Lab1
Optimization of a neural network

Carlos López Roa
me@mr3m.me

October 25, 2015

Abstract

In this report we describe the methods to optimize the parameters of a given neural network structure in order to correctly predict (classify) the labels of a four dimensional random generated vector. The true labels exhibit a nonlinear hidden relation which the neural network must *learn*. This is an example of supervised learning and the methods used are those of a nonlinear unconstraint optimization problem.

1 Problem

Given p random vectors $x \in \mathbb{R}^4$, $y \in \{0, 1\}$ and the structure of the neural network in figure 1,

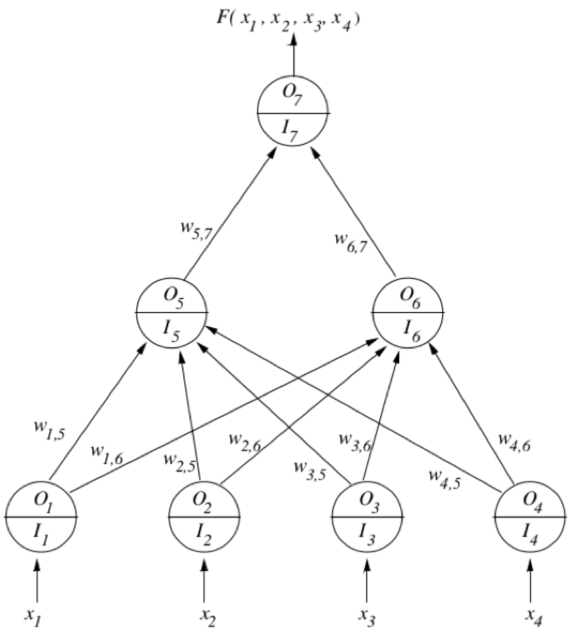

Figure 1: Given neural network structure. It consists in four input nodes, two hidden nodes and one exit node. With ten weights and cero bias parameters. In each node the activation function is a sigmoid function.

we can write the output function as

$$F(x) = \sigma_1(\sigma_{21}(\sigma_{31}(x) \cdot I_{1,4} \cdot \hat{w}) \cdot I_9 \cdot \hat{w} + \sigma_{22}(\sigma_{32}(x) \cdot I_{5,8} \cdot \hat{w}) \cdot I_{10} \cdot \hat{w}) \tag{1}$$

with

$$\sigma_i = \frac{1}{1 + e^{-x}} \tag{2}$$

the sigmoid function[1]

$$\hat{w}^T = (w_{1,5}, w_{2,5}, w_{3,5}, w_{4,5}, w_{1,6}, w_{2,6}, w_{3,6}, w_{4,6}, w_{5,7}, w_{6,7}) \tag{3}$$

the ordered weight vector[2], and $I_{i,j}$[3] is a matrix of dimension $(4, 10)$ which entries from column i to j are an identity matrix and the other entries are zero e.g.

$$I_{5,8} = \left.\begin{pmatrix} 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix}\right| \left.\begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix}\right| \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \end{pmatrix} \tag{4}$$

and I_j[4] is a vector of dimension (10) which j-th entrie is one and the others are zero, e.g.

$$I_9^T = (0, 0, \cdots, 0, 1, 0)\,. \tag{5}$$

We want to find a set of $\hat{w}^*$ such that

$$F(x) \to y \quad \forall\, x \in \mathbb{R}^4. \tag{6}$$

Thus we can define the optimization[5] problem to be solved:

[1] Here the subscript i labels the correspondant sigmoid function

[2] It is more convenient to treat w as a vector rather than an incomplete matrix or another extructure such as a dictionary

[3] This complicated matrix is constructed in such way in order that the product $I_{i,j} \cdot \hat{w}$ is a $(j - i + 1)$ vector with the entries of $\hat{w}$ from i to j that is a matrix which selects the desired entries of $\hat{w}$

[4] Similarly this vector selects the j-th entrie of $\hat{w}$

[5] Despite the fact that there exists several particular methods to solve neural networks (e.g. Backpropagation) we chose to use nonlinear optimization of a cost function in order to put in practice the concepts of this class

Problem 1 [6] *Find $\hat{w}^*$ such that*

$$\min_{\hat{w}} \sum_{i=1}^{p} d(F_i(x;\hat{w}) - y_i) \tag{7}$$

with $d(\cdot)$ a cost function[7].

Since $y \in \{0,1\}$ we may ask d to have this desired properties:

Let $e = F(x) - y$ then we ask for d:

$$\lim_{x\to -1} d(e) = \lim_{x\to 1} d(e) = +\infty$$

and

$$f(0) = 0 = \arg\min_{e} d(e)$$

and since we need to find the $\nabla d(e)$ we ask d to be differentiable thus continuous.

With this in mind we propose the distance function

$$d(e) = \frac{1}{1+e} + \frac{1}{1-e} - 2 \tag{8}$$

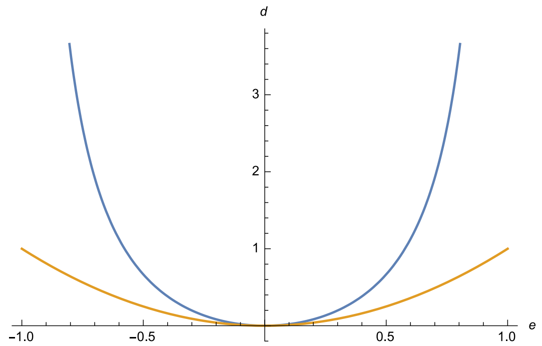

Figure 2: Graph of the function in equation 8 (blue) in comparation with the squared euclidean distance (SED), (orange). The proposed function bounded in the interval (-1,1) penalices the maximum error (± 1) with infinite cost, the SED is more permisive with big differences. It holds the properties asked and is differentiable in $(-1,1)$

Now, we propose to do gradient descent with a non linear solver in order to find the global minima. Thus we need to find the gradient explicitly:

$$\begin{aligned}
\nabla_{\hat{w}_i} d(e) =& \frac{4e}{(e^2-1)^2}\sigma_1(\cdot)(1-\sigma_1(\cdot)) \\
& \sigma_{21}(\cdot)(1-\sigma_{21}(\cdot)) \cdot I_9 \cdot \hat{w} \\
& \sigma_{31}(I_i \cdot x) \\
&& \text{for } i = 1,2,3,4 \\
\nabla_{\hat{w}_i} d(e) =& \frac{4e}{(e^2-1)^2}\sigma_1(\cdot)(1-\sigma_1(\cdot)) \\
& \sigma_{22}(\cdot)(1-\sigma_{22}(\cdot)) \cdot I_{10} \cdot \hat{w} \\
& \sigma_{32}(I_{i-4} \cdot x) \\
&& \text{for } i = 5,6,7,8 \\
\nabla_{\hat{w}_i} d(e) =& \frac{4e}{(e^2-1)^2}\sigma_1(\cdot)(1-\sigma_1(\cdot)) \\
& \sigma_{2(i-8)}(\cdot) \\
&& \text{for } i = 9,10
\end{aligned} \tag{9}$$

Recall: $\nabla_x \sigma(g(x)) = \sigma(g(x))(1-\sigma(g(x)))g'(x)$.

After finding $\hat{w}^*$ solution of the problem (7) and using it in expression (1) we need to apply the threshold function as follows:

$$\hat{F}(x) = \begin{cases} 1 & F(x) \geq 0.5 \\ 0 & F(x) < 0.5 \end{cases} \tag{10}$$

Then we can use expressions (7) for the objective function, (9) for the gradient and (10) for the predicted value, to find the solution of the problem.

2 Implementation

The random values were supplied using the provided binary executable `genxndat` with seed `26071991`

With this in mind we chose to use the non-linear unconstrained optimization solver of `MATLAB 7.12.0.635 (R2011a)` to solve the problem. The code used is as follows in the section 5

3 Tests and results

To determine the error of prediction we first want to obtain the optimal training set size. We will evaluate the sum of the number of false positives plus the number of false negatives and divide by the size of the training set. In this particular case this reduces to:

$$\begin{aligned}
\epsilon &= \frac{1}{p}\sum_{i=1}^{p} |\hat{e}_i| \\
&= \frac{1}{p}\sum_{i=1}^{p} |\hat{F}_i(x) - y_i|
\end{aligned} \tag{11}$$

We evaluated[8] $\bar{\epsilon} \pm \frac{\sigma_\epsilon}{2}$ for a 10 batch test in each

[6] Non linear unconstrained neural network optimization

[7] At first we tried with the squared euclidian distance function in a gradient descent with momentum implementation, finding convergence to $F(x) = 0.5$ which is not desired. Thus we designed the funcion described above

[8] That is, the mean of the error described with incertainty equal to the standard deviation of the error

order of magnitude of p from 1 to 1,000[9]. Exhibiting the results in figure 3.

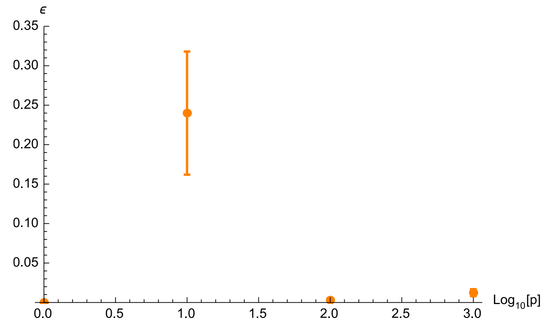

Figure 3: Normalized absolute error for diferent training set sizes. For sample size 10^0 the error es 0 in 10 tries in a row. While with a sample size of 10^1 the error grows to $24\% \pm 7.8$. With 10^2 training observations we get $0.30\% \pm 0.24$ but with 10^3 the error grows again to $1.24\% \pm 0.44$.

Having observed this, we decided to train the weights with a sample size of 100 observations and test the accuracy with the 1,000 sample size. Thus instead of using the Pareto approach[10] we adopt a more ambitious goal of training with just 10% of the set.

3.1 Training

The output of the program after 90 iterations:

Initial random weights and final weights

$$\hat{w}_0 = \begin{pmatrix} 0,352 \\ -0,144 \\ -0,260 \\ 0,911 \\ 2,306 \\ 0,847 \\ 0,642 \\ 0,004 \\ 0,129 \\ 0,581 \end{pmatrix} \quad \hat{w}^* = \begin{pmatrix} -0,750 \\ -0,940 \\ -0,986 \\ -0,892 \\ 4803,786 \\ 4565,815 \\ 4122,838 \\ 4515,028 \\ -16840,135 \\ 1662,110 \end{pmatrix} \tag{12}$$

Convergence graph in figure 4.

The error rate is equal to $\epsilon = 0\%$

3.2 Testing

We know that the *true* relation between input vectors x and labels y is described by

$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{4} x_i > 2 \\ 0 & \text{if } \sum_{i=1}^{4} x_i \leq 2 \end{cases} \tag{13}$$

[9]After 10^4 observations the computation takes too much time.

[10]Train with 80% of the set and test with 20%

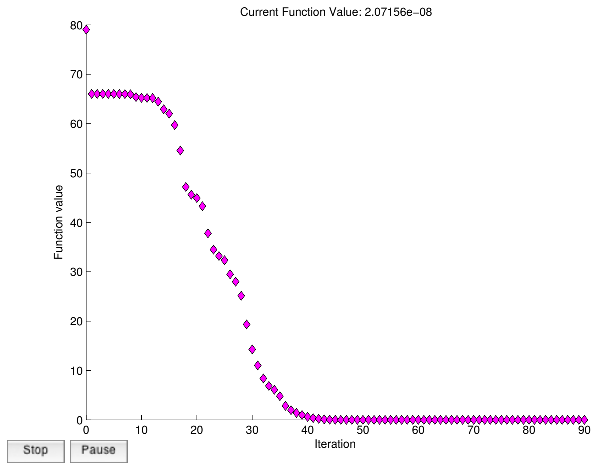

Figure 4: Value of the objetive function (7) through the iterations.

With this in mind we computed the

$$\sum_{i=1}^{4} x_i$$

and compared

$$\hat{e} = \hat{F}_i(x) - y_i$$

Taking the weights learned $\hat{w}^*$ but with the 10^3 test set.

The prediction got an error rate of $\epsilon = 2.9\%$. If we graph the pairs

$$\left(\sum_{i=1}^{4} x_i, \hat{e} \right)$$

we get figure 5

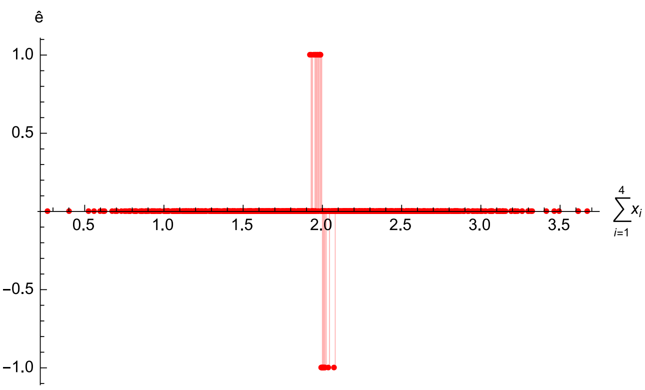

Figure 5: Graphical representation of the points that get missclassified in the test set. we observe clearly that they are near the change point.

In fact if we take the missclassified points minus two and see their ditribution we observe figure 6

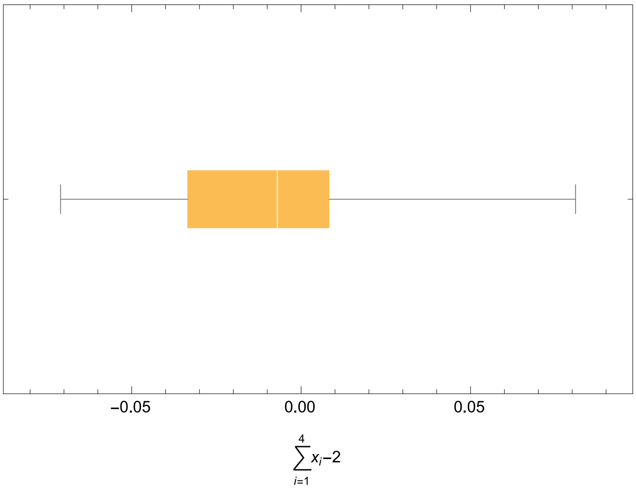

Figure 6: Taking the distance of missclassified points to the change point we observe that they are at most at 0.081 distance in median at -0.007 and in average at -0.0103, that is, very close

4 Conclusions

1. Giving a special treatment to the problem, we were able to pose an unconstrained non linear optimization problem and we found a solution using a proprietary solver
2. Evaluating the test sample size, we were able to reduce computation time and over-fitting issues.
3. The training set achieved 0% error while the test set achieved 2.9% even though the training set is just 10% portion of the test set.
4. The false positive and false positive points correspond entirely of the region around the transition of label. That is, the neural network encounters problem classifying points near the transition point.

5 Implementation Code

```
cd
/Users/poincare/Dropbox/Courseware_/Op/
Lab/Lab001/ds1 %Change directory
clear all %Clear workspace
clc %Clear prompt

function f = myfun(wi) %Cost Function
x = importdata('data_x.txt'); %obs x
y = importdata('data_y.txt'); %label y
d = %error function
(sigmf(sigmf(sigmf(x,[1,0])*([eye(4)
zeros(4,6)]*wi),[1,0])*([zeros(1,8) 1
0]*wi) +
sigmf(sigmf(x,[1,0])*([zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0])*([zeros(1,9)
1]*wi),[1,0])- y);
f = %cost function
sum((1./(1+d)+1./(1-d))-2);
g = %Gradient
[((4*d)./((d.^2-1).^2))'*(sigmf(sigmf(
sigmf(x,[1,0])*([eye(4)
zeros(4,6)]*wi),[1,0])*([zeros(1,8) 1
0]*wi) +
sigmf(sigmf(x,[1,0])*([zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0])*([zeros(1,9)
1]*wi),[1,0])*(1-sigmf(sigmf(sigmf(x,[1,
0])*([eye(4)
zeros(4,6)]*wi),[1,0])*([zeros(1,8) 1
0]*wi) +
sigmf(sigmf(x,[1,0])*([zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0])*([zeros(1,9)
1]*wi),[1,0]))'*sigmf(sigmf(x,[1,0])*([
eye(4)
zeros(4,6)]*wi),[1,0])*((1-sigmf(sigmf(x
,[1,0])*([eye(4)
zeros(4,6)]*wi),[1,0]))*([zeros(1,8) 1
0]*wi))'*sigmf(x*[1 0 0 0]',[1,0]));
((4*d)./((d.^2-1).^2))'*(sigmf(sigmf(
sigmf(x,[1,0])*([eye(4)
zeros(4,6)]*wi),[1,0])*([zeros(1,8) 1
0]*wi) +
sigmf(sigmf(x,[1,0])*([zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0])*([zeros(1,9)
1]*wi),[1,0])*(1-sigmf(sigmf(sigmf(x,[1,
0])*([eye(4)
zeros(4,6)]*wi),[1,0])*([zeros(1,8) 1
0]*wi) +
sigmf(sigmf(x,[1,0])*([zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0])*([zeros(1,9)
1]*wi),[1,0]))'*sigmf(sigmf(x,[1,0])*([
eye(4)
zeros(4,6)]*wi),[1,0])*((1-sigmf(sigmf(x
,[1,0])*([eye(4)
zeros(4,6)]*wi),[1,0]))*([zeros(1,8) 1
0]*wi))'*sigmf(x*[0 1 0 0]',[1,0])) ;
((4*d)./((d.^2-1).^2))'*(sigmf(sigmf(
sigmf(x,[1,0])*([eye(4)
zeros(4,6)]*wi),[1,0])*([zeros(1,8) 1
0]*wi) +
sigmf(sigmf(x,[1,0])*([zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0])*([zeros(1,9)
1]*wi),[1,0])*(1-sigmf(sigmf(sigmf(x,[1,
0])*([eye(4)
zeros(4,6)]*wi),[1,0])*([zeros(1,8) 1
0]*wi) +
sigmf(sigmf(x,[1,0])*([zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0])*([zeros(1,9)
1]*wi),[1,0]))'*sigmf(sigmf(x,[1,0])*([
eye(4)
zeros(4,6)]*wi),[1,0])*((1-sigmf(sigmf(x
,[1,0])*([eye(4)
zeros(4,6)]*wi),[1,0]))*([zeros(1,8) 1
0]*wi))'*sigmf(x*[0 0 1 0]',[1,0])) ;
((4*d)./((d.^2-1).^2))'*(sigmf(sigmf(
sigmf(x,[1,0])*([eye(4)
zeros(4,6)]*wi),[1,0])*([zeros(1,8) 1
0]*wi) +
sigmf(sigmf(x,[1,0])*([zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0])*([zeros(1,9)
1]*wi),[1,0])*(1-sigmf(sigmf(sigmf(x,[1,
0])*([eye(4)
zeros(4,6)]*wi),[1,0])*([zeros(1,8) 1
0]*wi) +
sigmf(sigmf(x,[1,0])*([zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0])*([zeros(1,9)
1]*wi),[1,0]))'*sigmf(sigmf(x,[1,0])*([
eye(4)
zeros(4,6)]*wi),[1,0])*((1-sigmf(sigmf(x
,[1,0])*([eye(4)
zeros(4,6)]*wi),[1,0]))*([zeros(1,8) 1
0]*wi))'*sigmf(x*[0 0 0 1]',[1,0])) ;
((4*d)./((d.^2-1).^2))'*(sigmf(sigmf(
sigmf(x,[1,0])*([eye(4)
zeros(4,6)]*wi),[1,0])*([zeros(1,8) 1
0]*wi) +
sigmf(sigmf(x,[1,0])*([zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0])*([zeros(1,9)
1]*wi),[1,0])*(1-sigmf(sigmf(sigmf(x,[1,
0])*([eye(4)
zeros(4,6)]*wi),[1,0])*([zeros(1,8) 1
0]*wi) +
sigmf(sigmf(x,[1,0])*([zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0])*([zeros(1,9)
1]*wi),[1,0]))'*sigmf(sigmf(x,[1,0])*([
zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0])*((1-sigmf(sigmf(x
,[1,0])*([zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0]))*([zeros(1,9)
1]*wi))'*sigmf(x*[1 0 0 0]',[1,0])) ;
((4*d)./((d.^2-1).^2))'*(sigmf(sigmf(
sigmf(x,[1,0])*([eye(4)
zeros(4,6)]*wi),[1,0])*([zeros(1,8) 1
0]*wi) +
sigmf(sigmf(x,[1,0])*([zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0])*([zeros(1,9)
1]*wi),[1,0])*(1-sigmf(sigmf(sigmf(x,[1,
0])*([eye(4)
zeros(4,6)]*wi),[1,0])*([zeros(1,8) 1
0]*wi) +
sigmf(sigmf(x,[1,0])*([zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0])*([zeros(1,9)
1]*wi),[1,0]))'*sigmf(sigmf(x,[1,0])*([
zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0])*((1-sigmf(sigmf(x
```

```
,[1,0])*([zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0]))*([zeros(1,9)
1]*wi))'*sigmf(x*[0 1 0 0]',[1,0])) ;
((4*d)./((d.^2-1).^2))'*(sigmf(sigmf(
sigmf(x,[1,0])*([eye(4)
zeros(4,6)]*wi),[1,0])*([zeros(1,8) 1
0]*wi) +
sigmf(sigmf(x,[1,0])*([zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0])*([zeros(1,9)
1]*wi),[1,0])*(1-sigmf(sigmf(sigmf(x,[1,
0])*([eye(4)
zeros(4,6)]*wi),[1,0])*([zeros(1,8) 1
0]*wi) +
sigmf(sigmf(x,[1,0])*([zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0])*([zeros(1,9)
1]*wi),[1,0]))'*sigmf(sigmf(x,[1,0])*([
zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0])*((1-sigmf(sigmf(x
,[1,0])*([zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0]))*([zeros(1,9)
1]*wi))'*sigmf(x*[0 0 1 0]',[1,0])) ;
((4*d)./((d.^2-1).^2))'*(sigmf(sigmf(
sigmf(x,[1,0])*([eye(4)
zeros(4,6)]*wi),[1,0])*([zeros(1,8) 1
0]*wi) +
sigmf(sigmf(x,[1,0])*([zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0])*([zeros(1,9)
1]*wi),[1,0])*(1-sigmf(sigmf(sigmf(x,[1,
0])*([eye(4)
zeros(4,6)]*wi),[1,0])*([zeros(1,8) 1
0]*wi) +
sigmf(sigmf(x,[1,0])*([zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0])*([zeros(1,9)
1]*wi),[1,0]))'*sigmf(sigmf(x,[1,0])*([
zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0])*((1-sigmf(sigmf(x
,[1,0])*([zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0]))*([zeros(1,9)
1]*wi))'*sigmf(x*[0 0 0 1]',[1,0])) ;
((4*d)./((d.^2-1).^2))'*(sigmf(sigmf(
sigmf(x,[1,0])*([eye(4)
zeros(4,6)]*wi),[1,0])*([zeros(1,8) 1
0]*wi) +
sigmf(sigmf(x,[1,0])*([zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0])*([zeros(1,9)
1]*wi),[1,0])*(1-sigmf(sigmf(sigmf(x,[1,
0])*([eye(4)
zeros(4,6)]*wi),[1,0])*([zeros(1,8) 1
0]*wi) +
sigmf(sigmf(x,[1,0])*([zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0])*([zeros(1,9)
1]*wi),[1,0]))'*sigmf(sigmf(x,[1,0])*([
eye(4) zeros(4,6)]*wi),[1,0])) ;
((4*d)./((d.^2-1).^2))'*(sigmf(sigmf(
sigmf(x,[1,0])*([eye(4)
zeros(4,6)]*wi),[1,0])*([zeros(1,8) 1
0]*wi) +
sigmf(sigmf(x,[1,0])*([zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0])*([zeros(1,9)
1]*wi),[1,0])*(1-sigmf(sigmf(sigmf(x,[1,
0])*([eye(4)
zeros(4,6)]*wi),[1,0])*([zeros(1,8) 1
0]*wi) +
sigmf(sigmf(x,[1,0])*([zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0])*([zeros(1,9)
1]*wi),[1,0]))'*sigmf(sigmf(x,[1,0])*([
zeros(4,4) eye(4)
zeros(4,2)]*wi),[1,0]))]; end

w0= randn(10,1); %initial random w
opt = %options
optimset('GradObj','on');
[wi,fval] = %optimizer call
fminunc(@myfun,w0)
```