

Avoiding local minima in Deep Learning: a nonlinear optimal control approach

Jan Scheers

Thesis voorgedragen tot het behalen
van de graad van Master of Science
in de ingenieurswetenschappen:
wiskundige ingenieurstechnieken

Promotor:
Prof. dr. ir. Panos Patrinos

© Copyright KU Leuven

Without written permission of the thesis supervisor and the author it is forbidden to reproduce or adapt in any form or by any means any part of this publication. Requests for obtaining the right to reproduce or utilize parts of this publication should be addressed to the Departement Computerwetenschappen, Celestijnenlaan 200A bus 2402, B-3001 Heverlee, +32-16-327700 or by email info@cs.kuleuven.be.

A written permission of the thesis supervisor is also required to use the methods, products, schematics and programmes described in this work for industrial or commercial use, and for submitting this publication in scientific contests.

Zonder voorafgaande schriftelijke toestemming van zowel de promotor als de auteur is overnemen, kopiëren, gebruiken of realiseren van deze uitgave of gedeelten ervan verboden. Voor aanvragen tot of informatie i.v.m. het overnemen en/of gebruik en/of realisatie van gedeelten uit deze publicatie, wend u tot het Departement Computerwetenschappen, Celestijnenlaan 200A bus 2402, B-3001 Heverlee, +32-16-327700 of via e-mail info@cs.kuleuven.be.

Voorafgaande schriftelijke toestemming van de promotor is eveneens vereist voor het aanwenden van de in deze masterproef beschreven (originele) methoden, producten, schakelingen en programma's voor industrieel of commercieel nut en voor de inzending van deze publicatie ter deelname aan wetenschappelijke prijzen of wedstrijden.

Preface

I would like to thank everybody who kept me busy the last year, especially my promoter and my assistants. I would also like to thank the jury for reading the text. My sincere gratitude also goes to my friends and my family.

Jan Scheers

Contents

Preface	i
Abstract	iii
1 Introduction	1
1.1 Artificial Neural Networks	1
1.2 Backpropagation	2
1.3 Neural Networks in Optimal Control Theory	3
2 Initial exploration	5
2.1 First experiment	5
2.2 Second experiment	5
2.3 Further exploration	6
3 Python implementation	7
3.1 Augmented Lagrangian Method	7
3.2 Jacobian	8
3.3 Algorithmic Verification of Jacobian	10
3.4 Alternative Representation	10
4 Tests	13
4.1 The First Topic of this Chapter	13
4.2 The Second Topic	13
4.3 Conclusion	13
5 Conclusion	15
A Source code	19
A.1 First experiment source	19

Abstract

Chapter 1

Introduction

1.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are a very popular machine learning model. They are known to be very expressive, leading to low statistical bias. With enough neurons, ANNs can approximate any function. They are especially useful for learning from very large data sets. But it is not entirely clear what the optimization of an ANN converges to, as the loss surface is highly non-convex. Nonetheless a number of results show that for wide enough networks, there are few "bad" local minima.

ANNs are composed of 'neurons', which are in some ways analogous to biological neurons. Each neuron is a nonlinear function transforming the weighted sum of its inputs:

$$y = \sigma(w_1x_1 + w_2x_2 + \dots + w_nx_n)$$

With w_i the weights, x_i the inputs and σ the activation function. This is the McCulloch-Pitts neuron model.[REF] The most commonly used activation function σ is the Rectified Linear Unit (ReLU):

$$\sigma(x) = x^+ = \max(0, x)$$

A visual representation is shown in figure 1.1b. A full network is built by connecting layers of neurons as shown in figure 1.1a. An ANN can also be expressed as a combination of function composition and matrix multiplication.

$$f(W, x) = W_L\sigma(W_{L-1}\sigma(\dots W_1\sigma(W_0x)\dots))$$

where W_n are the matrixes of the connection weights and L is the depth of the network.

Optimal Control	Neural Network
decision variables	weight parameters
state variables	(neuron) activation

1.2 Backpropagation

Backpropagation is the usual algorithm for training an ANN. Training of a network is done by minimizing the loss function C . For regression problems this will be often be a squared error loss, while for classification tasks the cross entropy is used.

$$\underset{W}{\text{minimize}} \quad C(W) = \sum_{j=0}^N \|f(W, x^j) - y^j\|^2$$

with x^j the input vectors, y^j the target output and N the number of data points

Backpropagation computes the gradient of the loss function C with respect to the weight matrices W . It has two steps: in the first step the output of the network is calculated using the current weights and input v . The activation values and derivatives of each Then the error is propagated backward and the gradient is calculated.

TODO: backpropagation beter uitleggen

Then any gradient descent method can be used to find the step update.

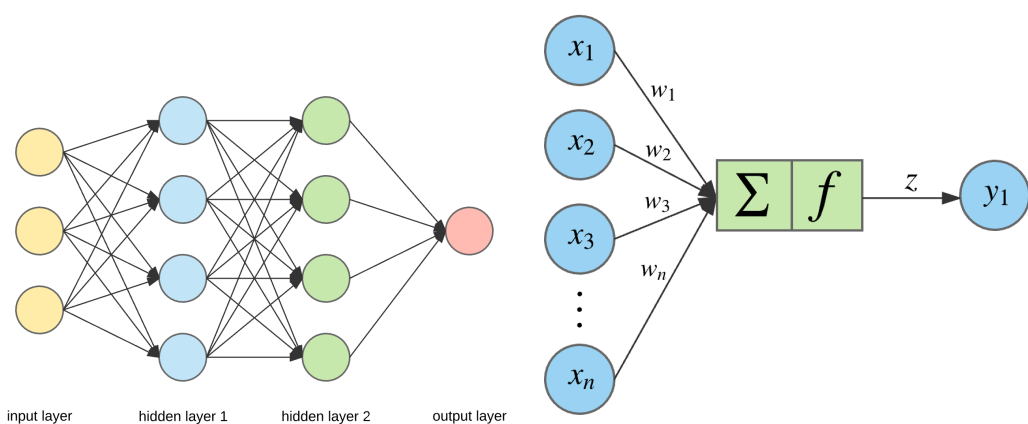


FIGURE 1.1: Feedforward Deep Neural Network and Single Neuron - McCulloch-Pitts model. (Retrieved from <https://towardsdatascience.com>)

1.3 Neural Networks in Optimal Control Theory

A neural network can also be interpreted as a dynamical system.

$$\begin{aligned} z_0 &= x \\ z_{k+1} &= \sigma(W_k z_k), \quad k = 0, \dots, L-1 \\ y &= W_H z_L \end{aligned}$$

In this way optimization methods from control theory can be applied. In particular, training a neural network can be formulated as the following Optimal Control Problem (OCP)

$$\begin{aligned} \underset{W}{\text{minimize}} \quad & \sum_{j=0}^N \|W_L z_L^j - y^j\|^2 \\ \text{subject to} \quad & z_{k+1}^j = \max(W_k z_k^j, 0), \quad k = 0, \dots, L-1, j = 1, \dots, N \end{aligned}$$

There are two main direct approaches to solving an OCP. First is the sequential approach, where the states are eliminated using the dynamics. This is equivalent to the backpropagation algorithm which is the current standard method. [ref mizutani]

The other approach is the simultaneous approach, where the state variables and the dynamics are kept as constraints. In control theory this approach often works better for highly nonlinear problems, which is certainly the case for training neural networks. The simultaneous approach is novel to neural networks and will be the topic of this thesis.

The disadvantage of this method is the number of variables that need to be optimized is much larger. For a fully connected neural network of width W , each layer will contain W^2 weights. Combined with a depth D , that gives approximately $W^2 D$ weight variables to be optimized for both the backpropagation and simultaneous approach. Adding the states as variables however adds another $W D N$ variables, where N is the number of samples in a training batch. The advantage of this method is that relaxing the states makes the problem more smooth, and will hopefully allow the optimization to converge more often to a good solution and not land in a bad local minimum. Topic of the thesis to investigate this

Chapter 2

Initial exploration

In this chapter an initial exploration of the problem is done. This was done in MATLAB with the neural network toolbox for the backpropagation algorithm, and YALMIP with fmincon for the simultaneous approach.

TODO version numbers, hardware

2.1 First experiment

The first experiment conducted was to apply both algorithms to a test problem. A small neural network is constructed with 2 hidden layers with each layer containing 3 nodes with tansig activation function. The activation function was chosen because it works well for a small network. This network is then trained to approximate a piece of a sine function.

The backpropagation algorithm converges every time for this problem, taking anywhere between 10 and 1000 iterations. The stopping criteria is that the validation error increases for 6 consecutive iterations.

The simultaneous approach also converges if the initial conditions are set correctly. The weight variables are randomly initialized and the state variables are initialized by simulating the network once using the input vector, so the initial point is feasible. Letting fmincon run for 200 iterations gives a good solution every time, but the algorithm runs much slower. Its not known if this is due to the difference in optimization of the algorithms and the fact that the backpropagation algorithm is running on my GPU instead of the CPU.

2.2 Second experiment

TODO make it more scientific.

For this experiment the same test problem is considered but with a different network. Here a shallow network 1 layer deep and 10 neurons wide is considered, with the ReLU activation function.

The backpropagation function will sometimes get stuck in a bad solution while training this network, but usually finds a good approximation.

For the simultaneous approach the RELU constraints will have to be reformulated. The ReLU function can be transformed into smooth constraints as follows:

$$\begin{aligned}
x_{k+1}^j &= \max(W_k x_k^j, 0) \\
&\Downarrow \\
x_{k+1}^j &= -\min(-W_k x_k^j, 0) \\
&\Downarrow \\
\min(x_{k+1}^j - W_k x_k^j) &= 0 \\
&\Downarrow \\
(x_{k+1}^j - W_k x_k^j)^\top x_{k+1}^j &= 0, \\
x_{k+1}^j &\geq 0, x_{k+1}^j - W_k x_k^j \geq 0
\end{aligned}$$

Even using these smooth constraints, the algorithm very often gets stuck in a bad local minimum.

2.3 Further exploration

There are many options that can be explored to further compare these algorithms.

- Activation function: ReLU is the most popular activation function in the field. Others are tansig, sigmoid, SoftPlus, leaky ReLU, etc.
- Network size: Networks can be up to thousands of neurons wide and hundreds of layers deep.
- Network architecture: There is a huge variety of possible network architectures. Convolutional neural networks for example are very popular for image recognition.
- Test problems: Neural networks have many applications. Some applications could benefit more from this training method than others.
- Optimization algorithm: fmincon is quite a general method, a more specific method might perform better
- Stopping criteria and initial conditions
These will be explored in the next chapters

Chapter 3

Python implementation

In this chapter the simultaneous approach is examined more closely and implemented into python.

3.1 Augmented Lagrangian Method

Instead of using fmincon, now an Augmented Lagrangian Method is used to solve the optimal control problem. The OCP equations

$$\begin{aligned} \underset{W}{\text{minimize}} \quad & \frac{1}{2} \sum_{j=0}^N \|W_H z_H^j - y^j\|^2 \\ \text{subject to} \quad & 0 = z_{k+1}^j - \sigma(W_k x_k^j + b_k), \quad k = 0, \dots, H-1, j = 1, \dots, N \end{aligned}$$

are represented as a nonlinear least squares problem

$$\begin{aligned} \min \quad & \frac{1}{2} \|F(x)\|_2^2 \\ \text{s. t.} \quad & h(x) = 0 \end{aligned}$$

The ALM method solves this problem by adding both a penalty and a lagrangian term. After rewriting, the nonlinear least squares problem looks like this

$$\begin{aligned} \mathcal{L}_c(x, \lambda) &= \frac{1}{2} \|F(x)\|_2^2 + \langle \lambda, h(x) \rangle + \frac{c}{2} \|h(x)\|_2^2 \\ &= \frac{1}{2} \|F(x)\|_2^2 + \frac{c}{2} \|h(x) + \lambda/c\|_2^2 - \frac{1}{2c} \|\lambda\|_2^2 \\ &= \frac{c}{2} \left\| \begin{bmatrix} F(x)/\sqrt{c} \\ h(x) + \lambda/c \end{bmatrix} \right\|^2 \end{aligned}$$

In the inner loop of the algorithm the least squares problem is solved for x^k up to a certain tolerance.

$$x^k \text{ s. t. } \|\nabla_k \mathcal{L}_{c_k}(x^k, \lambda^k)\|_2 \leq \epsilon \quad (3.1)$$

The langrangian parameters λ are then updated using the rule:

$$\lambda^{k+1} = \lambda^k + c_k h(x^k) \quad (3.2)$$

A new penalty parameter c^{k+1} is chosen and the algorithm continues. REFERNOCEDAL WRIGHT,

3.2 Jacobian

To solve the least squares problem, the Jacobian matrix must be calculated. It has a relatively sparse structure because there are no distant connections in the neural net, each layer is only connected to the next one and the previous one.

Neural networks obviously have many possible architectures, so a simple fully connected rectangular feedforward network is considered. It has an input dimension I , an output dimension O , and it has D hidden layers of width W . The weight matrixes have $IxW + OxW + (D - 1)xWxW$ parameters, the bias vectors have $DxW + O$ parameters and the state vectors have $DxWxN$ parameters. On the other hand $\mathcal{L}_c(x, \lambda)$ will have an output dimension of $DxWxN + OxN$. The dimension of the Jacobian will be $(D.W.N + O.N)x(D.W.N + O + (D + I + O)W + (D - 1)W^2)$. Therefore the Jacobian will be taller than it is wide when $N \geq 1 + (D + I + O)W/O + (D - 1)W^2/O$. Written out completely the Jacobian will look like Figure

3.2. Jacobian

$\nabla \mathcal{L}$		W_{0_1}	W_{0_2}	...	W_{0_W}	b_0
		I	I	...	I	W
F	O*N	0	0	...	0	0
h_1	N	$-x\sigma'(W_{0_1}x + b_{0_1})$	0	...	0	$-\sigma'(W_{0_1}x + b_{0_1})$
	N	0	$-x\sigma'(W_{0_2}x + b_{0_2})$...	0	$-\sigma'(W_{0_2}x + b_{0_2})$

	N	0	0	...	$-x\sigma'(W_{0_W}x + b_{0_W})$	$-\sigma'(W_{0_W}x + b_{0_W})$
h_2	W*N	0	0	...	0	0
...
h_D	W*N	0	0	...	0	0

$\nabla \mathcal{L}$		W_{i_1}	W_{i_2}	...	W_{i_W}	b_1
		W	W	...	W	W
F	O*N	0	0	...	0	0
h_1	W*N	0	0	...	0	0
...
h_{i+1}	N	$-z_1\sigma'(W_{i_1}z + b_{i_1})$	0	...	0	$-\sigma'(W_{i_1}x + b_{i_1})$
	N	0	$-z_1\sigma'(W_{i_2}z + b_{i_2})$...	0	$-\sigma'(W_{i_2}x + b_{i_2})$

	N	0	0	...	$-z_1\sigma'(W_{i_W}z + b_{i_W})$	$-\sigma'(W_{i_W}x + b_{i_W})$
...
h_D	W*N	0	0	...	0	0

$\nabla \mathcal{L}$		W_{D_1}	W_{D_2}	...	W_{D_O}	b_D
		W	W	...	W	O
F	N	$-\frac{z_D}{\sqrt{c}}\sigma'_O(W_{D_1}x + b_{D_1})$	0	...	0	$-\frac{1}{\sqrt{c}}\sigma'_O(W_{D_1}x + b_{D_1})$
	N	0	$-\frac{z_D}{\sqrt{c}}\sigma'_O(W_{D_2}x + b_{D_2})$...	0	$-\frac{1}{\sqrt{c}}\sigma'_O(W_{D_2}x + b_{D_2})$

	N	0	0	...	$-\frac{z_D}{\sqrt{c}}\sigma'_O(W_{D_O}x + b_{D_O})$	$-\frac{1}{\sqrt{c}}\sigma'_O(W_{D_O}x + b_{D_O})$
h_1	W*N	0	0	...	0	0
...
h_D	W*N	0	0	...	0	0

3. PYTHON IMPLEMENTATION

Square Diagonal Matrices

$\nabla \mathcal{L}$		z_{i_1}	z_{i_2}	...	z_{i_W}
		N	N	...	N
F	O*N	0	0	...	0
h_1	W*N	0	0	...	0
...
h_i	N	1	0	...	0
	N	0	1	...	0

	N	0	0	...	1
h_{i+1}	N	$-W_{i_1,1}\sigma'(W_{i_1}z_i + b_{i_1})$	$-W_{i_1,2}\sigma'(W_{i_1}z_i + b_{i_1})$...	$-W_{i_1,W}\sigma'(W_{i_1}z_i + b_{i_1})$
	N	$-W_{i_2,1}\sigma'(W_{i_2}z_i + b_{i_2})$	$-W_{i_2,2}\sigma'(W_{i_2}z_i + b_{i_2})$...	$-W_{i_2,W}\sigma'(W_{i_2}z_i + b_{i_2})$

	N	$-W_{i_W,1}\sigma'(W_{i_W}z_i + b_{i_W})$	$-W_{i_W,2}\sigma'(W_{i_W}z_i + b_{i_W})$...	$-W_{i_W,W}\sigma'(W_{i_W}z_i + b_{i_W})$
...
h_D	W*N	0	0	...	0

$\nabla \mathcal{L}$		z_{D_1}	z_{D_2}	...	z_{D_W}
		N	N	...	N
F	N	$-W_{D_1,1}\sigma'_O(W_{D_1}z_D + b_{D_1})$	$-W_{D_1,2}\sigma'_O(W_{D_1}z_D + b_{D_1})$...	$-W_{D_1,W}\sigma'_O(W_{D_1}z_D + b_{D_1})$
	N	$-W_{D_2,1}\sigma'_O(W_{D_2}z_D + b_{D_2})$	$-W_{D_2,2}\sigma'_O(W_{D_2}z_D + b_{D_2})$...	$-W_{D_2,W}\sigma'_O(W_{D_2}z_D + b_{D_2})$

	N	$-W_{D_O,1}\sigma'_O(W_{D_O}z_D + b_{D_O})$	$-W_{D_O,2}\sigma'_O(W_{D_O}z_D + b_{D_O})$...	$-W_{D_O,W}\sigma'_O(W_{D_O}z_D + b_{D_O})$
h_1	W*N	0	0	...	0
...
h_D	N	1	0	...	0
	N	0	1	...	0

	N	0	0	...	1

How to read figure

3.3 Algorithmic Verification of Jacobian

3.4 Alternative Representation

In this section we explain an alternative representation which is mathematically the same.

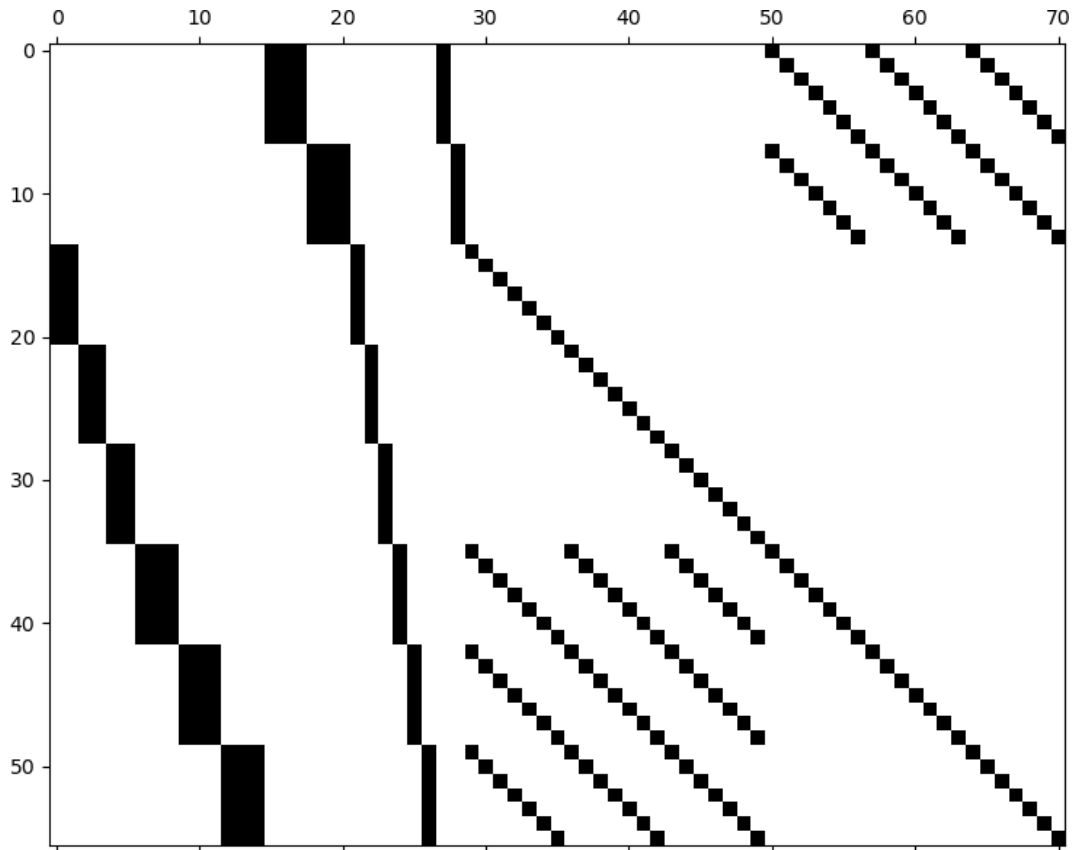


FIGURE 3.1: Nonzero elements of jacobian, for network with $I=2, O=2, W=3, D=2, N=7$

Chapter 4

Tests

This chapter discusses several tests to compare the newly proposed method with backpropagation. Each test will be described in detail. The reference code to which we compare our experiments is the backpropagation method of Tenserflow.

The code was executed on ...

4.1 The First Topic of this Chapter

4.1.1 Item 1

Sub-item 1

Sub-item 2

4.1.2 Item 2

4.2 The Second Topic

4.3 Conclusion

Chapter 5

Conclusion

The final chapter contains the overall conclusion. It also contains suggestions for future work and industrial applications.

Appendices

Appendix A

Source code

This appendix contains the source code of the experiments.

A.1 First experiment source

```
N = 21;
xin = linspace(0,1,21);
y = -sin(.8*pi*x);

w1 = sdpvar(3,1);
b1 = sdpvar(3,1);
x1 = sdpvar(3,N);
w2 = sdpvar(3,3,'full');
b2 = sdpvar(3,1);
x2 = sdpvar(3,N);
w3 = sdpvar(3,1);
b3 = sdpvar(1,1);

assign(w1,2*rand(3,1)-1);
assign(b1,2*rand(3,1)-1);
assign(x1,tansig(value(w1*xin+repmat(b1,1,N))));
assign(w2,2*rand(3,3)-1);
assign(b2,2*rand(3,1)-1);
assign(x2,tansig(value(w2*x1 +repmat(b2,1,N))));
assign(w3,2*rand(3,1)-1);
assign(b3,2*rand(1,1)-1);

res = w3'*x2 + b3 - y;
obj = res*res';
%
f1 = (x1-(w1*xin+repmat(b1,1,N)));
f2 = (x2-(w2*x1 +repmat(b2,1,N)));
% con = [f1 >= 0; x1 >= 0; f1.*x1 <= 0;
%         f2 >= 0; x2 >= 0; f2.*x2 <= 0];
% con = [x1 == max(w1*xin+repmat(b1,1,N),0);
%         x2 == max(w2*x1 +repmat(b2,1,N),0)];
con = [x1 == tansig(w1*xin+repmat(b1,1,N));
       x2 == tansig(w2*x1 +repmat(b2,1,N))];
```

A. SOURCE CODE

```
ops = sdpsettings('usex0',1);
ops.fmincon.MaxFunEvals = 20000;
ops.fmincon.MaxIter = 200;

optimize(con,obj,ops);

%%
x1s = tansig(value(w1)*xin+value(b1));
x2s = tansig(value(w2)*x1s+value(b2));
ys = value(w3)'*x2s+value(b3);

hold off
plot(xin,y,'Linewidth',3);
hold on
plot(xin,ys,'g--','Linewidth',4);
```

A.1.1 Second experiment source

```
clear
N = 21;W = 10;
xin = linspace(0,1,21);
y = -sin(.8*pi*xin);

w1 = sdpvar(W,1);
b1 = sdpvar(W,1);
x1 = sdpvar(W,N);
w3 = sdpvar(W,1);
b3 = sdpvar(1,1);

assign(w1,2*rand(W,1)-1);
assign(b1,2*rand(W,1)-1);
assign(x1,poslin(value(w1*xin+repmat(b1,1,N))));
assign(w3,2*rand(W,1)-1);
assign(b3,2*rand(1,1)-1);

res = w3'*x1 + b3 - y;
obj = res*res';
%
f1 = (x1-(w1*xin+repmat(b1,1,N)));
%f2 = (x2-(w2*x1 +repmat(b2,1,N)));
con = [f1 >= 0; x1 >= 0; f1.*x1 <= 0];
%
f2 >= 0; x2 >= 0; f2.*x2 <= 0];
% con = [x1 == max(w1*xin+repmat(b1,1,N),0);
%
x2 == max(w2*x1 +repmat(b2,1,N),0)];
% con = [x1 == tansig(w1*xin+repmat(b1,1,N));
%
x2 == tansig(w2*x1 +repmat(b2,1,N))];
ops = sdpsettings('usex0',1);
ops.fmincon.MaxFunEvals = 20000;
ops.fmincon.MaxIter = 200;

optimize(con,obj,ops);

%%
x1s = poslin(value(w1)*xin+value(b1));
```

```
%x2s = tansig(value(w2)*x1s+value(b2));  
ys = value(w3)*x1s+value(b3);  
  
hold off  
plot(xin,y,'Linewidth',3);  
hold on  
plot(xin,ys,'g--','Linewidth',4);
```

