THE NEWTON NEURAL NET: A NEW APPROXIMATING NETWORK

A.M. Ugena

Department of Applied Mathematic

ETSI Telecommunication

Universidad Politécnica de Madrid

Avenida Complutense, 30 “Ciudad Universitaria”

28040, Madrid, SPAIN

Abstract: In this paper we present the Newton neural network. We state several theorems that reveal this trained net as the best approximator to any curve or surface with the least number of training patterns. The low number of training samples is an advantage in this type of neural networks, as it affects substantially lowering the computational cost and training time.

AMS Subject Classification: 68T05

Key Words: backpropagation, artificial neural networks, adaptive learning, functional-link neural networks

1. Introduction

The Stone-Weierstrass theorem states that every continuous function can be approximated by means of polynomials with any arbitrary degree of accuracy and, as we know, that this result can be extended to bounded measurable functions. These theoretical conclusions have been used to implement artificial neural networks like multilayer perceptron or polynomial neural networks, for example. We can see different models about last ones in [7]. Until the moment

and regarding approximation, classification and regression it is known the efficiency of these nets. They have been found superior to statistical methods and traditional perceptron network on various practical tasks that require higher order nonlinear descriptions such as biological modeling, image processing, noisy real-world time series modeling, financial time series prediction, etc.

The bibliography and the research relied on polynomials neural nets is very broad and it can be found some nets that use polynomial terms on its hidden layers in multilayer perceptron [8], [2], [9], [3], polynomial nets dependent on the time [5], linear polynomial nets with adaptive exponent entrance [7], self-organizing polynomial neural networks [6], to extract feature extraction [10], applied to financial time series prediction [4], applied to domain complex [11] etc.

If one consider the Walsh theorem and the Weierstrass theorem, one can obtain the best approximation of functions based on interpolation on $L^2(\mu)$ space, with the Lebesgue measure. Therefore, one could apply, as approximator polynomials that appear in the Stone-Weierstrass theorem, the interpolator polynomial of Newton, Lagrange, etc according to the Walsh theorem.

On this line of reasoning we developed the neural net that we present in this paper and others which can be consulted in [12], [13], [14]. We agreed to call the net shown in this case as Newton neural network.

The development of the interpolator polynomials of Newton, Lagrange, etc, relies on the contained information in the training patterns, that is, in the values of the function graph we wants to approach. So, being the done information by the training patterns so outstanding, we thought we would use them as a constituent part of the activation functions of the nodes of the input layer.

Therefore, with a reduced number of training patterns, we can achieve to approximate a function or classify patterns. This reduction on the number of used training patterns, in contrast with other neural net models, implies a decline in the training time, a lower computational cost, a easy generalization and a better approximation for the output function of the net.

By means of the proposed theorems, we demonstrate that this net, with the corresponding training and learning rule, is the best approximator on compact sets to functions that belong to the L^2 space with the Lebesgue measure.

The paper is organized this way: first we describe the architecture and learning algorithm of the Newton neural net; Section 3 explains necessary and sufficient conditions to show that the proposed net is an universal approximator on compact sets and that it is the best mean squared approximator. Finally, the latest section includes experimental studies.

2. The Newton Neural Network. Network Architecture and Learning Algorithm

Let us consider P training patterns

$$\{(x_0, T(x_0)), (x_1, T(x_1)), \dots, (x_P, T(x_P))\}, \quad (1)$$

where $T(x_i)$ is the target output given the x_i input patterns and let us take, amongst themselves, $x_0, \dots, x_n$, $n \leq P$.

In some cases, when we are pretending to approximate a function $f \in L^2(\mu)$ we will suppose that x_i belongs to domain of f and $T(x_i)$ is its image $f(x_i)$.

If $f \in L^2(\mu)$, the Walsh and Stone-Weierstrass theorems establish that given P distinct points $z_1, \dots, z_P$ and given any $\epsilon > 0$ there exist only one polynomial $p(z)$ such that $f(z_i) = p(z_i)$ $i = 1, \dots, P$ and $\|f(z) - p(z)\| < \epsilon$.

Speaking about the Newton neural net, the input layer has n nodes, $n < P$, where n will be established depending on the function to approximate or the number of parameters to recognize. For example, if we want to approximate a polynomial of m degree and we have m distinct training patterns belonging to this polynomial, we will obtain, after the net training, an output function that approximates this polynomial with any degree of accurate on its whole domain of definition. In this case we would implement m nodes for the input layer.

The activation function associated to every j node of the input layer, which we have noted by $F_j(x)$, is a term of the interpolating polynomial of Newton $\prod_{i=0}^{j-1} (x - x_i)$ being $F_{-1}(x) = 1$

The activation function of the output layer is the identity function and the Newton neural net is a flat net, therefore the output of the net $O(x)$ will be

$$\begin{aligned} O(x) &= F \left(\sum_{j=0}^n w_j F_{j-1}(x) \right) = Id \left(\sum_{j=0}^n w_j F_{j-1}(x) \right) \\ &= w_0 + \sum_{j=1}^n w_j \prod_{i=0}^{j-1} (x - x_i), \quad (2) \end{aligned}$$

which is a truncated series of Newton. Every one of coefficients w_j is a weight of the net that will change until the net may be trained. On this way, the net achieves the generalization from the P training patterns to all points belonging to the done graph function, that is, we could recognize all points belonging to this function.

Graphically, the architecture can be like that

Figure 1: Newton's net architecture

2.1. Net Training

First the weights w_j , $j = 0, \dots, n$ are chosen at random such that $w_j \in [-0.5, 0.5]$.

Let us consider the set of training pairs $\{(x_0, T(x_0)), \dots, (x_P, T(x_P))\}$. For every training pattern $(x_k, T(x_k))$ we calculate the net output $O(x_k)$ and the corresponding squared error

$$E_k = \frac{1}{2} (T(x_k) - O(x_k))^2 = \frac{1}{2} (\delta_k)^2. \quad (3)$$

By using the delta rule of learning we are supposing $T(x_k) - O(x_k) = \delta_k$. Let lms consider the bound of mean squared error:

$$E = \frac{1}{P} \sum_{k=1}^P E_k. \quad (4)$$

If $lms > E$ then the net has been trained and we consider the weights w_j as the coefficients for the net output $O(x)$.

If $lms \leq E$, by applying the backpropagation algorithm and the delta rule learning, the weights will go changing according to next method.

We will consider the next notation with respect to the $m + 1$ step.

$w_j(m + 1)$ is the value of the weight w_j at the $m + 1$ iteration.

$O(x_k)(m + 1)$ is the net output at the $m + 1$ iteration for the input pattern x_k , and

$$\delta_k(m + 1) = T(x_k) - O(x_k)(m + 1) \quad (5)$$

The change of every weight w_j at the $m + 1$ iteration becomes calculated like this:

For every x_k it will be calculated the increase

$$\Delta_k w_j(m+1) = \eta \delta_k(m+1) F_{j-1}(x_k) + \alpha \Delta_k w_j(m)$$

being $\eta \in [0, 1]$ the learning rate and α a proportionality constant.

Then the total increase for the weight w_j at the $m + 1$ iteration

$$\Delta w_j(m+1) = \sum_{k=1}^P \Delta_k w_j(m+1) \quad (6)$$

The value of the changed weight at the $m + 1$ iteration will be

$$w_j(m+1) = w_j(m) + \Delta w_j(m+1). \quad (7)$$

With these news values of each one of the weights, we have a new net output for every input pattern and we calculate the new total error E that we will compare with the *lms* bound, once more.

3. Necessary and Sufficient Conditions for the Convergence of Newton Neural Network

In order to present the Newton neural network as the best least squared approximator on compact sets, this section proposes a set of theorems as a result of the Walsh and the Stone-Weierstrass theorems.

Theorem 1. (Newton Neural Net as an Approximator on Compact Sets) *Let a function $f \in L^2$ and $\{x_n\}_{n \in N}$ a sequence of distinct points and $F_j(x)$ defined as before. The function f can be uniformly approximated by means of polynomials p*

$$p \in L\left(\{1, F_j(x)\}_{j \in N \cup \{0\}}\right)$$

which verify: $p(x_i) = f(x_i)$, $i \in \{1, 2, \dots, n\}$.

Theorem 2. (Newton Neural Net as Least Square Approximator) *Let $\{x_0, x_2, \dots, x_n\}$ be $n + 1$ distinct points and a function $f \in L^2$. If f can be uniformly approximated by polynomials $p \in L\left(\{F_j(x)\}_{j \in N}\right)$ then f can be least square approximated by these polynomials p .*

To finish off we give this later theorem which aim is to explain the taken backpropagation training as the method to get the best approximation in the sense of the least square power.

Theorem 3 (Newton Neural Net As The Best Least Square Approximator). *Let $\{x_1, x_2, \dots, x_P\}$ be P distinct points, with $P > n + 1$. The problem of finding*

$$\min_{\{w_0, \dots, w_n\}} \left\{ \sum_{j=1}^P \left(f(x_j) - \left(\sum_{i=0}^n w_i \prod_{k=0}^{i-1} (x_j - x_k) \right) \right)^2 \right\} \quad (8)$$

has a solution. Such a solution yields a best approximation to $f(x)$ in the sense of the least square power.

4. Experimental Studies

4.1. Approximating Polynomial Functions

If it is a polynomial function with n degree the one which you want to recognize by means of the neural net of Newton then, by using the set of functions $\{F_0, \dots, F_{n-1}\}$ and the one of n distinct input patterns $\{x_0, \dots, x_{n-1}\}$ belonging to a compact set A to train the net, you will obtain the recognition of the full function. For example:

$$p(x) = x^4 - 8x^3 + 2x^2 + 3x - 5. \quad (9)$$

The training patterns have been taken on the interval $[0, 2]$:

x_i	$p(x_i)$
0	-5
0.5	-3.9375
1	-7
1.5	-17.9375
2	-39

Table 1

Taking the first four patterns then the polynomial terms are

$$\begin{aligned}
 F_{-1}(x) &= 1 \\
 F_0(x) &= x \\
 F_1(x) &= x(x - 0.5) \\
 F_2(x) &= x(x - 0.5)(x - 1) \\
 F_3(x) &= x(x - 0.5)(x - 1)(x - 1.5)
 \end{aligned} \tag{10}$$

Considering the training parameters $\alpha = 0.9$, $\eta = 0.01$, with the total error $E = 494707 \times 10^{-8}$ and 336 steps, the weights obtained from the trained net are:

$$\begin{aligned}
 w_0 &= -4.9998 \\
 w_1 &= 2.12363 \\
 w_2 &= -8.24679 \\
 w_3 &= -5.00418 \\
 w_4 &= 1.00365,
 \end{aligned} \tag{11}$$

which constitute the coefficients of the output of the net $O(x)$. The one $O(x)$ according to (2) will be

$$\begin{aligned}
 O(x) &= -4.9998 + 2.12363x - 8.24679x(x - 0.5) \\
 &\quad - 5.00418x(x - 0.5)(x - 1) \\
 &\quad + 1.00365x(x - 0.5)(x - 1)(x - 1.5)
 \end{aligned} \tag{12}$$

The following figure shows the good approximation between the graph of $O(x)$, represented by means of a continuous line, and the one of the function $f(x) = p(x)$, represented by means of cross points, on the interval $[0, 2]$.

Considering the output of the trained net, $O(x)$, and the given function $f(x)$ taking values out of the interval $[0, 2]$, it can be observed the complete accuracy between both functions. The following figure shows this result on the $[-5, 5]$ domain.

Now taking into account the training patterns from the table 1, we are going to compare this results with the ones of the trained two-layer perceptron. After 8000 steps and with an error $E = 0.515$ the output of the trained perceptron is:

The following figure shows the generalization of the output for the trained two-layer perceptron. The cross points are the training pairs.

As we can observe, with the same number of training patterns, we can obtain that the neural network of Newton outperforms the two-layer perceptron.

Figure 2

Figure 3

x_i	$T(x_i) = f(x_i)$	$O(x_i)$
0	-5	-4.68
0.5	-3.9375	-4.36
1	-7	-7.19
1.5	-17.9375	-17.52
2	-39	-39.34

Table 2

4.2. Non Polynomial Functions

If the function to approximate is not a polynomial function then we must focus on a compact set choosing some input patterns:

Figure 4

1 Let

$$f(x) = \frac{\sin(x)}{x}$$

Let $[0.2, 4]$ be the interval (compact set) to take the input patterns. These are shown in the following table:

x_i	$T(x_i) = \frac{\sin(x_i)}{x_i}$
0.2	0.99334665
1.8	0.54102646
3	0.04704
3.4	-0.07515914
3.5	-0.100223
3.7	-0.1431989757
3.8	-0.16
4	-0.1892

Table 3

$\{0.2, 1.8, 3\}$ are the chosen input patterns to form the enhanced functions:

$$\begin{aligned}
 F_{-1}(x) &= 1 \\
 F_0(x) &= (x - 0.2) \\
 F_1(x) &= (x - 0.2)(x - 1.8) \\
 F_2(x) &= (x - 0.2)(x - 1.8)(x - 3)
 \end{aligned} \tag{13}$$

With $\alpha = 0.99$, $\eta = 0.01$ and the mean square error was $3.553e - 06$ with 373 steps the weights obtained are:

$$\begin{aligned} w_0 &= 0.993364 \\ w_1 &= -0.284151 \\ w_2 &= -0.0449441 \\ w_3 &= 0.0330348 \end{aligned} \tag{14}$$

Then, the output Newton net $O(x)$ is:

$$\begin{aligned} O(x) &= 0.993364 - 0.284151(x - 0.2) - \\ &\quad - 0.0449441(x - 0.2)(x - 1.8) + \\ &\quad + 0.0330348(x - 0.2)(x - 1.8)(x - 3) \end{aligned} \tag{15}$$

In the next figure we see the graphs of the $\frac{\sin(x)}{x}$ function by means of the continuous line and the output net $O(x)$ by points:

Figure 5

Using the same training patterns for the two-layer perceptron, with 8000 steps and a training error $E = 0.000739$ we obtain the next output $O(x)$

The continuous line corresponds to trained two-layer perceptron. The cross points are the training pairs $(x_i, O(x_i))$.

2 Let

x_i	$T(x_i)$	$O(x_i)$
0.2	0.99334665	0.9966
1.8	0.54102646	0.5306
3	0.04704	0.0490
3.4	-0.07515914	-0.0677
3.5	-0.100223	-0.0934
3.7	-0.1431989757	-0.1412
3.8	-0.16	-0.1635
4	-0.1892	-0.2052

Table 4

Figure 6

$$f(x)=\frac{4x^3-1.2x^2-3x+1.2}{0.4x^5+0.8x^4-1.2x^3+0.2x^2-3},$$

where the input variable belongs to the interval $[-1, 1]$. Graphically, it can be expressed

This function is taken from the paper [8]. The authors approximate this function by means of a polynomial neural net. Results were obtained taking into account 100 samples to train the net. The average approximation error were about 0.02.

Figure 7

By means of Newton neural net we can approximate this function with only 9 samples, that is, 9 training patterns.

We divide the interval $[-1, 1]$ in two segments $[-1, 0]$ and $[0, 1]$, then we take 5 training patterns, repeating the input pattern 0 in both ones, of each interval that we expose in the tables 5 and 6. We are going to consider these intervals separately.

On the interval $[-1, 0]$:

x_i	$f(x_i)$
-1	0.8333
-0.8	-0.380
-0.5	-0.687
-0.2	-0.577
0	-0.4

Table 5

So that, the expansion functions will be expressed:

$$\begin{aligned}
 F_{-1}(x) &= 1 \\
 F_0(x) &= (x + 1) \\
 F_1(x) &= (x + 1)(x + 0.5) \\
 F_2(x) &= (x + 1)(x + 0.5)(x) \\
 F_3(x) &= (x + 1)(x + 0.5)(x)(x + 0.2).
 \end{aligned} \tag{16}$$

The error E and the steps at the end are $9.9e - 10$, and 17377 respectively with $\alpha = 0.99$ and $\eta = 0.2$. The weights achieved are

$$\begin{aligned} w_0 &= 0.832985 \\ w_1 &= -3.0401 \\ w_2 &= 3.61419 \\ w_3 &= -3.22384 \\ w_4 &= 8.1015. \end{aligned} \tag{17}$$

Therefore the output Newton net will be

$$\begin{aligned} O(x) &= 0.832985 - 3.0401(x + 1) + \\ &\quad + 3.61419(x + 1)(x + 0.5) - \\ &\quad - 3.22384(x + 1)(x + 0.5)(x) \\ &\quad + 8.1015(x + 1)(x + 0.5)(x)(x + 0.2). \end{aligned} \tag{18}$$

The following figure shows the generalization of the training set to the whole interval $[-1, 0]$ for the trained net of Newton and the graph of the function

$$f(x) = \frac{4x^3 - 1.2x^2 - 3x + 1.2}{0.4x^5 + 0.8x^4 - 1.2x^3 + 0.2x^2 - 3}.$$

The gray points for the function $f(x)$ and the black points for the output net $O(x)$

Figure 8

If we apply the same process to the interval $[0, 1]$:

x_i	$f(x_i)$
0	-0.4
0.65	0.0518
1	-0.357
0.4	-0.021
0.8	-0.026

Table 6

$$\begin{aligned}
F_{-1}(x) &= 1 \\
F_0(x) &= x \\
F_1(x) &= (x)(x - 0.65) \\
F_2(x) &= (x)(x - 0.65)(x - 1) \\
F_3(x) &= (x)(x - 0.65)(x - 1)(x - 0.4)
\end{aligned} \tag{19}$$

Being $\alpha = 0.9$, $\eta = 0.2$, the error $E = 9.9e - 8$ and 16292 the spent steps, the weights achieved

$$\begin{aligned}
w_0 &= -0.3999 \\
w_1 &= 0.696134 \\
w_2 &= -1.86579 \\
w_3 &= -1.43875 \\
w_4 &= -0.603902
\end{aligned} \tag{20}$$

And the output of the Newton net:

$$\begin{aligned}
O(x) &= -0.3999 + 0.696134(x) - 1.86579(x)(x - 0.65) - \\
&\quad - 1.43875(x)(x - 0.65)(x - 1) \\
&\quad - 0.603902(x)(x - 0.65)(x - 1)(x - 0.4)
\end{aligned} \tag{21}$$

As the previous interval, the following figure shows the generalization of the training set and the graph of the

$$f(x) = \frac{4x^3 - 1.2x^2 - 3x + 1.2}{0.4x^5 + 0.8x^4 - 1.2x^3 + 0.2x^2 - 3}$$

with the same colour points

If we compare the graphics on the interval $[-1, 1]$:

4.3. Another Curves in the Plain

In the case of regular curves or surfaces, we can observe that the results obtained are so good. We give the next cases

Figure 9

Figure 10

We are going to consider the upper curve of the following figure

Figure 11

The training pairs $(x_i, y_i) = (x_i, T(x_i))$ were obtained drawing a mesh over

this curve and dividing it into subintervals for x_i

Figure 12: Mesh over upper curve

The next table shows the chosen subintervals

<i>Subinterval</i>	<i>Section</i>
$0 \leq x \leq 3$	1
$3 \leq x \leq 6$	2
$6 \leq x \leq 8$	3
$8 \leq x \leq 11$	4
$11 \leq x \leq 14$	5
$14 \leq x \leq 17$	6
$17 \leq x \leq 20$	7
$20 \leq x \leq 23$	8
$23 \leq x \leq 26$	9
$26 \leq x \leq 27.7$	10
$27.7 \leq x \leq 30$	11

Table 7

The following table describes, for each subinterval and section, the chosen training patterns $(x_i, T(x_i))$

The number of steps and the mean squared errors spent to train the net were:

And the following outputs of the trained net of Newton with the weights

Taking the weights and the outputs the obtained curve is

<i>Section</i>	$\{x_0, x_1, x_2, x_3\}$	$T(x_i)$
1	$\{0, 1, 2, 3\}$	$\{3, 3.7, 3.8, 3.8\}$
2	$\{4, 5, 5.4, 6\}$	$\{3.8, 3.9, 4, 4.2\}$
3	$\{6, 6.5, 7, 8\}$	$\{4.2, 5, 5.7, 6.6\}$
4	$\{8, 9, 10, 11\}$	$\{6.6, 7, 7.1, 7.01\}$
5	$\{11, 12, 13, 14\}$	$\{7.01, 6.99, 6.7, 6.2\}$
6	$\{14, 15, 16, 17\}$	$\{6.2, 5.7, 5, 4.5\}$
7	$\{17, 18, 19, 20\}$	$\{4.5, 6, 6.8, 7\}$
8	$\{20, 21, 22, 23\}$	$\{7, 6.95, 6.7, 6.1\}$
9	$\{23, 24, 25, 26\}$	$\{6.1, 5.6, 5.8, 5.8\}$
10	$\{26, 26.5, 27, 27.7\}$	$\{5.8, 5.5, 5.2, 4.1\}$
11	$\{27.7, 28, 29, 30\}$	$\{4.1, 4.3, 4.1, 3\}$

Table 8

<i>Section</i>	<i>Steps</i>	<i>SquareError</i>
1	176	$4.70449 \cdot 10^{-8}$
2	159	$4.55144 \cdot 10^{-8}$
3	161	3.4298310^{-8}
4	204	4.9578710^{-8}
5	213	4.9272310^{-8}
6	210	4.7619510^{-8}
7	179	1.2321610^{-8}
8	209	4.857610^{-8}
9	212	4.9710410^{-8}
10	164	7.2993810^{-9}
11	1632	3.135410^{-8}

Table 9

4.4. Functions of Two Variables

Considering the function

$$f\left(x,y\right)=\frac{\sin\sqrt{x^2+y^2}}{\sqrt{x^2+y^2}}+1$$

<i>Section</i>	$O(x_i)$
1	{2.99982, 3.70054, 3.8, 3.79977}
2	{3.79958, 3.90003, 4.00026, 4.20035}
3	{4.20034, 4.9997, 5.70027, 6.6003}
4	{6.59952, 7.0004, 7.09997, 7.01002}
5	{7.00955, 6.99038, 6.69979, 6.20008}
6	{6.19954, 5.70035, 4.99979, 4.50001}
7	{4.50001, 6.0001, 6.79999, 7.0003}
8	{6.99954, 6.95031, 6.69975, 6.09987}
9	{6.09956, 5.60037, 5.79982, 5.80016}
10	{5.80016, 5.50001, 5.20002, 4.10013}
11	{4.09984, 4.29993, 4.10035, 2.99969}

Table 10

<i>Section</i>	$\{w_0, w_1, w_2, w_3\}$
1	{2.99982, 3.70054, 3.8, 3.79977}
2	{3.79958, 0.10045, 0.0666698, 0.0665242}
3	{4.20034, 4.9997, -0.197596, -0.0683265}
4	{6.59952, 0.400886, -0.150661, 0.0186344}
5	{7.00955, -0.0191752, -0.135707, 0.0103825}
6	{6.19954, -0.499188, -0.100688, 0.0670275}
7	{4.50001, 1.50009, -0.350099, 0.0167685}
8	{6.99954, -0.0492296, -0.100666, -0.0246636}
9	{6.09956, -0.499183, 0.349312, -0.14962}
10	{5.80016, -0.600143, -0.571248, 0}
11	{4.09984, 0.666985, -0.666589, 0.0939339}

Table 11

To train the net, 16 training pairs and 15 enhanced inputs were taken

The training pairs were chosen such that only one coefficient could be obtained for each pattern and the input patterns belonged to $[0, 3] \times [0, 3]$. With the next parameters it was obtained the accuracy trained net

This table shows the set of training pairs and the net objective output after

Figure 13: Mesh over upper curve

1	$(x - x_0) (x - x_1) (y - y_0)$
$(x - x_0)$	$(x - x_0) (y - y_0) (y - y_1)$
$(x - x_0) (x - x_1)$	$(x - x_0) (x - x_1) (y - y_0) (y - y_1)$
$(x - x_0) (x - x_1) (x - x_2)$	$(x - x_0) (x - x_1) (x - x_2) (y - y_0)$
$(y - y_0)$	$(x - x_0) (x - x_1) (x - x_2) (y - y_0) (y - y_1)$
$(y - y_0) (y - y_1)$	$(x - x_0) (y - y_0) (y - y_1) (y - y_2)$
$(y - y_0) (y - y_1) (y - y_2)$	$(x - x_0) (x - x_1) (y - y_0) (y - y_1) (y - y_2)$
$(x - x_0) (y - y_0)$	$(x - x_0) (x - x_1) (x - x_2) (y - y_0) (y - y_1) (y - y_2)$

Table 12

$\alpha = 0.99$	$\eta = 0.0005$	$E = 9.33445 \cdot 10^{-7}$	$Steps = 1229$

Table 13

the training:

The weights will be

$$\begin{aligned} O(x, y) = p(x, y) = & 1.99792 - 0.158019x - 0.113211x(x - 1) + \\ & + 0.0340444x(x - 1)(x - 2) \\ & - 0.157396y - 0.113816y(y - 1) + 0.034216y(y - 1)(y - 2) + \\ & + 0.0154977xy + 0.0116215x(x - 1)y + 0.011703xy(y - 1) + \\ & + 0.00939871x(x - 1)y(y - 1) - 0.00229217x(x - 1)(x - 2)y - \\ & - 0.00167421x(x - 1)(x - 2)y(y - 1) - \\ & - 0.00212448xy(y - 1)(y - 2) - \end{aligned}$$

	(x_i, y_i)	$T(x_i, y_i)$	$O(x_i, y_i)$
(x_0, y_0)	$(0, 0)$	2	1.99792
(x_1, y_0)	$(1, 0)$	1.841470985	1.8399
(x_2, y_0)	$(2, 0)$	1.454648713	1.45546
(x_3, y_0)	$(3, 0)$	1.047040003	1.04886
(x_0, y_1)	$(0, 1)$	1.841470985	1.84052
(x_0, y_2)	$(0, 2)$	1.454648713	1.45549
(x_0, y_3)	$(0, 3)$	1.047040003	1.04813
(x_1, y_1)	$(1, 1)$	1.698455999	1.698
(x_2, y_1)	$(2, 1)$	1.351844908	1.3523
(x_1, y_2)	$(1, 2)$	1.351844908	1.35187
(x_2, y_2)	$(2, 2)$	1.108919809	1.10591
(x_3, y_1)	$(3, 1)$	0.993459293	0.993932
(x_3, y_2)	$(3, 2)$	0.875888118	0.874284
(x_1, y_3)	$(1, 3)$	0.993459293	0.994071
(x_2, y_3)	$(2, 3)$	0.875888118	0.874248
(x_3, y_3)	$(3, 3)$	0.789828477	0.788832

Table 14

$$-0.00182161x(x-1)y(y-1)(y-2)-7.1114610^{-5}x(x-1)(x-2)y(y-1)(y-2)$$

Following pictures show both functions, $O(x, y) = p(x, y)$, in lightbluegreen color and $f(x, y) = \frac{\sin \sqrt{x^2 + y^2}}{\sqrt{x^2 + y^2}} + 1$, in lightgrey color.

Figure 14: $p(x, y)$: green; $f(x, y)$: grey

w_i
1.99792
-0.158019
-0.113211
0.0340444
-0.157396
-0.113816
0.034216
0.0154977
0.0116215
0.011703
0.00939871
-0.00229217
-0.00167421
-0.00212448
-0.00182161
$-7.11146 \cdot 10^{-5}$

Table 15

Figure 15: $p(x, y)$: green $f(x, y)$: grey

As we can observe, both the output net $O(x, y)$ and the function $f(x, y)$ match up into the domain $[0, 3] \times [0, 3]$ where it has taken the training patterns

Compared to another neural networks that they need to train more than 100 training patterns in order to approximate the function

$$f(x, y) = \frac{\sin \sqrt{x^2 + y^2}}{\sqrt{x^2 + y^2}} + 1,$$

the neural net of Newton only needs 16 training patterns to get such result.

5. Conclusions

This results are corroborated by several theorems elaborated for that and presented in this paper.

From the results so far obtained it can be stated that this neural net is most suitable for a sort of problems needing the reduction of the training period, the reduction of the error level or of the computing time for solution.

The Newton neural network has been presented, in this paper, like a flat polynomial neural net of supervised learning.

It is demonstrated that it can be a good tool to rebuild curves and surfaces using some points belonging to those.

Acknowledgments

The economic support provided by the research group of the Universidad Politécnica of Madrid, Life Supporting Technologies.

The author is grateful to F. de Arriaga and A. González for reading a preliminary version of the article.

References

- [1] Richard L. Burden, J. Douglas Faires, *Numerical Analysis*, Fifth Edition, PWS Publishing Company, Boston.
- [2] M. Chen, M.T. Manry, Power series analysis of back-propagation neural networks, *IEEE* (1991), 0-7803-0164-1/91.
- [3] K.H. Chon, R.J. Cohen, Linear and nonlinear ARMA model parameter estimation using an artificial neural network, *IEEE Trans. on Biomedical Engineering*, **44**, No. 3 (1997), 168-174.
- [4] R. Ghazaly, A. Hussain, W. El-Deredy, Application of Ridge polynomial neural network to financial time series prediction, In: *Int. Joint Conf. on Neural Networks*, Vancouver B.C. Canada, Tydale House Press (2006) 235-239.
- [5] N. Hoffman, *A Neural Feed-Forward Network with a Polynomial Nonlinearity* (0-7803-0557-4/92).

- [6] D. Kim, G.T. Park, Advanced self-organizing polynomial neural network, *Neural Comput. and Applic.* (2007), 443-452, **doi:** 10.1007/s00521-006-0070-x
- [7] N.Y. Nikolaev, H. Iba, Learning polynomial feedforward neural networks by genetic programming and backpropagation, *IEEE Transactions on Neural Networks*, **14**, No. 2 (2003), 337-350.
- [8] A. Patrikar, J. Provence, Pattern classification using polynomial networks, *Electronics Letters*, **28**, No. 12 (1992), 1109-1110.
- [9] Y. Shin, J. Ghosh, Approximation of multivariate functions using ridge polynomial, *IEEE*, (1992), 0-7803-0559-0/92.
- [10] E.M. Tamil, H.M. Radzi, M.Y.I. Idris, A.M. Tamil, A review on feature extraction and classification techniques for biosignal processing, In: *Proc. 21 Biomed* (2008), 113-116.
- [11] B.K. Tripathi, B. Chandra, P.K. Kalra, The generalized product neuron model in complex domain, In: *ICONIP 2008; Part II*, (Ed-s: M. Köppen et al.), LNCS 5507 (2009), 867-876.
- [12] A. Ugena, F. De Arriaga, El Alami, Speaker-independent speech recognition by means of functional link neural networks, In: *Proc. 15-th Int. Conf. on Pattern Recognition*, **2** (2000), 1022-1025.
- [13] A. Ugena, F. De Arriaga, El Alami, Phoneme recognition by means of functional link neural networks, In: *Conference Internationale de Mathematiques Appliquees et Sciences de L'Ingenieur*, ENSEM Eds., Casablanca (2000), 106-116.
- [14] A. Ugena, F. De Arriaga, El Alami, Neural network architecture: New strategies for real time problems, In: *Int. Conf. on Computing and Information Technologies* (Ed-s: G. Antoniou, D. Deremer), New Jersey, CA, World Scientific (2001), 247-253.