ON THE GREEDY RIDGE FUNCTION NEURAL NETWORKS FOR APPROXIMATION MULTIDIMENSIONAL FUNCTIONS

Reyadh S. Naoum and Najlaa M. Hussein*
Department of Mathematics, College of Science, University of Baghdad. Baghdad -Iraq.
**Department of Computer Science, College of Science, University of Baghdad. Baghdad -Iraq.*

Abstract
The aim of this paper is to approximate multidimensional functions $f \in C(\mathbb{R}^s)$ by developing a new type of Feedforward neural networks (FFNNs) which we called it Greedy ridge function neural networks (GRGFNNs). Also, we introduce a modification to the greedy algorithm which is used to train the greedy ridge function neural networks. An error bound are introduced in Sobolev space. Finally, a comparison was made between the three algorithms (modified greedy algorithm, Backpropagation algorithm and the result in [1]).

حول تقريب الدوال متعددة الابعاد باستخدام الشبكات العصبية ذات الدوال الصلبة مع خوارزمية كريدي

الخلاصة
الهدف الرئيسي من هذا البحث هو تقريب الدوال المتعددة الابعاد $f \in C(\mathbb{R}^s)$ باستخدام نوع جديد من الشبكات العصبية التقدمية (FFNNs) والتي اطلقنا عليها اسم الشبكات العصبية ذات الدوال الصلبة (GRGFNNs). ايضاً، قدمنا تعديل لخوارزمية كريدي (Greedy) والتي استخدمت لتدريب الشبكات العصبية ذات الدوال الصلبة. تم تقديم حد للخطأ في فضاء سبولوف (Sobolev). اخيراً، قدمنا مقارنة بين الخوارزميات الثلاث (خوارزمية كريدي المعدلة وخوارزمية Backpropagation والنتائج المستحصلة في [14]).

Introduction

Approximations of multidimensional function have been studied by many researchers such as Burger and Neubauer [2], Ciesielski and Sacha [3], Ellacott [4] and Pinkus [5]. Burger and Neubauer, [2], they gave an error bound in Sobolev space $W^{m,p}(\Omega)$ for approximation multidimensional functions by a linear combination of ridge functions. Ciesielski and Sacha, [3], focused on a development of a constructive formula for the upper bound of L_∞ error approximation. Ellacott, [4], proved that a semilinear (multilayer perceptron MLP) feedforward network with one hidden layer can uniformly approximate any continuous function in C(K) where K is a compact set in $\mathbb{R}^s$ and s is a positive integer. Pinkus, [5], presented an algorithm for approximating the multidimensional function by using feedforward neural network.

The main result of this paper is the construction of a new type of Feedforward neural networks (greedy ridge function neural networks GRGFNNs) to approximate multidimensional functions $f \in C(\mathbb{R}^s)$ and modifies the greedy algorithm to train greedy ridge function neural networks. Also, a comparison was made between the three algorithms (modified greedy algorithm, Backpropagation algorithm and the result in [1]), where the method in [1] use the

Radon Transform and ridge function neural networks to approximate $f \in C(\mathbb{R}^s)$.

Greedy Ridge Function Neural Networks (GRGFNNs)

An artificial neural network is a mathematical model of the human brain. In many literatures, [6], [7], and [4], different types of neural network models had been studied. In this section we present a new type of Feedforward Neural Networks (FFNNs) which we called it greedy ridge function neural networks (GRGFNNs). Our neural network consists of a large number of computing units (processing elements PEs) arranged schematically in three layers as shown in figure (1), i.e. input layer, one hidden layer and output layer, with s inputs, n hidden PEs and ℓ outputs PEs.

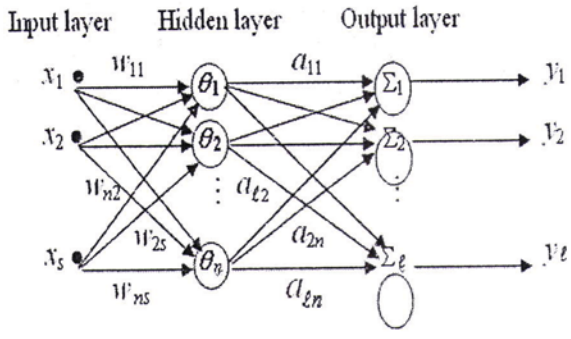

Figure (1): Greedy Ridge Function Neural Networks.

Each unit of the input layer can be connected to each unit (PE) of the hidden layer, this connection has been associated with a weight, which is a real number. The weight attached to the link from input unit j to unit i on the hidden layer is denoted by w_{ij} ($j = 1, 2, \dots, s$ and $i = 1, 2, \dots, n$). In a typical operation, each unit on the input layer will contain a real number. Let the jth input unit contain the real number x_j. Then unit i on the hidden layer will receive from unit j on the input layer the quantity $w_{ij}\, x_j$. The total input that unit i receives from all the input units is then $\sum_{j=1}^{s} w_{ij}\, x_j$, $i = 1, 2, \dots, n$. Unit i on the hidden layer now processes this input with a continuous ridge function θ (activation function).

Definition 2.1, [7]:

A ridge function is a multivariate function $h : \mathrm{R}^s \to \mathrm{R}$ of the form

$$h(x) = \theta(\phi(x)) = (\theta \circ \phi)(x),$$

$x \in \mathbb{R}^s$

where $\phi : \mathbb{R}^s \to \mathbb{R}$ and $\theta : \mathbb{R} \to \mathbb{R}$ are continuous functions.

Since every continuous linear functional on $\mathbb{R}^s$ has the form,

$\phi(x) = \xi \cdot x = \xi_1\, x_1 + \xi_2\, x_2 + \dots + \xi_s\, x_s$,

$\xi, x \in \mathbb{R}^s$. Then a ridge function on $\mathbb{R}^s$ has the form $h(x) = \theta(\xi \cdot x)$, [7].

Then the output is the real number $\theta_i\left(\sum_{j=1}^{s} w_{ij}\, x_j + b_i\right) = \theta(net_i)$. This output is then transmitted, with a weight a_{pi}, from the unit i in the hidden layer to the output unit p in the output layer. Each output unit implements a linear combination of these ridge functions. The output is then have the form

$$y_p = net_p = \sum_{i=1}^{n} a_{pi}\, \theta_i\left(\sum_{j=1}^{s} w_{ij}\, x_j + b_i\right)$$

$$, \quad p = 1, 2, \dots, \ell \tag{1}$$

The above formula can be generalized to neural network with multiple hidden layers.

In our network we take the case when $\ell = 1$, $s = 2$, $n = 9$, $n = 18$ and $x \in [-4, 4] \times [-4, 4]$. We choose ridge function and hyperbolic tangent function as the activation functions. Also, we compute the error by using the mean square error function, i.e.

$$E = \frac{1}{2}(d - y)^2,$$

where d is the desired output (exact) and y is the approximation output which came from our network.

Function Approximation

Function approximation can be described as follows: For a function f, known exactly or approximately, find an approximation that has a more simply computable form, where the error of the approximation within a given error tolerance. Often the function f is not known exactly. For example, if the function comes from a physical experiment, that is, only we have a table of function values. In our paper we will consider these properties of f which can be use to serve our goal of understanding how neural networks can be used to approximate an arbitrary function f. A classical result, over $[a,b]$, is the well-known Weierstrass theorem (Let $f \in C[a,b]$, let $\varepsilon > 0$. Then there exists a polynomial $p(x)$ for which $\| f - p \|_\infty \le \varepsilon$ for all $x \in [a,b]$), [8].

Legendre and Gauss used polynomials to approximate continuous functions over $\mathbb{R}$. Chebyshev developed the concept of best uniform approximation. *Series* expansions (i.e. Taylor series) have been utilized for many years to approximate and compute the value of a function in a neighborhood of the operating point. Trigonometric polynomials are also widely used as function approximators, but their computation is a bit more involved, [6].

To explain how the artificial neural networks (ANNs) can be used to approximate real-valued $f \in C(\mathbb{R}^s)$, we will consider a model called Greedy ridge function neural networks (GRGFNNs) with input layer, output layer and one hidden layer, given in section (1). Thus our neural network consists of a large number of computing units (PEs) arranged schematically in three layers, as in Figure (1). Then the total output of the neural network has the form

$$\sum_{i=1}^{n} a_i \, \theta_i \left(\sum_{j=1}^{s} w_{ij} \, x_j + b_i \right) = \tilde{f}(x_1, x_2, \ldots, x_s). \tag{2}$$

It is known, [6], that any continuous function of s variables can be approximated with arbitrary precision on compact set by a function given in equation (2). Thus, by suitable adjusting the parameters n, b_i, a_i and w_{ij}, we can reproduce approximately any desired output with GRGFNNs just described above.

Till now, the best achievable approximation accuracy (rate of convergence) for approximating s-variable function with d continuous derivatives is $O(n^{-d/s})$, [7]. Thus we note that for a given error approximation the number of parameters n, number of basis functions, exponentially increases with s (for a fixed measure of "complexity" d). It implies that the number of samples needed for accurate estimation of n parameters also grows exponentially with dimensionality s. This result constitutes the curse of dimensionality. It is more accurate to view the ratio s/d as the complexity index of the possible trade-off between the smoothness and dimensionality, which is the rate of convergence and the number of samples needed, for training, for accurate estimation that increases exponentially with the complexity index. Thus fast rate of convergence for high dimensional problems can be obtained, in principle, by imposing stronger smoothness constraints.

Mathematically, the neural network depending upon its architecture can be used to approximate any continuous function,. For example, if $s, n \ge 1$ are integers, the output of a neural network with one hidden layer comprising n processing elements PEs (neurons), each evaluating a nonlinear function θ, and receiving an input vector $\boldsymbol{x} \in \mathbb{R}^s$ can be expressed in the form $\sum_{i=1}^{n} a_i \, \theta_i \left(\boldsymbol{w}_i \cdot \boldsymbol{x} + b_i \right)$, where for $i = 1, 2, \ldots, n$, the *weights* $\boldsymbol{w}_i \in \mathbb{R}^s$, and the *thresholds* b_i and the coefficients a_i are real numbers. In the sequel, the class of all such output functions will be denoted by $\prod_{\theta,n,s}$, which often refers to the output function of the neural network.

The following questions arises, in a theoretical study, when we approximate a function $f \in C(\mathbb{R}^s)$ by using an artificial neural networks procedure, are the following:

1. *Density*, Given a continuous (real-valued) function f on a compact subset $K \subset \mathbb{R}^s$ and a positive number ε, is it possible to find some integer n, and a network $\aleph \in \prod_{\theta,n,s}$ such that

$$| f(x) - \aleph(x) | \le \varepsilon \tag{3}$$

   What are the necessary and sufficient conditions should be putted on θ for the property (3) to hold?
   If it does not hold, what the type of functions that can be approximated by this way?

2. *Complexity*, If we know a priori assumption about the target function f, formulated

mathematically by the statement $f \in W$, for some function space W, can one obtain necessary number of neurons, n, in the neural network of (2) in terms of ε? How does the choice of θ affect this bound?

3. *Construction*, How does one construct a neural network with, theoretically, minimal size that approximates any function from W within a prescribed accuracy?
4. *Limitations*, Is there any advantage to be gained by using a more complicated architecture, such as networks with multiple hidden layers; i.e., are there any limitations on the networks with one hidden layer?

The density problem is perhaps the most widely investigated problem. In the context of neural networks, the works of Cybenko [9] and Hornik et al. [10] are often cited. Hornik, et al. showed that feedforward neural networks (FFNNs) are universal approximators, the proof is based on an extension of the Weierstrass theorem (the Stone-Weierstrass theorem). The study of Cybenko [9], was concerned with the use of ridge functions and radial basis functions. In [11], Mhaskar and Micchelli have formulated necessary and sufficient conditions for the function θ (where θ is a ridge functions) so as to achieve density. Also, they have given similar conditions for the radial basis function networks. From the point of view of function approximation, the hidden units provide a set of activation functions that constitute an arbitrary "basis" for representing input patterns in the space spanned by the hidden units.
However another important characteristic, of approximating a function by the use of neural network, is the study of complexity which means how the error decreases with the number of layers, number of neurons, and the dimension of the input space. The importance of Feedforward neural networks (FFNNs) for function approximation was reinforced by the work of Barron, [12]. He showed that the asymptotic accuracy of the approximation with Feedforward neural networks (FFNNs) is approximately independent of the dimension of the input space s, i.e. the order of the error $O(1/n)$ depends only on the number of neurons. This is unlike approximation with polynomials, spline, trigonometric expansions and series expansions with n terms, where the error convergence rate is exponentially related to the number of dimensions of the input, i.e. the order of the error $O(1/n)^{2/S}$ where s is the dimension of the input space of the function f. This means that Feedforward neural networks (FFNNs) become much more efficient, for approximating $f \in C(\mathbb{R}^s)$ in high dimensional space, than using polynomials, spline, trigonometric expansions and series expansions.

Greedy Approximation

Greedy process is a technique which can be used to approximate large classes of functions. Cheney, [6], use such technique to produce a sequence of functions, f_n, which can be used to approximate the original function f, where he proved the following result.

$$\| f - f_n \| \leq C\, n^{-\frac{1}{2}}, \tag{4}$$

where n denotes the number of neurons (PEs), in the proposed neural networks and C is a constant depending on the function f to be approximated. However, Cheney [6], in his work does not apply the above result and its just a theoretical results. Our numerical result, for approximation of a function with two variables, conform the above Cheney result.
The pay off for this nice convergence behavior is the necessity to compute global minimum for high-dimensional nonlinear optimization problems (in particular for large n) in order to obtain the approximating functions f_n. In the context of neural networks, the so-called *backpropagation algorithm* is the most popular approach to solve the arising optimization problems (called training), mainly due to its simple realization. But since the backpropagation (BP) algorithm is a version of the gradient method, *steepest descent method*, so one cannot expect global convergence and this is due to lack of losing the density property. Thus Cheney result, [6], can not be used.
Moreover, the performance of such iterative algorithms is limited by the inherent ill-posedness of the training problem. Our numerical results in section (6) conform such conclusion and such, iterative, convergence is very slow and our conclusion coincide with the conclusion given in Burger, [13].
Greedy process is an iterative algorithm which can be used, also, to train neural networks and such training algorithm basically depends on the

function property without the need of using the derivatives of the function to be approximated, also it can be implemented efficiently. This iterative algorithm, greedy algorithm, is also called *projection pursuit* or *convex approximation techniques*. The main idea of such algorithm is to increase the number of nodes in the network, Figure (1), step by step by one neuron (by using suitable convex combination) and to optimize only over the parameters of the new node, which yields a sequence of low-dimensional optimization problems. The original motivation for such method is the possibility to maintain the convergence rate $n^{-\frac{1}{2}}$ with low computational effort.

To give a detail analysis to the above algorithm, we need the following definition and lemma given in [6].

Definition 4.1, [7]

Let G be a subset of an inner product space H with induced norm $\|\cdot\|$. Then the convex hull of G is the set

$$\mathrm{co}(G)=\left\{\sum_{i=1}^{n}\alpha_i g_i : n\in\mathbb{N}, g_i\in G, \alpha_i\geq 0, \sum_{i=1}^{n}\alpha_i=1\right\}$$
.

The closure of this set is denoted by $\overline{\mathrm{co}}(G)$.

Furthermore let the elements of the set G be bounded in the norm, by a constant b, which may be abbreviated as $G\subset \mathrm{B}(0,b)$.

Lemma 4.2, [7]

Let G be a subset of an inner product space H, and let f be an element of H. In order that $f\in\overline{\mathrm{co}}(G)$ it is necessary and sufficient that there exists a sequence $\{g_n\}$ in G and a sequence $\{f_n\}$, with $f_n\in\mathrm{co}\{g_1,\ldots,g_n\}$ for each n, such that $f_n\to f$.

For further analysis we define the constant γ as

$$\gamma=\inf_{v\in\mathrm{H}}\ \sup_{g\in G}\left(\|g-v\|^2-\|f-v\|^2\right). \qquad (5)$$

This value is in some sense a measure for the number of different elements of G that are needed to represent f. If the norm of f is close to the bound b and therefore f is close to the boundary of $\overline{\mathrm{co}}(G)$ then the value of γ will be very small. In this case f can be represented by few different elements of G, see Figure (2).

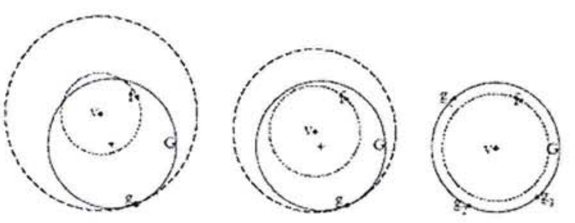

Figure (2): Interpretation of condition (5). The objective function is the difference of the radius of the two dashed circles. In this symmetric case the infimum is attained for *v* lying in the center of *G*.

Note that the value of γ can be bounded from above by $b^2-\|f\|^2$ since $0\in\mathrm{H}$. In our paper we combine the two results of Cheney, [6], and Burger, [5], so that the constant γ can be used to provide an estimate for the rate of convex approximation to the function f. The following lemma in [14] is useful to illustrate the above claim.

Lemma 4.3, [5]

Let $G\subset\mathrm{B}(0,b)$, $f\in\overline{\mathrm{co}}(G)$, $h\in\mathrm{co}(G)$ and let γ be defined as in equation (5). Then the estimate

$$\inf_{g\in G}\|f-\lambda h-(1-\lambda)g\|^2\leq\lambda^2\|f-h\|^2+(1-\lambda)^2\gamma$$

holds for $\lambda\in[0,1]$.

From the above lemma, the function f_n which is defined in the error bound (4) has the form $f_n=\lambda h-(1-\lambda)g$. Now we shall present the Greedy algorithm, as a result of above analysis.

1. The Greedy Algorithm:

Initialization:

Choose a constant M, such that $M>\gamma$ (as defined in (5)).

Choose a positive sequence ε_k, tending to zero that fulfills

$$\varepsilon_k\leq\frac{M-\gamma}{k^2} \qquad \text{for } k=1,2,\ldots$$

Choose a positive integer *max*.

```
Set $f_0 := 0$.
*Iteration:*
**For** $k := 1$ to *max* **do**
    Find an element $g_k \in G$ such that
    $\left\| f - \frac{k-1}{k} f_{k-1} - \frac{1}{k} g_k \right\|^2 \le \inf_{g \in G} \left\| f - \frac{k-1}{k} f_{k-1} - \frac{1}{k} g \right\|^2 + \varepsilon_k$
    is satisfied and define $f_k$ as
    $f_k = \frac{k-1}{k} f_{k-1} - \frac{1}{k} g_k$.
**End For**.
```

Note that in each step only one element of G is chosen, the other components of f_k are fixed. Nevertheless with the greedy algorithm, still, the rate of convergence is independent of the dimension. Such claim can be justified by using the following theorem which is given in [5].

Theorem 4.4, [5]

Let the conditions of Lemma (4.3) be satisfied. Then the approximating functions f_k generated by the above algorithm fulfill an error estimate $\| f - f_k \|^2 \le \frac{M}{k}$.

The ridge function feedforward neural networks, with one hidden layer of ridge functions f_n, can be represented as in section (3) by

$$f_n = \sum_{i=1}^{n} a_i \, \theta(w_i \, x + b_i),$$

where w_i , $i = 1, 2, \ldots, n$, is the connections (weights). Hence for the above neural networks we can write

$$f_n = \sum_{i=1}^{n} a_i \, \Theta(x, w_i). \tag{6}$$

In order to apply the above algorithm, using equation (6), we consider the Hilbert space H which is a Lebesgue space $L^2(\Omega)$, where Ω is a (not necessarily bounded) domain in $\mathbb{R}^s$, and G be the set

$$G_b = \{ a\,\Theta(x, w) : |a| \le b,\ w \in \mathrm{T} \} \subset L^2(\Omega),$$

where T is the compact set of parameters in $\mathbb{R}^s$. The set G_b can be interpreted as the set of all possible nodes of the neural networks. If the function Θ is scaled such that its L^2-norm is bounded above by 1, uniformly in w, i.e.,

$$\int_{\Omega} |\Theta(x, w)|^2 dx \le 1 \qquad \text{for all } w \in \mathrm{T} , \tag{7}$$

then G_b is bounded and $G_b \subset \mathrm{B}(0, b)$. For simplicity, assuming equation (7), one can use the bound $\tilde{b} = b \big/ \sup_{w \in \mathrm{T}} \| \Theta(x, w) \|_{L^2(\Omega)}$ for the factor a in G_b. Observe that $\tilde{b}$ can not be zero because the set T is compact. The value of γ, given in equation (5), is now has the value

$$\gamma = \inf_{v \in L^2} \sup_{|a| \le b,\, w \in \mathrm{T}} \left(\| a\,\Theta(x, w) - v \|^2 - \| f - v \|^2 \right) . \tag{8}$$

The convex hull of the set G_b is defined as

$$, \operatorname{co}(G_b) = \left\{ f \in L^2(\Omega) : f = \sum_{i=1}^{n} |a_i| \le b,\ w \in \mathrm{T},\ n \in \mathbb{N} \right\}$$

the sign of the parameters a_i is not important, because the original set G_b is symmetric. Further the sum needs not to be equal to b but can be smaller, because the zero function is an element of G_b. Note, that $\overline{\operatorname{co}}(G_b)$ contains all functions having a representation of the form $\sum_{i=1}^{n} a_i \, \Theta(x, w_i)$ and $\int_{\mathrm{T}} a\,\Theta(x, w)\, dw$.

Now we shall consider the Greedy algorithm for training greedy ridge function neural networks (GRGFNNs).

1.1 The Greedy Algorithm with Ridge Functions:

In this section we shall explain how the greedy algorithm, using ridge functions, can be used to produce a sequence f_k which approximates $f \in C(\mathbb{R}^s)$.

```
*Initialization*:
    Choose $\varepsilon > 0$, say $10^{-5}$.
    Choose a constant $M$, such that $M > \gamma$ (as defined in (8)), e.g. choose
    $M$ equal to $M = \frac{3}{2}\left(b^2 - \| f \|_2^2\right)$ and
    set $M_{opt} = M$, $k_{opt} = \infty$.
    Choose a positive integer *max*.
    Set $f_0 := 0$.
*Iteration*:
    **For** $k := 1$ to Min( $k_{opt}$ , *max*) **do**
        **If** $\| f - f_{k-1} \|_2^2 \le \frac{M}{(k+1)}$ **then**
            set $f_k := f_{k-1}$ and go to the next step of the iteration.
        **End If**
            Find parameters **w**, $b$ and $a$ such that
            $\left\| f - \frac{k-1}{k} f_{k-1} - \frac{1}{k} a\,\theta\left(w^{\mathrm{T}} x + b\right) \right\|_2^2 \le \frac{M_{opt}}{k}$,
            and define $f_k$ as
            $f_k := \frac{k-1}{k} f_{k-1} - \frac{1}{k} a\,\theta\left(w^{\mathrm{T}} x + b\right)$.
    **End for**
*Check*:
        **If** $\| f - f_k \|_2 < \varepsilon$ **then**
            go to end.
        **Else**
            Reduce $M$ and repeat the iteration.
        **End If**
    **End.**
```

Complexity

As we know, the Backpropagation training algorithm depends on the derivative of the activation function θ and it's easy to be computed if the activation function is logistic function or hyperbolic tangent function. Cheney, [6], in his work use the property that the function θ has the derivative θ', which can be putted in term of θ. But this is not always true that the derivative can be expressed in term of the original function. The weakness of the above analysis is that the norm defined in Hilbert space which has been used does not include the derivative of θ and thus we may loss the completeness, or density, property. Thus we consider the problem of approximating a function $\theta \in W^{m,p}(\Omega)$, where $W^{m,p}(\Omega)$ denote the Sobolev spaces and Ω is a (not necessarily bounded) domain in $\mathbb{R}^s$. For example, if θ' cannot be putted in term of θ then we need to add a stage to the neural network so that to handle the derivative as in Figure (3).

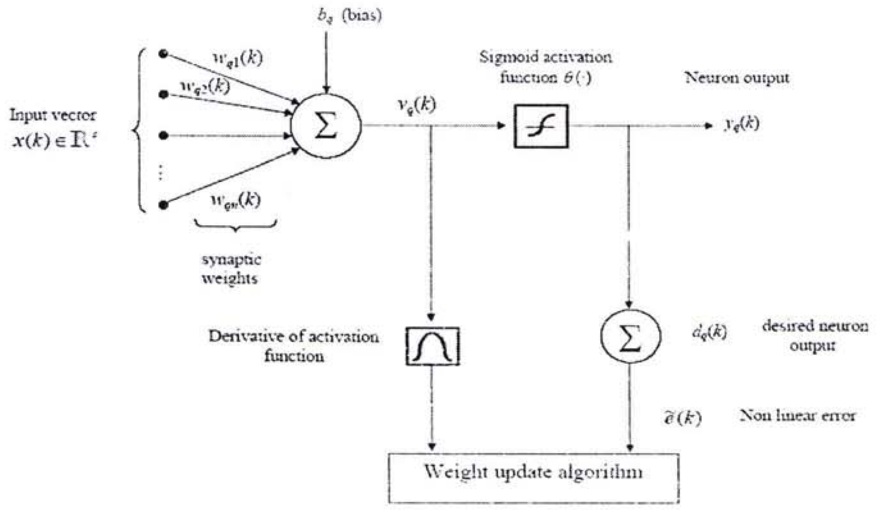

Figure (3): The Single q^{th} Neuron in MLP with Sigmoid Activation

Definition 5.1, [1]

Let m be a non negative integer, $p \in [1, \infty]$. The Sobolev space $W^{m,p}(\Omega)$ is the set of all functions $v \in L^1_{loc}(\Omega)$ such that for each multi-index α with $|\alpha| \le m$, the α^th weak derivative $D^\alpha v$ exists and $D^\alpha v \in L^p(\Omega)$. The norm in the space $W^{m,p}(\Omega)$ is defined as

$$\| v \|_{W^{m,p}(\Omega)} = \begin{cases} \left(\sum_{|\alpha| \le m} \| D^\alpha v \|^p_{L^p(\Omega)} \right)^{\frac{1}{p}} & , 1 \le p < \infty \\ \max_{|\alpha| \le m} \| D^\alpha v \|_{L^\infty(\Omega)} & , p = \infty \end{cases}$$

.

When $p = 2$, we write $H^m(\Omega) \equiv W^{m,2}(\Omega)$. Note that the Sobolev space $H^m(\Omega)$ is a Hilbert space with the inner product

$$< u, v >_m = \int_\Omega \sum_{|\alpha| \le m} D^\alpha u(x)\, D^\alpha v(x)\, dx \quad ,$$

$u, v \in H^m(\Omega)$.

Thus $H^m(\Omega)$ is a completion of the set of all real-valued functions $f \in C^\infty(\Omega)$.

Now, the problem of approximating f can be written in the form

$$\inf_{g \in X_n} \| f - g \|_X , \tag{9}$$

where X is referred as a Sobolev spaces and X_n denotes the set of all functions of the form (6), i.e.

$$X_n = \left\{ g = \sum_{i=1}^{n} a_i\, \Theta(x, w_i) : w_i \in T \subset \mathbb{R}^t, a_i \in \mathbb{R} \right\}$$

. (10)

Usually, the convergence rate of using g in equation (9) if it exists (note that X_n is not a finite dimensional subspace of X) is arbitrarily slow, since the approximation problem is asymptotically ill-posed, i.e. arbitrarily small errors in the observation can lead to arbitrarily large errors in the approximation as $n \to \infty$, our numerical results in section (6) confirm such observation, slow rate of convergence, and such observation coincide with that given in [3]. However, it was shown in [4] that the set of functions to which neural networks of the form in equation (6) converge is just the closure of the range of the integral operator

$$h \mapsto \int_T h(w)\, \Theta(x, w)\, dw .$$

To improve rates of convergence, or concord the ill-posed approximation to *f*, usually one should impose additional conditions on *f*. A natural condition seems to be that f is in the range of the above operator, i.e.

$$f(x) = \int_T h(w)\, \Theta(x, w)\, dw . \tag{11}$$

It was shown in [16] that under this condition the convergence rate

$$\inf_{g \in X_n} \| f - g \|_{L^p(\Omega)} = O(n^{-\frac{1}{2}}), \tag{12}$$

is obtained if Θ is a continuous function.

In [6], they improved the above error bound, equation (12), by imposing additional smoothness assumptions on the basis function Θ defined in $H^m(\Omega)$ as in the following theorem. Moreover, they gave error bounds in $W^{m,p}(\Omega)$ that depend on the dimension, *r*, of T and this is the weakness of such approach. Also, they gave a sufficient conditions on f so that condition in equation (11) hold.

Theorem 5.2, [6]

Let X_n be defined as in (10) with $T \subset \mathbb{R}^t$ bounded and Θ such that

$$\| \Theta(x, w) - \Theta(x, u) \|_{H^m(\Omega)} \le c \| w - u \|^\rho ,$$

$\rho \in (0, 1]$, $c > 0$.

Moreover, let $f \in H^m(\Omega)$ satisfy (11) with $h \in L^\infty(T)$. Then the convergence rate is

$$\inf_{g \in X_n} \| f - g \|_{H^m(\Omega)} = O(n^{-\frac{1}{2} - \frac{\rho}{t}}) .$$

Now, assume $\Theta \in W^{m,p}(\Omega, \mathrm{Y})$ with $\mathrm{Y} = H^k(\mathrm{T})$ or $\mathrm{Y} = \mathrm{C}^k(\mathrm{T})$, where $k > \frac{t}{2}$, and if we use the following norms

$$\|\Theta\|_{W^{m,p}(\Omega,Y)} = \begin{cases} \left(\sum_{|\alpha| \le m} \int_{\Omega} \left\| \frac{\partial^{|\alpha|}}{\partial x^{\alpha}} \Theta(x, w) \right\|_Y^p dx \right)^{\frac{1}{p}} & , 1 \le p < \infty \\ \max_{|\alpha| \le m} \operatorname{ess\,sup}_{x \in \Omega} \left\| \frac{\partial^{|\alpha|}}{\partial x^{\alpha}} \Theta(x, w) \right\|_Y & , p = \infty \end{cases}.$$

Then the following result hold.

Theorem 5.3, [6]

Let X_n be defined as in (10) with $\mathrm{T} \subset \mathbb{R}^t$ bounded and let $\Theta \in W^{m,p}(\Omega, \mathrm{Y})$ with $\mathrm{Y} = H^k(\mathrm{T})$, $k > \frac{t}{2}$, or $\mathrm{Y} = \mathrm{C}^k(\mathrm{T})$. Moreover, let $f \in W^{m,p}(\Omega)$ satisfy (11) with $h \in L^2(\mathrm{T})$ if $\mathrm{Y} = H^k(\mathrm{T})$ and $h \in L^1(\mathrm{T})$ if $\mathrm{Y} = \mathrm{C}^k(\mathrm{T})$. Then the convergence rate is

$$\inf_{g \in \mathrm{X}_n} \| f - g \|_{W^{m,p}(\Omega)} = O(n^{-\frac{k}{t}}). \qquad (13)$$

We think the above result can be generalized if the condition given in equation (11) has the form

$$f(\boldsymbol{x}) = \sum_{|\alpha| \le \tau} \int_{\mathrm{T}} h_\beta(w) \frac{\partial^{|\beta|}}{\partial \boldsymbol{x}^\beta} \Theta(\boldsymbol{x}, w)\, dw,$$

$\tau < k$.

Then the convergence rate is

$$\inf_{g \in \mathrm{X}_n} \| f - g \|_{W^{m,p}(\Omega)} = O(n^{-\frac{(k-\tau)}{t}}). \qquad (14)$$

The convergence rates, in equations (13) and (14), decrease with increasing the dimension t. The above conjecture requires further study but, from our numerical result in section (6), we feel its true and is the best possible result, error bound, when we use the space $W^{m,p}(\Omega)$.

Numerical Example

In this section we verify the above theoretical results by considering a numerical example. Let us consider the following two dimensional function $f : \mathbb{R}^2 \to \mathbb{R}$.

$$z = f(x_1, x_2) = 3(1-x_1)^2 \exp[-x_1^2 - (x_2+1)^2] - 10 \left(\frac{1}{5}x_1 - x_1^3 - x_2^5\right) \exp[-x_1^2 - x_2^2] - \frac{1}{3} \exp[-(x_1+1)^2 - x_1^2].$$

A three dimensional plot of the function f for $-4 \le x_1 \le 4$ and $-4 \le x_2 \le 4$ is shown in figure (4). A two dimensional problem is chosen so that to explain how the steps of the algorithm can be illustrated.

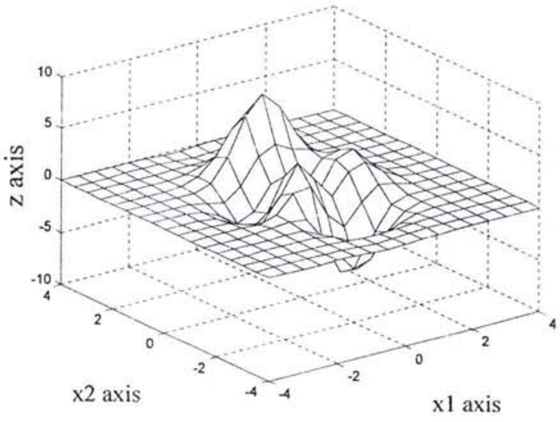

Figure (4): The Function $f(x_1, x_2)$.

The above algorithm (Greedy Algorithm with Ridge Functions) was, numerically, implemented by using MATLAB version (7.0). Figure (5) shows the approximation to the function $f(x_1, x_2)$ and figure (6) shows a comparison between the exact data of the function $z = f(x_1, x_2)$ and the approximation data with $k = 9$, i.e. number of neurons, $n = 9$, in the hidden layer.

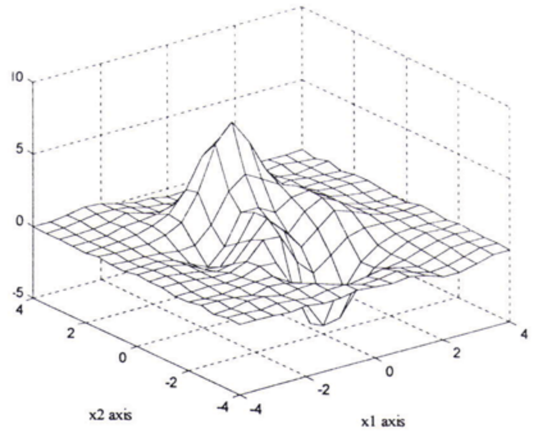

Figure (5): Approximation with $n=9$.

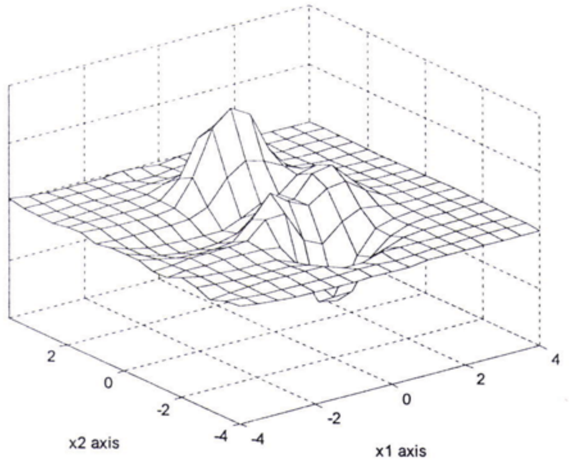

Figure (7): Approximation with $n=9$.

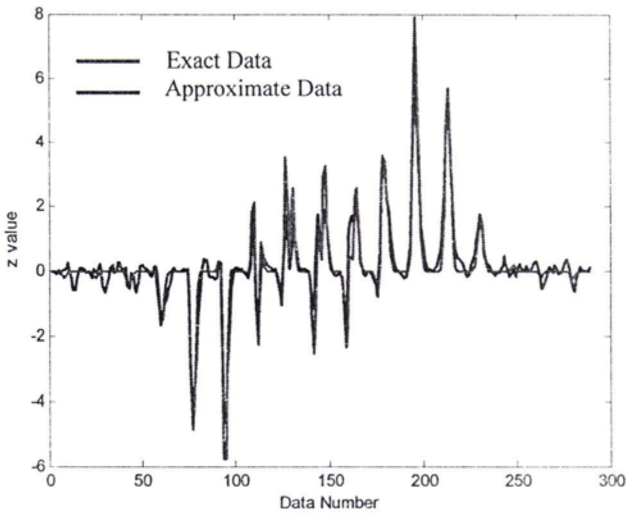

Figure (6): Comparison between exact data and approximate data with $n=9$.

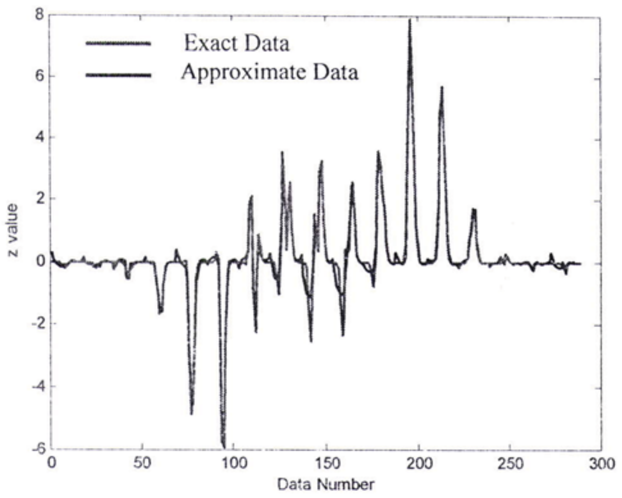

Figure (8): Comparison between exact data and

Also, the GRGFNNs use the Backpropagation algorithm with hyperbolic tangent function in the hidden layer and pureline function in the output layer. Figure (7) shows the approximation to the function $f(x_1,x_2)$ and figure (8) shows a comparison between the exact data of the function $z=f(x_1,x_2)$ and the approximation data with $k=9$, i.e. number of neurons, $n=9$, in the hidden layer.

Figure (9) shows the approximation to the function $f(x_1,x_2)$ by using the method in [14] and figure (10) shows a comparison between the exact data of the function $z=f(x_1,x_2)$ and the approximation data with $k=18$, i.e. number of neurons, $n=18$, in the hidden layer.

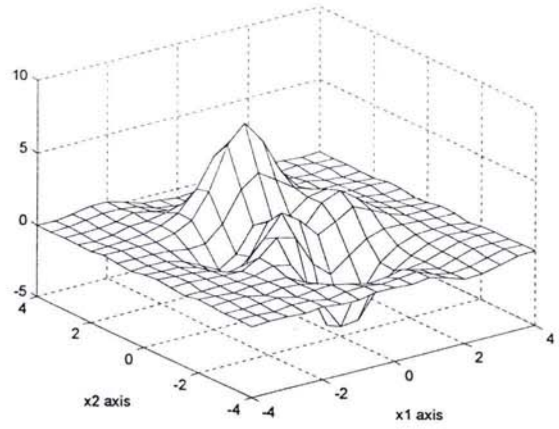

Figure (9): Approximation with $n = 18$.

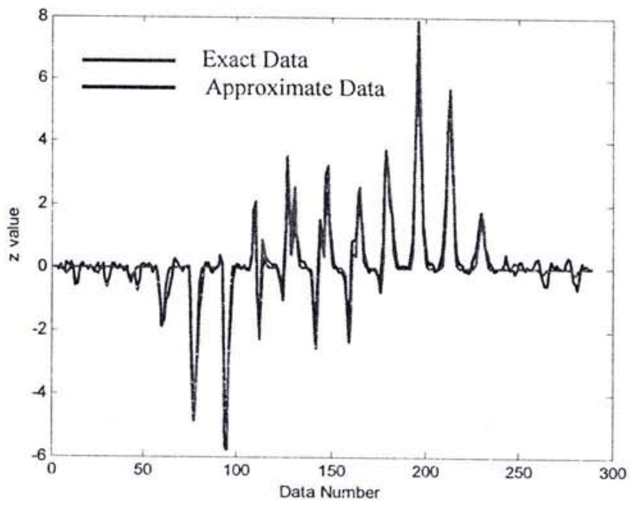

Figure (10) : Comparison between exact data and approximate data with $n = 18$.

Conclusions

Our numerical results shows that Backpropagation (BP) algorithm procedure more accurate numerical approximation to *f* than Greedy algorithm and the method in [1], but the Greedy algorithm is faster and use less flops than Backpropagation algorithm and the method in [1] overcome the problem of dimesionality which discussed in section (3). Also, if the derivative of f can be putted in term of f and the derivative of the sigmoidal function can be putted in term of the sigmoidal function then the Greedy algorithm, and its modifications, for training greedy ridge function neural networks (GRGFNNs) procedure almost the same accurate as we use Backpropagation algorithm. However, if we adjust the rate of convergence η in Backpropagation algorithm, then the BP algorithm converge faster.

References

1. Naoum, R. S. and Hussein, N. M., **2006,** "*Approximation of Multidimensional Functions by Radon Ridge Feedforward Neural Networks*" Journal of AL-Nahrain University_Science (JNUS), 9, 1, 100-107.
2. Burger, M. and Neubauer, A., **2001,** "*Error Bounds for Approximation with Neural Networks*" *Journal* of Approximation Theory, 112, 2, 235-250.
3. Ciesielski, K., Sacha, J. P. and Cios, K. J., **2000,** "*Synthesis of feedforward networks in supremum error bound*" IEEE Transactions on neural networks, 11, 6, 1213-1227.
4. Ellacot, S. W., **1994,** "*Aspects of the numerical analysis of neural networks*" Acta Numerica, 145-202.
5. Pinkus, A., **1997,** "*Approximation by ridge functions*" surface fitting and multiresolution methods,A. Le Méhauté, C. Rabut, and L.L. Schumaker(eds.),Vanderbilt University Press, Nashville, TN, 1-14.
6. Cheney, E. W. and Light, W. A., **2000,** "*A Course in Approximation Theory*" The Brooks/Cole Publishing Company, USA.
7. Cherkassky, V. and Mulier, F., **1998,** "*Learning From Data: Concepts, Theory, and Methods*" John Wiley and Sons, Inc., USA.
8. Atkinson, K. and Han, W., **2001,** "*Theoretical Numerical Analysis: A Functional Analysis Framework*" Springer-Verlag, New York, Inc.
9. Cybenko, G., **1989,** "*Approximation by Superposition of Sigmoidal Functions*" Mathematics of Control, Signal and Systems, 2, 303-314.
10. Hornik, K., Stinchcombe, M. and White, H., **1989,** "*Multilayer Feedforward Networks are Universal Approximators*" Neural Networks, 2, 359-366.
11. Mhaskar, H. N. and Micchelli, C. A., **1992,** "*Approximation By Superposition of A Sigmoidal Function and Radial Basis Functions*", Advances in Applied Mathematics, 13, 350-373.
12. Barron, A. R., **1993,** "*Universal Approximation Bounds For Superposition of A Sigmoidal Function*" IEEE Transaction on Information Theory, 39, 3, 930-945,.
13. Burger, M. and Engl, H. W., **2000,** "*Training Neural Networks with Noisy Data as an ill-*