A Neural Network Algorithm for Solving Quadratic Programming Based on Fibonacci Method

Jingli Yang and Tingsong Du

Institute of Nonlinear and Complex System, China Three Gorges University, YiChang, Hubei 443002, China
jennyyang_2008@163.com

Abstract. In this paper, a novel neural network algorithm is proposed, which solve the quadratic programming problem with linear constraints based on Fibonacci method. Compared with the existing models for solving the quadratic programming problem with linear constraints, it is more universal, since the objective function of the quadratic programming not only can be convex function but also can be quasi convex function. Finally, example is provided to show the applicability of the proposed neural network algorithm.

Keywords: Quadratic programming, Fibonacci method, Neural network, Learning algorithm.

1 Introduction

Optimization problems arise in a wide variety of scientific and engineering applications, including regression analysis, signal processing, system identification, filter design, robot control, function approximation, *etc*[1-2]. Many engineering problems can be solved by transforming the original problems into linearly constrained quadratic programming.

Compared with traditional numerical methods, the neural network approach can solve the optimization problems much faster in running times. Therefore, neural network methods for optimization problems have been received considerable attention. In 1985, Hopfield and Tank first proposed a neural network for solving Traveling Salesman Problem(TSP)[3]. Kennedy and Chua proposed a modified model and canonical circuit models that are superior to the Hopfield and Tank model. By using penalty parameter, they proposed a neural network for solving nonlinear programming problems[4]. Lately, many researchers successively proposed a number of models. In [5], Bouzerdoum and Pattison presented a neural network for solving convex quadratic optimization problems with bounded constraints. Liang &Wang and Xia &Wang presented several neural networks for solving nonlinear convex optimization with bounded constrains and box constraints, respectively [6-9]. He D.X.[10] presented a neural network for solving linear program based on bisection method. Nevertheless, majority of

L. Zhang, J. Kwok, and B.-L. Lu (Eds.): ISNN 2010, Part I, LNCS 6063, pp. 118–125, 2010.

these methods are virtue of energy function, used back propagation(BP) algorithm. While BP algorithm is one order algorithm, it's slowly and usually reach the local solution.

In this paper, by employing Fibonacci method and the characteristic of common network topology, we present a neural network algorithm for solving the quadratic programming with linear constraints. The objective function of the quadratic programming is broad, which can be convex function or quasi convex function.

The paper is organized as follows. In Section 2, we introduce the theoretical foundation about Fibonacci method and quadratic programming. In Section 3, we present the neural network based on Fibonacci method. The neural network algorithm is proposed in Section 4. In Section 5, a example is discussed to evaluate the effectiveness of the proposed neural network algorithm. Finally, the conclusions are made in Section 6.

2 Preliminaries

2.1 Quadratic Programming Problem

We consider the following quadratic programming problem:

$$\begin{cases} \text{minimize} & f(x) = \frac{1}{2}x^T Ax + c^T x \\ \text{subject to} & l \leq Dx \leq h \end{cases} \tag{1}$$

where x=$(x_1, x_2, \cdots, x_n)^T \in R^n$, $A \in R^{n\times n}$, $D \in R^{n\times n}$is a nonsingular positive matrix, $c \in R^n$,and $l, h \in R^n$.

Remark. Clearly, when $D = I$, the problem (1) is quadratic programming problem with bound constraints[9].

2.2 Description of the Fibonacci Method

The Fibonacci numbers are defined as follows:

$$\begin{cases} F_0 = F_1 = 1, \\ F_{k+1} = F_k + F_{k-1}, k = 1, 2, \ldots \end{cases} \tag{2}$$

the formula can be showed as

$$F_k = \frac{1}{\sqrt{5}}[(\frac{1+\sqrt{5}}{2})^{k+1} - (\frac{1-\sqrt{5}}{2})^{k+1}], k = 0, 1, 2, \ldots \tag{3}$$

The basic idea for Fibonacci method to solve optimization problem is when we know the first search interval [a_1,b_1] and the iteration precision ε, we can obtain the last search interval [a_n,b_n] after iterating n times. We can find that

$$b_n - a_n \leq \varepsilon.$$

The shortening rate for the search interval is that

$$b_{k+1} - a_{k+1} \leq \frac{F_{n-k}}{F_{n-k+1}}(b_k - a_k).$$

Since

$$b_n - a_n = \frac{F_1}{F_2}(b_{n-1} - a_{n-1}) = \frac{F_1}{F_2} \cdot \frac{F_2}{F_3} \cdots \frac{F_{n-1}}{F_n}(b_1 - a_1) = \frac{1}{F_n}(b_1 - a_1).$$

Then

$$\frac{1}{F_n}(b_1 - a_1) \leq \varepsilon,$$

thus

$$F_n \geq \frac{b_1 - a_1}{\varepsilon}. \tag{4}$$

According to (4), we can get the minimum F_n. By using (3), we can obtain n, then the search times n can be found. By using the procedure below we can find the optimal solution for optimization problems:

Step 1: Give the constrain condition $a_1 \leq x \leq b_1$ and the iteration precision ε, according to (3)and (4)we know n, then we can get F_n, F_{n-1}, F_{n-2}, set

$$\lambda_1 = a_1 + (1 - \frac{F_{n-1}}{F_n})(b_1 - a_1) = \frac{F_{n-1}}{F_n}a_1 + (1 - \frac{F_{n-1}}{F_n})b_1 = \frac{F_{n-1}}{F_n}a_1 + \frac{F_{n-2}}{F_n}b_1,$$

$$\mu_1 = a_1 + \frac{F_{n-1}}{F_n}(b_1 - a_1) = (1 - \frac{F_{n-1}}{F_n})a_1 + \frac{F_{n-1}}{F_n}b_1 = \frac{F_{n-2}}{F_n}a_1 + \frac{F_{n-1}}{F_n}b_1,$$

put $k = 1$.

Step 2: If $|b_k - a_k| < \varepsilon$, end. The optimal solution $x^* \in [a_k, b_k]$, let $x^* = (a_k + b_k)/2$, otherwise, let

$$\begin{aligned}\lambda_k = a_k + (1 - \frac{F_{n-k}}{F_{n-k+1}})(b_k - a_k) &= \frac{F_{n-k}}{F_{n-k+1}}a_k + (1 - \frac{F_{n-k}}{F_{n-k+1}})b_k \\ &= \frac{F_{n-k}}{F_{n-k+1}}a_k + \frac{F_{n-k-1}}{F_{n-k+1}}b_k,\end{aligned}$$

$$\begin{aligned}\mu_k = a_k + \frac{F_{n-k}}{F_{n-k+1}}(b_k - a_k) &= (1 - \frac{F_{n-k}}{F_{n-k+1}})a_k + \frac{F_{n-k}}{F_{n-k+1}}b_k \\ &= \frac{F_{n-k-1}}{F_{n-k+1}}a_k + \frac{F_{n-k}}{F_{n-k+1}}b_k,\end{aligned}$$

then we can get $f(\lambda_k)$, $f(\mu_k)$. If $f(\lambda_k) > f(\mu_k)$, turn to *step 3*, otherwise, turn to *step 4*.

Step 3: Let $a_{k+1} = \lambda_k$, $b_{k+1} = b_k$, and let

$$\lambda_{k+1} = \mu_k,$$

$$\mu_{k+1} = a_{k+1} + \frac{F_{n-k}}{F_{n-k+1}}(b_{k+1} - a_{k+1}) = \frac{F_{n-k-1}}{F_{n-k+1}}a_{k+1} + \frac{F_{n-k}}{F_{n-k+1}}b_{k+1},$$

calculate $f(\mu_{k+1})$.

Step 4: Let $a_{k+1} = a_k$, $b_{k+1} = \mu_k$, and let

$$\lambda_{k+1} = a_{k+1} + (1 - \frac{F_{n-k}}{F_{n-k+1}})(b_{k+1} - a_{k+1}) = \frac{F_{n-k}}{F_{n-k+1}} a_{k+1} + \frac{F_{n-k-1}}{F_{n-k+1}} b_{k+1},$$

$$\mu_{k+1} = \lambda_k,$$

calculate $f(\lambda_{k+1})$.

Step 5: Let $k = k + 1$, turn to *Step 2*.

3 Neural Network Structure

According to Fibonacci method, we design the neural network as follow:

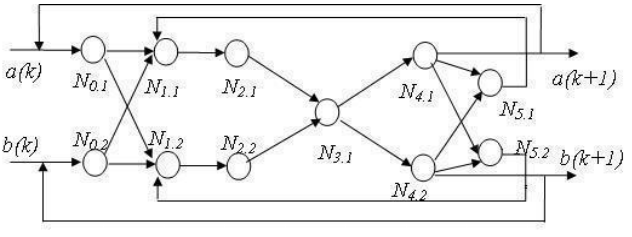

Fig. 1. Neural network structure based on Fibonacci method

We suppose that:$N_{i,j}$ means the neuron j in layer i, $net_{i,j}$ denote as the input value of neuron $N_{i,j}$, $O_{i,j}$ means the output value of neuron $N_{i,j}$, and $\omega_{(i,j),(k,l)}$ means the connection weight from $N_{i,j}$ to $N_{k,l}$. Biasing threshold vector is defined by $\theta = 0$.

The neural network structure in Fig.1 contains 6 layers, including input layer, three hidden layers, one feedback layer and output layer. We arrange calculation steps below to solve quadratic programming.

(I) Input Layer: Let $[a_k, b_k]$ as the input of $N_{0,1}$ and $N_{0,2}$, then

$$net_{0,1} = a_k = a_1,$$
$$net_{0,2} = b_k = b_1,$$

the activation functions are defined by $\varphi_{0,1}(x) = x$, $\varphi_{0,2}(y) = y$, then the outputs of $N_{0,1}$, $N_{0,2}$ are

$$O_{0,1} = \varphi_{0,1}(net_{0,1}) = a_1,$$
$$O_{0,2} = \varphi_{0,1}(net_{0,2}) = b_1.$$

(II) Hidden Layer: For the first hidden layer,$N_{1,1}$ enforces λ_k , $N_{1,2}$ enforces μ_k, the corresponding connection weights are

$$\omega_{(0,1),(1,1)} = \omega_{(0,2),(1,2)} = \frac{F_{n-k}}{F_{n-k+1}},$$

$$\omega_{(0,2),(1,1)} = \omega_{(0,1),(1,2)} = 1 - \frac{F_{n-k}}{F_{n-k+1}} = \frac{F_{n-k-1}}{F_{n-k+1}},$$

then the output of neurons $N_{1,1}$ and $N_{1,2}$ are

$$O_{1,1} = \omega_{(0,1),(1,1)} * a_k + \omega_{(0,2),(1,1)} * b_k = \lambda_k,$$
$$O_{1,2} = \omega_{(0,1),(1,2)} * a_k + \omega_{(0,2),(1,2)} * b_k = \mu_k.$$

For the second hidden layer, $N_{2,1}$ and $N_{2,2}$ are used to enforce the output of λ_k, μ_k, respectively. Set $\omega_{(1,1),(2,1)} = \omega_{(1,2),(2,2)} = 1$, then

$$O_{2,1} = f(\omega_{(1,1),(2,1)} * \lambda_k) = f(\lambda_k),$$
$$O_{2,2} = f(\omega_{(1,2),(2,2)} * \mu_k) = f(\mu_k).$$

For the third hidden layer, the input is

$$net_{3,1} = (\omega_{(2,1),(3,1)} * O_{2,1}) - (\omega_{(2,2),(3,1)} * O_{2,2}),$$

let $\omega_{(2,1),(3,1)} = \omega_{(2,2),(3,1)} = 1$, setting the activation function is

$$\varphi(net_{3,1}) = \begin{cases} 1, & \text{if } net_{3,1} > 0, \\ 0, & \text{otherwise,} \end{cases}$$

then the output is

$$O_{3,1} = f(\lambda_k) - f(\mu_k).$$

(III) Output Layer: There are two neurons in the output layer,which are used to calculate a_{k+1}, b_{k+1}, set

$$\omega_{(3,1),(4,1)} = \omega_{(3,1),(4,2)} = 1,$$

then the outputs are

$$O_{4,1} = O_{3,1} * O_{1,1} + (1 - O_{3,1}) * O_{0,1} = \begin{cases} a_{k+1} = \lambda_k = O_{1,1}, & O_{3,1} = 1, \\ a_{k+1} = a_k = O_{0,1}, & O_{3,1} = 0, \end{cases}$$

$$O_{4,2} = O_{3,1} * O_{0,2} + (1 - O_{3,1}) * O_{1,2} = \begin{cases} b_{k+1} = b_k = O_{0,2}, & O_{3,1} = 1, \\ b_{k+1} = \mu_k = O_{1,2}, & O_{3,1} = 0. \end{cases}$$

(IV) Feedback Layer: The neurons $N_{5,1}$ and $N_{5,2}$ in feedback layer are used to calculate λ_{k+1} and μ_{k+1},

$$O_{5,1} = \frac{F_{n-k}}{F_{n-k+1}}(1 - O_{3,1}) * O_{4,1} + \frac{F_{n-k-1}}{F_{n-k+1}}(1 - O_{3,1}) * O_{4,2} + O_{3,1} * O_{1,2}$$
$$= \begin{cases} \lambda_{k+1} = O_{1,2}, & O_{3,1} = 1, \\ \lambda_{k+1} = \frac{F_{n-k}}{F_{n-k+1}} O_{4,1} + \frac{F_{n-k-1}}{F_{n-k+1}} O_{4,2}, & O_{3,1} = 0, \end{cases}$$

$$O_{5,2} = \frac{F_{n-k-1}}{F_{n-k+1}} O_{3,1} * O_{4,1} + \frac{F_{n-k}}{F_{n-k+1}} O_{3,1} * O_{4,2} + (1 - O_{3,1}) * O_{1,1}$$
$$= \begin{cases} \mu_{k+1} = \frac{F_{n-k-1}}{F_{n-k+1}} O_{4,1} + \frac{F_{n-k}}{F_{n-k+1}} O_{4,2}, & O_{3,1} = 1, \\ \mu_{k+1} = O_{1,1}, & O_{3,1} = 0. \end{cases}$$

(V) Iteration: Let the output neurons as the input of the feedback layer, If $f(\lambda_k) > f(\mu_k)$, then $a_{k+1} = \lambda_k$, $b_{k+1} = b_k$, otherwise, let $a_{k+1} = a_k$, $b_{k+1} = \mu_k$.

Let $k = 1$, $k = k + 1$,circulation,until $|b_k - a_k| \leq \varepsilon$.

4 The Neural Network Algorithm Based on Fibonacci Method

For the quadratic programming subject to linear constraints, we can present the algorithm based on the neural network structure in Section 3.

Step 1: According to the constrains condition $l \leq Dx \leq h$, we can obtain the upper bound and lower bound of x, denote a_1, b_1. Let $net_{0,1} = a_1, net_{0,2} = b_1$, give $\varepsilon \geq 0$ as iteration precision. If $|b_1 - a_1| < \varepsilon$, let optimal solution $x^* = (a_1 + b_1)/2$, otherwise, let the initial solution is $x(1) = (a_1 + b_1)/2$, turn to *Step 2*.

Step 2: Calculate $O_{1,1} = \lambda_1$, $O_{1,2} = \mu_1$, $O_{2,1} = f(\lambda_1)$, $O_{2,2} = f(\mu_1)$, let $O_{3,1} = f_(\lambda_1) - f(\mu_1)$.

Step 3: If $O_{3,1} > 0$, let $O_{4,1} = a_2 = \lambda_1$,$O_{4,2} = b_2 = b_1$, otherwise, let $O_{4,1} = a_2 = a_1$, $O_{4,2} = b_2 = \mu_1$.

Step 4: When $O_{3,1} > 0$, let

$$O_{5,1} = \lambda_2 = \mu_1,$$
$$O_{5,2} = \mu_2 = a_2 + \frac{F_{n-2}}{F_{n-2+1}}(b_2 - a_2) = \frac{F_{n-2-1}}{F_{n-2+1}}a_2 + \frac{F_{n-2}}{F_{n-2+1}}b_2.$$

When $O_{3,1} \leq 0$, let

$$O_{5,1} = \lambda_2 = a_2 + (1 - \frac{F_{n-2}}{F_{n-2+1}})(b_2 - a_2) = \frac{F_{n-2}}{F_{n-2+1}}a_2 + \frac{F_{n-2-1}}{F_{n-2+1}}b_2,$$
$$O_{5,2} = \mu_2 = \lambda_1.$$

Step 5: Let $k = 1$, $k = k+1$, and let $net_{0,1} = a_k$, $net_{0,2} = b_k$, until $|b_k - a_k| \leq \varepsilon$, then we can obtain $x^* = (a_k + b_k)/2$.

Theorem. Used the neural network algorithm based on Fibonacci method to solve quadratic programming problem, the error of the approximate solution is less than $\tau^k|b_1 - a_1|$, where $\tau = \frac{F_{n-k}}{F_{n-k+1}}$. And we can conclude that

$$\lim_{k\to\infty} \frac{F_{k-1}}{F_k} = \frac{\sqrt{5}-1}{2} = 0.618. \tag{5}$$

Proof. According to the neural network algorithm we proposed, we can get $[a_k, b_k]$,$k = 1, 2, \cdots, n$, while $|b_k - a_k| = \tau^k|b_1 - a_1| \to 0$,$(k \to \infty)$, then when $|b_k - a_k| < \varepsilon$, let $x^* = \frac{a_k+b_k}{2}$ as the approximate optimal solution, then the error is less then $\tau^k|b_1 - a_1|$, where $\tau = \frac{F_{n-k}}{F_{n-k+1}}$.

According to (2), we can easily obtain (5).

5 Simulation Result

In this section, we give an illustrative example. The simulation is conducted in MATLAB.

Consider a quadratic programming example as

$$\begin{cases} \min & f(x) = x_1^2 + 4x_2^2 - 4x_1x_2 - 2x_1 \\ \text{s. t.} & 1 \le -x_1 + x_2 \le 2 \\ & 2 \le 2x_1 + 3x_2 \le 3. \end{cases} \tag{6}$$

Then

$$A = \begin{pmatrix} 2 & -4 \\ -4 & 8 \end{pmatrix}, \quad c = \begin{pmatrix} -2 \\ 0 \end{pmatrix},$$

and

$$D = \begin{pmatrix} -1 & 1 \\ 2 & 3 \end{pmatrix}, \quad l = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \quad h = \begin{pmatrix} 2 \\ 3 \end{pmatrix}.$$

This problem has a unique optimal solution $x^* = (-0.2, 0.8)^T$ and the optimal value $f^* = 3.64$ when solved by the neural network model in [9].

We used 'quadprog' in Matlab toolbox to solve this problem, then we can get the optimal solution $x^* = (0, 1.0)^T$, and the optimal value $f^* = 4$. We use the algorithm in Section 4 to solve the above problem. Let $x(1) = (-0.4, 1.1)^T$, $\varepsilon = 10^{-5}$, iterate 23 times, the problem is globally converge to an unique optimal solution $x^* = (-0.200003, 0.800004)^T$, and the optimal value of the objective function is $f^* = 3.6405$. Compared with the results in [9], the results in this paper has higher accuracy. Figure.2 show the transient behaviors of the decision variable.

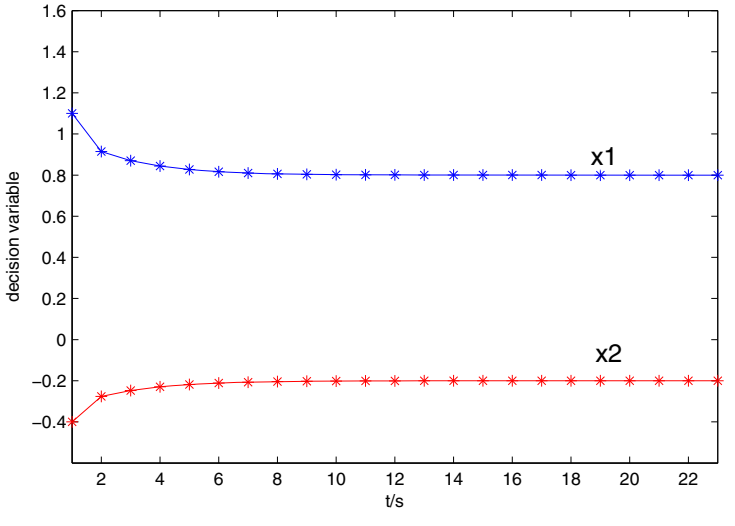

Fig. 2. Transient Behaviors of the decision variable

6 Conclusion

This paper presents a new neural network algorithm for solving quadratic programming problems by using Fibonacci method and neural network. It has also been substantiated that the proposed neural network algorithm is able to generate optimal solution to linear programming with bound constraints. Compared with other neural network models, the objective function of the quadratic programming in this algorithm is universal, which can be convex function or quasi convex function. With the dimension increasing, the advantage of this algorithm is more clearly. It has been shown that the proposed algorithm is easy to implement in computer.

Acknowledgments

This work is supported by the Graduate Scientific Research Creative Foundation of Three Gorges University, China(200949), the Scientific Innovation Team Project of Hubei Provincial Department of Education(T200809), the Natural Science Foundation of Hubei Province, China(2008CDZ046) and the National Natural Science Foundation, China (10726016).

References

1. Horst, R., Pardalos, P.M., Thoai, N.V.: Introduction to Global Optimization. Tsinghua University Publishing House, Beijing (2005)
2. Bertsekas, D.P.: Parallel and Distributed Numerical Methods. Prentice Hall, Englewood Cliffs (1989)
3. Hopfield, J.J., Tank, D.W.: Neural Computation of Decisions in Optimization Problems. Biol. Cybern. 52, 141–152 (1985)
4. Kennedy, M.P., Chua, L.O.: A Neural Networks for Nonlinear Programming. IEEE Trans. on Circuits and Systems 35, 554–562 (1988)
5. Bouzerdoum, A., Pattison, T.R.: Neural Network for Quadratic Optimization with Bound Constraints. IEEE Trans. on Neural Networks 4, 293–304 (1993)
6. Xia, Y.S., Wang, J.: A Recurrent Neural Network for Solving Linear Projection Equation. Neural Networks 13, 337–350 (2000)
7. Xia, Y.S., Wang, J.: A Dual Neural Network for Kinematic Control of Redundant Robot Manipulators. IEEE Trans. Syst.,Man Cybern. Pan B 31, 147–154 (2001)
8. Liang, X.B., Wang, J.: A Recurrent Neural Network for Nonlinear Optimization with a Conntinuously Differentiable Objective Function and Bound Constrains. IEEE Trans. Neural Networks 11, 1251–1262 (2000)
9. Xia, Y.S., Wang, J.: Primal Neural Networks for Solving Convex Quadratic Programs. In: Neural Networks, IJCNN, pp. 582–587 (1999)
10. He, D.X.: Neural Network for Solving Linear Program Based on Bisection Method. Computer Engineering and Applications 20, 102, 74–75 (2006)