of neurons at each layer, are joined together, there is an infinite number of possibilities. If the chosen network architecture is not suitable for the fixed size network, this will cause under-fitting or over-fitting. A network architecture that is neither too large nor too small is required for a better performance in generalization and less training time.

It might be reasonable to choose a NN architecture through manual design if there are qualified human experts with sufficiently advanced knowledge of the problem to be solved. However, this is clearly not the case for some real-world situations where several advanced data are not available. NNs make a fixed architecture in the back-propagation algorithm, while a constructive algorithm performs dynamic NN architectures.

The constructive algorithm is a supervised learning algorithm, and its dynamic models are commonly used to solve real-world problems. Even so, its mechanism for achieving dynamic NN architectures is totally the opposite. As such, the use of an acceptable NN architecture is still an issue for dealing with real-world problems. A constructive algorithm starts a NN with a limited design, i.e., with a limited number of hidden layers, neurons and links. Firstly, it starts with a simple NN approach, then gradually attaches hidden nodes and weights until an optimal methodology is identified.

Among these constructive algorithms are the cascade-correlation neural networks (CCNNs), which are a well-known class of discriminatory (as opposed to generative) models that have been effective in simulating several processes in development research. The CCNN algorithm is a rapid algorithm, as the new candidate neuron is learned just before connecting to the network, and the weights of the hidden input nodes are frozen in the corresponding process. Consequently, the weights are frequently trained at the output nodes after each new hidden neuron has been added.

In this paper, we proposed a method called "cascade-correlation growing deep learning neural network algorithm (CCG-DLNN)" which uses the growing phase at "growing pruning deep learning neural network algorithm (GP-DLNN)" [2], 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 frozen. Consequently, the weights of the output neurons are constantly trained after adding each new hidden neuron as in the CCNN algorithm.

The proposed CCG-DLNN algorithm has two benefits: firstly, the cascade architecture adds the hidden node to the neural network each time and it is not changed after inserting it; secondly, it is a learning algorithm that creates and adds a new hidden neuron. Trials are made to optimize the amount of correlation among the output of the node and the residual error signal.

We aim to increase the accuracy of the deep neural network when we perform it in the context of different real classification problems by combining some cascade-correlation algorithm features and (GP-DLNN) algorithm [2].

The paper is set out as following: Section 2 presents a state of the art with respect to the adaptive algorithms. Section 3 includes a comprehensive overview of the CCG-DLNN algorithm, and proof of convergence is given of the algorithm. Section 4 is about results and experimental analysis and a conclusion is made in Section 5.

## 2. State of the Art

With expanded computational power, artificial neural networks have demonstrated great strength in solving data classification and regression problems in big data fields. Traditional neural network training uses the error-back propagation strategy for the iterative alignment of all parameters. There are many challenges when the number of hidden layers increases, like local minima, slow convergence, and its time-consuming nature [3].

Unlike traditional neural network (NNs) algorithms that demand the NN architecture description before the start of the training, the dynamic neural network structure allows the network architecture to be designed and trained. A number of algorithms have been suggested to evaluate the optimal network model, including different algorithms for pruning, constructive, and hybrid constructive-pruning algorithm. A constructive algorithm combines neurons, hidden layers, and links into a small neural network architecture within the training. However, a pruning algorithm discards unnecessary hidden layers, nodes, and connections. it begins with large NNs during preparation [4].

In this paper, two methods were combined which are the cascade-correlation neural network algorithm and a constructive algorithm. At the following two subsections, the advantages of the two methods will be shown in detail to argue why we used them.

### 2.1. Constructive Neural Networks

Constructive neural networks (CoNNs) are a set of algorithms that change the design of the network, automatically generating a network with an acceptable dimension. The algorithms used for the CoNN learning method are called constructive algorithms. A constructive algorithm begins with a small network design and adding layers, nodes, and connections as appropriate during training. The architecture adaptation process resumes until the training algorithm obtains an almost best model that provides a sufficient problem solution [4].

Constructive algorithms have many huge benefits over pruning algorithms, such as, for constructive algorithms, it is fairly simple to define the initial network architecture, whereas in pruning algorithms, the initial network does not have an accurate number of layers [5]. In constructive algorithms, at the initial step of the training phase a smaller number of parameters (weights) are to be modified, thus requiring less training data for good generalization; however, we need large amounts of training data in pruning algorithms [4]. One popular feature of constructive algorithms is to suggest that the hidden nodes that are already installed in the network are useful for modeling part of the underlying task. In such situations, the weights fed to such implemented nodes may be frozen in order to prevent the problem of moving the target. The number of optimized weights is reduced over time to reduce the time and requirements of the memory [4].

For all these previous reasons, we used a constructive algorithm rather than the pruning algorithm for the proposed algorithm.

CoNN algorithms are different from each other, in terms of [6]:

1. Number of nodes that have been added per layer at each iteration;
2. The network grows forward or backward;
3. Whether all neurons make the same function;
4. Stopping criteria;
5. Newly added neuron connectivity pattern;
6. Kind of input patterns that deal with binary valued or categorical attributes;
7. Type of problems including classification and regression problems or clustering.

We preferred to make our proposed constructive algorithm according to the following characteristics:

1. The network grows forward;
2. Stopping criteria are based on when the desired accuracy of the network is achieved;
3. Kind of input patterns is binary valued;
4. This algorithm can deal with classification problems.

Within neural network literature, several CoNN algorithms that are only suitable for classification problems have been proposed. The well-known two-class CoNN algorithms are the tower and pyramid [7], the tiling [8], the upstart [9] and the cascade of perceptron [10]. The MTower, MPyramid, Mtiling, MUpstart and MPerceptron cascade are the multi-class versions of these algorithms. One can see these algorithms in [11,12]. The major constructive algorithms for regression problems, as mentioned in [13], are dynamic node creation (DNC) [14–22], cascade-correlation algorithm (CCA) [23], projection pursuit regression based on the statistical technique [24], resource-allocating network [25], group method of data handling [26] and miscellaneous [27].

Among these algorithms, the CCA was chosen as the proposed algorithm and the next subsection will show how CCA works and its advantages over the other algorithms.

### 2.2. Cascade-Correlation Algorithm

One of the supervised learning models for neural networks is a cascade correlation. Cascade correlation starts with a small number of layers, and then gradually trains and installs new hidden nodes, one at each step, and thus generating many layer models rather than only changing the weights in a fixed topology network. After adding a new hidden node to the network, the weights of this node are fixed at the input side. This node is a fixed feature detector inside the network, ready for output production or many other complex feature detectors to be produced, as seen in Figure 1. There are two ideas within the CCNN algorithm: firstly, the cascade model, where hidden nodes are connected to the network only once and do not alter after they have been added; secondly, the learning algorithm, which produces and installs new hidden units. The algorithm aims to maximize the magnitude of the correlation between the new unit's output and the residual error signal for each new hidden unit [23].

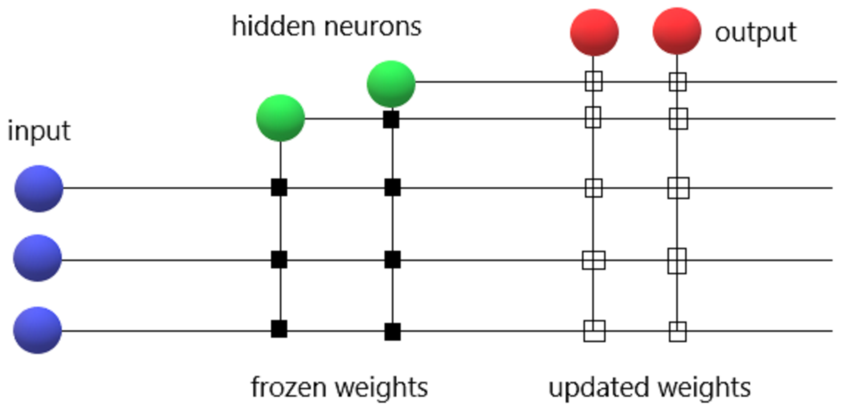

**Figure 1.** Traditional CCNN.

In our proposed algorithm, the CCNN algorithm was chosen due to its following advantages:

1. It learns faster than other back-propagation algorithms (CCNN does not use the back-propagation of error signals);
2. The network chooses the suitable size and topology at learning step;
3. At incremental learning, CCNNs algorithm is useful as it can add new information to the trained network.

The following section will illustrate in detail our proposed algorithm with all its steps.

## 3. New Proposed Cascade-Correlation Growing Deep Learning Neural Network Algorithm (CCG-DLNN)

The conclusion of the previous section is that the constructive algorithms have many benefits over pruning algorithms in terms of building the NN structure. Usually, constructive neural networks are used to solve the problems of classification. Using this type of network has been shown to result in fewer computation requirements, smaller topologies, faster learning and better classification. Furthermore, this reveals that such constructive neural networks will still develop to a point where 100% of the training data can be correctly identified [28].

In this section, we present a growing algorithm based on the deep neural architectures (CCG-DLNN).

### *3.1. Notations and Assumptions*

- $N_{HL}$: number of hidden layers as in Figure 2;
- $N_u^l$: number of units within the lth layer as in Figure 2;
- $S$: is the attributes number at the training data set;
- $l$: Layer number, $l = 0$ is the input layer, $l = N_{HL} + 1$ is the output layer and numbers of hidden layers is defined as $1 \leq l \leq N_{HL}$;
- $w_{ij}^l$: is the weight between the ith node at lth layer to the jth node at the (lth $-$ 1) layer $1 \leq l \leq N_{HL} + 1$ as in Figure 2;
- $W^l = (w_{ij}^l)_{N_u^l \times N_u^{l-1}}$: is the matrix of weights which links the lth layer to the (lth $-$ 1) layer as in Figure 2;
- $\varphi = (w_{ij}^l)_n$: is all the neural network weight connections $n = \sum_{l=1}^{N_{HL}+1} N_u^l \times N_u^{l-1}$;

- $\hat{\varphi}$: is the new weights when the network is increased;
- $L(\varphi)$: loss function;
- $E^m(\varphi)$: is the output error resulting from the mth input $x^m$;
- $\nabla L(\varphi)$: the gradient of $L(\varphi)$ based on the parameter $\varphi$ and given by Equation (1):

$$\nabla L(\varphi) = \frac{\partial E^m(\varphi)}{\partial w^l_{ij}} \tag{1}$$

To train the deep neural network, we use an online backpropagation algorithm and update the weights using Equation (2):

$$\Delta w^l_{ij}|_m = -\eta \nabla L(\varphi) \tag{2}$$

where the learning rate is $\eta$:

- $x^m$: is the mth input sample of the training input data set $X = (x^m)_S, 1 \le m \le S$;
- $o^m$: is the mth desired output of the training output data set $O = (o^m)_S, 1 \le m \le S$;
- $y^m$: is the output of the NN obtained for the input $x^m$, $Y = (y^m)_S$ as in Figure 2;
- $h^{l,m}_i$: the output of the ith hidden unit related to the lth hidden layer related to the mth input $x^m$ and given by Equations (3) and (4):

$$h^{l,m}_i = f(g^m_i) \tag{3}$$

$$g^m_i = \sum_{j=1}^{N_u^{l-1}} (w^l_{ij} h^{l-1,m}_i) \tag{4}$$

where $g^m_i$ is the summation of the product of previous layers' neurons outputs and weights and $f(.)$ is the nonlinear activation function that could be sigmoid, tanh or ReLU; and $y^m = g^m_i$ when $l = N_{HL} + 1, i = 1$.

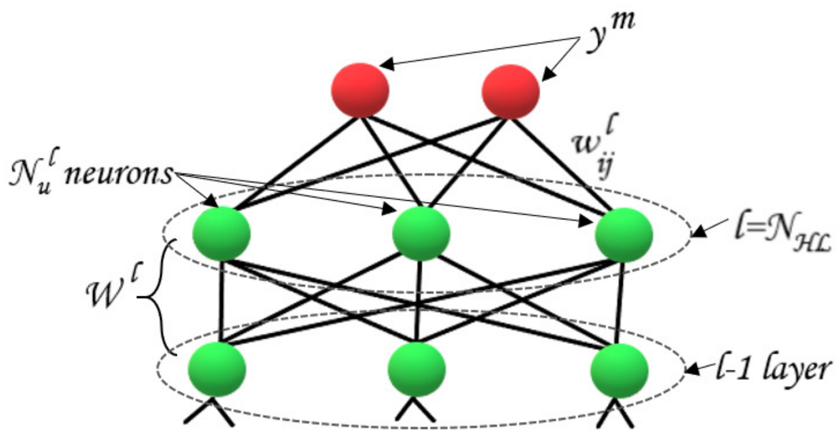

**Figure 2.** NN notations.

## 3.2. *The CCG-DLNN Algorithm*

The CCG-DLNN algorithm uses the same strategy like CCNN but by adding more than one hidden layer—that has more than one neuron—between the input and output layers. Therefore, at each iteration, we add a new hidden layer with one neuron or add a new neuron at the last added hidden layer $(l = N_{HL})$.

At the beginning, the learning algorithm begins with a simple network that has only the input and output layers and does not have any hidden layers. Due to the lack of hidden neurons, this network can be learned by a simple gradient descent algorithm individually applied to each output neuron.

New neurons are connected to the network one by one during the learning process. Each of them is put in a new hidden layer and linked to all the previous hidden neurons at the previous hidden layer. When the neurons are eventually connected to the network (activated), their input connections will freeze and no longer change.

Adding a new neuron can be divided into two sections. First, we start with a candidate neuron and receive the input connections from all pre-existing hidden units at the last hidden layer $(l = N_{HL})$ as in Figure 3. The candidate unit's output is not yet added to the current network. We do a sequence of passes over the examples set for training, updating the weights of the candidate unit for each pass at the input side. Second, the candidate is linked to the output neurons (activated) and then all output connections are trained. The entire process is repeated multiple times until the desired accuracy of the network is achieved.

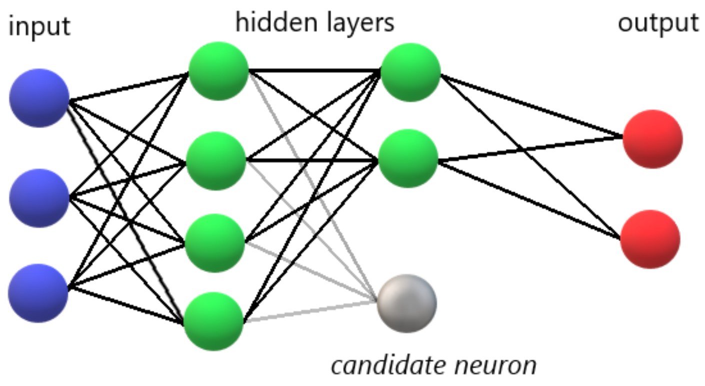

**Figure 3.** Candidate neuron at neural network.

### 3.3. *Steps of CCG-DLNN Algorithm*

As shown by Algorithm 1 and Figure 4, the CCG-DLNN algorithm begins with minimal architecture (input and output layers) without any hidden layers, and this simple network is trained and the loss function $L(\varphi_t)$ is calculated. Let us consider that DE is the desired error [2], $L(\varphi_t) \geq$ DE denotes that more neural hidden layers and more nodes are required to fit the training data set correctly. To do so, we add a new completely linked neuron to the last hidden layer or add a new hidden layer. We let $C$ be introduced to measure changes in the loss function when a new neuron is added to the current hidden layer and $\tau$ is the threshold to add a new hidden layer. As shown in Equations (5) and (6), if $L(\varphi_{t-1})$ is degraded compared to $L(\hat{\varphi}_t)$ before adding the new neuron, then increases $C$ by one. Otherwise, $C$ will be zero. While $C$ is less than the threshold $\tau$, we add one new neuron to the last hidden layer. If $C$ reaches the threshold value $\tau$, then we add a new hidden layer that has one neuron as in Figure 5:

$$\begin{cases} C = 0 & if L(\varphi_{t-1}) - L(\hat{\varphi}_t) > \epsilon \\ C = C + 1 & otherwise \end{cases} \quad (5)$$

$$\begin{cases} l = l + 1 & if C \geq \tau \\ N_{ul} = N_{ul} + 1 & otherwise \end{cases} \quad (6)$$

where $\epsilon$ is a growing process threshold.

The stage of the neuron can be divided into two parts. First, we start with a candidate neuron that accepts input links at the last hidden layer from all pre-existing hidden units. The candidate unit's output is not yet connected to the active network. We run several passes over the training set examples, adjusting the candidate unit's input weights for each pass. Second, the candidate is connected to the (activated) output neurons, and then all output connections are learned. The entire cycle is repeated until the network's required accuracy is achieved. Increasing the number of hidden layers may or may not increase the accuracy, as it really depends on the difficulty of the solved problem.

Increasing the number of hidden layers much more than a sufficient number of layers would reduce the accuracy of the test set. It will make the network over-fit the training set, that is, it will know the training data, but it will not be able to generalize to new unknown data [29].

The large weights in the neural network are a sign of over-fitting. The research in [30] defined weights in a qualified network in a more coherent manner, by preventing them from taking broad values using Equation (7):

$$min(E(\varphi)), where(l \leq \varphi \leq u) \tag{7}$$

where $\varphi$ is all the neural network weight connections, $E(\varphi)$ is the output error, and $l$ is a lower bound and $u$ is an upper bound.

**Algorithm 1:** Cascade-Correlation Growing Deep Learning Neural Network

```
Input:
$X$ : training data set
$S$ : the training data set size
$iter$: iteration number for the NN training
$t$: the index of training step
$l$: current hidden layer index
$N_{ul}$: the count of neurons at layer $l$
$W^l$: Matrix of the links from layers $l$ to $(l-1)$
$L(\varphi_t)$: The loss function at the t step
DE: The threshold Error
$\tau$: adding a new hidden layer Threshold
$C$: Condition for adding a new hidden layer
Output:
Deep Neural Network
1  $l = 0$, $N_{ul} = 0$, $C$=0, $t = 0$
2  $feedforward(X)$
3  Calculate $L(\varphi_t)$
4  $backpropagation(X, L(\varphi_t))$
5  update weights $W^l$ of the initial architecture that has input and output layers only.
6  $t = t + 1$
7  while $L(\varphi_t) \geq DE$ and $t \leq iter$ do
8      if $L(\varphi_{t-1}) - L(\hat{\varphi}_t) > \epsilon$ then
9          break
10     else
11         if $C \geq \tau$ then
12             $l = l + 1$
13             C= 0
14             $N_{ul} = 1$
15         else
16             C= C+1
17             $N_{ul} = N_{ul} + 1$
18     Train the new Candidate unit with all neurons in layer $l - 1$
19     Connect the trained Candidate unit with output layer
20     The weights $W^l$ is trained for the new architecture
21     Calculate $L(\varphi_t)$
22     perform Weight Constrained Neural Network Training Algorithm [30]
23     $t = t + 1$
24 return The Deep Neural Network
```

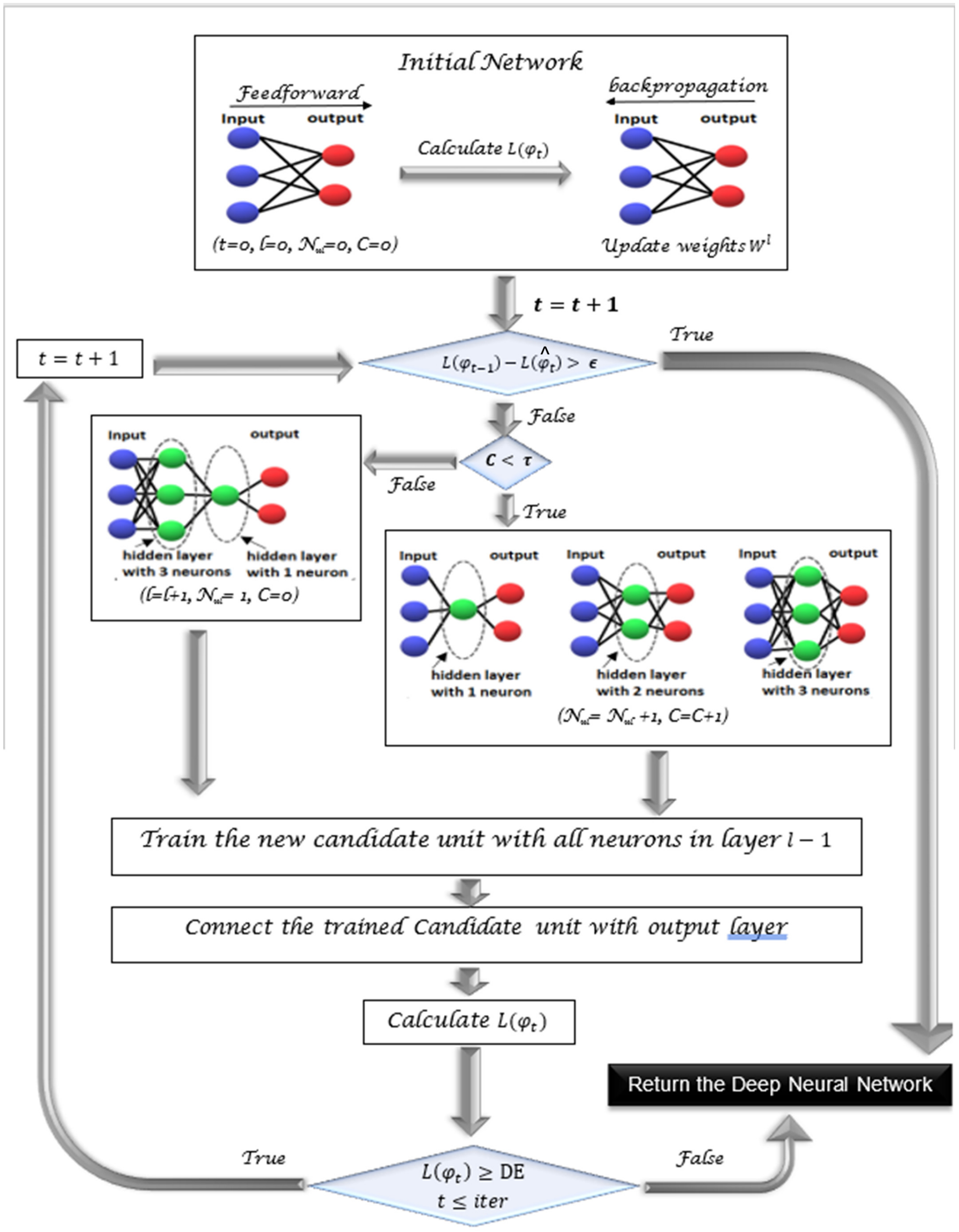
Initial Network
Feedforward
backpropagation
Input
output
Calculate $L(\varphi_t)$
(t=0, l=0, $\mathcal{N}_{ul}$=0, C=0)
Update weights $W^l$
$t = t + 1$
True
$L(\varphi_{t-1}) - L(\hat{\varphi_t}) > \epsilon$
False
$C < \tau$
hidden layer with 3 neurons
hidden layer with 1 neuron
(l=l+1, $\mathcal{N}_{ul}$= 1, C=0)
hidden layer with 2 neurons
($\mathcal{N}_{ul}$= $\mathcal{N}_{ul}$ +1, C=C+1)
Train the new candidate unit with all neurons in layer $l-1$
Connect the trained Candidate unit with output layer
Calculate $L(\varphi_t)$
Return the Deep Neural Network
$L(\varphi_t) \geq$ DE
$t \leq iter$

**Figure 4.** Steps of CCG-DLNN algorithm.

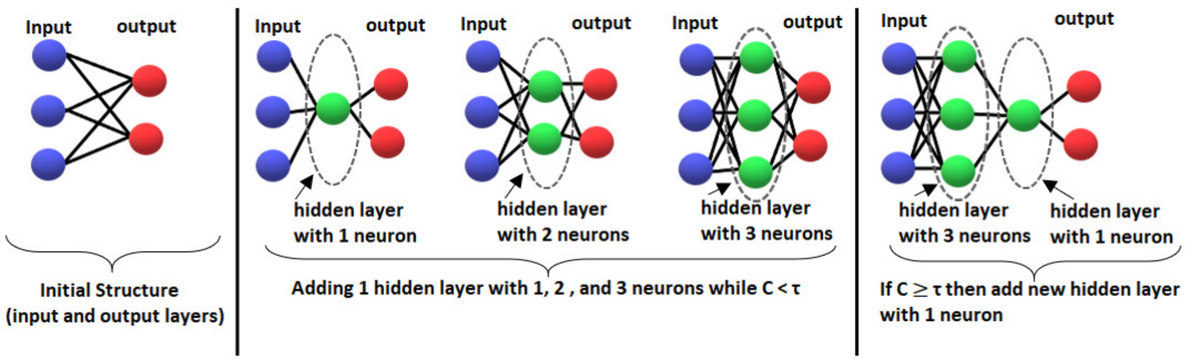
Input
output
Initial Structure (input and output layers)
hidden layer with 1 neuron
hidden layer with 2 neurons
hidden layer with 3 neurons
Adding 1 hidden layer with 1, 2 , and 3 neurons while C < τ
hidden layer with 3 neurons
hidden layer with 1 neuron
If C ≥ τ then add new hidden layer with 1 neuron

**Figure 5.** Steps of CCG-DLNN algorithm.

### *3.4. The Convergence Proof*

**Theorem 1.** *There exists a weight setting for neurons within the newly added layer l and the output neurons in the CCG-DLNN algorithm such that the number of patterns misclassified by the network after the addition of the N neurons to the l layer and the retraining of the output weights is less than the number of patterns misclassified before that (i.e.,* $E_t < E_{t-1}$*).*

**Proof.** At iteration t, the loss function $L(\varphi_t)$ will be:

$$L(\varphi_t) = \frac{1}{S}\sum_{m=1}^{S}(E^m(\varphi_t))^2 \tag{8}$$

where:

$$E^m(\varphi_t) = y_t^m - o^m \tag{9}$$

For hidden layer $l$:

$$g_t^{l,m} = \varphi_t h_t^{l-1,m} \tag{10}$$

if $l = N_{HL} + 1$, then:

$$y_t^m = \varphi_t h_t^{N_{HL},m} \tag{11}$$

from [31], we have:

$$\lim_{t\to\infty} \|\nabla L(\varphi_t)\| = 0 \tag{12}$$

So:

$$L(\varphi_t) \le L(\varphi_{t-1}) \tag{13}$$

from Equation (8):

$$\frac{1}{S}\sum_{m=1}^{S}(E^m(\varphi_t))^2 \le \frac{1}{S}\sum_{m=1}^{S}(E^m(\varphi_{t-1}))^2 \tag{14}$$

Let us dispose of the summing symbol for simplicity. The summation symbol is important, particularly for the definition of the stochastic gradient descent (SGD) versus batch gradient descent. In the batch gradient descent, we focus only on the error of all the training examples at once when we look at each error at a time in the SGD. However, just to keep things easy, we are going to assume that we are referring to each error one at a time:

$$\frac{1}{S}(E^m(\varphi_t))^2 \le \frac{1}{S}(E^m(\varphi_{t-1}))^2 \tag{15}$$

$$(E^m(\varphi_t))^2 \le (E^m(\varphi_{t-1}))^2 \tag{16}$$

$$E^m(\varphi_t) \le E^m(\varphi_{t-1}) \tag{17}$$

from Equation (9):

$$y_t^m - o^m \le y_{t-1}^m - o^m \tag{18}$$

$$y_t^m \le y_{t-1}^m \tag{19}$$

from Equation (11):

$$\varphi_t h_t^{N_{HL},m} \le \varphi_{t-1} h_{t-1}^{N_{HL},m} \tag{20}$$

If we add new one neuron to the last hidden layer, then $y_t^m$ will be:

$$y_t^m = y_{t-1}^m + \varphi_{new} h_{new} \tag{21}$$

where $\varphi_{new}$ is the weight from the newly added neuron to the output neuron and $h_{new}$ is the output of the newly added neuron.

So Equation (19) will be:

$$y_{t-1}^m + \varphi_{new} h_{new} \le y_{t-1}^m \tag{22}$$

So:

$$\varphi_{new} h_{new} \leq 0 \tag{23}$$

After adding a new neuron and from Equation (9):

$$E^m_{new}(\varphi_t) = y^m_{new,t} - o^m \tag{24}$$

from Equation (21):

$$E^m_{new}(\varphi_t) = y^m_{t-1} + \varphi_{new} h_{new} - o^m \tag{25}$$

$$E^m_{new}(\varphi_t) = E^m(\varphi_{t-1}) + \varphi_{new} h_{new} \tag{26}$$

from Equation (23):

$$E^m_{new}(\varphi_t) \leq E^m(\varphi_{t-1}) \tag{27}$$

□

So the number of unclassified patterns by the network after adding the $N$ neurons in layer $l$ and the extra training of the output weights $E^m new(\varphi t)$ is less than the amount of patterns unclassified before that.

## 4. Experimental Results

The proposed CCG-DLNN algorithm was tested for a neural network implementation. The CCG-DLNN algorithm was evaluated on some real-world data sets benchmark problems. The original CCNN are compared with the CCG-DLNN results and GP-DLNN algorithms. All simulations were performed in the Python 3.7 environment and run in Core i7 with CPU 3.60GHZ and 32.0 GB of RAM.

### *4.1. Two Spirals Classification Problem*

As shown in Figure 6, the two-spiral problem consists of two interconnected spirals, which has been commonly used to test the network algorithms. In this experiment, we built and trained CCG-DLNN, which can correctly make a classification and make a good generalization of test results. The training set size is 200, 50% of which will generate a $-1$ and leave 1 remaining. The size of the test data set is 100, which is generated from equal space points in space $[-6, 6] \times [-6, 6]$. The maximum number of installed hidden neurons is 22. Figure 7 displayed the results obtained from the three separate neural architectures shown in Table 1.

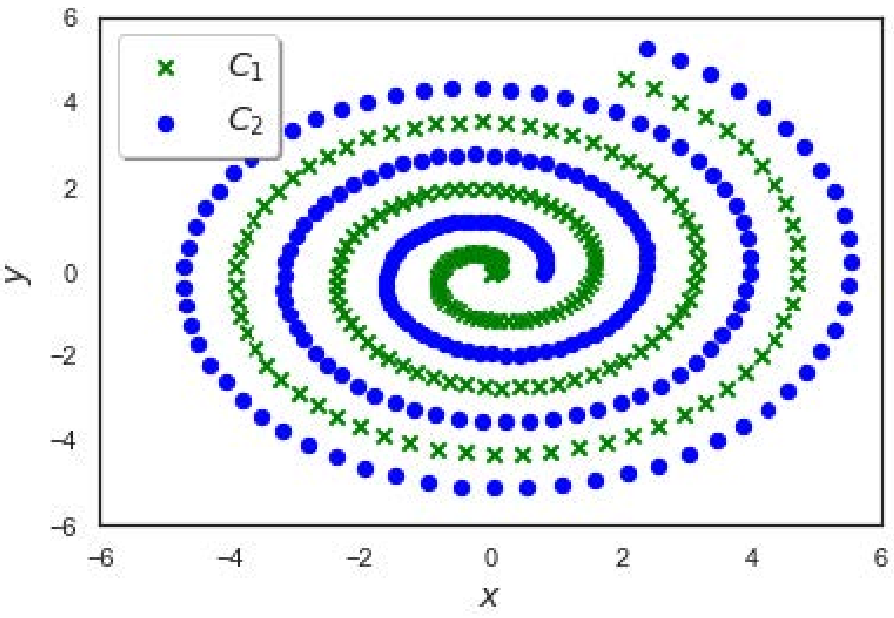

**Figure 6.** Training sample of two-spiral problem.

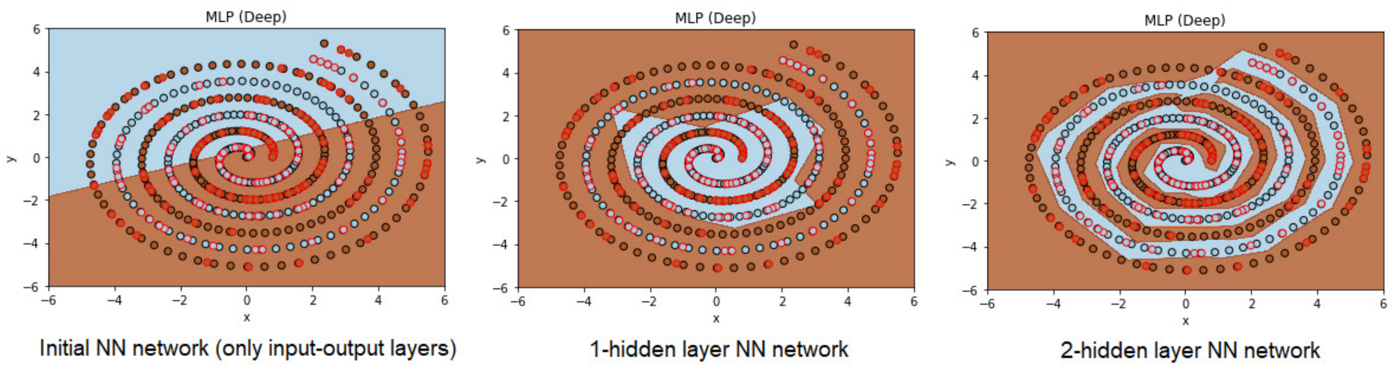

**Figure 7.** Testing two-spiral problem using CCG-DLNN algorithm.

**Table 1.** Testing two-spiral problem using CCG-DLNN.

| Two-Spiral Architecture | Number of Neurons | Accuracy (%) | Avg Cost |
|---|---|---|---|
| Initial NN | 0 | 44 | 1.004625 |
| Adding 1st Hidden Layer | 10 | 53 | 0.932671 |
| Adding 2nd Hidden Layer | 22 | 99.5 | 0.054119 |

As shown in Table 1, the third neural architecture which has two hidden layers with 22 neurons has the highest accuracy (99.5%) over the other two architectures with an average loss of 0.054119. The results resulted from CCG-DLNN, CCNN and GP-DLNN, which are in Table 2.

**Table 2.** Simulation results of CCG-DLNN, CCNN and GP-DLNN on the two-spiral problem.

| | # Hidden Layers | # Neurons | Accuracy (%) | Avg Cost |
|---|---|---|---|---|
| CCG-DLNN | 2 | 22 | 99.50 | 0.054119 |
| CCNN | 10 | 10 | 87.21 | 0.254895 |
| GP-DLNN (growing step) [2] | 3 | 70 | 92.23 | 0.075852 |

The proposed algorithm CCG-DLNN has the accuracy of 99.50% after adding two layers with total 22 neurons, However, the growing step at the GP-DLNN algorithm gives an accuracy of 92.23% after adding three layers with a total 70 neurons. The CCNN algorithm performed 87.21% accuracy after adding 10 neurons at 10 layers.

### 4.2. *Classification Problems at Real-World Data Sets*

The proposed CCG-DLNN algorithm was tested on a number of different real classification problems such as: the Lung Image Database Consortium image collection (LIDC-IDRI) [32], and some benchmark problems from the UCI Machine Learning Repository [33]: Diabetes 130-US hospitals (DH) [34], Hepatitis C Virus (HCV) for Egyptian patients, Heart Disease (HD), Breast Cancer Wisconsin (Diagnostic) (BCW) [33], Breast Cancer Coimbra (BCC) [35], Liver Disorders Data Set (LD) [33], Acute Inflammations Data Set (AI) [36], and Pima Indians Diabetes Database (PID) [33].

Table 3 explains the description of all data sets as it shows the number of training and testing samples and the overlapping level which is visually represented in Figure 8 using the t-distributed stochastic neighbor embedding (t-SNE) projection. Figure 8 shows the 2-D representation of LIDC-IDRI, DH, HCV, HD, and BCC data sets. The t-SNE 2D representation of BCW, AI, LD, and PID data sets will be founded at [2].

For each event, 100 proceedings are performed and the resulting averages are shown in Table 4 as well as Figures 9 and 10, where the best of each test are in bold face. The three cases have the same condition for stopping. We can observe that the CCG-DLNN algorithm obtains lower errors for all cases than the other two algorithms. The loss rate for the three algorithms on the LIDC-IDR data set is shown at Figure 11. It can be seen from Figure 11 that the loss of CCG-DLNN strictly decreases faster than CCNN and slightly faster or equal to GP-DLNN [2]. Figure 12 gives the curves of loss obtained by CCG-DLNN, GP-DLNN and CCN on the LIDC-IDR data set. Figure 12 shows that the loss of CCG-DLNN decreases as the number of hidden units increases. In addition, the CCG-DLNN loss curve will reach zero at limited steps of the CCG-DLNN, which is the number of training data samples. We observe that the CCG-DLNN algorithm will prevent a local minimum problem.

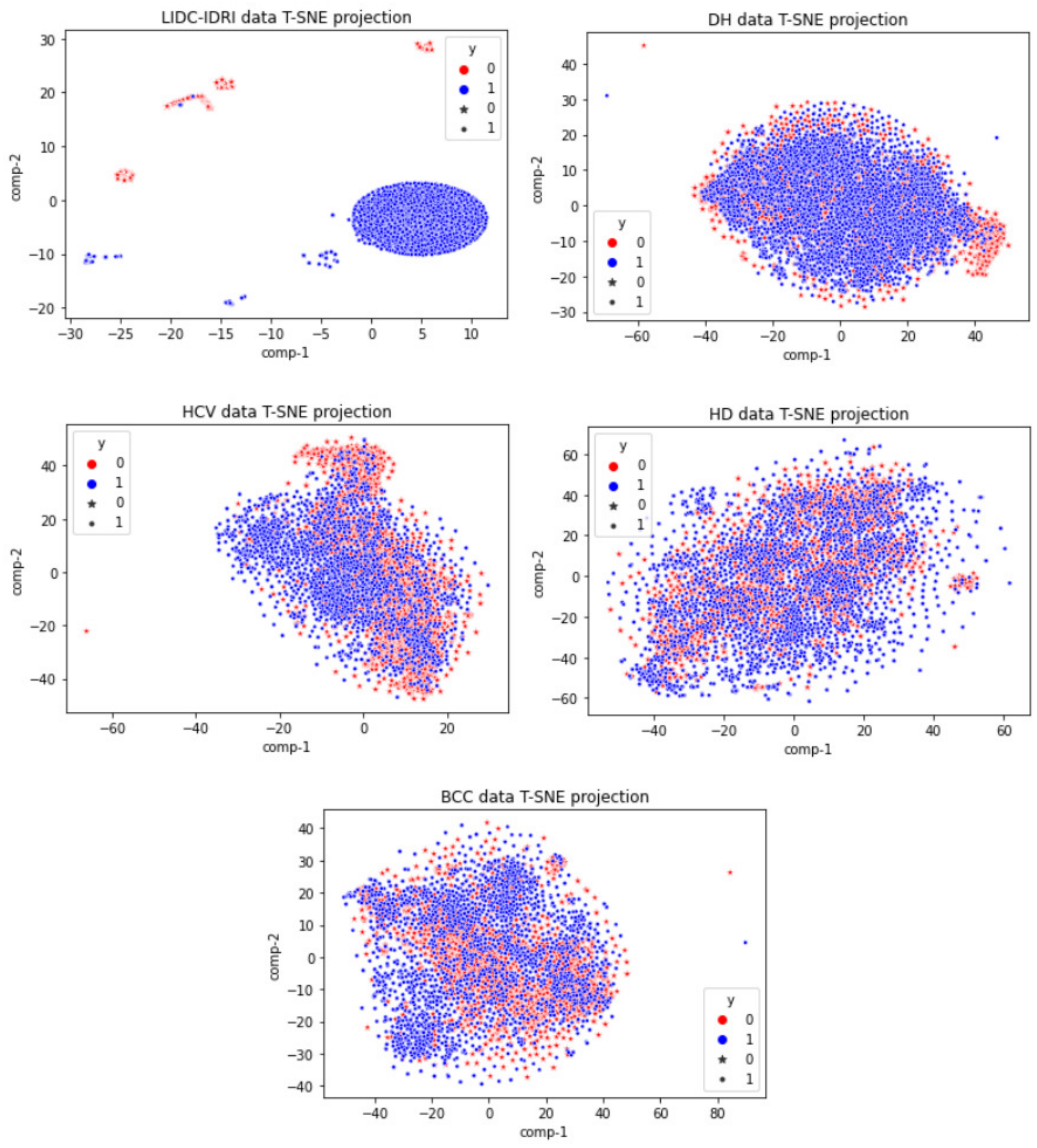

**Figure 8.** The 2-D distribution of some used data sets using t-distributed stochastic neighbor embedding (t-SNE).

**Table 3.** Specification of real-world classification problems.

| Data Set | Training Sample | Testing Sample | Overlapping Level |
|---|---|---|---|
| LIDC-IDRI | 20,000 | 20,000 | No |
| DH | 15,000 | 15,000 | High |
| HCV | 500 | 500 | High |
| HD | 150 | 150 | High |
| BCW | 547 | 136 | Low |
| BCC | 50 | 50 | High |
| LD | 276 | 69 | High |
| AI | 96 | 24 | No |
| PID | 615 | 153 | High |

**Table 4.** Simulation results of CCG-DLNN, GP-DLNN and CCNN on real-world classification problems.

| Data Set | Training Rate | | | Testing Rate | | | Number of Hidden Units | | |
|---|---|---|---|---|---|---|---|---|---|
| | CCG-DLNN | CCNN | GP-DLNN | CCG-DLNN | CCNN | GP-DLNN | CCG-DLNN | CCNN | GP-DLNN |
| LIDC-IDRI | **0.98** | 0.93 | 0.95 | **0.96** | 0.95 | 0.94 | 10 | 13 | **24** |
| DH | **0.89** | 0.85 | 0.86 | **0.75** | 0.69 | 0.70 | 15 | 20 | **34** |
| HCV | **1** | 0.96 | 0.98 | **0.98** | 0.92 | 0.95 | 20 | 25 | **44** |
| HD | **0.93** | 0.88 | 0.89 | **0.90** | 0.86 | 0.84 | 24 | 30 | **50** |
| BCW | 0.83 | 0.80 | **0.85** | 0.79 | 0.72 | **0.85** | 6 | 10 | **14** |
| BCC | 0.95 | **0.97** | 0.93 | **0.94** | 0.92 | 0.91 | 11 | 14 | **28** |
| LD | **0.99** | 0.95 | 0.98 | **0.93** | 0.89 | 0.93 | 9 | 15 | **22** |
| AI | **1** | 0.93 | 0.95 | **1** | 0.95 | 0.99 | 8 | 19 | **19** |
| PID | **0.98** | 0.95 | 0.96 | **0.98** | 0.91 | 0.97 | 14 | 19 | **23** |

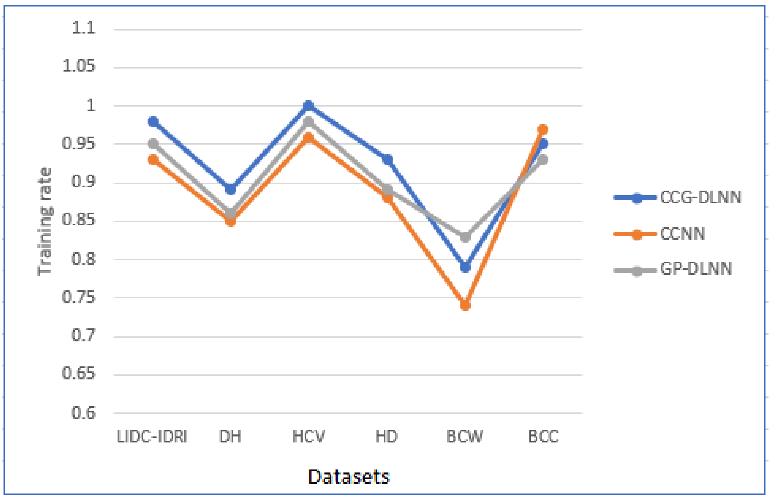

**Figure 9.** Curves of training rate obtained by CCG-DLNN, GP-DLNN and CCNN on real-world classification problems.

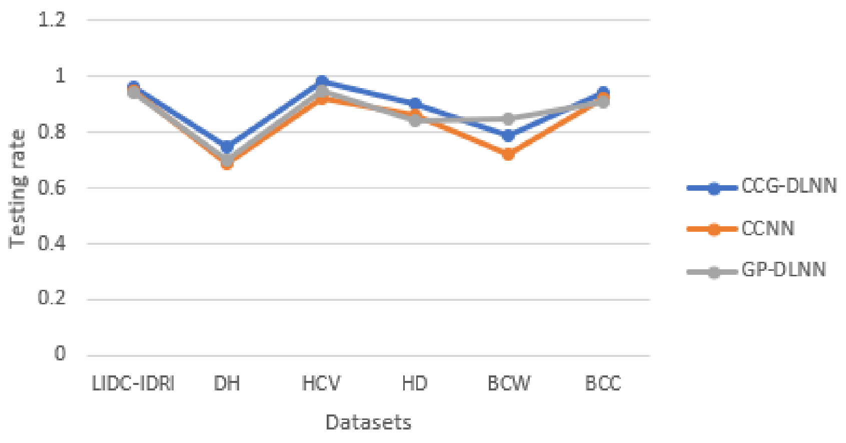

**Figure 10.** Curves of testing rate obtained by CCG-DLNN, GP-DLNN and CCNN on real-world classification problems.

The criteria used to measure the efficiency of the tests were sensitivity, specificity and accuracy. The first performance, testing the accuracy of the proposed CCG-DLNN algorithm, was compared to the GP-DLNN (growing step) [2] and CNN algorithms. The best outcomes for all data sets resulted from the CCG-DLNN algorithm, as shown in Table 5.

The classification performance related to the CCG-DLNN, CCNN and GP-DLNN on the LIDC-IDRI data set is presented in Figure 13 and the classification performance for all data sets is represented in Table 6. All methods achieved good performance in the classification. The highest AUC value was obtained for the proposed algorithm CCG-DLNN. Sensitivity, specificity and accuracy were determined using the ROC curve threshold which was the closest point to the upper left corner of the ROC plot (0, 1).

However, based on the Delong test, this difference in AUC values was statistically significant ($p$-value $< 0.05$). The value of AUC obtained for the CCG-DLNN algorithm was significantly greater than that for the GP-DLNN method (Delong test $p$ values $< 0.05$). The significance level and Z statistic between the AUC of CCG-DLNN and GP-DLNN algorithms for all data sets is presented in Table 7.

To evaluate the speed of the three algorithms (CCG-DLNN, GP-DLNN, and CCNN), we compared the runtime of these algorithms when solving the LIDC-IDRI problem. Figure 14 shows that our proposed algorithm CCG-DLNN is not faster than the CCNN algorithm at the beginning steps and it is faster at the final stages. GP-DLNN is the slowest algorithm compared to the other two algorithms.

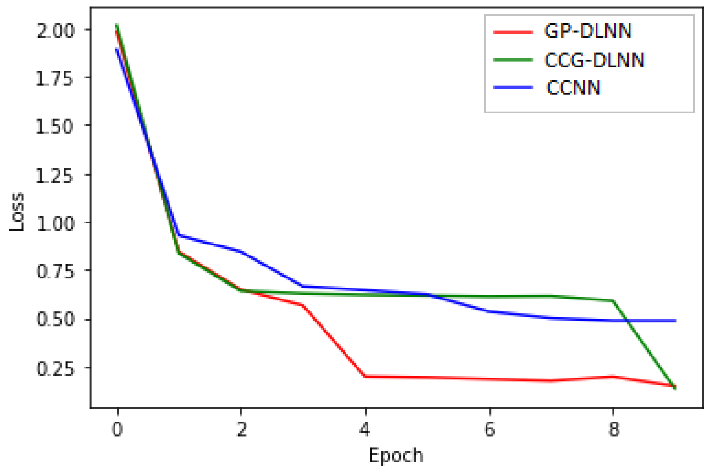

**Figure 11.** Curves of loss obtained by CCG-DLNN, GP-DLNN and CCNN on the LIDC-IDRI problem.

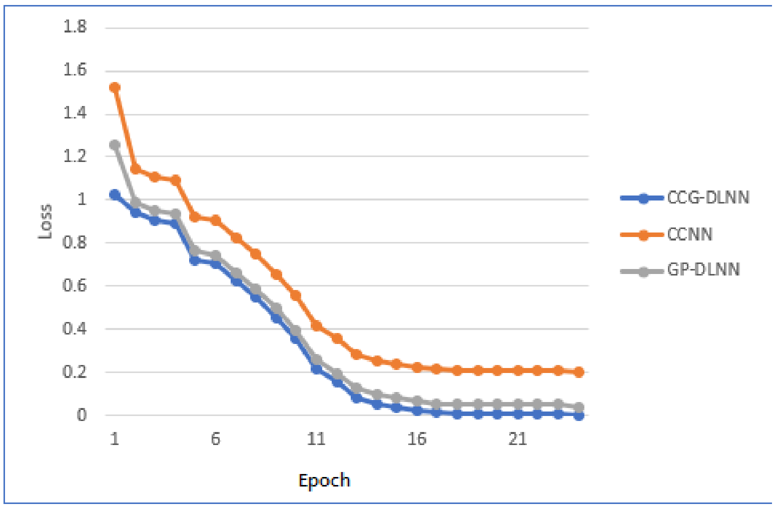

**Figure 12.** Curves of loss obtained by CCG-DLNN, GP-DLNN and CCNN on the LIDC-IDRI problem.

**Table 5.** Summary of simulation results.

| **Data Sets** | | **Accuracy (%)** | |
|---|---|---|---|
| | **CCG-DLNN** | **CCNN** | **GP-DLNN** |
| LIDC-IDRI | **95** | 93.6 | 92.3 |
| DH | **84** | 77 | 82 |
| HCV | **94** | 83 | 90 |
| HD | **90** | 83 | 86 |
| BCW | **93** | 92 | 90 |
| BCC | **100** | 90 | 91 |
| LD | **87** | 69 | 73 |
| AI | **100** | 92 | **100** |
| PID | **80** | 65 | 74 |

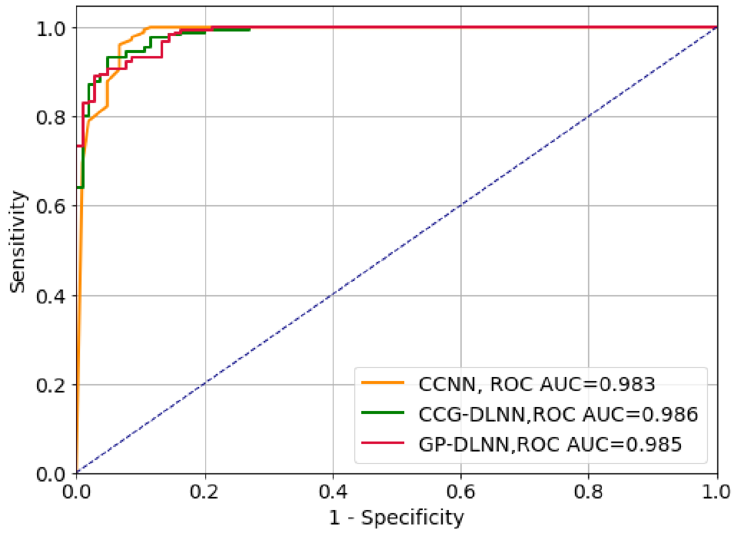

**Figure 13.** The ROC curves for CCG-DLNN, GP-DLNN and CCNN algorithms on the LIDC-IDR data set.

**Table 6.** Classification performance summary.

| Data Sets | AUC | | | Sensitivity (Avg.) | | | Specificity (Avg.) | | |
|---|---|---|---|---|---|---|---|---|---|
| | **CCG-DLNN** | **CCNN** | **GP-DLNN** | **CCG-DLNN** | **CCNN** | **GP-DLNN** | **CCG-DLNN** | **CCNN** | **GP-DLNN** |
| LIDC-IDRI | **0.92** | 0.84 | 0.86 | **0.93** | 0.89 | 0.87 | **0.79** | 0.67 | 0.71 |
| DH | **0.91** | 0.74 | 0.80 | **0.91** | 0.65 | 0.66 | 0.78 | 0.76 | **0.89** |
| HCV | **0.97** | 0.79 | 0.80 | **0.97** | 0.78 | 0.76 | **0.83** | 0.72 | 0.72 |
| HD | **0.98** | 0.80 | 0.97 | 0.92 | 0.78 | **0.97** | **0.90** | 0.65 | 0.82 |
| BCW | **0.95** | 0.74 | 0.88 | **1** | 0.81 | 0.98 | **1** | 0.80 | 0.97 |
| BCC | **0.98** | 0.74 | 0.94 | 0.90 | 0.77 | **0.94** | **0.94** | 0.60 | 0.80 |
| AI | **0.91** | 0.73 | 0.80 | 1 | 0.98 | **1** | **1** | 0.95 | 1 |
| LD | **0.85** | 0.74 | 0.79 | **0.90** | 0.61 | 0.85 | **0.85** | 0.58 | 0.60 |
| PID | **0.93** | 0.74 | 0.90 | **0.75** | 0.58 | 62 | **0.92** | 0.80 | 0.86 |

**Table 7.** AUC significance level between CCG-DLNN and GP-DLNN algorithms.

| Data Sets | Z Statistic | Significance Level (P) |
|---|---|---|
| LIDC-IDRI | 2.78 | 0.0054 |
| DH | 3.81 | 0.0001 |
| HCV | 5.84 | <0.0001 |
| HD | 2.95 | 0.0032 |
| BCW | 5.36 | <0.0001 |
| BCC | 2.89 | 0.0039 |
| AI | 2.36 | 0.0035 |
| LD | 5.03 | <0.0001 |
| PID | 3.22 | 0.0042 |

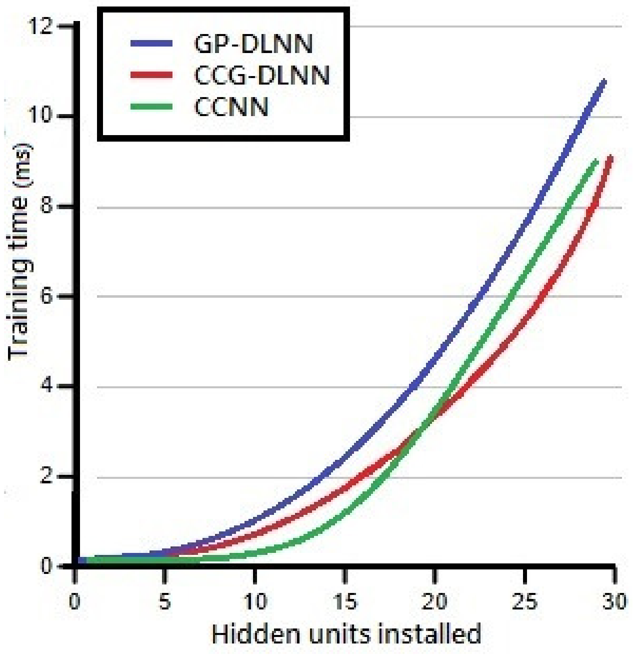

**Figure 14.** Runtime of GP-DLNN, CCG-DLNN and CCNN algorithm on the LIDC-IDRI data set.

## 5. Conclusions

In this paper, we proposed a method called (CCG-DLNN) that uses the growing step shown in [2], but by training the latest hidden unit (candidate unit) when connecting to the existing network, and freezing the hidden input weights of the network. Therefore, we repeatedly trained the weights linked to the output neurons after adding each new hidden node. To prevent the problem of over-fitting, we defined weights in a qualified network in a more coherent manner as in [28]. Theoretically, the convergence analysis of this algorithm has been proven. The proposed CCG-DLNN showed good results compared to standard algorithms. Its strength was evaluated on many data sets and the two mixed spirals were checked on a standard difficult benchmark. The structure of the neural network at our proposed algorithm was changed based on some cascade correlation algorithm features, so: We start with a candidate neuron and receiving input connections from all pre-existing hidden units at the last hidden layer. The candidate unit's output was not yet added to the current network until we do a sequence of passes over the data set for training, updating the weights of the candidate unit for each pass at the input side. The candidate neuron is linked to the output neurons (activated) and then all output connections are trained. The entire process is repeated multiple times until the desired accuracy of the network is achieved. Therefore, we are certain that each newly added neuron will improve the accuracy of the neural network, and so the proposed algorithm performs better than the others. Finally, our future research is reducing the running time of the CCG-DLNN algorithm.

**Author Contributions:** Methodology, M.F.F.; Software, S.A.E.-M.M. and M.H.M. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Data Availability Statement:** Not applicable.

**Conflicts of Interest:** The authors declare that they have no conflict of interest.

## References

1. Irsoy, O.; Alpaydin, E. Continuously constructive deep neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* **2019**, *31*, 1124–1133. [CrossRef] [PubMed]
2. Zemouri, R.; Omri, N.; Fnaiech, F.; Zerhouni, N.; Fnaiech, N. A new growing pruning deep learning neural network algorithm (GP-DLNN). *Neural Comput. Appl.* **2019**, *32*, 18143–18159. [CrossRef]
3. Cao, W.; Wang, X.; Ming, Z.; Gao, J. A review on neural networks with random weights. *Neurocomputing* **2018**, *275*, 278–287. [CrossRef]
4. Sharma, S.K.; Chandra, P. Constructive neural networks: A review. *Int. J. Eng. Sci. Technol.* **2010**, *2*, 7847–7855.
5. Alam, K.; Karmokar, B.C.; Siddiquee, M.K. A comparison of constructive and pruning algorithms to design neural networks. *Indian J. Comput. Sci. Eng.* **2011**, *2*. [CrossRef]
6. do Carmo Nicoletti, M.; Bertini, J.R.; Elizondo, D.; Franco, L.; Jerez, J.M. Constructive neural network algorithms for feedforward architectures suitable for classification tasks. In *Constructive Neural Networks*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 1–23.
7. Gallant, S.I. Three constructive algorithms for network learning. In Proceedings of the 8th Annual Conference of the Cognitive Science Society, Amherst, MA, USA, 15–17 August 1986; pp. 849–852.
8. Mezard, M.; Nadal, J.P. Learning in feedforward layered networks: The Tiling algorithm. *J. Phys. Math. Gen.* **1989**, *22*, 2191–2203. [CrossRef]
9. Frean, M. The Upstart algorithm: A method for constructing and training feed-forward neural networks. *Neural Netw.* **1990**, *2*, 198–209. [CrossRef]
10. Burgess, N. A constructive algorithm that converges for real-valued input patterns. *Int. J. Neural Syst.* **1994**, *5*, 59–66. [CrossRef]
11. Parekh, R.; Yang, J.; Honavar, V. *Constructive Neural Network Learning Algorithms for Multi-Category Pattern Classification*; Iowa Sate University: Ames, IA, USA, 1997.
12. Parekh, R.; Yang, J.; Honavar, V. Constructive neural-network learning algorithms for pattern classification. *IEEE Trans. Neural Netw.* **2000**, *11*, 436–451. [CrossRef]
13. Kwok, T.Y.; Yeung, D.Y. Constructive algorithms for structure learning in feedforward neural networks for regression problems. *IEEE Trans. Neural Netw.* **1997**, *8*, 630–645. [CrossRef]
14. Moody, J. Prediction risk and architecture selection for neural networks. In *From Statistics to Neural Networks*; Springer: Berlin/Heidelberg, Germany, 1994; pp. 147–165.

15. Chung, F.L.; Lee, T. Network-grwoth approach to design of feedforward neural networks. *IEE Proc. Control. Theory Appl.* **1995**, *142*, 486–492. [CrossRef]
16. Azimi-Sadjadi, M.R.; Sheedvash, S.; Trujillo, F.O. Recursive dynamic node creation in multilayer neural networks. *IEEE Trans. Neural Netw.* **1993**, *4*, 242–256. [CrossRef]
17. Setiono, R.; Hui, L.C.K. Use of a quasi-Newton method in a feedforward neural network construction algorithm. *IEEE Trans. Neural Netw.* **1995**, *6*, 273–277. [CrossRef]
18. Ash, T. Dynamic node creation in backpropagation networks. *Connect. Sci.* **1989**, *1*, 365–375. [CrossRef]
19. Bartlett, E.B. Dynamic node architecture learning: An information theoretic approach. *Neural Netw.* **1994**, *7*, 129–140. [CrossRef]
20. Hirose, Y.; Yamashita, K.; Hijiya, S. Backpropagation algorithm which varies the number of hidden units. *Neural Netw.* **1991**, *4*, 61–66. [CrossRef]
21. Khorasani, K.; Weng, W. Structure adaptation in feedforward neural networks. In Proceedings of the 1994 IEEE International Conference on Neural Networks (ICNN'94), Orlando, FL, USA, 28 June–2 July 1994; Volume 3, pp. 1403–1408.
22. Zhang, B.T. An incremental learning algorithm that optimizes network size and sample size in one trial. In Proceedings of the 1994 IEEE International Conference on Neural Networks (ICNN'94), Orlando, FL, USA, 28 June–2 July 1994; Volume 1, pp. 215–220.
23. Fahlman, C.L. The cascade-correlation learning architecture. *Adv. Neural Inf. Process. Syst.* **1990**, *2*, 524–532.
24. Friedman, J.H.; Stuetzle, W. Projection pursuit regression. *J. Am. Stat. Assoc.* **1981**, *76*, 817–823. [CrossRef]
25. Platt, J. A resource-allocating network for function interpolation. *Neural Comput.* **1991**, *3*, 213–225. [CrossRef]
26. Farlow, S.J. (Ed.) Self-Organizing Methods in Modeling: GMDH Type Algorithms. In *Statistics: Textbooks and Monographs*; Marcel Dekker: New York, NY, USA, 1984; Volume 54.
27. Nabhan, T.M.; Zomaya, A.Y. Toward generating neural network structures for function approximation. *Neural Netw.* **1994**, *7*, 89–90. [CrossRef]
28. Huemer, A.; Elizondo, D.; Gongora, M. A Constructive Neural Network for Evolving a Machine Controller in Real-Time. In *Constructive Neural Networks*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 225–242.
29. Heaton, J. *Ian Goodfellow, Yoshua Bengio and Aaron Courville. Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
30. Livieris, I.E. Improving the classification efficiency of an ANN utilizing a new training methodology. *Informatics* **2019**, *6*, 1. [CrossRef]
31. Kwok, T.-Y.; Yeung, D.-Y. Objective functions for training new hidden units in constructive neural networks. *IEEE Trans. Neural Netw.* **1997**, *8*, 1131–1148. [CrossRef] [PubMed]
32. Armato, S.G., 3rd; McLennan, G.; Bidaut, L.; McNitt-Gray, M.F.; Meyer, C.R.; Reeves, A.P.; Zhao, B.; Aberle, D.R.; Henschke, C.I.; Hoffman, E.A.; et al. The Lung Image Database Consortium (LIDC) and Image Database Resource Initiative (IDRI): A completed reference database of lung nodules on CT scans. *Med. Phys.* **2011**, *38*, 915–931. [CrossRef] [PubMed]
33. Dua, D.; Graff, C. *UCI Machine Learning Repository*; University of California, School of Information and Computer Science: Irvine, CA, USA, 2019. Available online: http://archive.ics.uci.edu/ml (accessed on 5 May 2021).
34. Strack, B.; DeShazo, J.P.; Gennings, C.; Olmo, J.L.; Ventura, S.; Cios, K.J.; Clore, J.N. Impact of HbA1c Measurement on Hospital Readmission Rates: Analysis of 70,000 Clinical Database Patient Records. *BioMed Res. Int.* **2014**, *2014*, 781670. [CrossRef]
35. Patrício, M.; Pereira, J.; Crisóstomo, J.; Matafome, P.; Gomes, M.; Seiça, R.; Caramelo, F. Using Resistin, glucose, age and BMI to predict the presence of breast cancer. *BMC Cancer* **2018**, *18*, 29. [CrossRef]
36. Czerniak, J.; Zarzycki, H. Application of rough sets in the presumptive diagnosis of urinary system diseases In *Artifical Inteligence and Security in Computing Systems*; Kluwer Academic Publishers: Dordrecht, The Netherlands, 2003; pp. 41–51.