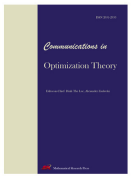
Commun. Optim. Theory 2022 (2022) 3 https://doi.org/10.23952/cot.2022.3

MATHRES Communications in Optimization Theory
Available online at http://cot.mathres.org



# RATIONAL ACTIVATION FUNCTIONS IN NEURAL NETWORK WITH UNIFORM NORM BASED LOSS FUNCTION AND ITS APPLICATION IN CLASSIFICATION

VINESHA PEIRIS

Department of Mathematics, Swinburne University of Technology, Hawthorn, Victoria, Australia

Dedication to the memory of Professor Hoang Tuy

**Abstract.** In this paper, we demonstrate an application of generalised rational uniform (Chebyshev) approximation to neural networks. In particular, our activation functions are rational functions of degree $(1,1)$ and the loss function is based on the uniform norm. In this setting, when the coefficients of the rational activation function are fixed, overall optimisation problem of the neural network can be formulated as a generalised rational approximation problem with additional linear constraints. In this case, the weights and the bias of the network are the decision variables. To optimise the decision variables, we suggest using two prominent methods: bisection method and differential correction algorithm. We illustrate the efficiency of this application by performing numerical experiments on classification problems with two classes and report the classification accuracy obtained by the network using the bisection method, differential correction algorithm along with the standard MATLAB toolbox which uses least square loss function. We show that the choice of the uniform norm based loss function with rational activation function leads to a better classification accuracy when the training dataset is either very small or if the classes are imbalanced.

**Keywords.** Deep learning; Generalised rational uniform approximation; Quasiconvex optimisation; Rational activation functions.

## 1. INTRODUCTION

Deep learning is a branch of machine learning which is inspired by artificial neural networks. Recently, the study of deep learning has been an interesting area for many researchers due to its excellent performance in a variety of practical applications [5, 18, 21, 34, 37]. In general, a neural network contains layers (input, hidden, output) of interconnected nodes and these connections are known as weights. Hidden layers refine the weights coming from the previous layer in composition with a nonlinear activation function. From the point of view of optimisation, the main goal of a network is to optimise the weights by minimising the loss function. Hence, the

E-mail address: vpeiris@swin.edu.au.
Received October 12, 2021; Accepted February 15, 2022.

choice of the activation function, the format of the loss function and the optimisation algorithm which minimises the loss function have a significant impact on the efficiency of the network.

Rational activation functions have drawn much attention lately due to their flexibility and smoothness [3, 6, 14, 28]. Telgarsky's theoretical work [39] on error bounds of the approximation to the ReLU (Rectifies Linear Unit) network by rational functions and vice versa influenced several recent rational function adaptations to neural networks as an activation function [3, 28]. In the most recent work [3], authors study rational neural networks, whose activation functions are low degree rational functions with trainable coefficients. In all prior work on rational activation functions, the loss function is based on the least squares formulation.

The least squares loss function is often employed for optimising the parameters (weights) of a neural network against a training set. This is due to the fact that basic optimisation techniques such as gradient descent may be effectively utilised to optimise the loss function. In some specific cases, the loss function can be successfully adapted to a uniform (Chebyshev) approximation based model where the maximum error is minimised. In particular, if the size of the data available for training is small but the data (themselves) are reliable or if the data is greatly imbalanced, the uniform approximation based model performs much better than least square based models [33].

In this paper, we consider a simple neural network model with no hidden layers (single–layer feedforward neural network, that is, the inputs are fed directly to the outputs) whose activation function is a classical rational function (ratio of two polynomials, the basis functions are monomials) of degree (1,1) and the loss function is in the form of uniform approximation. The coefficients of the rational activation function are obtained by the best rational approximation to the ReLU function. In these circumstances, the overall optimisation problem of the network is equivalent to the generalised rational uniform approximation problem whose basis functions are the inputs of the neural network. Weights and the bias term are the decision variables to be optimised.

The generalised rational uniform approximations in the sense of Cheney and Loeb [9] are well-studied and there are many efficient techniques to construct these approximations [1, 7, 22, 29, 31]. The differential correction method [1, 7, 8] is an iterative scheme which has guaranteed convergence properties to find the optimal generalised rational approximation. Bisection method is another simple, robust and modern optimisation based approach which utilises the fact that the corresponding optimisation problems are quasiconvex [4]. The class of quasiconvex functions includes all the functions whose sublevel set is convex. Authors of [31] use this method to construct univariate generalised rational uniform approximations and later, it was extended in several directions [26, 32]. In this paper, we prove that the relevant optimisation problems appearing in a simple neural network with rational functions of degree $(1,1)$ as activation functions and uniform based loss function are quasiconvex and therefore, they can be solved using the bisection method.

We implement the differential correction algorithm and bisection method in MATLAB adapted to a simple neural network with no hidden layers. We perform our numerical experiments on classification problems with two classes and report the classification accuracy obtained by the network using the bisection method, differential correction algorithm along with the standard MATLAB toolbox which uses the least squares loss function. We highlight the benefits of the

bisection method and the differential correction algorithm in connection with neural networks by comparing the computational time and the classification accuracy.

This paper is organised as follows. The motivation for the current study is discussed in Section 2. Section 3 provides the notations and the model formulation of a simple neural network. In Section 4, we define generalised rational uniform approximations and includes definitions of quasiconvex functions. In Section 5, we introduce rational functions of degree $(1,1)$ as activation functions and incorporate them into a simple neural network whose loss function is based on uniform approximation. Section 6 discusses training of the network using the bisection method and the differential correction algorithm. Section 7 provides numerical experiments. Finally, Section 8 explains conclusions and future research directions.

## 2. Motivation

The main motivation for the current study comes from the work done by Telgarsky in [39], where he raised a few open problems related to rational functions and neural networks. One of the open questions was 'Can rational functions be used to design algorithms for training neural networks?' and it was first addressed in [28] by utilising rational activation functions in a multilayer neural network framework where the loss function is based on the least squares formula and the network parameters are optimised using the gradient descent approach.

Rational functions and rational approximations techniques are very efficient and powerful tools. In this study, we address the above open problem by demonstrating how we can utilise rational functions and rational approximation techniques to design algorithms for training neural networks with specific features rather than using the traditional framework.

The choice of the activation function (nonlinear) impacts the performance of a neural network. ReLU activation function has been a popular choice lately and in [3], the authors demonstrate the efficiency of neural networks with rational activation functions over the networks with ReLU activation functions. Rational functions as activation function are considered to be an attractive alternative to ReLU activation functions since rational functions are flexible and smooth.

Moreover, the motivation for the current framework comes from the Kolmogorov–Arnold representation theorem [42] which says that any continuous function of several variables could be expressed by sums and composition of functions of one variable. But the construction of the representation of these functions still remains as an open problem. Therefore, deep learning researchers replaced this representation by an approximation which is convenient to deal with when there is a need to compute gradients. Traditionally, this approximation consists of compositions between a simple function (also called activation function) and an affine transformation (linear combinations between the weights and the inputs of the network) since this representation makes computing the gradients easier [43]. In this study, we do this slightly differently by changing the format of the loss functions, so that it does not involve gradients. From the optimisation point of view, this approach is interesting and reasonable since the resulting function is a rational function which has better approximation powers.

## 3. Formulating a General Neural Network Model

Consider a neural network which consists of an input layer with $n$ nodes and an output layer with $m$ nodes. Let $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ represent the input of the neural network and $y \in \mathbb{R}^m$

represent the output computed by the neural network. The weights between the input nodes and the output nodes are denoted by $w_{ij}$ where $i = 1:m$, $j = 1:n$. The forward propagation that passes the input to the next layer are computed by taking the linear combination between inputs and the corresponding weights

$$z_i(x) = \sum_{j=1}^{n} w_{ij}x_j + b_j, \quad i = 1:m,$$

where $b_j, j = 1:n$ is known as the bias term. If the activation function defined on the output layer is $\sigma$, then the output of the network is a single composition between $\sigma$ and $z_i$

$$y_i(x) = \sigma(z_i(x)) = \sigma\left(\sum_{j=1}^{n} w_{ij}x_j + b_j\right), \quad i = 1:m.$$

This can also be formulated in matrix notation as follows:

$$y(x) = \sigma(\mathbf{W}x + b),$$

where

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \dots & w_{mn} \end{bmatrix}.$$

The training of a neural network is done by minimising the loss function (optimising the weights of the network). The loss function elaborates the error of the network approximation, where the training set input values are used to calculate the outputs. .

Let $\{(x^1,y^1),(x^2,y^2),\dots,(x^N,y^N)\}$ be a training set where $x^k, k = 1:N$ are the inputs and $y^k, k = 1:N$ are the desired outputs. Weights and the bias term are the decision variables of the corresponding optimisation problems of the network and one needs to optimise the decision variables ($\mathbf{W}$, $b$) such that the error between the neural network output $y(x^k) = \sigma(\mathbf{W}x^k + b)$, and the desired output of the training set $y^k$ has to be minimised in some sense. A common loss function used in neural networks is the least squares loss function,

$$L(\mathbf{W},b) = \sum_{k=1}^{N} (y^k - \sigma(\mathbf{W}x^k + b))^2.$$

Gradient descent algorithms can be accompanied to optimise the decision variables of the least squares loss function when training the network. There are many different types of loss functions used in neural networks, and [18] goes over a few of them in detail.

In this paper, we use uniform approximation based formula for the loss function,

$$L(\mathbf{W},b) = \max_{\substack{k\in\{1:N\} \\ i\in\{1:m\}}} |y_i^k - \sigma_i(\mathbf{W}x^k + b)|.$$

This is also known as Chebyshev norm and max norm. This loss function is much more efficient than the least squares based approach when the training set is reliable, but limited in size [33].

In this research, we only consider a simple neural network with no hidden layers, and we assume that the output layer consist of a single node. In the case of several output nodes,

the network can be constructed as a parallel structure. Our uniform approximation based loss function is

$$L(\mathbf{W},b) = \max_{k\in 1:N} |y^k - \sigma(\mathbf{W}x^k + b)|,$$

and $\mathbf{W}$ is now reduced to a row vector. One can also write this formula explicitly as follows:

$$L(w_j,b) = \max_{k\in\{1:N\}} \left| y^k - \sigma\left(\sum_{j=1}^{n} w_j x_j^k + b\right)\right|,$$

where $w_j,\ j = 1:n$ are the elements of the row vector $\mathbf{W}$ and $x_j^k,\ j = 1:n$ are the components of the input vector in the $k$-th sample of the training set.

## 4. Generalised Rational Uniform Approximations

In this section, we define the optimisation problems appearing in generalised rational uniform approximations and discuss some important definitions of quasiconvex functions. First, let us define the generalised rational functions in terms of Cheney and Loeb [9]. In particular, these generalised rational functions are the ratios of linear forms,

$$\bar{R}_{n,m}(x) = P(x)/Q(x) = \frac{\sum_{i=0}^{n} p_i g_i(x)}{\sum_{j=0}^{n} q_j h_j(x)}, \tag{4.1}$$

where $g_i(x),\ i = 1:n$ and $h_j(x),\ j = 1:m$ are basis functions and $x \in [c,d]$, a closed interval on the real line. Functions from (4.1) reduces to the classical rational functions when the basis functions are just monomials, i.e., ratio of two polynomials. Functions from $\bar{R}_{n,m}(x)$ are also known as rational functions of degree $(n,m)$ or rational functions of type $(n,m)$.

In the case of generalised rational uniform approximation, the optimisation problem is

$$\min_{\mathbf{A},\mathbf{B}} \quad \max_{x\in[c,d]} \left| f(x) - \frac{\mathbf{P}^T\mathbf{G}(x)}{\mathbf{Q}^T\mathbf{H}(x)}\right| \tag{4.2}$$

subject to

$$\mathbf{Q}^T\mathbf{H}(x) > 0, \quad x \in [c,d],$$

where

- $f(x)$ is the function to approximate,
- $\mathbf{P} = (p_0, p_1, \ldots, p_n)^T \in \mathbb{R}^{n+1}, \mathbf{Q} = (q_0, q_1, \ldots, q_m)^T \in \mathbb{R}^{m+1}$ are the decision variables,
- $\mathbf{G}(x) = (g_0(x), \ldots, g_n(x))^T, \mathbf{H}(x) = (h_0(x), \ldots, h_m(x))^T$, where $g_j(x),\ j = 1:n$ and $h_i(x),\ i = 1:m$ are known basis functions.

Therefore, the approximations are ratios of linear combinations of basis functions. When all the basis functions are monomials, the problem is reduced to the best rational uniform approximation.

In [31], the authors proved that the objective function of (4.2) forms a quasiconvex function. The definition of quasiconvex functions and associated preliminary findings are included in the next section.

**Definition 4.1.** Function $f(t)$ is quasiconvex if and only if its sublevel set

$$S_\alpha = \{x | f(x) \le \alpha\}$$

is convex for any $\alpha \in \mathbb{R}$. The set $S_\alpha$ is also called an $\alpha$-sublevel set.

It can be shown that this Definition 4.1 is equivalent to the following definition [16].

**Definition 4.2.** A function $f : D \to \mathbb{R}$ defined on a convex subset $D$ of a real vector space is called quasiconvex if and only if, for any pair $x$ and $y$ from $D$ and $\lambda \in [0,1]$, one has

$$f(\lambda x + (1-\lambda)y) \leq \max\{f(x), f(y)\}.$$

**Definition 4.3.** Function $f$ is quasiconcave if and only if $-f$ is quasiconvex.

This means that every superlevel set $\bar{S}_\alpha = \{x | f(x) \geq \alpha\}$ is convex.

**Definition 4.4.** Functions that are quasiconvex and quasiconcave at the same time are called quasiaffine (sometimes called quasilinear).

Since the problem of generalised rational uniform approximation forms a quasiconvex function, it can be treated using a number of computational methods developed for quasiconvex optimisation [4, 12, 13, 15, 23, 35, 36]. One such method (called bisection method for quasiconvex functions [4]) will be described in Section 6.1.

The bisection method that we use in this paper is simple and robust, but it is not as efficient as some other methods, in particular, the differential correction method. Differential correction method does not require any special properties of the objective function to process. We discuss this further in Section 6.2.

## 5. Network with Rational Activation Functions and Uniform Norm Based Loss Function

We start this section by introducing rational activation functions. Activation functions play a crucial role in the theory of neural networks. The main goal of defining an activation function on the output of a particular layer is to introduce nonlinearity to the network. The choice of the activation function depends on the type of application. Sigmoid, logistic or hyperbolic tangent are popular smooth activation functions, but their derivatives are zero for large inputs which cause problems when the gradient descent algorithm is employed to optimise the loss function [2]. An attractive alternative is polynomial activation function [11, 19, 30]. At the same time, they are not efficient when one needs to approximate nonsmooth (or non-Lipschitz) functions: high order polynomials are required, leading to severe oscillations and numerical instability. Recently, Rectified Linear Unit (ReLU) and its variants (Leaky ReLU (LReLU), Exponential Linear Unit, Parametric ReLU (PReLU), etc.) has been widely used by deep learning community due to its reduced likelihood of vanishing gradient.

### 5.1. Rational activation function.

Let $R_{n,m}$ be the set of all rational functions of degree $n$ and $m$ where $n$ is the degree of the numerator and $m$ is the degree of the denominator ($n$ and $m$ are nonnegative integers) and $x \in [c,d]$ is a closed segment on the real line.

$$R_{n,m}(x) = \frac{P(x)}{Q(x)} = \frac{\sum_{i=0}^{n} p_i x^i}{\sum_{j=0}^{n} q_j x^j} = \frac{p_0 + p_1 x + p_2 x^2 + \ldots + p_n x^n}{q_0 + q_1 x + q_2 x^2 + \ldots + q_n x^m}.$$

Functions from $R_{n,m}$ are also known as classical rational functions (ratio of two polynomials) where the denominator is strictly positive.

Telgarsky's theoretical work [39] inspired the use of rational functions in deep learning as an activation function. His work highlights the tight connections between rational functions

and neural networks with ReLU activation functions. In [28], rational activation functions were introduced for the first time as Padé Activation Units (PAU). In the most recent work, low degree rational functions have been employed as rational activation functions [3]. Moreover, their numerical experiments suggest that networks with rational activation functions are an attractive alternation to networks with ReLU actiavtion function.

In both [3, 28], $p_i, i = 0:n$ and $q_j, j = 0:m$ are learnable parameters of the network, meaning they are not fixed, and they are optimised with the rest of the decision variables (weights and bias) of the optimisation problems of the network where the loss function is in the form of the least squares. However, the optimisation process of [3] is initialised by the coefficients of the best rational approximation to ReLU (Rectified Linear Unit) function while [28] uses the LReLU (Leaky Rectified Linear Unit) function. The idea behind this is to initialise the rational networks near a network with ReLU or LReLU activation functions [3] since rational functions effectively approximates neural networks with ReLU activations [39]. The definitions of ReLU and LReLU functions are given below in Definition 5.1 and Definition 5.2 respectively. [3, 28] also use slightly different normalisation conditions which are imposed to guarantee that the optimal approximation is unique. In particular, authors of [28] fixes the coefficient of the lowest degree monomial of the denominator at 1 and authors of [3] uses $max|q_j| = 1, j = 0:m$.

**Definition 5.1.** ReLU function is defined as follows:

$$f(x) = \max\{0, x\}.$$

**Definition 5.2.** LReLU function is defined as follows:

$$f(x) = \max\{x, 0.01x\}.$$

5.2. **Network with rational function of degree (1,1) as an activation function.** Our neural network model is different from the above discussed ones from few different perspectives.

(1) Our loss function is based on uniform approximation.

Because of the nature of the uniform norm based loss function, counting the contribution of outliers in the dataset is possible, but least squares formulation discounts them by averaging. Therefore, our loss function is designed to produce better classification accuracy if the training data is reliable while being either limited in size or imbalanced [33].

This type of limited training datasets are available if the labelling of the dataset is done manually which can be both costly and time consuming in general. Sometimes, if the data are produced by a very expensive procedure, then the available number of data is limited in size [38]. Moreover, in many robotics applications, it is not enough to have accurate performance on average; one requires the maximum variation to be within a specific range [25]. There are many research devoted to classification accuracy with limited data or imbalanced class data [24, 27], however, most studies are based on the popular least squares loss function.

(2) We utilise rational functions of degree $(1,1)$ as activation functions and the coefficients of the rational activation are fixed by the best uniform rational approximation to the ReLU function between $[-1,1]$.

In particular, our activation function is

$$R(x) = \frac{p_0 + p_1 x}{q_0 + q_1 x}, \tag{5.1}$$

where $q_0 + q_1 x > 0$. This function is smooth, simple and easy to work within the uniform approximation based loss function. Most importantly, the objective function of the corresponding optimisation problem forms a quasiconvex function when the coefficients of the activation function are fixed.

We formulate our loss function as follows:

$$L(w_j, b) = \max_{k\in\{1:N\}} \left| y^k - \frac{p_0 + p_1(\sum_{j=1}^n w_j x_j^k + b)}{q_0 + q_1(\sum_{j=1}^n w_j x_j^k + b)} \right|, \quad j = 1:n, \tag{5.2}$$

where $q_0 + q_1(\sum_{j=1}^n w_j x_j^k + b) > 0$ for all $k = 1:N$.

**Theorem 5.3.** *Function*

$$L(w_j, b) = \max_{k\in\{1:N\}} \left| y^k - \frac{p_0 + p_1(\sum_{j=1}^n w_j x_j^k + b)}{q_0 + q_1(\sum_{j=1}^n w_j x_j^k + b)} \right|, \quad j = 1:n$$

*is quasiconvex.*

*Proof.* Let

$$R(w_j, b) = \frac{p_0 + p_1(\sum_{j=1}^n w_j x_j^k + b)}{q_0 + q_1(\sum_{j=1}^n w_j x_j^k + b)}, \quad k = 1:N. \tag{5.3}$$

When $p_0, p_1$ and $q_0, q_1$ are fixed, for any known input value $x^k$, the numerator and the denominator of $R$ consist of linear combinations between known basis functions and the parameters $w_j, b, j = 1:n$. Hence, $R$ is a ratio of two linear functions (ratio of linear forms). It was shown in [4] that the ratios of linear forms are quasilinear. Therefore, $R(w_j, b)$ is quasilinear. Since $|z| = max\{z, -z\}$, one can see that

$$L(w_j, b) = \max_{k\in\{1:N\}} \max_{j\in\{1:n\}} \{y^k - R(w_j, b), -y^k + R(w_j, b)\}$$

is the maximum of quasilinear functions. Quasilinear functions are quasiconvex. Maximum of quasiconvex functions is quasiconvex and hence, $L(w_j, b), j = 1:n$ is quasiconvex. □

When $p_0, p_1$ and $q_0, q_1$ are fixed, for any pair of training input and desired output $(x^i, y^i)$, the loss function (5.2) is equivalent to the objective function of the generalised rational uniform approximation problem. Various techniques can be used to optimise the decision variables of such approximation problems [1, 7, 31].

(3) We use two different algorithms to optimise the parameters of the loss function with two different normalisation conditions. Namely, the bisection method and the differential correction algorithm.

The decision variables of the optimisation problems appearing in our simple neural network are the weights $w_j, j = 1:n$ and the bias term $b$. We select two prominent methods to find the optimal weights of the network. In subroutines of both methods, the underlying optimisation problems can be solved using linear programming techniques which are easy to implement in most programming environments. Bisection method uses the quasiconvex characteristic of the loss function (5.2).

By multiplying the denominator and the numerator of (5.1) by a constant number, one can obtain infinitely many solutions. To avoid this ambiguity, we impose normalisation procedures which are aiming at ensuring that the solution is unique. In our numerical experiments, we

fix one of coefficients in the denominator, namely, $q_0 = 1$ for the bisection method. We found in our experiments that this normalisation condition is efficient for the Bisection approach. Differential correction algorithm obtains the optimal solution by finding the descent direction at each step of the process and the usual normalisation condition for this method in practice is $max|q_j| = 1,\ j = 0,1$ [1].

**Remark 5.4.** If one needs to use rational activation functions with a higher degree (degree $\geq 2$ in the numerator or in the denominator), then the quasiconvex property is immediately lost and the application of bisection method is not possible. Moreover, if the coefficients of (5.1) are considered to be learnable parameters of the network, rather than fixing them, then the corresponding optimisation problems are more complex and may need polynomial optimisation techniques to solve them. We leave these research directions open for future studies.

## 6. Training The Model

In this section, our goal is to develop the implementation of the bisection method and the differential correction method for our setting. The optimisation problem that we need to solve is as follows:

$$\min_{\substack{w_j,b \\ j\in\{1:n\}}} \quad \max_{k\in\{1:N\}} \left| y^k - \frac{p_0 + p_1(\sum_{j=1}^n w_j x_j^k + b)}{q_0 + q_1(\sum_{j=1}^n w_j x_j^k + b)} \right| \tag{6.1}$$

subject to

$$q_0 + q_1 \left( \sum_{j=1}^n w_j x_j^k + b \right) > 0, \quad k = 1:N, \tag{6.2}$$

where $w_j,\ j = 1:n$ are the weights, $b$ is the bias term and $(x^k, y^k),\ k = 1:N$ is the training set.

6.1. **Bisection method.** The bisection method is a simple, but efficient approach developed for minimising quasiconvex functions [4]. The convergence of this method is linear. Theorem 5.3 guarantees the quasiconvexity of the objective function (6.1) and therefore, we use the bisection algorithm to find the decision variables of the problem (6.1)–(6.2). As the normalisation condition, we use $q_0 = 1$.

Problem (6.1)–(6.2) can be also formulated (equivalently) as follows:

$$\min \quad z$$

subject to

$$y^k - \frac{p_0 + p_1(\sum_{j=1}^n w_j x_j^k + b)}{q_0 + q_1(\sum_{j=1}^n w_j x_j^k + b)} \leq z, \quad k = 1:N, \tag{6.3}$$

$$\frac{p_0 + p_1(\sum_{j=1}^n w_j x_j^k + b)}{q_0 + q_1(\sum_{j=1}^n w_j x_j^k + b)} - y^k \leq z, \quad k = 1:N, \tag{6.4}$$

$$q_0 + q_1 \left( \sum_{j=1}^n w_j x_j^k + b \right) > 0, \quad k = 1:N. \tag{6.5}$$

Without loss of generality, the constraint (6.5) can be replaced by

$$q_0 + q_1 \left( \sum_{j=1}^{n} w_j x_j^k + b \right) \geq \delta, \quad k = 1:N,$$

where $\delta$ is an arbitrary small positive number. The bisection method includes the following steps.

Step 1: To initialise the bisection method, one needs to define the following parameters:

- The lower bound, $l$.
  Since the objective function is nonnegative, one can choose $l = 0$ as the lower bound for the optimal solution.
- The upper bound, $u$.
  One can substitute any value of the decision variables $w_j, j = 1:n$ and $b$ in the objective function, for instance, $w_j = 0, j = 1:n$ and $b = 0$, then

$$u = \max_{k \in \{1:N\}} \left| y^k - R(0,0) \right|.$$

  The definition of $R(w_j, b)$ can be found in the Equation (5.3). $R(0,0) = p_0$ due to the normalisation condition and therefore, the upper bound is

$$u = \max_{k \in \{1:N\}} \left| y^k - p_0 \right|.$$

- The parameter for the stopping criterion, $\varepsilon_1$.
  This parameter ($\varepsilon_1$) for the bisection method is the difference between the upper and lower bounds of the bisection interval.

Step 2: Set $z = \frac{u+l}{2}$ and check if the set of constraints (6.3)–(6.5) has a feasible solution.

Step 3: If (6.3)–(6.5) has a feasible point, update the upper bound $u = z$, otherwise update the lower bound $l = z$. If $u - l \geq \varepsilon_1$, go to Step 2. This procedure terminates when $u - l < \varepsilon_1$.

In general, checking if a set is nonempty (Step 2) may be difficult [45, 46, 47, 48]. However, in this case, this problem is reduced to solving a linear programming problem. When $z$ is fixed, one needs to solve the following problem:

$$\min \quad u \tag{6.6}$$

subject to

$$y^k - \frac{(p_0 + p_1(\sum_{j=1}^{n} w_j x_j^k + b))}{(q_0 + q_1(\sum_{j=1}^{n} w_j x_j^k + b))} \leq z + u, \quad k = 1:N, \tag{6.7}$$

$$\frac{(p_0 + p_1(\sum_{j=1}^{n} w_j x_j^k + b))}{(q_0 + q_1(\sum_{j=1}^{n} w_j x_j^k + b))} - y^k \leq z + u, \quad k = 1:N, \tag{6.8}$$

$$q_0 + q_1 \left( \sum_{j=1}^{n} w_j x_j^k + b \right) \geq \delta, \quad k = 1:N. \tag{6.9}$$

If the optimal solution $u \leq 0$, then the set (6.3)–(6.5) has a feasible point, otherwise the set is empty. This feasibility problem (6.6)–(6.9) is a linear programming problem with respect to its decision variables $w_j, j = 1:n$ and $b$ and hence, it can be solved by using standard linear programming techniques.

6.2. **Differential correction method.** There are a few different versions of the differential correction algorithm. The original differential correction method was first developed by Cheney and Loeb [7]. Later on, slightly modified versions of the same method were published in [8, 10, 20]. In 1986, it was discovered that the generalised Dinkelbach's method applied to rational approximation problems [41] is similar to the differential correction method.

In general, differential correction method has guaranteed convergence properties; it was proved in [1] that the original differential correction method is converging quadratically while the modified versions have linear convergence. Thus, in this section, we wish to present the original version of the differential correction method adapted to our network setting.

This algorithm is also an iterative process which includes the following steps.

Step 1: To initialise the differential correction method, we need the following:

- The parameter for the stopping criterion, $\varepsilon_2$.
  This parameter ($\varepsilon_2$) for the differential correction method is the gap between the error terms of two consecutive iterations.
- Initialisation of the rational function.
  One can substitute any value of the decision variables, $w_j, j = 1:n$ and $b$, since this method guarantees the convergence from any initial point as long as the denominator is positive. Therefore, we suggest using the simplest possible initialisation which is $w_j = 0, j = 1:n$ and $b = 0$. Then, the initial ratio will be $R^0 = R(0,0) = \frac{p_0}{q_0}$. The definition of $R(w_j, b)$ can be found in Equation (5.3). By using this initial ratio, the maximal absolute deviation, $\Delta^0$ can be found.

Step 2: $r$–th step of the iterative process.

- At the $r$–th step of this iterative scheme, the decision variables $w_j^{r-1}, j = 1:n$ and $b^{r-1}$ are always available (known values) from the previous step. By using these known values, one can compute the ratio, $R^{r-1} = \frac{P(w_j^{r-1}, b^{r-1})}{Q(w_j^{r-1}, b^{r-1})}$ and hence, the maximum absolute error, $\Delta^{r-1}$.
- The ratio, $R^r = \frac{P(w_j^r, b^r)}{Q(w_j^r, b^r)}$ of the current iteration can be found by minimising the auxiliary expression

$$\max_{k=1:N} \left\{ \frac{\left| y^k Q(w_j^r, b^r) - P(w_j^r, b^r) \right| - \Delta^{r-1} Q(w_j^r, b^r)}{Q(w_j^{r-1}, b^{r-1})} \right\} \tag{6.10}$$

  subject to the normalisation condition of $\max_{j=1:n}\{|w_j|, |b|\} = 1$. Here, $\Delta^{r-1}$ is the maximum absolute error, $Q(w_j^{r-1}, b^{r-1})$ is the denominator and they are both known values from the previous step. Once the ratio, $R^r = \frac{P(w_j^r, b^r)}{Q(w_j^r, b^r)}$ is found, the maximum absolute error at the $r$–th step, $\Delta^r$ can be found.

Step 3: The Step 2 of this procedure needs to be repeated until $|\Delta^{r-1} - \Delta^r| < \varepsilon_2$.

Without loss of generality, the normalisation condition can be substituted by

$$-1 \le w_j^r \le 1, \quad -1 \le b^r \le 1, \quad \text{for all} \quad j = 1:n,$$

see [1, 7]. The minimisation of the expression (6.10) can be formulated as an equivalent linear programing problem as follows:

$$\min \quad \bar{z}$$

subject to

$$y^k Q(w_j^r, b^r) - P(w_j^r, b^r) - \Delta^{r-1} Q(w_j^r, b^r) \leq \bar{z} Q(w_j^{r-1}, b^{r-1}), \quad k = 1:N,$$
$$P(w_j^r, b^r) - y^k Q(w_j^r, b^r) - \Delta^{r-1} Q(w_j^r, b^r) \leq \bar{z} Q(w_j^{r-1}, b^{r-1}), \quad k = 1:N,$$
$$-1 \leq w_j^r, b^r \leq 1, \quad j = 1:n.$$

This auxiliary function in (6.10) automatically maintains the constraint

$$q_0 + q_1 \left( \sum_{j=1}^{n} w_j x_j^k + b \right) > 0, \quad k = 1:N,$$

and hence it needs not be incorporated into the linear programming problem formulation [1]. Since the constraints are linear with respect to its decision variables, one can use linear programming techniques to find the solution to this problem.

## 7. Numerical Experiments

Our experiments are done in two stages. In the first stage, we find the coefficients of the rational activation function as the coefficients of the best rational approximation of degree $(1,1)$ to the ReLU function. In the second stage, we train the simple neural network by using the bisection method and the differential correction method and calculate the corresponding classification accuracy.

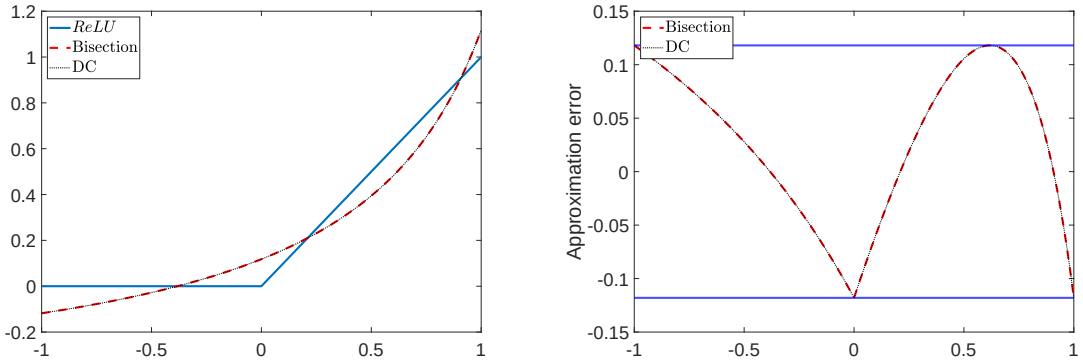
7.1. **Finding the coefficient of the rational activation function.** The coefficients of our rational activation function of degree $(1,1)$ need to be fixed before we start the optimisation process of the neural network. In this section, we present the best rational approximations to ReLU (Rectified Linear Unit) and LReLU (Leaky Rectified Linear Unit) functions computed via the bisection method and the differential correction algorithm. The parameter for the stopping criterion of both bisection and differential correction algorithms is set at $\varepsilon_1 = \varepsilon_2 = 10^{-5}$ to ensure the termination of these iterative processes. The computed coefficients are then used for the experiments in the next section.

In [3], authors use the coefficients of the best rational approximation to ReLU function to initialise the network while in [28], authors use the coefficients of the approximation to the LReLU function. Therefore, in this section, we demonstrate the results of the approximations for both ReLU and LReLU functions. Note that in this study, we use a two-stage procedure rather than using the coefficients as an initialisation procedure of the neural network.

Table 1 includes the coefficients of the rational approximation of degree $(1,1)$ for the ReLU function, computed by the bisection method and the differential correction algorithm. We report the results up to 7 decimal places to demonstrate the difference in the sets of coefficients derived by these two rational uniform approximation approaches. The maximum absolute error of the approximation and the coefficients are comparable for both techniques up to the 5th decimal place in most cases. To determine which approach computed the best approximation with the lowest maximum deviation, one needs to examine the 6th decimal place and therefore, we use a lengthier format to report the results. It is not surprising to see very similar approximations computed by the bisection method and the differential correction algorithm since the main

| Coefficients | Bisection method | Diff correction algorithm |
|---|---|---|
| $p_0$ | 0.1180331 | 0.1180329 |
| $p_1$ | 0.3090156 | 0.3090145 |
| $q_0$ | 1 | 1 |
| $q_1$ | -0.6180350 | -0.6180367 |
| Max abs error | 0.1180336 | 0.1180329 |

TABLE 1. The coefficients of the best rational approximation to ReLU function.




(a) Rational approximations.

(b) Error curves.

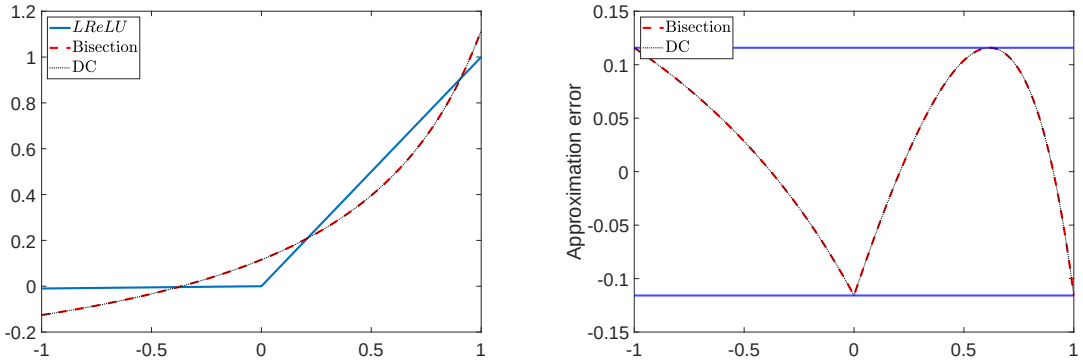
FIGURE 1. Rational $(1,1)$ approximations computed by bisection method and differential correction algorithm and its error curves for the ReLU function.

| Coefficients | Bisection method | Diff correction algorithm |
|---|---|---|
| $p_0$ | 0.1158096 | 0.1158089 |
| $p_1$ | 0.3184661 | 0.3184621 |
| $q_0$ | 1 | 1 |
| $q_1$ | -0.6107951 | -0.6108016 |
| Max abs error | 0.1158114 | 0.1158089 |

TABLE 2. The coefficients of the best rational approximation to LReLU function.

objective of both methods is the same. Figure 1 shows the approximations computed by both methods side by side with the error curves. One can see that the approximations and the error curves are identical for both methods.

Table 2 includes the coefficients of the rational approximation of degree $(1,1)$ for the LReLU function, computed by the bisection method and the differential correction algorithm. The computed coefficients are comparable up to at least the 3rd decimal place. Figure 2 shows the approximations computed by both methods side by side with the error curves. Since the slope of the LReLU function is very close to zero, we provide a zoomed in section of Figure 2a in Figure 3 to clearly highlight the difference.




(a) Rational approximations.

(b) Error curves.

FIGURE 2. Rational $(1,1)$ approximations computed by bisection method and differential correction algorithm and its error curves for LReLU function.

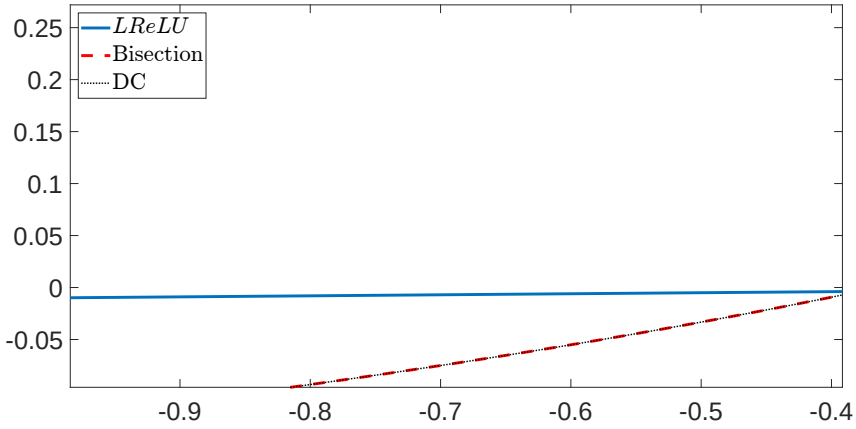



FIGURE 3. A small section of the Figure 2a zoomed in.

There are few things that one could notice. The difference between the sets of coefficients computed by the bisection methods and the differential correction algorithm is very small and it is at least within the precision $10^{-3}$, but for most cases it is within the precision $10^{-5}$. Even though we use a different normalisation condition for the differential correction algorithm, the coefficient of the smallest degree monomial turned out to be 1. Clearly, the approximations are smooth and the shape of the approximations are not very different for both ReLU and LReLU functions. However, the sets of coefficients are slightly different as expected. Both error curves comprise of 4 maximal and minimal alternating points which guarantee that the current approximations are optimal.

Since the ReLU coefficients lead us to better classification accuracy, we only report the results of the experiments conducted with ReLU coefficients with precision $10^{-5}$. Therefore, the activation for the experiments in the rest of this section is

$$R(z) = \frac{0.11803 + 0.30902z}{1 - 0.61804z}.$$

In our simple neural network, $z$ will be the affine transformation between the weights and the inputs of the training set, that is, $\sum_{j=1}^{m} w_j x_j^k + b$ where $k = 1:N$. Hence, our loss function is

$$\max_{k\in\{1:N\}} \left| y^k - \frac{0.11803 + 0.30902(\sum_{j=1}^{n} w_j x_j^k + b)}{1 - 0.61804(\sum_{j=1}^{n} w_j x_j^k + b)} \right|,$$

and the corresponding optimisation problem is

$$\min_{\substack{w_j,b \\ j\in 1:n}} \quad \max_{k\in\{1:N\}} \left| y^k - \frac{0.11803 + 0.30902(\sum_{j=1}^{n} w_j x_j^k + b)}{1 - 0.61804(\sum_{j=1}^{n} w_j x_j^k + b)} \right|$$

subject to

$$1 - 0.61804 \left( \sum_{j=1}^{n} w_j x_j^k + b \right) > 0, \quad k = 1:N.$$

**Remark 7.1.** Note that since $x^k$ are known values from the training set, we still have a ratio of two linear functions with respect to its decision variables, $w_j,\ j = 1:n$ and $b$. However, unlike the usual rational approximation, here we have the same set of decision variables in the numerator and also in the denominator ($w_j,\ j = 1:n$ and $b$ in the numerator and the denominator are the same). Therefore, imposing a restriction on the collection of decision variables in the denominator implies that the numerator's decision variables are also restricted the same way.

7.2. **Experiments with datasets.** The goal of this section is to compare the classification accuracy obtained by our simple neural network when the bisection method and the differential correction algorithm are employed to optimise the parameters of the network along with the classification accuracy obtained by the standard MATLAB toolbox. We intentionally use datasets with limited training data or imbalanced class data since the uniform approximation based loss function works better on these types of training sets (see [33]).

We use the default activation function (softmax on the output) for MATLAB deep learning toolbox whose loss function is based on the least squares formula, and we use rational activation function of degree $(1,1)$ for the networks with uniform approximation based loss functions. We report the classification accuracy, confusion matrix, computational time to train the network and computational time to calculate the classification accuracy for the test set.

A confusion matrix, also known as an error matrix, is a table layout that allows one to see how well a classification algorithm performs. The main diagonal of the confusion matrix corresponds to the correctly classified points whereas the off-diagonal corresponds to misclassified points. Each row of the matrix corresponds to the actual class and each column of the matrix corresponds to the predicted class. Hence, the top left element of the matrix includes the correctly classified points from Class 1, top right element includes the misclassified points from Class 1.

MATLAB deep learning toolbox does not allow us to compute training and testing times separately and therefore, we only present the combined time for training and testing. On the other hand, for uniform approximation based loss functions, we compute training and testing times separately and it allows us to compare the computational time of the bisection method and the differential correction method. We fix the parameter for the stopping criterion for both rational uniform approximation algorithms at $\varepsilon_1 = \varepsilon_2 = 10^{-5}$ for all the experiments in this section.

(a) Signal 0. (b) Signal 1.

FIGURE 4. Signals from the dataset TwoLeadECG data.

7.2.1. *Experiments with TwoLeadECG dataset.* We begin our numerical experiments with one of the datasets from the famous PhysioNet [17] database, and we only focus on MIT-BIH Long-Term ECG Database. This MIT-BIH Long-Term ECG data collection contains 7 long-term ECG recordings, with manually reviewed beat annotations. We use the dataset, TwoLeadECG which is the seventh set of recordings from the MIT-BIH Long-Term ECG data base. It contains two classes of signal; Class 1 includes signals of type signal 0 (Figure 4a) and Class 2 consist of signals of type signal 1 (Figure 4b). Each signal contains 82 features. The main goal is to distinguish the signals between these two classes.

The training set contains 23 recordings; 12 recordings from Class 1, 11 recordings from Class 2 and the test set contains 1139 recordings; 569 recordings from Class 1, 570 recordings from Class 2.

**Experiments with the original dataset.** We start with the original dataset. Results of the experiments are shown in Table 3.

TABLE 3. Classification results for the original dataset.

| Method | MATLAB Toolbox | | Uniform approximation based loss | |
|---|---|---|---|---|
| | | | Bisection | Diff Correction |
| Test classification accuracy | 70.2% | | 87.62% | 84.02% |
| Confusion matrix | 510 | 59 | 535 34 | 517 52 |
| | 280 | 290 | 107 463 | 130 440 |
| Training time (in seconds) | 4.89754 | | 1.89228 | 1.70678 |
| Testing time (in seconds) | | | 1.66403 | 1.79433 |

One can see from Table 3 that the bisection method is more accurate than the other two techniques. Classification accuracy obtained by the MATLAB toolbox is the lowest. Low classification accuracy of the differential correction method may be due to the fact that the differential correction method stops far from the optimal solution, since in general, it comes to

a stop when the improvement in the descent direction is insufficient. This can be verified by reducing the parameter for the stopping criterion ($\varepsilon_2$) of the algorithm. When $\varepsilon_2 = 10^{-10}$, the differential correction method computes the same accuracy (84.02%) as before. This implies that the current solution is optimal subjected to the given restrictions.

TABLE 4. Loss function values of the original dataset.

| Loss function value | Bisection | Diff correction |
|---|---|---|
| Training set | $1.345765 \times 10^{-5}$ | 0.0528 |

Table 4 contains the objective function values computed by the uniform approximation based loss functions. For the training set, the bisection method achieves a smaller loss function (objective function) value than the differential correction method. However, both objective function values are close to zero. The difference between the loss function values is (most likely) due to the choice of normalisation condition.

In the case of approximating a given continuous function by a rational function (degree $(n,m)$), the differential correction method only forces the decision variables in the denominator (the coefficients of the denominator) to be within the interval $[-1,1]$ as its normalisation condition. However, in the case of a simple neural network where we use the differential correction method to optimise the decision variables, restricting the decision variables in the denominator implies a natural restriction on the decision variables in the numerator (Remark 7.1). Therefore, in this model of a simple neural network, we can expect the bisection method and the differential correction method to work differently and produce two different outputs unlike in the case of general rational approximation. Moreover, the efficiency of the differential correction method can be significantly improved if it is used in conjunction with the 'equalisation of maxima' [44] method which makes the maximum deviations equal. We leave this for future research.

Note that there are no optimisation techniques employed when computing the classification accuracy for the test set. Only the training set requires the use of optimisation techniques to compute the parameters of the network (that is, weights and the bias). For each of the three techniques, we also calculated the classification accuracy for the training set. While MATLAB toolbox achieves only 82.6% percent accuracy for the training set, uniform approximation based loss functions perfectly separated the training set into the two classes. Since the number of training samples is very low in comparison to the number of features in a single training sample, both uniform approximation based models are likely to be overfitting in the training set. MATLAB toolbox performs poorly compared to the uniform approximation based methods during the training and testing.

**Experiments with reduced dataset.** Now we reduce the training set even further by randomly selecting 10 points from the original training set and compare the classification accuracy against the test set. The classification results can be found in Table 5.

Clearly, from Table 5, the classification accuracy for all three methods decreased compared to the original dataset. However, the bisection method outperforms both MATLAB toolbox and the differential correction method while requiring the same amount of time as the differential correction method to train and test the model. All three methods perfectly separated the training set into the two classes even though the MATLAB toolbox computes lower classification accuracy for the test set.

TABLE 5. Classification results for the randomly selected points from the training dataset.

| Method | MATLAB Toolbox | | Uniform approximation based loss | | | |
|---|---|---|---|---|---|---|
| | | | Bisection | | Diff Correction | |
| Test classification accuracy | 61.3% | | 79.10% | | 72.34% | |
| Confusion matrix | 177 | 392 | 504 | 65 | 543 | 26 |
| | 49 | 521 | 173 | 397 | 288 | 281 |
| Training time (in seconds) | 3.26634 | | 1.86108 | | 1.58480 | |
| Testing time (in seconds) | | | 1.81060 | | 1.98166 | |

**Experiments with imbalanced distribution between classes.** Our next step is to consider imbalanced distribution of the data points between classes. This means that one of the two classes is underrepresented in the corresponding training dataset. We consider two cases: in the first case, Class 1 is underrepresented and in the case two, Class 2 is underrepresented.

**Case 1:** Training set consist of all the points from Class 2 with 2 randomly selected points from Class 1. Size of the training set is now $13 (= 11 + 2)$. The classification accuracy is given in Table 6.

TABLE 6. Classification results when Class 1 is underrepresented in the training set.

| Method | MATLAB Toolbox | | Uniform approximation based loss | | | |
|---|---|---|---|---|---|---|
| | | | Bisection | | Diff Correction | |
| Test classification accuracy | 50.1% | | 66.54% | | 61.19% | |
| Confusion matrix | 10 | 559 | 300 | 269 | 135 | 434 |
| | 9 | 561 | 112 | 458 | 8 | 562 |
| Training time (in seconds) | 3.30496 | | 1.93120 | | 1.69338 | |
| Testing time (in seconds) | | | 2.04918 | | 1.77658 | |

From Table 6, we see that the classification accuracy for all three method decreased. However, the accuracy calculated using the bisection method, continues to be superior. At the same time, it takes much longer to test the network than the differential correction algorithm.

Classification accuracy for the training set computed by the MATLAB toolbox is 84.6% and it only correctly classified the points from Class 2 (Class 2 now has a higher number of points in the training set) and all the points coming from Class 1 were classified incorrectly. At the same time, uniform approximation based loss functions perfectly separated the training set into the two classes.

**Case 2:** Training set consist of all the points from Class 1 with 2 randomly selected points from Class 2. Size of the training set is now $14(=12+2)$. The classification accuracy is given in Table 7.

TABLE 7. Classification results when Class 2 is underrepresented in the training set.

| Method | MATLAB Toolbox | | Uniform approximation based loss | | | |
|---|---|---|---|---|---|---|
| | | | Bisection | | Diff Correction | |
| Test classification accuracy | 54.7% | | 63.92% | | 59.35% | |
| Confusion matrix | 63 | 506 | 540 | 29 | 566 | 3 |
| | 10 | 560 | 382 | 188 | 460 | 110 |
| Training time (in seconds) | 3.358267 | | 1.95700 | | 1.234968 | |
| Testing time (in seconds) | | | 2.04481 | | 1.638340 | |

The accuracy calculated using the bisection method, continues to be superior, but the differential correction method is not very far behind the highest classification accuracy. Bisection method takes more time to train and test the classification accuracy than the differential correction method.

7.2.2. *Experiments with SonyAIBORobotSurface1 dataset.* We now consider a different dataset from [40]. The SONY AIBO Robot is a small, dog-shaped robot equipped with multiple sensors. In the experimental setting, the robot walked on two different surfaces: carpet and cement. Class 1 comprise of the data when the robot walked on the carpet and Class 2 consist of the data of the robot when it walked on the cement floor. The main goal is to distinguish the type of floor that the robot walked on.

The training set contains 20 recordings; 6 recordings from Class 1, 14 recordings from Class 2 and the test set contains 601 recordings; 343 recordings from Class 1, 258 recordings from Class 2. Each recording contains 70 features.

**Experiments with the original dataset.** We start with the original dataset. Results of the experiments are shown in Table 8.

One can clearly see that the differential correction method outperform the bisection method in this case and takes even less time for training and testing. MATLAB toolbox computes the lowest classification accuracy in the shortest computational time. Low test classification accuracy of the MATLAB toolbox is not surprising since the training accuracy was only 90% while the uniform approximation based loss functions perfectly separated the training set.

This experiment also validates the differential correction algorithm's superior convergence properties, which have been reported in the literature. It is also worth noticing that the difference between the classification accuracy computed by the bisection method and the differential correction method is negligible. When the parameter for the stopping criterion for the bisection algorithm is reduced to $\varepsilon_1 = 10^{-10}$, it computes a slightly higher accuracy (77.20%) which is still lower than the differential correction method.

The objective function values (loss function values) of the training set computed by the bisection method and the differential correction method are reported in Table 9. Clearly, the bisection

TABLE 8. Classification results for the original dataset.

| Method | MATLAB Toolbox | Uniform approximation based loss | |
|---|---|---|---|
| | | Bisection | Diff Correction |
| Test classification accuracy | 58.9% | 77.04% | 78.04% |
| Confusion matrix | 101 242<br>5 253 | 246 97<br>41 217 | 247 96<br>36 222 |
| Training time (in seconds) | 1.44505 | 1.99841 | 1.76474 |
| Testing time (in seconds) | | 1.84173 | 1.77953 |

TABLE 9. Loss functions value for the original dataset.

| Loss function value | Bisection | Diff correction |
|---|---|---|
| Training set | $1.34582 \times 10^{-5}$ | 0.07295 |

method computes a lower loss function value, but the test set classification accuracy is lower than the differential correction method. This implies that a better objective function value does not mean the test set classification is better and overfitting of the training set may be the reason for this observation.

**Experiments with reduced dataset.** Now we make our training set smaller by randomly selecting 10 points from the original training set and compare the classification accuracy against the test set. The classification results can be found in Table 10.

TABLE 10. Classification results for the randomly selected points from the training dataset.

| Method | MATLAB Toolbox | Uniform approximation based loss | |
|---|---|---|---|
| | | Bisection | Diff Correction |
| Test classification accuracy | 55.2% | 67.05% | 68.22% |
| Confusion matrix | 86 257<br>12 246 | 246 97<br>101 157 | 193 150<br>41 217 |
| Training time (in seconds) | 2.65381 | 1.74184 | 1.60299 |
| Testing time (in seconds) | | 1.65028 | 1.65776 |

All three techniques have lower classification accuracy than the original dataset. The differential correction method again achieved the highest classification accuracy. At the same time, accuracy obtained by the bisection method is only slightly smaller than the highest accuracy. Both uniform approximation based procedures take about the same time to train and test the corresponding models, but the MATLAB toolbox is much faster.

**Experiments with imbalanced distribution between classes.** Now we consider imbalanced distribution of the data points between classes.

**Case 1:** Training set consist of all the points from Class 2 with 2 randomly selected points from Class 1. Size of the training set is now $16(=14+2)$. The classification accuracy is given in Table 11.

TABLE 11. Classification results when Class 1 is underrepresented in the training set.

| Method | MATLAB Toolbox | Uniform approximation based loss | |
|---|---|---|---|
| | | Bisection | Diff Correction |
| Test classification accuracy | 54.1% | 73.54% | 74.21% |
| Confusion matrix | 68 \| 275<br>1 \| 257 | 220 \| 123<br>36 \| 222 | 210 \| 133<br>22 \| 236 |
| Training time (in seconds) | 1.30874 | 1.77939 | 1.64759 |
| Testing time (in seconds) | | 1.64104 | 1.83456 |

Clearly, the classification accuracy computed by the MATLAB toolbox decreased even further, but the accuracy for the uniform based loss functions increased compared to the reduced dataset accuracy. The differential correction method, continues to be superior, but it takes longer to test the model than the bisection algorithm. Despite the fact that all three approaches achieved 100% training classification accuracy, the MATLAB toolbox earned the lowest classification accuracy.

**Case 2:** Training set consist of all the points from Class 1 with 2 randomly selected points from Class 2. Size of the training set is now $8(=6+2)$. The classification accuracy is given in Table 12.

TABLE 12. Classification results when Class 2 is underrepresented in the training set.

| Method | MATLAB Toolbox | Uniform approximation based loss | |
|---|---|---|---|
| | | Bisection | Diff Correction |
| Test classification accuracy | 44.3% | 59.90% | 61.40% |
| Confusion matrix | 208 \| 135<br>200 \| 58 | 222 \| 121<br>120 \| 138 | 147 \| 196<br>36 \| 222 |
| Training time (in seconds) | 4.52671 | 1.91397 | 1.79120 |
| Testing time (in seconds) | | 1.76283 | 1.96599 |

The accuracy obtained by applying the differential correction approach remains superior. The uniform approximation approaches take about the same amount of time to train and test the models, whereas the MATLAB toolbox requires more computational time than the other

two methods. Moreover, all three approaches perfectly separated the training set into the two classes.

In general, uniform approximation based methods perform consistently better than the MATLAB toolbox in classification problems. In particular, when the activation function is a rational function of degree $(1,1)$. The bisection approach outperforms the differential correction method in the first set of experiments, and vice versa in the second set.

7.2.3. *Approximating the outputs by a rational function directly.* We also computed the classification accuracy (for the testing set) by approximating the outputs of the training set directly by a rational function using the bisection method. In this case, the corresponding optimisation problem is as follows:

$$\min_{\substack{a_n,b_n \\ n\in\{0:n\}}} \quad \max_{k\in\{1:N\}} \left| y^k - \frac{a_0+\sum_{j=1}^{n} a_n x_n^k}{b_0+\sum_{j=1}^{n} b_n x_n^k} \right| \tag{7.1}$$

subject to

$$b_0 + \sum_{j=1}^{n} b_n x_n^k > 0, \quad k = 1:N.$$

$b_0 = 1$ can be imposed as the normalisation condition. In this case, the set of decision variables consists of different collections of decision variables coming from the numerator and the denominator. Therefore, the number of decision variables in the set is twice as large as the number of decision variables in the neural network that we considered earlier. This leads to overfitting of the training set which has a significant influence on the test set classification accuracy in this situation. Rational approximation in model in Equation (7.1) only computed approximately the same (in some cases) or lower classification accuracy results than we reported earlier. The results are reported in Table 13.

TABLE 13. Test set classification accuracy computed by the bisection method when the inputs were approximated directly by a rational function.

| Training set | ECG signal dataset | AIBORobot dataset |
|---|---|---|
| Original set | 49.96% | 67.72% |
| Reduced set | 53.03% | 42.43% |
| Class 1 underrepresented | 54.61% | 71.88% |
| Class 2 underrepresented | 49.96% | 59.07% |

## 8. CONCLUSIONS

In this paper, we used classical rational function of degree $(1,1)$ as the activation function in a simple neural network whose loss function is based on uniform approximation. In this setting, when the coefficients of the rational activation function are fixed, the corresponding optimisation problem appearing in the network is equivalent to the generalised rational uniform approximation problem. To find the weights and the bias of the network, we used two methods: the bisection methods and the differential correction algorithm. Our numerical experiments performed on classification problems confirm that the bisection method and the differential correction algorithm return superior accuracy results compared to the MATLAB toolbox when

the training set is either small or imbalanced between the classes. Moreover, the differential correction method deals with overfitting better than the bisection method.

In the future, we are planning to continue our work in the direction of extending the use of classical rational functions of degree $(1,1)$ as activation function in a neural network with one or more hidden layers. Because the quasiconvexity property is preserved when layers are sandwiched on top of each other, this generalisation is achievable. We would also like to study the optimisation problems appearing in neural network when the degree of the rational activation function is higher than $(1,1)$. Another interesting research direction is to investigate the efficiency of the differential correction method when used in conjunction with the so called "equalisation of maxima" [44] method in the case of a simple neural network.

**Acknowledgements**
This research was supported by the Australian Research Council (ARC), Solving hard Chebyshev approximation problems through nonsmooth analysis (Discovery Project DP180100602). The author would like to thank Dr. Nadezda Sukhorukova for many fascinating discussions about this study, as well as the anonymous reviewers for their helpful recommendations on improving this manuscript.

## REFERENCES

[1] I. Barrodale, et al., The differential correction algorithm for rational $l_\infty$-approximation, SIAM J. Math. Anal. 9 (1972), 493-504.
[2] Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult, IEEE Trans. Neural Netw. 5 (1994) 157-166.
[3] N. Boullé, Y. Nakatsukasa, A. Townsend, Rational neural networks, Conference on Neural Information Processing Systems, 2020.
[4] S. Boyd, L. Vandenberghe, Convex Optimization, Cambridge University Press, New York, 2010.
[5] C. Cao, et al., Deep learning and its applications in biomedicine, Genomics, Proteomics Bioinfo. 16 (2018) 17-32.
[6] Z. Chen, et al., Rational neural networks for approximating graph convolution operator on jump discontinuities, In: 2018 IEEE International Conference on Data Mining (ICDM), pp. 59-68, IEEE, 2018.
[7] E.W. Cheney, H.L. Loeb, Two new algorithms for rational approximation, Numer. Math. 3 (1961) 72-75.
[8] E.W. Cheney, H.L. Loeb, On rational chebyshev approximation, Numer. Math. 4 (1962) 124-127.
[9] E.W. Cheney, H.L. Loeb, Generalized rational approximation, J. Soc. Indust. Appl. Math. Series B 1 (1964) 11-25.
[10] E.W. Cheney, T.H. Southard, A survey of methods for rational approximation, with particular reference to a new method based on a forumla of darboux, SIAM Rev. 5 (1963) 219-231.
[11] X. Cheng, et al., Polynomial regression as an alternative to neural nets, arXiv preprint arXiv:1806.06850, 2018.
[12] J.X. Da Cruz Neto, J.O. Lopes, M.V. Travaglia, Algorithms for quasiconvex minimization, Optimization 60 (2011) 1105-1117.
[13] A. Daniilidis, N. Hadjisavvas, J.E. Martinez-Legaz, An appropriate subdifferential for quasiconvex functions, SIAM J. Optim. 12 (2002) 407-420.
[14] Q. Delfosse, et al., Recurrent rational networks, arXiv preprint arXiv:2102.09407, 2021.
[15] J. Dutta, A.M. Rubinov, Abstract Convexity, Handbook of Generalized Convexity and Generalized Monotonicity, 76 (2005) 293-333.
[16] W. Fenchel, D.W. Blackett, Convex cones, sets, and functions, Princeton University, Department of Mathematics, Logistics Research Project, 1953.

[17] A. Goldberger, et al., Physiobank, physiotoolkit, and physionet: Components of a new research resource for complex physiologic signals, 2000.
[18] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.
[19] M. Goyal, R. Goyal, B. Lall, Learning activation functions: A new paradigm for understanding neural networks, arXiv preprint arXiv:1906.09529, 2019.
[20] J.R. Rice, The Approximation of Functions: Nonlinear and multivariate theory, Addison-Wesley series in computer science and information processing. Mass., Addison-Wesley Publishing Company, 1969.
[21] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature, 521 (2015) 436-444.
[22] H.L. Loeb, Algorithms for Chebyshev approximations using the ratio of linear forms, J. Soc. Indust. Appl. Math. 8 (1960) 458-465.
[23] J.E. Martínez-Legaz, Quasiconvex duality theory by generalized conjugation methods, Optimization, 19 (1998) 603-652.
[24] M.A. Mazurowski, et al., Training neural network classifiers for medical decision making: The effects of imbalanced datasets on classification performance, Neural Netw. 21 (2008) 427-436.
[25] M. Meltser, M. Shoham, L.M. Manevitz, Approximating functions by neural networks: a constructive solution in the uniform norm, Neural Networks 9 (1996) 965-978.
[26] R. Díaz Millán, et al., Multivariate approximation by polynomial and generalised rational functions, arXiv preprint arXiv:2101.11786, 2021.
[27] B. Mohammad, M. Timothy, Neural network for regression problems with reduced training sets, Neural Networks, 95 (2017) 1-9.
[28] A. Molina, P. Schramowski, K. Kersting, Padé activation units: End-to-end learning of flexible activation functions in deep networks, In: m International Conference on Learning Representations, 2019.
[29] Y. Nakatsukasa, O. Sète, L.N. Trefethen, The AAA algorithm for rational approximation, SIAM J. Sci. Comput. 40 (2018) A1494–A1522.
[30] S.K. Oh, W. Pedrycz, B.J. Park, Polynomial neural networks architecture: analysis and design, Comput. Elect. Eng. 29 (2003) 703-725.
[31] V. Peiris, et al., Generalised rational approximation and its application to improve deep learning classifiers, Appl. Math. Comput. 389 (2021) 125560.
[32] V. Peiris, N. Sukhorukova, The extension of the linear inequality method for generalized rational chebyshev approximation to approximation by general quasilinear functions, Optimization, 10.1080/02331934.2021.1939342.
[33] V. Peiris, N. Sukhorukova, V. Roshchina, Deep learning with nonsmooth objectives, 2021.
[34] M. Raissi, Deep hidden physics models: Deep learning of nonlinear partial differential equations, J. Machine Learn. Res. 19 (2018), 932-955.
[35] A.M. Rubinov, Abstract Convexity and Global Optimization, Kluwer Academic Publishers, New York, 2000.
[36] A.M. Rubinov, B. Simsek, Conjugate quasiconvex nonnegative functions, Optimization, 35 (1995) 1-22.
[37] M. Seonwoo, L. Byunghan, Y. Sungroh, Deep learning in bioinformatics, Briefings in Bioinfo. 18 (2017) 851-869.
[38] I. Steponavičė, et al., Efficient identification of the pareto optimal set, In: International Conference on Learning and Intelligent Optimization, pp. 341-352, Springer, 2014.
[39] M. Telgarsky, Neural networks and rational functions, In: International Conference on Machine Learning, PMLR, pp. 3387-3393, 2017.
[40] D. Vail, M. Veloso, Learning from accelerometer data on a legged robot, IFAC Proceedings Volumes 37 (2004) 822-827.
[41] J.P. Crouzeix, J.A. Ferland, S. Schaible, A note on an algorithm for generalized fractional programs, J. Optim. Theory Appl. 50 (1986) 183-187.
[42] A. Kolmogorov, On the representation of continuous functions of several variables by superpositions of continuous functions of a smaller number of variables, Proc. USSR Acad. Sci. 108 (1956) 179–182. English translation: Amer. Math. Soc. Transl. 17 (1961) 369–373.
[43] V. Kůrková, Kolmogorov's theorem is relevant, Neural Comput. 3 (1991) 617-622.
[44] V.N. Malozemov, Equalization of maxima, USSR Comput. Math. Math. Phys. 16 (1976) 232-236.

[45] S.Y. Matsushita, L. Xu, On the finite termination of the Douglas-Rachford method for the convex feasibility problem, Optimization. 65 (2016) 2037-2047.
[46] Y. Yuning, Y. Qingzhi, Some modified relaxed alternating projection methods for solving the two-sets convex feasibility problem, Optimization, 62 (2019) 509–525.
[47] A.J. Zaslavski, Subgradient projection algorithms and approximate solutions of convex feasibility problems, J. Optim. Theory Appl. 157 (2013) 803–819.
[48] X. Zhao, M.A. Köbis, On the convergence of general projection methods for solving convex feasibility problems with applications to the inverse problem of image recovery, Optimization, 67 (2018) 1409–1427.