

# 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,

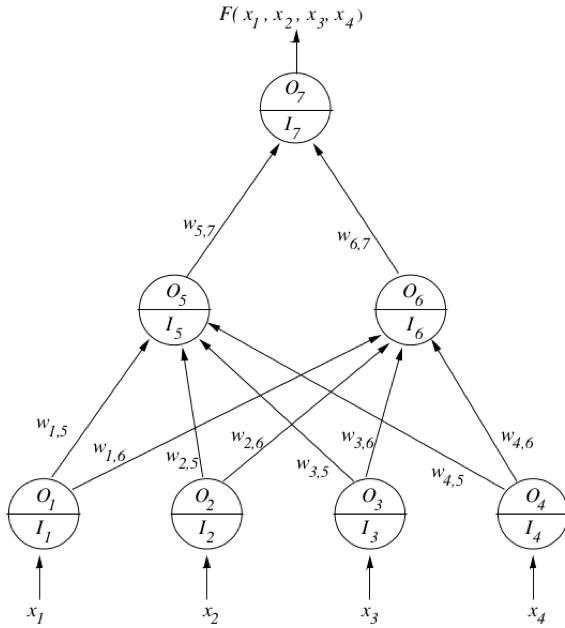


Figure 1: Given neural network structure. It consists in four input nodes, two hidden nodes and one exit node. With ten weights and zero bias parameters. In each node the activation function is a sigmoid function.

we can write the output function as

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

with

$$\sigma_i = \frac{1}{1 + e^{-x}} \quad (2)$$

the sigmoid function<sup>1</sup>

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

the ordered weight vector<sup>2</sup>, and  $I_{i,j}$ <sup>3</sup> 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.

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

and  $I_j$ <sup>4</sup> is a vector of dimension  $(10)$  which  $j$ -th entry is one and the others are zero, e.g.

$$I_9^T = (0, 0, \dots, 0, 1, 0). \quad (5)$$

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

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

Thus we can define the optimization<sup>5</sup> problem to be solved:

<sup>1</sup>Here the subscript  $i$  labels the correspondent sigmoid function

<sup>2</sup>It is more convenient to treat  $w$  as a vector rather than an incomplete matrix or another exstructure such as a dictionary

<sup>3</sup>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  $\hat{w}$

<sup>4</sup>Similarly this vector selects the  $j$ -th entrie of  $\hat{w}$

<sup>5</sup>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** <sup>6</sup> Find  $\hat{w}^*$  such that

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

with  $d(\cdot)$  a cost function<sup>7</sup>.

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

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

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

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

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

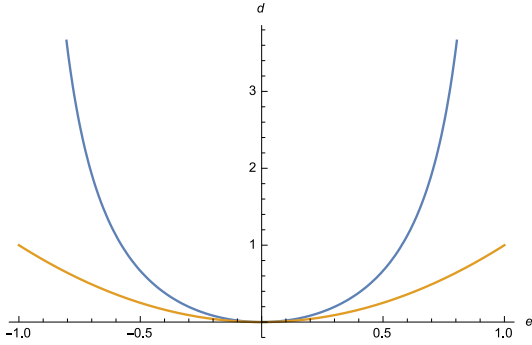


Figure 2: Graph of the function in equation 8 (blue) in comparison with the squared euclidean distance (SED), (orange). The proposed function bounded in the interval  $(-1, 1)$  penalizes the maximum error  $(\pm 1)$  with infinite cost, the SED is more permissive with big differences. It holds the properties 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:

<sup>6</sup>Non linear unconstrained neural network optimization

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

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

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:

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

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:

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

We evaluated<sup>8</sup>  $\bar{\epsilon} \pm \frac{\sigma_{\epsilon}}{2}$  for a 10 batch test in each

<sup>8</sup>That is, the mean of the error described with uncertainty equal to the standard deviation of the error

order of magnitude of  $p$  from 1 to 1,000<sup>9</sup>. Exhibiting the results in figure 3.

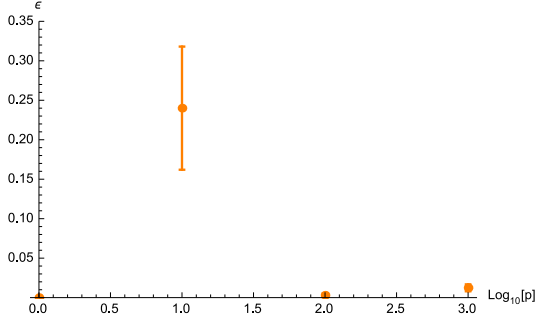


Figure 3: Normalized absolute error for diferent training set sizes. For sample size  $10^0$  the error es 0 in 10 tries in a row. While with a sample size of  $10^1$  the error grows to  $24\% \pm 7.8$ . With  $10^2$  training observations we get  $0.30\% \pm 0.24$  but with  $10^3$  the error grows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

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

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

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

<sup>9</sup>After  $10^4$  observations the computation takes too much time.

<sup>10</sup>Train with 80% of the set and test with 20%

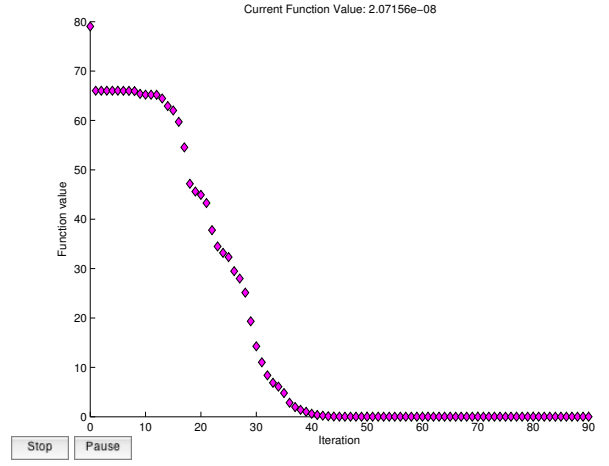


Figure 4: Value of the objective function (7) through the iterations.

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

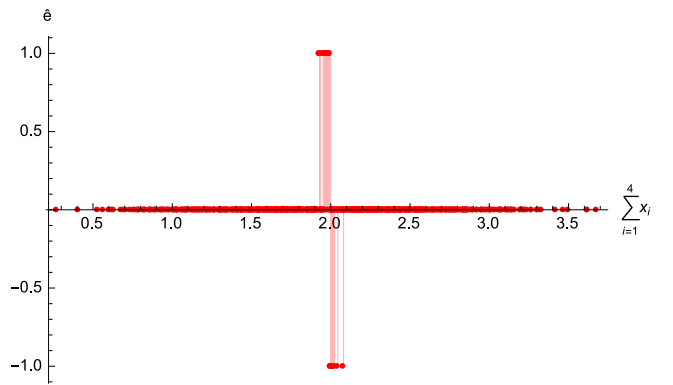


Figure 5: Graphical representation of the points that get misclassified in the test set. we observe clearly that they are near the change point.

In fact if we take the misclassified points minus two and see their distribution we observe figure 6

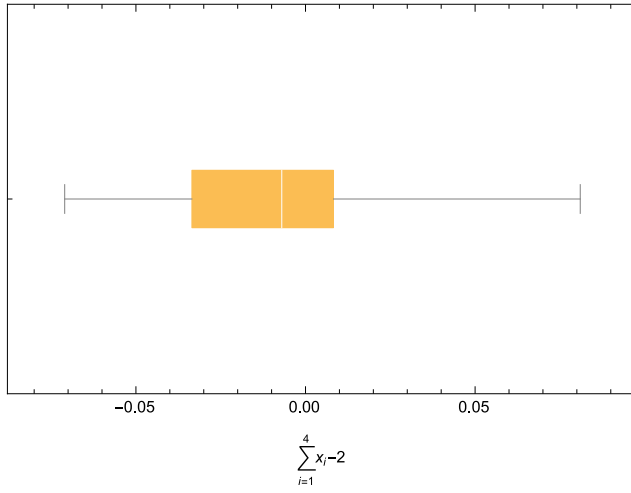


Figure 6: Taking the distance of missclassified points to the change point we observe that they are at most at 0.081 distance in median at  $-0.007$  and in average at  $-0.0103$ , that is, very close

## 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

```

1 cd
2 /Users/poincare/Dropbox/Courseware_/Op/
3 Lab/Lab001/ds1 %Change directory
4 clear all %Clear workspace
5 clc %Clear prompt
6
7 function f = myfun(wi) %Cost Function
8 x = importdata('data_x.txt'); %obs x
9 y = importdata('data_y.txt'); %label y
10 d = %error function
11 (sigmf(sigmoid(sigmoid(x,[1,0])*(eye(4)
12 zeros(4,6])*wi),[1,0])*(zeros(1,8) 1
13 0)*wi) +
14 sigmf(sigmoid(x,[1,0])*(zeros(4,4) eye(4)
15 zeros(4,2])*wi),[1,0])*(zeros(1,9)
16 1)*wi],[1,0]) - y);
17 f = %cost function
18 sum((1./(1+d)+1./(1-d))-2);
19 g = %Gradient
20 [((4*d)./((d.^2-1).^2))*sigmoid(sigmoid(
21 sigmf(x,[1,0])*(eye(4)
22 zeros(4,6])*wi),[1,0])*(zeros(1,8) 1
23 0)*wi) +
24 sigmf(sigmoid(x,[1,0])*(zeros(4,4) eye(4)
25 zeros(4,2])*wi),[1,0])*(zeros(1,9)
26 1)*wi],[1,0])*(1-sigmoid(sigmoid(sigmoid(x,[1,
27 0])*(eye(4)
28 zeros(4,6])*wi),[1,0])*(zeros(1,8) 1
29 0)*wi) +
30 sigmf(sigmoid(x,[1,0])*(zeros(4,4) eye(4)
31 zeros(4,2])*wi),[1,0])*(zeros(1,9)
32 1)*wi],[1,0]))'*sigmoid(sigmoid(x,[1,0])*(eye(4)
33 zeros(4,6])*wi),[1,0])*((1-sigmoid(sigmoid(x
34 ,[1,0])*(eye(4)
35 zeros(4,6])*wi),[1,0])*(zeros(1,8) 1
36 0)*wi))*sigmoid(x*[1 0 0 0]',[1,0]));
37 ((4*d)./((d.^2-1).^2))*sigmoid(sigmoid(
38 sigmf(x,[1,0])*(eye(4)
39 zeros(4,6])*wi),[1,0])*(zeros(1,8) 1
40 0)*wi) +
41 sigmf(sigmoid(x,[1,0])*(zeros(4,4) eye(4)
42 zeros(4,2])*wi),[1,0])*(zeros(1,9)
43 1)*wi],[1,0])*(1-sigmoid(sigmoid(sigmoid(x,[1,
44 0])*(eye(4)
45 zeros(4,6])*wi),[1,0])*(zeros(1,8) 1
46 0)*wi) +
47 sigmf(sigmoid(x,[1,0])*(zeros(4,4) eye(4)
48 zeros(4,2])*wi),[1,0])*(zeros(1,9)
49 1)*wi],[1,0]))'*sigmoid(sigmoid(x,[1,0])*(eye(4)
50 zeros(4,6])*wi),[1,0])*((1-sigmoid(sigmoid(x
51 ,[1,0])*(eye(4)
52 zeros(4,6])*wi),[1,0])*(zeros(1,8) 1
53 0)*wi))*sigmoid(x*[0 1 0 0]',[1,0]));
54 ((4*d)./((d.^2-1).^2))*sigmoid(sigmoid(
55 sigmf(x,[1,0])*(eye(4)
56 zeros(4,6])*wi),[1,0])*(zeros(1,8) 1
57 0)*wi) +
58 sigmf(sigmoid(x,[1,0])*(zeros(4,4) eye(4)
59 zeros(4,2])*wi),[1,0])*(zeros(1,9)
60 1)*wi],[1,0])*(1-sigmoid(sigmoid(sigmoid(x,[1,
61 0])*(eye(4)
62 zeros(4,6])*wi),[1,0])*(zeros(1,8) 1
63 0)*wi) +
64 sigmf(sigmoid(x,[1,0])*(zeros(4,4) eye(4)
65 zeros(4,2])*wi),[1,0])*(zeros(1,9)
66 1)*wi],[1,0]))'*sigmoid(sigmoid(x,[1,0])*(eye(4)
67 zeros(4,6])*wi),[1,0])*((1-sigmoid(sigmoid(x
68 ,[1,0])*(eye(4)
69 zeros(4,6])*wi),[1,0])*(zeros(1,8) 1
70 0)*wi) +
71 sigmf(sigmoid(x,[1,0])*(zeros(4,4) eye(4)
72 zeros(4,2])*wi),[1,0])*(zeros(1,9)
73 1)*wi],[1,0]))'*sigmoid(sigmoid(x,[1,0])*(eye(4)
74 zeros(4,6])*wi),[1,0])*((1-sigmoid(sigmoid(x
75 ,[1,0])*(eye(4)
76 zeros(4,6])*wi),[1,0])*(zeros(1,8) 1
77 0)*wi) +
78 sigmf(sigmoid(x,[1,0])*(zeros(4,4) eye(4)
79 zeros(4,2])*wi),[1,0])*(zeros(1,9)
80 1)*wi],[1,0])*(1-sigmoid(sigmoid(sigmoid(x,[1,
81 0])*(eye(4)
82 zeros(4,6])*wi),[1,0])*(zeros(1,8) 1
83 0)*wi) +
84 sigmf(sigmoid(x,[1,0])*(zeros(4,4) eye(4)
85 zeros(4,2])*wi),[1,0])*(zeros(1,9)
86 1)*wi],[1,0]))'*sigmoid(sigmoid(x,[1,0])*(eye(4)
87 zeros(4,6])*wi),[1,0])*((1-sigmoid(sigmoid(x
88 ,[1,0])*(eye(4)
89 zeros(4,6])*wi),[1,0])*(zeros(1,8) 1
90 0)*wi))*sigmoid(x*[0 0 0 1]',[1,0]));
91 ((4*d)./((d.^2-1).^2))*sigmoid(sigmoid(
92 sigmf(x,[1,0])*(eye(4)
93 zeros(4,6])*wi),[1,0])*(zeros(1,8) 1
94 0)*wi) +
95 sigmf(sigmoid(x,[1,0])*(zeros(4,4) eye(4)
96 zeros(4,2])*wi),[1,0])*(zeros(1,9)
97 1)*wi],[1,0])*(1-sigmoid(sigmoid(sigmoid(x,[1,
98 0])*(eye(4)
99 zeros(4,6])*wi),[1,0])*(zeros(1,8) 1
100 0)*wi) +
101 sigmf(sigmoid(x,[1,0])*(zeros(4,4) eye(4)
102 zeros(4,2])*wi),[1,0])*(zeros(1,9)
103 1)*wi],[1,0]))'*sigmoid(sigmoid(x,[1,0])*(eye(4)
104 zeros(4,6])*wi),[1,0])*((1-sigmoid(sigmoid(x
105 ,[1,0])*(eye(4)
106 zeros(4,2])*wi),[1,0])*(zeros(1,8) 1
107 0)*wi))*sigmoid(x*[1 0 0 0]',[1,0]));
108 ((4*d)./((d.^2-1).^2))*sigmoid(sigmoid(
109 sigmf(x,[1,0])*(eye(4)
110 zeros(4,6])*wi),[1,0])*(zeros(1,8) 1
111 0)*wi) +
112 sigmf(sigmoid(x,[1,0])*(zeros(4,4) eye(4)
113 zeros(4,2])*wi),[1,0])*(zeros(1,9)
114 1)*wi],[1,0])*(1-sigmoid(sigmoid(sigmoid(x,[1,
115 0])*(eye(4)
116 zeros(4,6])*wi),[1,0])*(zeros(1,8) 1
117 0)*wi) +
118 sigmf(sigmoid(x,[1,0])*(zeros(4,4) eye(4)
119 zeros(4,2])*wi),[1,0])*(zeros(1,9)
120 1)*wi],[1,0]))'*sigmoid(sigmoid(x,[1,0])*(eye(4)
121 zeros(4,6])*wi),[1,0])*((1-sigmoid(sigmoid(x
122 ,[1,0])*(eye(4)
123 zeros(4,6])*wi),[1,0])*(zeros(1,8) 1
124 0)*wi))*sigmoid(x*[1 0 0 0]',[1,0]));

```

```

125 , [1, 0]) * ([zeros(4,4) eye(4)
126 zeros(4,2)] * wi), [1, 0]) * ([zeros(1,9)
127 1] * wi))' * sigmf(x*[0 1 0 0]', [1, 0])) ;
128 ((4*d)./((d.^2-1).^2))' * (sigmf(sigmf(
129 sigmf(x, [1, 0]) * ([eye(4)
130 zeros(4,6)] * wi), [1, 0]) * ([zeros(1,8) 1
131 0] * wi) +
132 sigmf(sigmf(x, [1, 0]) * ([zeros(4,4) eye(4)
133 zeros(4,2)] * wi), [1, 0]) * ([zeros(1,9)
134 1] * wi), [1, 0]) * (1 - sigmf(sigmf(sigmf(x, [1,
135 0]) * ([eye(4)
136 zeros(4,6)] * wi), [1, 0]) * ([zeros(1,8) 1
137 0] * wi) +
138 sigmf(sigmf(x, [1, 0]) * ([zeros(4,4) eye(4)
139 zeros(4,2)] * wi), [1, 0]) * ([zeros(1,9)
140 1] * wi), [1, 0]))' * sigmf(sigmf(x, [1, 0]) * ([
141 zeros(4,4) eye(4)
142 zeros(4,2)] * wi), [1, 0]) * ((1 - sigmf(sigmf(x
143 , [1, 0]) * ([zeros(4,4) eye(4)
144 zeros(4,2)] * wi), [1, 0])) * ([zeros(1,9)
145 1] * wi))' * sigmf(x*[0 0 1 0]', [1, 0])) ;
146 ((4*d)./((d.^2-1).^2))' * (sigmf(sigmf(
147 sigmf(x, [1, 0]) * ([eye(4)
148 zeros(4,6)] * wi), [1, 0]) * ([zeros(1,8) 1
149 0] * wi) +
150 sigmf(sigmf(x, [1, 0]) * ([zeros(4,4) eye(4)
151 zeros(4,2)] * wi), [1, 0]) * ([zeros(1,9)
152 1] * wi), [1, 0]) * (1 - sigmf(sigmf(sigmf(x, [1,
153 0]) * ([eye(4)
154 zeros(4,6)] * wi), [1, 0]) * ([zeros(1,8) 1
155 0] * wi) +
156 sigmf(sigmf(x, [1, 0]) * ([zeros(4,4) eye(4)
157 zeros(4,2)] * wi), [1, 0]) * ([zeros(1,9)
158 1] * wi), [1, 0]))' * sigmf(sigmf(x, [1, 0]) * ([
159 zeros(4,4) eye(4)
160 zeros(4,2)] * wi), [1, 0]) * ((1 - sigmf(sigmf(x
161 , [1, 0]) * ([zeros(4,4) eye(4)
162 zeros(4,2)] * wi), [1, 0])) * ([zeros(1,9)
163 1] * wi))' * sigmf(x*[0 0 0 1]', [1, 0])) ;
164 ((4*d)./((d.^2-1).^2))' * (sigmf(sigmf(
165 sigmf(x, [1, 0]) * ([eye(4)
166 zeros(4,6)] * wi), [1, 0]) * ([zeros(1,8) 1
167 0] * wi) +
168 sigmf(sigmf(x, [1, 0]) * ([zeros(4,4) eye(4)
169 zeros(4,2)] * wi), [1, 0]) * ([zeros(1,9)
170 1] * wi), [1, 0]) * (1 - sigmf(sigmf(sigmf(x, [1,
171 0]) * ([eye(4)
172 zeros(4,6)] * wi), [1, 0]) * ([zeros(1,8) 1
173 0] * wi) +
174 sigmf(sigmf(x, [1, 0]) * ([zeros(4,4) eye(4)
175 zeros(4,2)] * wi), [1, 0]) * ([zeros(1,9)
176 1] * wi), [1, 0]))' * sigmf(sigmf(x, [1, 0]) * ([
177 eye(4) zeros(4,6)] * wi), [1, 0])) ;
178 ((4*d)./((d.^2-1).^2))' * (sigmf(sigmf(
179 sigmf(x, [1, 0]) * ([eye(4)
180 zeros(4,6)] * wi), [1, 0]) * ([zeros(1,8) 1
181 0] * wi) +
182 sigmf(sigmf(x, [1, 0]) * ([zeros(4,4) eye(4)
183 zeros(4,2)] * wi), [1, 0]) * ([zeros(1,9)
184 1] * wi), [1, 0]) * (1 - sigmf(sigmf(sigmf(x, [1,
185 0]) * ([eye(4)
186 zeros(4,6)] * wi), [1, 0]) * ([zeros(1,8) 1
187 0] * wi) +
188 sigmf(sigmf(x, [1, 0]) * ([zeros(4,4) eye(4)
189 zeros(4,2)] * wi), [1, 0]) * ([zeros(1,9)
190 1] * wi), [1, 0]))' * sigmf(sigmf(x, [1, 0]) * ([
191 zeros(4,4) eye(4)
192 zeros(4,2)] * wi), [1, 0]))]; end
193
194 w0= randn(10,1); %initial random w
195 opt = %options
196 optimset('GradObj','on');
197 [wi, fval] = %optimizer call
198 fminunc(@myfun, w0)

```