Neuro-Fuzzy Computing
Problem Set 02

Eleftherios P. Loukas 2029 - Panagiotis Nikitakis 1717

November 2018

Problem-01

Use the MATLAB function **fminunc** to minimize $f:R^2 \rightarrow R$ defined as follows:

$$\forall x \in R^2,\ f(x)=(x_1-1)^2 + (x_2-3)^2 - 1.8(x_1-1)(x_2-3).$$

You should write an MATLAB M-file to evaluate both f and ∇f. Specify that you are supplying the gradient ∇f by setting the **GradObj** option to **on** using the **optimset** function. Use the steepest descent algorithm by setting the **LargeScale** option to **off** and the **HessUpdate** option to **steepdesc** using the **optimset** function. Use initial guess $x^{(0)}= [3\ \ -5]^T$. Report the number of iterations required.

Answer:

Please refer to the attached files for the Python code.

The total number of iterations required was **74**.

Problem-02

Suppose that we have the following three reference patterns and their targets:

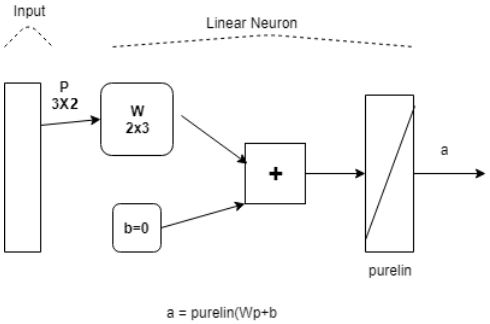
$$\left\{\mathbf{p}_1 = \begin{bmatrix}2\\4\end{bmatrix}, t_1 = \begin{bmatrix}26\end{bmatrix}\right\}, \left\{\mathbf{p}_2 = \begin{bmatrix}4\\2\end{bmatrix}, t_2 = \begin{bmatrix}26\end{bmatrix}\right\}, \left\{\mathbf{p}_3 = \begin{bmatrix}-2\\-2\end{bmatrix}, t_3 = \begin{bmatrix}-26\end{bmatrix}\right\}.$$

The probability of vector $\mathbf{p}_1$ is 0.25, the probability of vector $\mathbf{p}_2$ is 0.25, and the probability of vector $\mathbf{p}_3$ is 0.5.

A. Draw the network diagram for an ADALINE network with no bias that could be trained on these patterns.
B. Sketch the contour plot of the mean square error performance index.
C. Show the optimal decision boundary (for the weights that minimize mean square error), and verify that it separates the patterns into the appropriate categories.
D. Find the maximum stable learning rate for the LMS algorithm. If the target values are changed from 26 and -26 to 2 and -2, how would this change the maximum stable learning rate?
E. Perform one iteration of the LMS algorithm, starting with all weights equal to zero, and presenting input vector $\mathbf{p}_1$. Use a learning rate of α=0.5.

Answer:

A.



B.

Let's find first the MSE index through the slides' equation $F(x) = c - 2x^T h + x^T Rx$ where $c = E[t^2]$, $h = E[tz]$, $R = E[zz^T]$.

$$c = E[t^2] = 25^2 * 0.25 + 26^2(0.25(+(-26^2) * 0.5 = 676$$

$$h = E[tz] = 0.25 * 26 * \begin{vmatrix} 2 \\ 4 \end{vmatrix} + 0.25 * 26 \begin{vmatrix} 4 \\ 2 \end{vmatrix} + 0.5 * (-26) * \begin{vmatrix} -2 \\ -2 \end{vmatrix} = \begin{vmatrix} 65 \\ 65 \end{vmatrix}$$

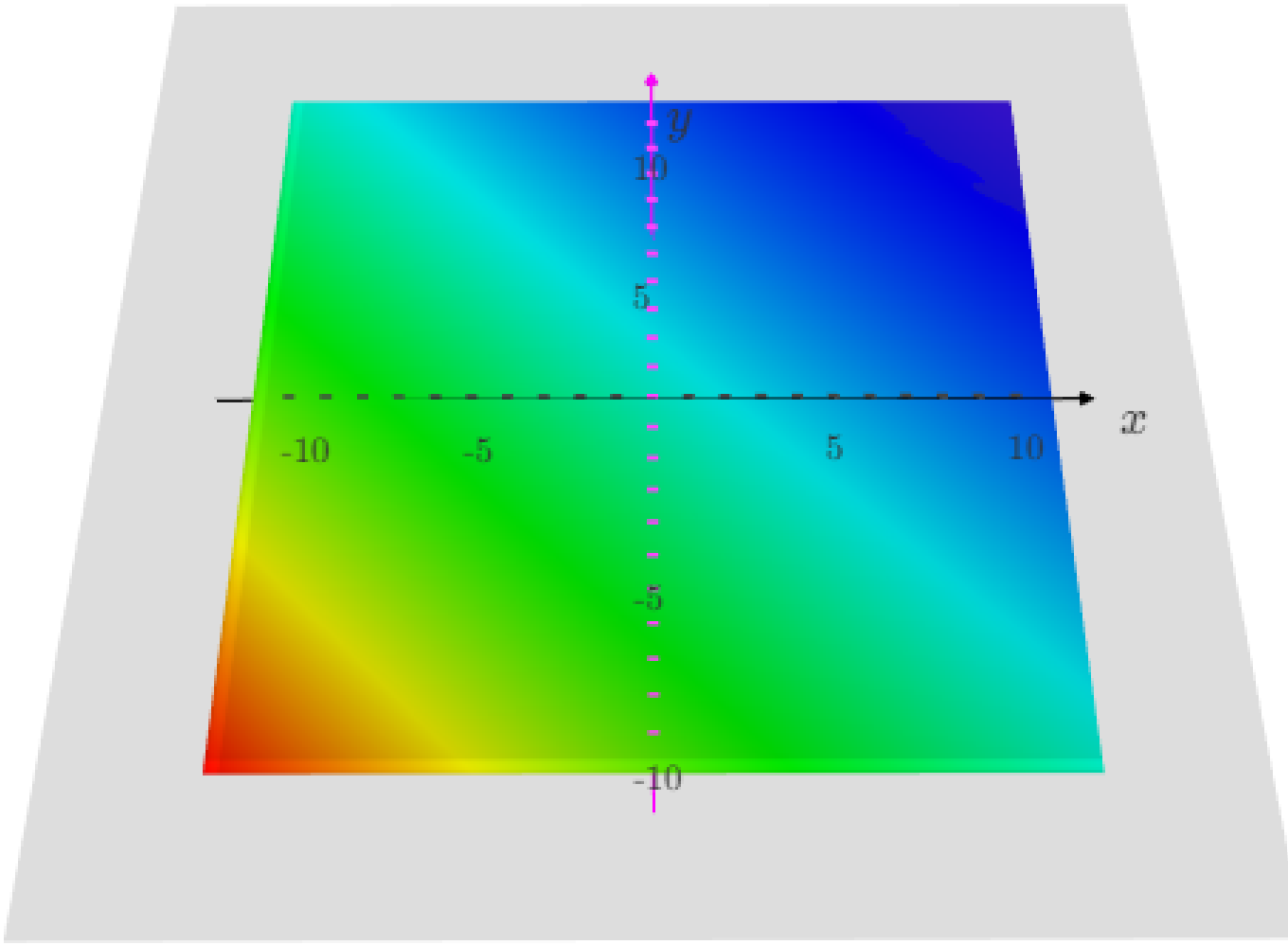
$$R = E[zz^T] = p_1 p_1^T * (Prob1) + p_2 p_2^T * (Prob2) + p_3 p_3^t * (Prob3) = \begin{vmatrix} 7 & 6 \\ 6 & 7 \end{vmatrix}$$

So, the MSE index is equal to $F(x) = c - 2x^T h + x^T Rx = 676 - 130(w_1 + w_2) + 2(w_1^2 + w_2^2) + 4(w_1 * w_2)$

To find the center of the contour, we need to solve for the minimum w*

$$w* = R^{-1} h = \begin{vmatrix} 7 & 6 \\ 6 & 7 \end{vmatrix}^{-1} \begin{vmatrix} 65 \\ 65 \end{vmatrix} = \begin{vmatrix} 5 \\ 5 \end{vmatrix}$$

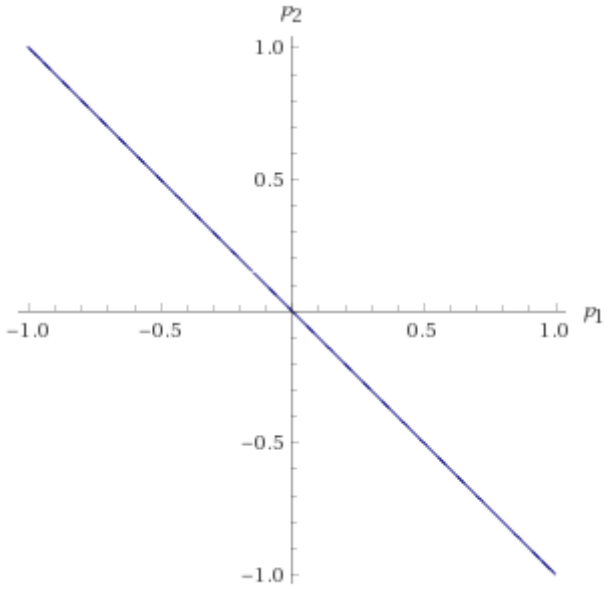
We can verify it below through its contour plot:



C.

For $bias = 0$ and $w_{minMSE} = \begin{vmatrix} 5 \\ 5 \end{vmatrix}$ we solve $w^T p + b = 0 => w^T p = 0 => 5p_1 = -5pw =>$ $p_1 = -p_2$

Let's visualize the optimal decision boundary (for min MSE):



D.

First, we need to find the eigenvalues of R. $R = \begin{vmatrix} 7 & 6 \\ 6 & 7 \end{vmatrix}$ so $\lambda_1 = 1$ and $\lambda_2 = 13$.

$$\lambda_{max} = 20$$

So, from the slides,

$$0 < a < 1/13$$

where a is the maximum stable learning rate.

Now, if target values to change to 2 and -2, there will be no change to $R = E[zz^T]$. Sincerely, there will be no changes to the learning rate.

E.

$$Wp = 0 * p = 0 = result$$

$$error = target - result = 26 - 0 = 26$$

From the slides, LMS algorithm is like this:

$$W(k+1) = W(k) + 2a * e(k)p^T(k) = \begin{vmatrix} 0 \\ 0 \end{vmatrix} + 2 * 0.5 * 26 * \begin{vmatrix} 2 \\ 4 \end{vmatrix} = \begin{vmatrix} 52 \\ 104 \end{vmatrix}$$

$$b(k+1) = b(k) + 2 * a * e(k) = 0 + 2 * 0.5 * 26 = 26$$

Problem-03

Consider a 1-2-1 RBF network (two neurons in the hidden layer and one output neuron). The first layer weights and biases are fixed as follows:

$$\mathbf{W}^1 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \mathbf{b}^1 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

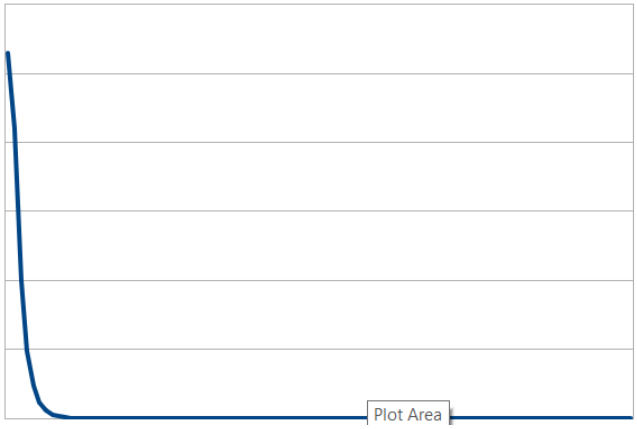
Assume that the bias in the second layer is fixed at 0 ($b^2=0$). The training set has the following input/target pairs: $\{p_1=1, t_1=-1\}$, $\{p_2=0, t_2=0\}$, $\{p_3=-1, t_3=1\}$.

A. Use linear least squares to solve for the second layer weights, assuming that the parameter $\varrho=0$.
B. Plot the contour plot for the sum squared error. Recall that it will be a quadratic function.
C. Write a MATLAB/python program to check your answers to parts (A) and (B).
D. Repeat tasks (A) to (C) with $\varrho=4$. Plot again the squared error.

Answer: ***Please refer to the attached files for the Octave code.***

We can observe that the iteration never ends. This happens because the patterns are non linear seperatable.

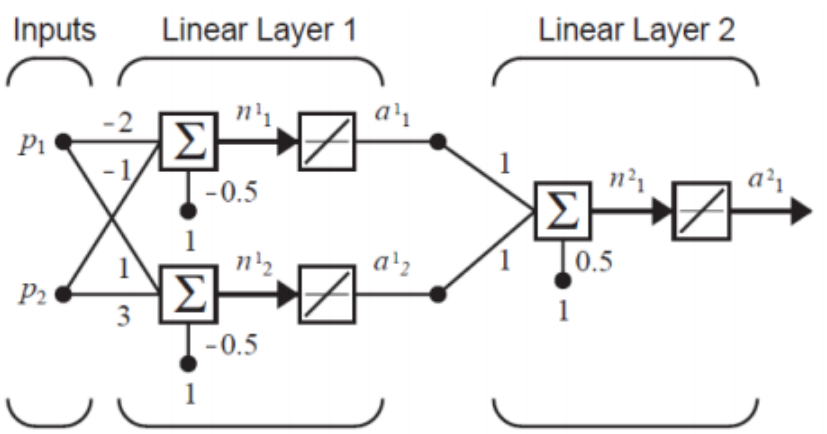
Problem-04



Answer:

Please refer to the attached files for the Python code.

Problem-05

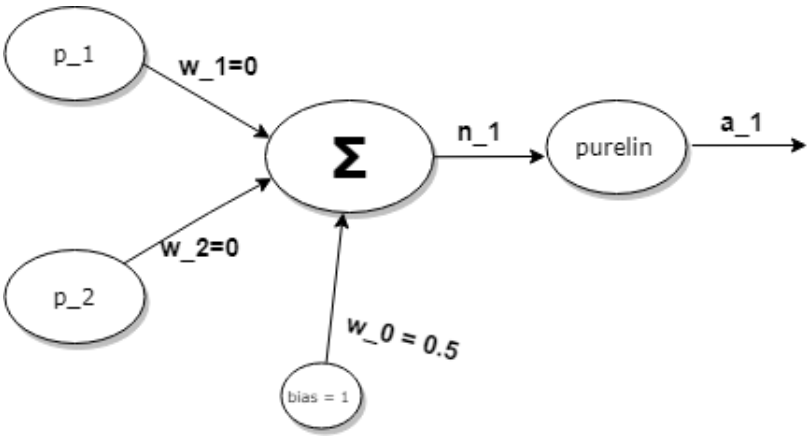


Answer: Find a single-layer network that has the same input/output characteristic as the network shown below.

If you do some example feedforward passes (e.g. $p_1 = 3$, $p_2 = 2$), you will see that the top row has an activation value of -8.5, while the bottom row has an activation value of 8.5.

If you pass this through a purelinear function with a $0.5 * 1$ bias, then the neural network will always output the value 0.5.

Summarizing, the above network could be replaced with this one:



Problem-06

Write a (MATLAB/python) program to implement the backpropagation algorithm for 1-S^1-1 network (logsigmoid-linear). Write the program using matrix operations, as we di in the class lecture. Choose the initial weights and biases to be random number uniformly distributed between -0.5 and 0.5, and train the network to approximate th function

$$g(p)=1+\sin[p(\pi/2)] \text{ for } -2 \leq p \leq 2.$$

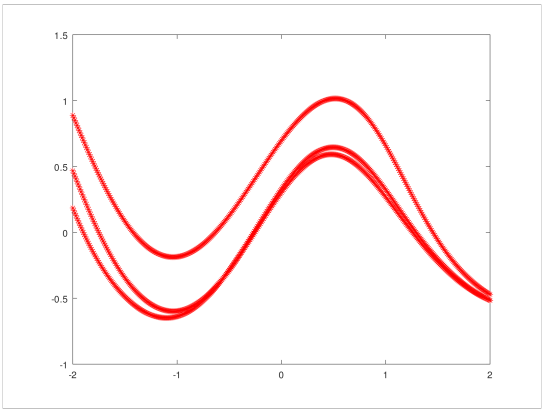
Use $S^1= 2$ and $S^1= 10$. Experiment with several different values for the learning rate α, and use several different initial conditions. Discuss the convergence properties of the algorithm as the learning rate changes.

Answer:

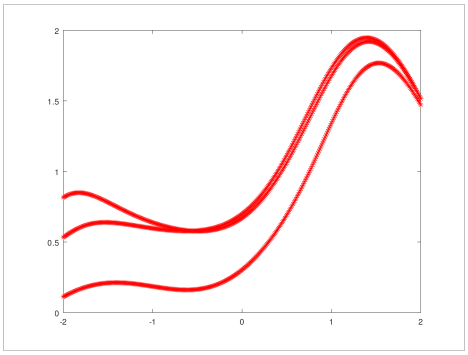
Please refer to the attached files for the OCTAVE code.

Following, we have 3 plots for each hidden layer architecture.
The first one is for 2 neurons in 1 hidden layer.
The second one is for 10 neurons in 1 hidden layer.
For each layer, we are going to plot:

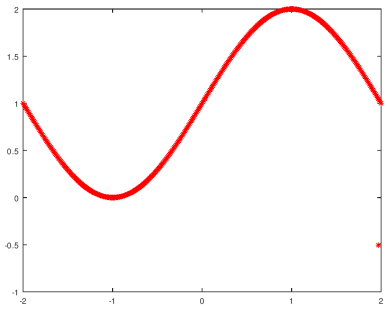
1. The error of the approximation according to the input
2. The approximation according to the input
3. The real value of the function according to the input



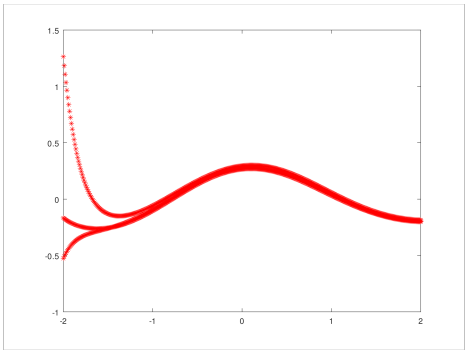
Error for S=2



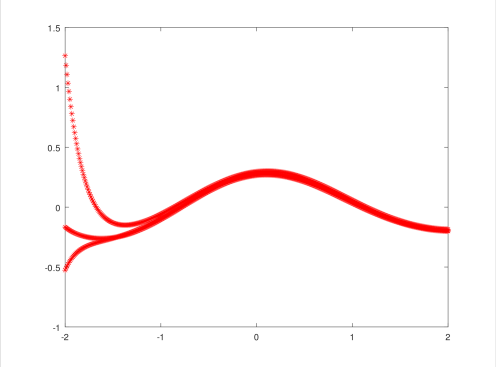
Approximation for S=2



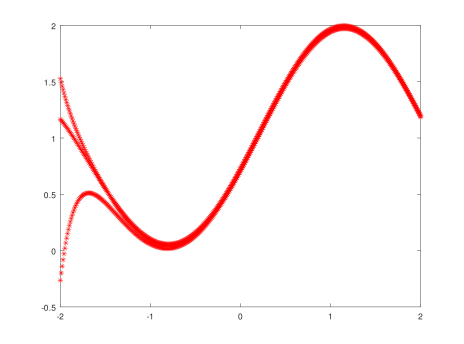
Real function value for S=2



Error for S=10



Approximation for S=10



Real value function for S=10

Comments:

- Error/cost seems to oscillate for S=2 and manages to get a lower value, comparing to the one for S=10
- The approximation for S=2 looks more realistic than for S=10
- By looking at the convex for S=2 and S=10, we could say that momentum backpropagation variance could help in this model.
- Of course, there are infinite choices of hyperparameters and we just present the most robust ones we tried.

Problem-07

The standard steepest descent backpropagation algorithm, which is summarized in the slide entitled "Summary of backpropagation algorithm" in Lecture-06, was designed to minimize the performance function that was the sum of squares of the network errors, as given in the last equation of slide 17 of Lecture-06. Suppose that we want to change the performance function to the sum of the fourth powers of the errors (e^4) plus the sum of the squares of the weights and biases in the network. Show how the equations in the slide entitled "Summary of backpropagation algorithm" will change for this new performance function. (You don't need to rederive any steps which are already derived in our lectures and do not change.)

Answer:

According to the description above, we have the similar change to the performance function:

$$F(x) = E[e^4 + w^2 + b^2] = E[(t-a)^4 + w^2 + b^2]$$

As lecture 06, slide 17 says: As with the LMS algorithm, we will approximate the mean square error with

$$F(x) = (t(k) - a(k))^4 + w^2(k) + b^2(k)$$

Notice that the expectation of the squared error has been replaced by the squared error at iteration k. Following, we will act like slide 26 in Lecture 06 and derive the sensitivity backpropagation:

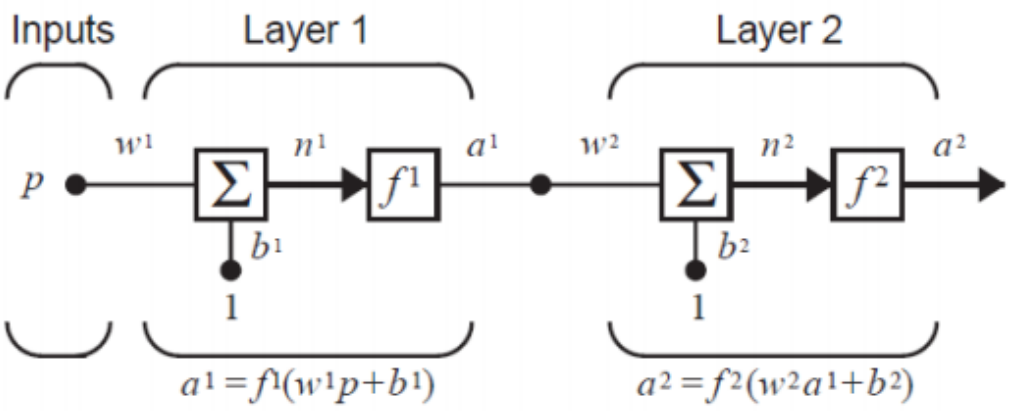
$$\begin{aligned} s_i^M &= \frac{\partial \hat{F}}{\partial n_i^M} = \frac{[\partial (t-a)^4 + w^2 + b^2]}{\partial n_i^m} \\ &= \frac{\partial (t-a)^4}{\partial n_i^m} + \frac{\partial w^2}{\partial n_i^m} + \frac{\partial b^2}{\partial n_i^m} \\ &= \frac{\partial (t-a)^4}{\partial n_i^m} + w^2 + 0 \\ &= -4 * (t-a)^3 * \frac{\partial a}{\partial n_i^m} + w^2 \\ &= -4(t-a)^3 * f'^M(n_i^M) + w^2 \end{aligned} \tag{1}$$

So, this means that in matrix form we have:

$$S_i^M = -4(t-a)^3 F'^M(n^M) + W_i^2$$

Problem-08

For the network shown below



the initial weights and biases are chosen to be

$$w^1(0)= 1,\ b^1(0)= -2,\ \ w^2(0)= 1,\ b^2(0)= 1.$$

The network transfer functions are:

$$f^1(n)= (n)^2,\quad f^2(n)= 1/n$$

and the input/target pair is given to be {p=1, t=1}.

Perform one iteration of backpropagation with $\alpha=1$.

Answer:

Feed-Forward Phase:

$$\begin{aligned} a^1 &= (1*1+(-2)*1)^2 = 1 \\ a^2 &= 1/(a^1*1+1*1) = 0.5 \end{aligned} \tag{2}$$

Compute Error:

$$error = e = t - output = 1 - 0.5 = 0.5 \tag{3}$$

Backpropagation:

Sensitivities:

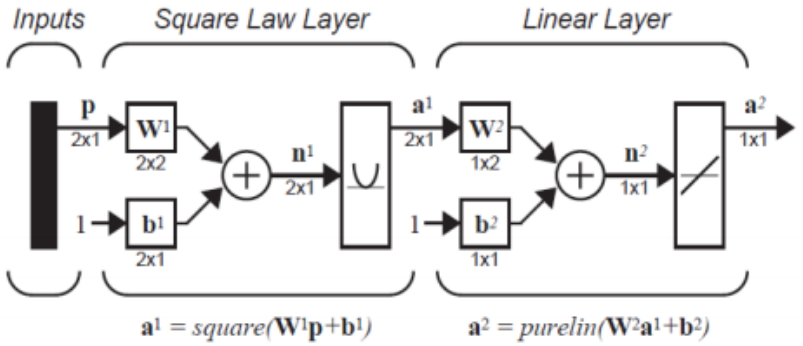
$$\begin{aligned} s^2 &= -2F'^2(n^2)(t-a) = -2(-\tfrac{1}{n^2})(e) = 0.25 \\ s^1 &= F'^1(n^1)(W^2)^T s^2 = 2n^1 * 1^t * 0.25 = -0.5 \end{aligned} \tag{4}$$

Update weights and biases:

$$
\begin{aligned}
&W^2(1) = W^2(0) - a * s^2 (a^1)^T = 1 - 1 * 0.25 * 1 = 0.75 \\
&b^2(1) = b^2(0) - a * s^2 = 1 - 1 * 0.25 = 0.75 \\
&W^1(1) = W^1(0) - a * s^1 (a^0)^T = 1 - 1 * (-0.5) = 1.5 \\
&b^1(1) = b^1(0) - a * s^1 = -2 - 1 * (-0.5) = 1.5
\end{aligned} \tag{5}
$$

Problem-09

Consider the following multilayer perceptron network. (The transfer function of the hidden layer is $f(n)= n^2$.)



The initial weights and biases are:

$$\mathbf{W}^1(0) = \begin{bmatrix} 1 & -1 \\ 1 & 0 \end{bmatrix}, \mathbf{W}^2(0) = \begin{bmatrix} 2 & 1 \end{bmatrix}, \mathbf{b}^1(0) = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \mathbf{b}^2(0) = \begin{bmatrix} -1 \end{bmatrix}$$

Perform one iteration of the standard steepest descent backpropagation (use matrix operations) with learning rate $\alpha= 0.5$ for the following input/target pair:

$$\left\{ \mathbf{p} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, t = \begin{bmatrix} 2 \end{bmatrix} \right\}$$

Answer:

Feed-Forward Phase:

For the first layer:

$$\begin{aligned} w^1 p + b^1 &= \begin{vmatrix} 1 & -1 \\ 1 & 0 \end{vmatrix} \begin{vmatrix} 1 \\ 1 \end{vmatrix} + \begin{vmatrix} 1 \\ -1 \end{vmatrix} \\ &= n^1 = \begin{vmatrix} 1 \\ 0 \end{vmatrix} \end{aligned} \tag{6}$$

$$a^1 = square(n^1) = \begin{vmatrix} 1 \\ 0 \end{vmatrix} \tag{7}$$

For the second layer:

$$\begin{aligned} w^2 a^1 + b^2 &= \begin{vmatrix} 2 & 1 \end{vmatrix} \begin{vmatrix} 1 \\ 0 \end{vmatrix} + \begin{vmatrix} 1 & -1 \end{vmatrix} \\ &= n^2 = \begin{vmatrix} 1 \end{vmatrix} \end{aligned} \tag{8}$$

Compute Error:

$$e = t - output = 2 - 1 = error \tag{9}$$

Perform standard steepest descent backpropagation:

Find the derivatives of the activation functions:

$$f'^2 = 1, f'^1 = 2n \tag{10}$$

Compute sensitivities:

$$
\begin{aligned}
s^2 &= -2f'^2(n^2)(e) = -2 * 1 * 1 = -2 \\
s^1 &= f'^1(n^1)(W^2)^T s^2 = 2n^1 \begin{vmatrix} 2 \\ 1 \end{vmatrix} (-2) = \begin{vmatrix} -8 \\ 0 \end{vmatrix}
\end{aligned} \tag{11}
$$

Finally, update weights and biases:

$$
\begin{aligned}
&W^2(1) = W^2(0) - a * s^2(a^1)^T = \begin{vmatrix} 2 & 1 \end{vmatrix} - 0.5(-2) \begin{vmatrix} 1 & 0 \end{vmatrix} = \begin{vmatrix} 3 & 1 \end{vmatrix} \\
&b^2(1) = b^2(0) - a * s^2 = -1 - 0.5(-2) = 0 \\
&W^1(1) = W^1(0) - a * s^1(a^0)^T = \begin{vmatrix} 1 & -1 \\ 1 & 0 \end{vmatrix} - 0.5 \begin{vmatrix} -8 & -4 \\ 1 & 0 \end{vmatrix} \begin{vmatrix} 1 \\ 1 \end{vmatrix} = \begin{vmatrix} 5 & 3 \\ 1 & 0 \end{vmatrix} \\
&b^1(1) = b^1(0) - a * s^1 = \begin{vmatrix} 1 \\ 1 \end{vmatrix} - 0.5 * \begin{vmatrix} -9 \\ 1 \end{vmatrix} = \begin{vmatrix} 5 \\ -1 \end{vmatrix}
\end{aligned} \tag{12}
$$

Problem-10

In the multilayer network, the net input is computed as follows:

$$\mathbf{n}^{m+1} = \mathbf{W}^{m+1}\mathbf{a}^m + \mathbf{b}^{m+1} \text{ or } n_i^{m+1} = \sum_{j=1}^{S^m} w_{i,j}^{m+1} a_j^m + b_i^{m+1}$$

If the net input calculation is changed to the following equation (squared distance calculation):

$$n_i^{m+1} = \sum_{j=1}^{S^m} (w_{i,j}^{m+1} - a_j^m)^2$$

how will the sensitivity backpropagation (i.e., $\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T\mathbf{s}^{m+1}$) change?

Answer:

In lecture 6 slides, we can get the original derivative and modify it:

$$\begin{aligned} &\frac{\partial n_i^{m+1}}{\partial n_j^m} = \frac{\partial(\sum_{j=1}^m (w_{ij}^{m+1} - a_j^m)^2)}{\partial n_j^m} \\ &= \frac{\partial (w_i^{m+1} - a^m)^2}{\partial n_j^m} \\ &= -2(w_i^{m+1} - a^m)(-\frac{\partial a^m}{\partial n_i^m} \\ &= -2(w_i^{m+1} - a^m)F'^m(n^m) \end{aligned} \tag{13}$$

Now, using the above equation, we can modify the sensitivity backpropagation:

$$\begin{aligned} &s^m = \frac{\partial \hat{F}}{\partial n^m} \\ &= (\frac{\partial n^{m+1}}{\partial n^m})^T \frac{\partial \hat{F}}{\partial n^{m+1}} \\ &= (-2(w_i^{m+1} - a^m)F'^m(n^m))^T \frac{\partial \hat{F}}{\partial n^{m+1}} \\ &= (-2(w_i^{m+1} - a^m)F'^m(n^m))^T s^{m+1} \end{aligned} \tag{14}$$

Problem-11

Consider again the net input calculation, as described in **Problem-10**. If the net input calculation is changed to the following equation (multiply by the bias, instead of add),

$$n_i^{m+1} = \left(\sum_{j=1}^{S^m} w_{i,j}^{m+1} a_j^m \right) \times b_i^{m+1}$$

how will the sensitivity backpropagation change?

Answer:

Just like in the previous problem, in lecture 6 slides, we can get the original derivative and modify it:

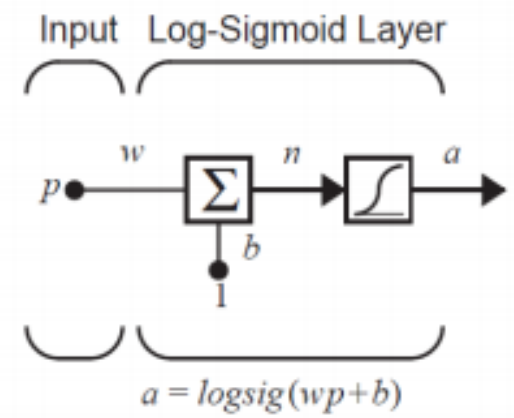
$$\begin{aligned} &\frac{\partial n_i^{m+1}}{\partial n_j^m} = \frac{\partial (w_{ij}^{m+1} a_j^m b_i^{m+1})}{\partial n m} \\ &= (w_i^{m+1} b^{m+1}) \frac{\partial a_j^m}{\partial n_j^m} \\ &= (w_i^{m+1} b^{m+1}) F'^m (n^m) \end{aligned} \tag{15}$$

Now, using the above equation, we can modify the sensitivity backpropagation:

$$\begin{aligned} &s^m = \frac{\partial \hat{F}}{\partial n^m} \\ &= (\frac{\partial n^{m+1}}{\partial n^m})^T \frac{\partial \hat{F}}{\partial n^{m+1}} \\ &= (w_i^{m+1} b^{m+1}) F'^m (n^m)^T \frac{\partial \hat{F}}{\partial n^{m+1}} \\ &= (w_i^{m+1} b^{m+1}) F'^m (n^m)^T s^{m+1} \end{aligned} \tag{16}$$

Problem-12

We want to train the network shown below on the training set $\{(\mathbf{p}_1=[-2]), (\mathbf{t}_1=[0.8])\}$, $\{(\mathbf{p}_2=[2]), (\mathbf{t}_2=[1])\}$, where each pair is equally likely to occur. Demonstrate the effect of batching by computing the direction of the initial step for SDBP with and without batching, starting from the initial guess: w(0)=0, b(0)=0.5. Do the steps point towards the same direction?



Answer: First, we will compute the direction of the first step without batching. The first step is computed from the first input/target pair. The forward and backpropagation steps will be:

$$a = logsig(wp + b) = logsig(0 + b) = logsig(b) = \frac{1}{1 + e(-0.5)} = 0.622 = a$$

$$e = t - a = 0.177$$

$$s = -2 * f`(n) * e = -2a(1 - a)e = -0.0832$$

Now, the direction of the weight will be:

$$-s * p = -(-0.0832) * 2 = 0.1664$$

The direction of the bias will be:

$$-s = 0.0832$$

So, finally, the direction plane without batching will be

$$Dir(w, b) = \begin{vmatrix} 0.1664 \\ 0.0832 \end{vmatrix} \tag{17}$$

Now, let's consider the direction of the gradient by batching. This will be computed by adding together the 2 individual gradients of the input/target pairs, multiplying by their probability of course, which is 0.5.

So, we will need to do the feed-forward/backpropagation phase for the second input too, since we already have computed this for the first input.

$$a = logsig(wp + b) = logsig(0 + b) = logsig(b) = \frac{1}{1 + e(-0.5)} = 0.622 = a$$

$$e = t - a = 0.378$$

$$s = -2a(1 - a)e = -0.1777$$

Like previously, the direction of the weight for input 2 will be:

$$-s * p_2 = 0.3553$$

The direction of the bias for input 2 will be:

$$-s = 0.1777$$

Since we are in 'batching mode', we need to sum up the direction vectors by respecting their probabilities occuring:

$$Dir(w, b)_{batch} = 1/2 \begin{vmatrix} 0.1664 \\ 0.0832 \end{vmatrix} + \begin{vmatrix} 0.3554 \\ 0.1777 \end{vmatrix} = \begin{vmatrix} 0.2609 \\ 0.1304 \end{vmatrix} \tag{18}$$

By comparing the 2 direction vectors, we can see that the steps do not point towards the same direction.

Problem-13

Consider the following quadratic function: $x_1^2+2x_2^2$
Perform 3 iterations of the variable learning rate algorithm, with initial guess: $x_0=[0, -1]^T$.
Use the algorithm parameters: α=1, γ=0.2, η=1.5, ϱ=0.5, ζ=5%.
(Count an iteration each time the function is evaluated after the initial guess.)

Answer: The first step is to evaluate the function at the initial guess x_0:

$$F(x_0) = 0^2 + 2(-1)^2 = 0 + 2 = 2$$

Next, we will find the gradient for the function:

$$\nabla F(x) = \begin{vmatrix} \frac{\partial}{\partial x_1} F(x) \\ \frac{\partial}{\partial x_2} F(x) \end{vmatrix} = \begin{vmatrix} 2x_1 \\ 4x_2 \end{vmatrix} \tag{19}$$

Next, we must evaluate the gradient at the initial guess x_0:

$$g_0 = \nabla F(x)|_{x=x_0} = \begin{vmatrix} 2*0 \\ 4*(-1) \end{vmatrix} = \begin{vmatrix} 0 \\ -4 \end{vmatrix} \tag{20}$$

Followingly, we must compute the tentative first step of the variable learning rate algorithm with a learning rate $a = 1$:

$$\nabla x_0 = \gamma * \nabla x_{-1} - (1-\gamma)\alpha g_0 = 0.2 * \begin{vmatrix} 0 \\ 0 \end{vmatrix} - (1-0.2) * 1 * \begin{vmatrix} 0 \\ -4 \end{vmatrix} = \begin{vmatrix} 0 \\ 3.2 \end{vmatrix} \tag{21}$$

$$x_1^t = x_0 + \nabla x_0 = \begin{vmatrix} 0 \\ -1 \end{vmatrix} + \begin{vmatrix} 0 \\ 3.2 \end{vmatrix} = \begin{vmatrix} 0 \\ 2.2 \end{vmatrix}$$

To verify the validation of this tentative step, we must test the value of the function at this new point:

$$F(x_1^t) = 0^2 + 2 * 2.2^2 = 9.68$$

This computed value is more than 5% larger than $F(x_0)$ so we must reject the tentative step, reduce the learning rate and set the momentum coefficient to zero.

$$x_1 = x_0$$
$$F(x_1) = F(x_0) = 2$$
$$\alpha = \rho * \alpha = 0.5 * 1 = 0.5$$
$$\gamma = 0$$
$$g_1 = g_0$$

Now a new tentative step is computed with zero momentum this time.

$$\nabla x_1 = -\alpha g_1 = -0.5 * \begin{vmatrix} 0 \\ -4 \end{vmatrix} = \begin{vmatrix} 0 \\ +2 \end{vmatrix} = \nabla x_1$$

$$x_2^t = x_1 + \nabla x_1 = \begin{vmatrix} 0 \\ -1 \end{vmatrix} + \begin{vmatrix} 0 \\ +2 \end{vmatrix} = \begin{vmatrix} 0 \\ +1 \end{vmatrix}$$

Let's evaluate the function at this point so we can see if we reject or accept the step:

$$F(x_2^t) = 0^2 + 2 * 1^2 = 2$$

This is the same value with the previous value $F(x_1)$ which is an edge case. We are stuck in this "local optimum" and the algorithm failed to get us find to find the global one.

Problem-14

Consider the following quadratic function:

$$F(x)=\frac{1}{2}x^T\begin{bmatrix}10 & -6\\ -6 & 10\end{bmatrix}x+\begin{bmatrix}4 & 4\end{bmatrix}x.$$

We want to use the steepest descent algorithm with momentum to minimize this function.

A. Suppose that the learning rate is α=0.2. Find a value for the momentum coefficient γ for which the algorithm will be stable. [Wait for the exercise in the class.]
B. Suppose that the learning rate is α=20. Find a value for the momentum coefficient γ for which the algorithm will be stable. [Wait for the exercise in the class.]
C. Write a MATLAB program to plot the trajectories of the algorithm for the α and γ values of both part (A) and part (B) on the contour plot of F(x), starting from the initial guess: $\mathbf{x}_0$= [-1 -2.5]^T.

Answer:

A) $A = \begin{bmatrix} 10 & -6 \\ -6 & 10 \end{bmatrix}$

$\det(A - \lambda I) = \begin{vmatrix} 10-\lambda & -6 \\ -6 & 10-\lambda \end{vmatrix} = (10-\lambda)^2 - 36 = 0$

$100 - 20\lambda + \lambda^2 - 36 = 0 \Rightarrow \lambda^2 - 20\lambda + 64 = 0$

$\Delta = 400 - 4 \cdot 64 = 64 \quad \lambda = \frac{20 \pm 8}{2} \begin{matrix} 14 \\ 6 \end{matrix}$

$|(1+\gamma) - (1-\gamma)a\lambda_i| < 2\gamma$

Έστω $a = 0.2$

Για $\lambda = 6$: $|(1+\gamma) - (1-\gamma) \cdot 1.2| < 2\gamma \Rightarrow 0.0082 < \gamma < 1$

Για $\lambda = 14$: $|(1+\gamma) - (1-\gamma) \cdot 2.8| < 2\gamma \Rightarrow 0.224 < \gamma < 1$

Επομένως οι τιμές για το γ είναι στο $(0.224, 1)$. Μέση τιμή $\gamma = 0.5$

B) Έστω $a = 20$

Για $\lambda = 6$: $|(1+\gamma) - 120(1-\gamma)| < 2\gamma \Rightarrow 0.96 < \gamma < 1$

Για $\lambda = 14$: $|(1+\gamma) - 280(1-\gamma)| < 2\gamma \Rightarrow 0.98 < \gamma < 1$

Επομένως οι τιμές του γ είναι στο $(0.98, 1)$. Μέση τιμή $\gamma = 0.99$

Problem-15

Consider the following quadratic function:

$$F(x)=\frac{1}{2}x^T\begin{bmatrix}3 & -1\\ -1 & 3\end{bmatrix}x+\begin{bmatrix}4 & -4\end{bmatrix}x.$$

We want to use the steepest descent algorithm with momentum to minimize this function.

A. Perform two iterations (finding $\mathbf{x}_1$ and $\mathbf{x}_2$) of steepest descent with momentum, starting from the initial condition $\mathbf{x}_0=[0\ 0]^T$. Use a learning rate of $\alpha=1$ and a momentum coefficient of $\gamma=0.75$.

B. Is the algorithm stable with this learning rate and this momentum? [Wait for the exercise in the class.]

C. Would the algorithm be stable with this learning rate, if the momentum were zero?

Answer:

A) $A = \begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix}$ $d^T = [4 \ -4]$

$grad = Ax + d = \begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix} x + \begin{bmatrix} 4 \\ -4 \end{bmatrix}$

$\Delta x_0 = \gamma \Delta x_{k-1} - (1-\gamma) a g x_0$

$x_1 - x_0 = \gamma x_0 - (1-\gamma) a g x_0$

$x_1 = -(1-0.75)\left(\begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix} x_0 + \begin{bmatrix} 4 \\ -4 \end{bmatrix}\right) = -0.25 \begin{bmatrix} 4 \\ -4 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$

$\Delta x_1 = \gamma \Delta x_0 - (1-\gamma) a g x_1$

$x_2 - x_1 = \gamma (x_1 - x_0) - (1-\gamma) a g x_1$

$x_2 - \begin{bmatrix} -1 \\ 1 \end{bmatrix} = 0.75 \begin{bmatrix} 1 \\ 1 \end{bmatrix} - 0.25 \left(\begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} + \begin{bmatrix} 4 \\ -4 \end{bmatrix}\right)$

$x_2 - \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.75 \\ 0.75 \end{bmatrix} - 0.25 \left(\begin{bmatrix} -4 \\ 4 \end{bmatrix} + \begin{bmatrix} 4 \\ -4 \end{bmatrix}\right) = 0$

$x_2 = \begin{bmatrix} -1.75 \\ 1.75 \end{bmatrix}$

B) $\det(A - \lambda I) = \begin{vmatrix} 3-\lambda & -1 \\ -1 & 3-\lambda \end{vmatrix} = 9 - 6\lambda + \lambda^2 - 1 = 0$

$\lambda^2 - 6\lambda + 8 = 0$ $\quad \Delta = 36 - 32 = 4$ $\quad \lambda = \frac{6 \pm 2}{2} \begin{matrix} \nearrow 4 \\ \searrow 2 \end{matrix}$

$|(1+\gamma) - (1-\gamma) a \lambda_i| < 2\sqrt{\gamma}$

για $\lambda = 2$: $|1.75 - 0.25 \cdot 2| = 1.25$, $2\sqrt{\gamma} = 2\sqrt{0.75} = 1.62$ $\Rightarrow 1.25 < 1.62$ ισχύει!

για $\lambda = 4$: $|1.75 - 0.25 \cdot 4| = 0.75$, $2\sqrt{\gamma} = 1.62$ $\Rightarrow 0.75 < 1.62$ ισχύει!

Άρα ο αλγόριθμος είναι stable.

C) Έστω $\gamma = 0$.

για $\lambda = 2$ $|1 - 2| < 0$ Άτοπο!

για $\lambda = 4$ $|1 - 4| < 0$ Άτοπο!

Άρα δεν είναι stable για $\gamma = 0$.

Problem-16

Consider an RBF network with the weights and biases in the first layer fixed. Show how the LMS algorithm of ADALINE could be modified for learning the second layer weights and biases.

Answer:

$W^2(1) = W^2(0) - \alpha s^2 (a^1)^T \overset{(1)}{\Rightarrow} W^2(1) = W^2(0) + 2e\alpha (a^1)^T$

$b^2(1) = b^2(0) - \alpha s^2 \overset{(1)}{\Rightarrow} b^2(1) = b^2(0) + 2\alpha e$

$s^2 = -2 \dot{f}^2(n^2)(t - a^2) = -2e$ (1)

LMS: $W_{k+1} = W_k + 2\alpha e_k p_k^T$

Ο αλγόριθμος έχει ως είσοδο το p_k^T ενώ στο backpropagation έχει ως είσοδο το $(a^1)^T$.

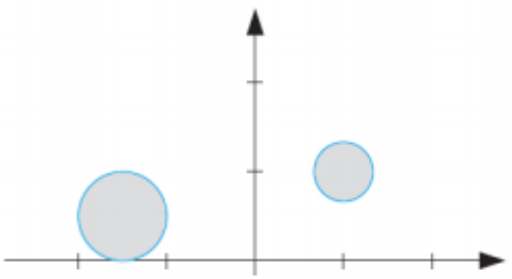
Επομένως όπου $p_k^T = (a^1)^T$

$b_{k+1} = b_k + 2\alpha e_k$

Τα b παραμένουν ίδια, άρα δεν χρειάζεται κάποια επιπλέον αλλαγή

Problem-17

Design an RBF network to perform the classification illustrated in the following figure. The network should produce a positive output whenever the input vector is in the shaded region and a negative output otherwise.



Answer:

For this problem, we will have 2 inputs and we can use 1 output to distinguish these 2 classes.

Everything inside the shaded region will be considered Class I and everything outside the shaded region will be considered Class 2. Judging by the image, two neurons will be enough for this classification problem.

The rows of the 1st layer weight matrix will be created by the centers of the two basis functions. By centering a basis function in each of these 2 shaded regions, the first layer weight matrix will be like this:

$$W^1 = \begin{vmatrix} -1.5 & 0.5 \\ 1 & 1 \end{vmatrix} \tag{22}$$

About the biases in the first layer, we need to consider the width of each basis function. Looking at the problem, the 1st basis function should be wider than the second. You can verify that by looking at the size of the circles/shaded regions.

Therefore, the first bias will be smaller than the second bias. The decision boundary formed by the 1st basis function should have a radius 0.5 while the second decision boundary should have a radius 0.25

Next, we want the basis function to drop significantly from their peaks in these distances. Let's try a bias of 2 for the first neuron and a bias of 4 for the second neuron.

For the first neuron:

$$a = e^{-n^2} = e^{(-2*0.5)^2} = 0.3679$$

For the second neuron:

$$a = e^{-n^2} = e^{(-4*0.25)^2} = 0.3679$$

This seems good so we will select first layer bias equal to:

$$b^1 = \begin{vmatrix} 2 \\ 4 \end{vmatrix}$$

We want the output to be negative for inputs outside the decision regions, so we will use a bias of -1 for the second layer.

$$b^2 = \begin{vmatrix} -1 \end{vmatrix}$$

The weights in the second layer scale the height of the hills. Since our first weights vector has them on the same level, there is really no problem here defining the second weight vector.. so the second layer weights could, for example, be like this:

$$W^2 = \begin{vmatrix} 1 & 1 \end{vmatrix}$$

Problem-18

Consider a 1-2-1 RBF network (two neurons in the hidden layer and one output neuron). The first layer weights and biases are fixed as follows:

$$\mathbf{W}^1 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \mathbf{b}^1 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

Assume that the bias in the second layer is fixed at 0 ($b^2=0$). The training set has the following input/target pairs: $\{p_1=1, t_1=-1\}$, $\{p_2=0, t_2=0\}$, $\{p_3=-1, t_3=1\}$.

A. Use linear least squares to solve for the second layer weights, assuming that the parameter $\varrho=0$.
B. Plot the contour plot for the sum squared error. Recall that it will be a quadratic function.
C. Write a MATLAB/python program to check your answers to parts (A) and (B).
D. Repeat tasks (A) to (C) with $\varrho=4$. Plot again the squared error.

Answer:

A) $W^1 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$ $b^1 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$ $b^2 = 0$ $\varphi = 0$

$q_1 = 1$ $q_2 = 0$ $q_3 = -1$ $t_1 = -1$ $t_2 = 0$ $t_3 = 1$

$n_{1,1} = \|w_1^T - q_1\| b_1 = \sqrt{(-1)^2} \cdot 0.5 = 0.5$

$n_{1,2} = \|w_1^T - q_1\| b_1 = \sqrt{(1-1)^2} \cdot 0.5 = 0$

$a_1 = \begin{bmatrix} e^{-n_{1,1}^2} \\ e^{-n_{1,2}^2} \end{bmatrix} = \begin{bmatrix} 0.36 \\ 1 \end{bmatrix}$

$n_{2,1} = \sqrt{(-1)^2} \cdot 0.5 = 0.5$ $n_{2,2} = \sqrt{1} \cdot 0.5 = 0.5$

$a_2 = \begin{bmatrix} e^{-n_{2,1}^2} \\ e^{-n_{2,2}^2} \end{bmatrix} = \begin{bmatrix} 0.77 \\ 0.77 \end{bmatrix}$

$n_{3,1} = \sqrt{(-1+1)^2} \cdot 0.5 = 0$

$n_{3,2} = \sqrt{(1+1)^2} \cdot 0.5 = 1$

$a_3 = \begin{bmatrix} e^{-n_{3,1}^2} \\ e^{-n_{3,2}^2} \end{bmatrix} = \begin{bmatrix} 1 \\ 0.36 \end{bmatrix}$

$U = \begin{bmatrix} z_1^T \\ z_2^T \\ z_3^T \end{bmatrix} = \begin{bmatrix} 0.36 & 1 & 1 \\ 0.77 & 0.77 & 1 \\ 1 & 0.36 & 1 \end{bmatrix}$

$U^T U = \begin{bmatrix} 0.36 & 0.77 & 1 \\ 1 & 0.77 & 0.36 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0.36 & 1 & 1 \\ 0.77 & 0.77 & 1 \\ 1 & 0.36 & 1 \end{bmatrix} = \begin{bmatrix} 1.72 & 1.31 & 2.13 \\ 1.31 & 1.72 & 2.13 \\ 2.13 & 2.13 & 3 \end{bmatrix}$

$x^+ = [U^T U + q I]^{-1} U^T t = \begin{bmatrix} 1.72 & 1.31 & 2.13 \\ 1.31 & 1.72 & 2.13 \\ 2.13 & 2.13 & 3 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 0.36 & 0.77 & 1 \\ 1 & 0.77 & 0.36 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} =$

$= \begin{bmatrix} 0.26 \\ -0.26 \\ 0 \end{bmatrix}$. Άρα $W_2 = \begin{bmatrix} 0.26 \\ -0.26 \end{bmatrix}$ και $b_2 = 0$

B) $f(x) = t^T t - 2 t^T U x + x^T [U^T U + q I] x = [-1 \; 0 \; 1] \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} - 2 [-1 \; 0 \; 1] \begin{bmatrix} 0.36 & 1 & 1 \\ 0.77 & 0.77 & 1 \\ 1 & 0.36 & 1 \end{bmatrix}$

$\cdot \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} + [x \; y \; 0] \begin{bmatrix} 1.72 & 1.31 & 2.13 \\ 1.31 & 1.72 & 2.13 \\ 2.13 & 2.13 & 3 \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} =$

$= 1.72x^2 + 1.72y^2 + 2.62xy - 1.28x + 1.28y + 2$

Problem-19

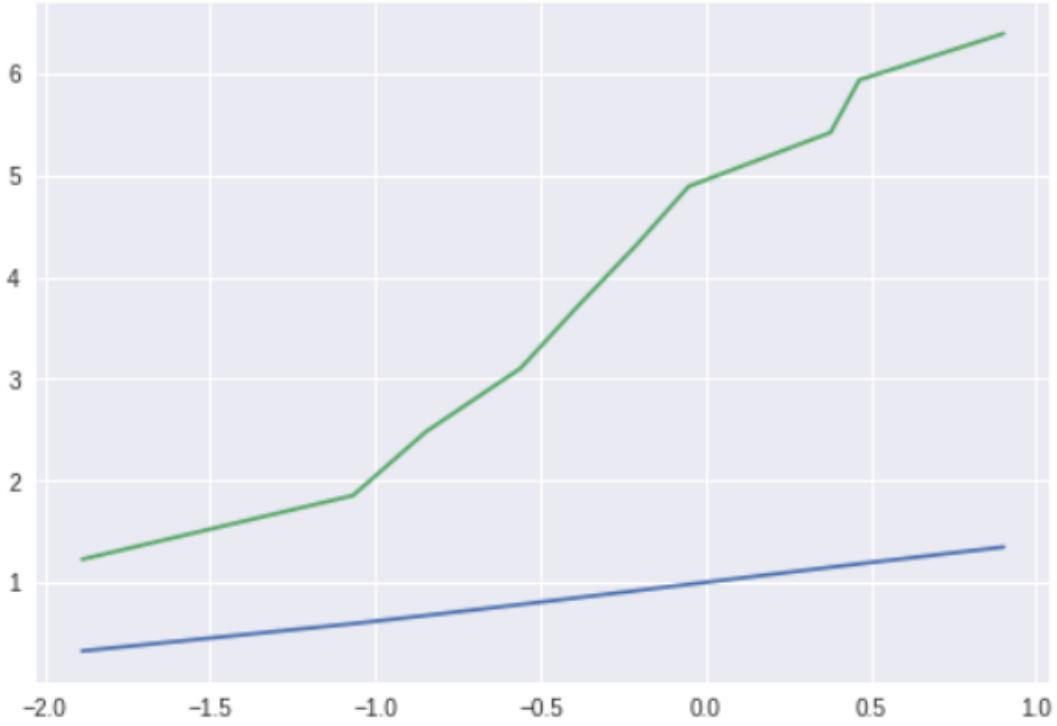
Write a MATLAB/python program to implement the steepest descent algorithm for the 1-S^1-1 RBF network. Train the network to approximate the function:

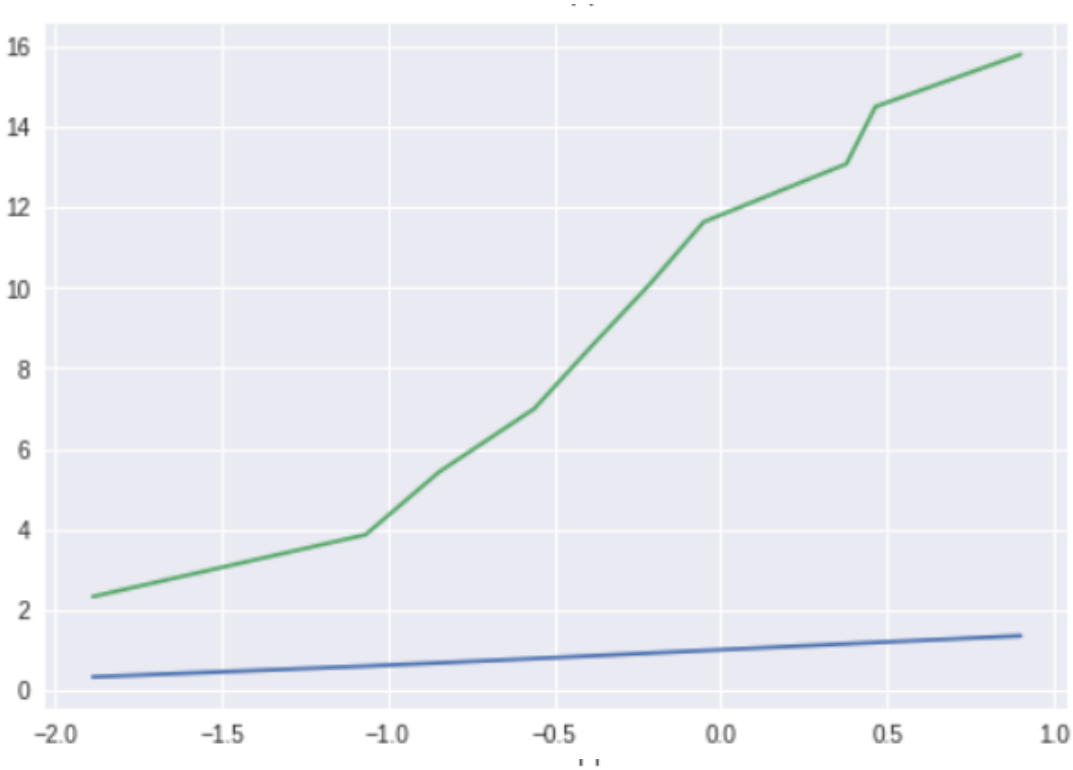
$$g(p) = 1 + \sin(p\pi/8) \text{ for } -2 \leq p \leq 2.$$

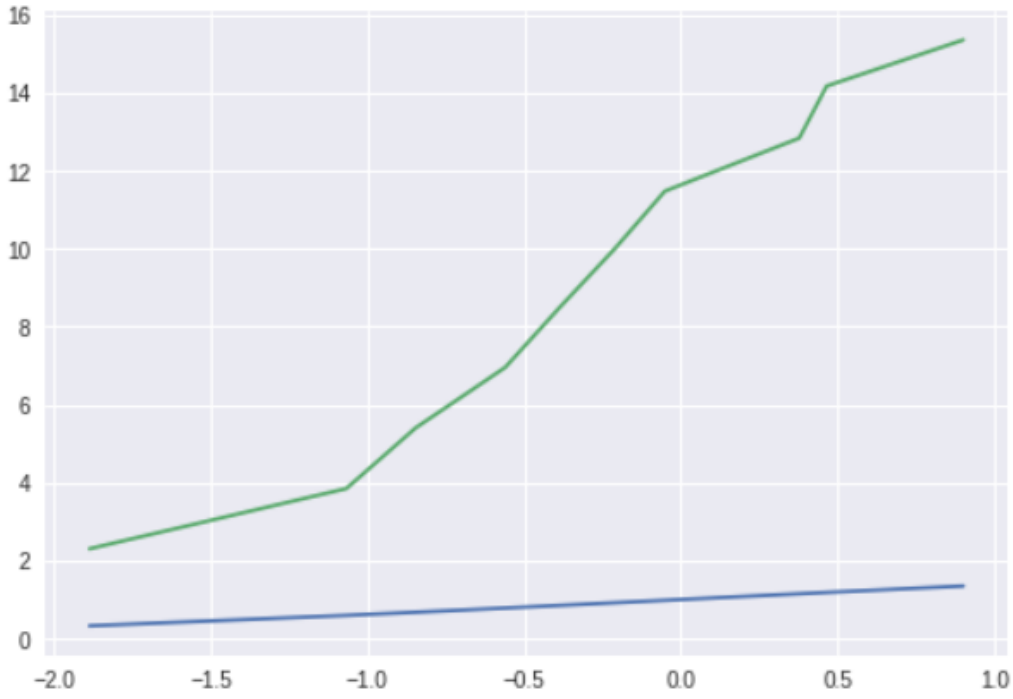
A. Select 10 data points at random from the interval $-2 \leq p \leq 2$.
B. Initialize all parameters (weights and biases in both layers) as small random numbers, and then train the network to convergence. (Experiment with the learning rate α, to determine a stable value.) Plot the network response for $-2 \leq p \leq 2$, and show the training points on the same plot. Compute the sum squared error over the training set. Use 2, 4 and 8 centers. Try different sets of initial weights.

Answer:

Please refer to the attached files for the Python code.




16
14
12
10
8
6
4
2
0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0


16
14
12
10
8
6
4
2
0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0