Backpropagation

Hamidreza Baradaran Kashani

Neural Networks Course (Fall 2022)

Objectives

- Seeking a way to train Multilayer Neural Networks
- Getting familiar with the Backpropagation algorithm
- Investigating the capacity of different network architectures for applying BP.
- Understanding the issues of vanilla BP and trying to improve it by variations like MOBP & VLBP algorithm

List of Contents

1. Multilayer Perceptron
2. Backpropagation Algorithm
3. Choice of network architecture
4. Drawbacks of SDBP
5. Momentum (MOBP)
6. Variable Learning Rate (VLBP)

Multilayer Perceptron

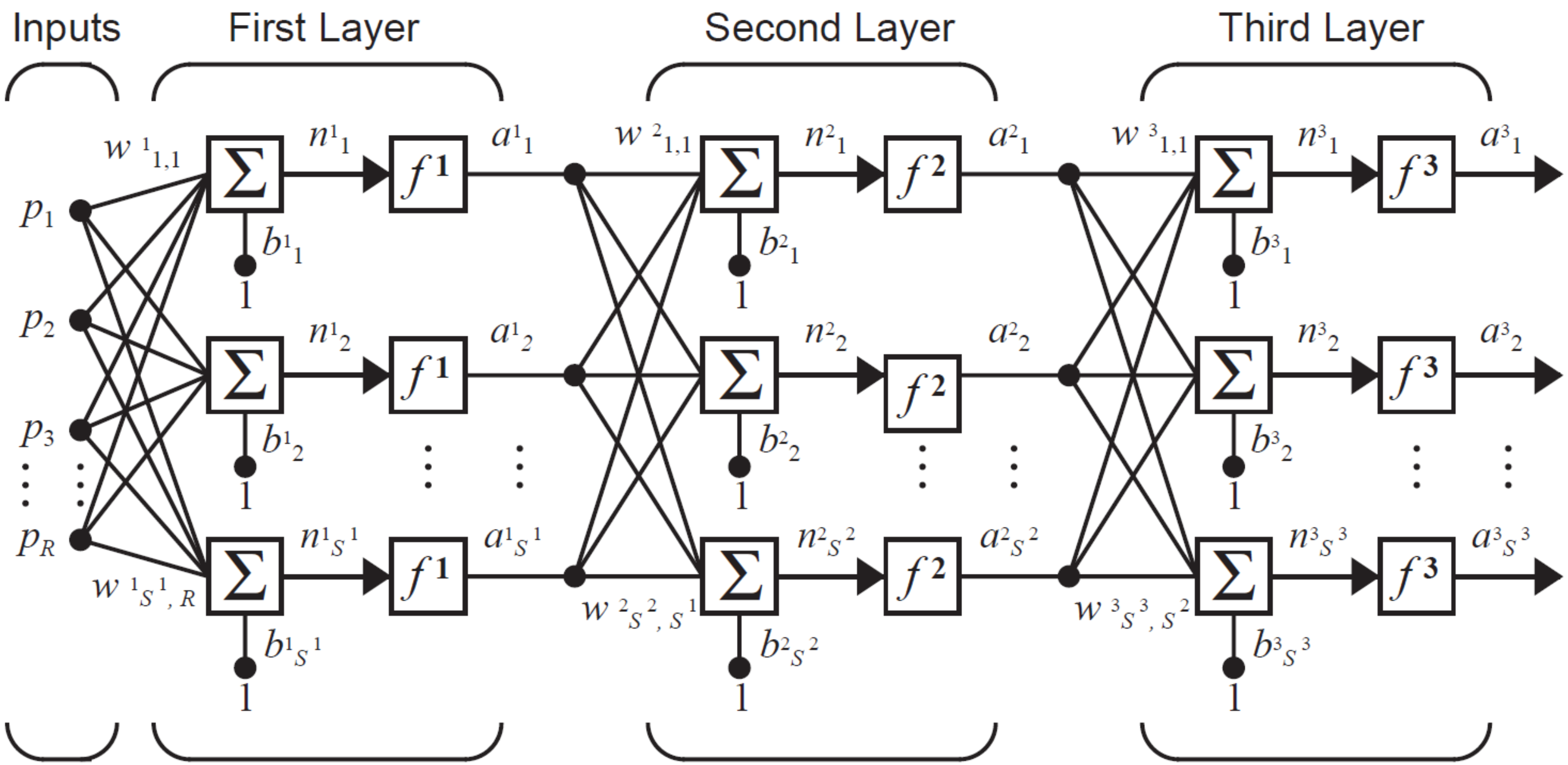
- ✓ We can cascade some single-layer perceptrons to form a multilayer perceptron.

- ✓ Each layer may have a different number of neurons, and even a different activation function.

- ✓ We use superscripts to identify the layer number. The weight matrix for the first layer and the second layer are written as $\mathbf{W}^1$ and $\mathbf{W}^2$ respectively.

- ✓ To identify the structure of a multilayer network, we will sometimes use this shorthand notation, where the number of inputs is followed by the number of neurons in each layer: $R - S^1 - S^2 - S^3$

Multilayer Perceptron



$$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1) \qquad \mathbf{a}^2 = \mathbf{f}^2(\mathbf{W}^2\mathbf{a}^1+\mathbf{b}^2) \qquad \mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{a}^2+\mathbf{b}^3)$$

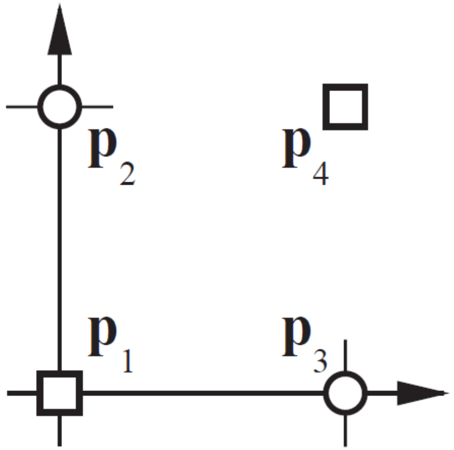
$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1)+\mathbf{b}^2)+\mathbf{b}^3)$$

Multilayer Perceptron

- **MLP for classification:** One of the first problems to demonstrate the limitations of the single-layer perceptron was the XOR classification problem.

- In this problem, since the two categories are not linearly separable, a single-layer perceptron can not perform the classification.

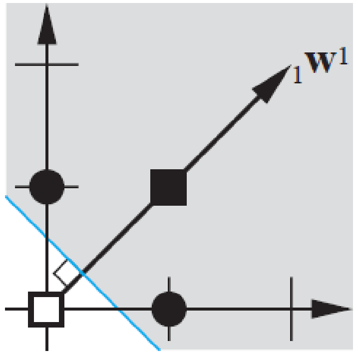
$$\left\{\mathbf{p}_1 = \begin{bmatrix}0\\0\end{bmatrix}, t_1 = 0\right\} \left\{\mathbf{p}_2 = \begin{bmatrix}0\\1\end{bmatrix}, t_2 = 1\right\} \left\{\mathbf{p}_3 = \begin{bmatrix}1\\0\end{bmatrix}, t_3 = 1\right\} \left\{\mathbf{p}_4 = \begin{bmatrix}1\\1\end{bmatrix}, t_4 = 0\right\}$$



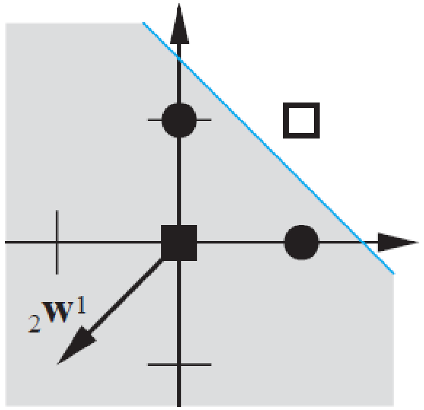
Multilayer Perceptron

- There are many different multilayer solutions to solve the XOR problem. One solution is to use a two-layer perceptron.

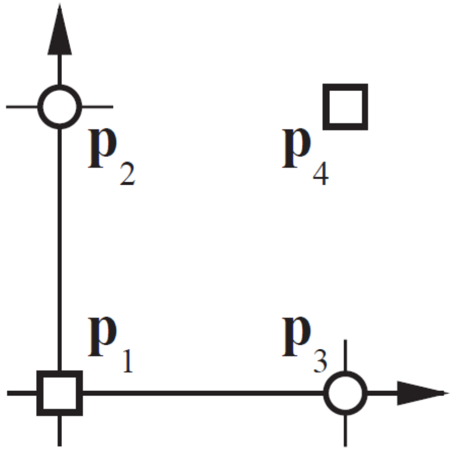
- In the first layer there are two neurons to create two decision boundaries. The first boundary separates $\mathbf{p}_1$ from the other patterns, and the second boundary separates $\mathbf{p}_4$.



Layer 1/Neuron 1

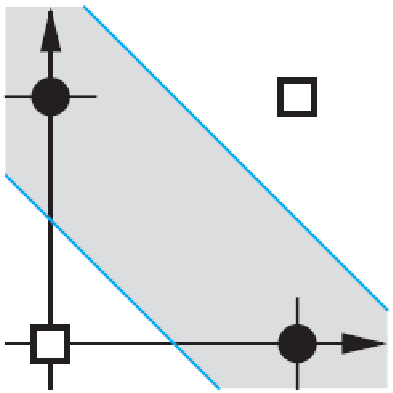


Layer 1/Neuron 2

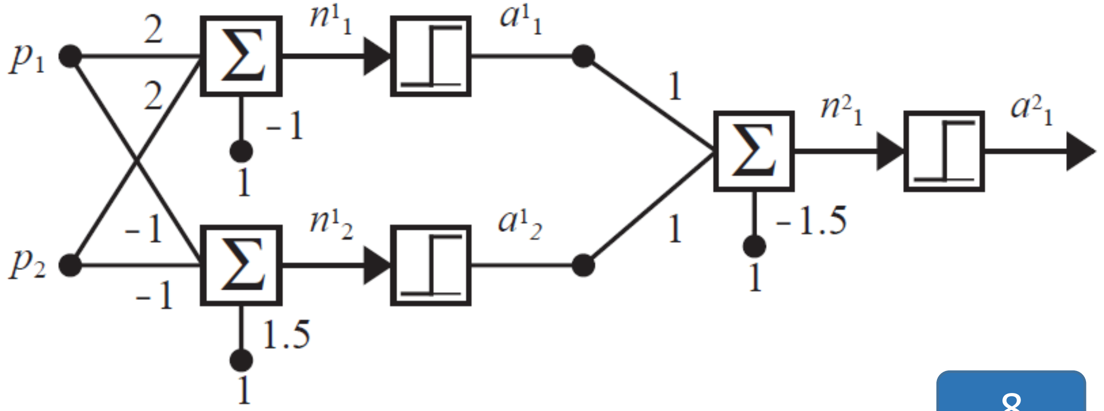


Multilayer Perceptron

- Then the second layer is used to combine the two boundaries together using an AND operation. The shaded region indicates those inputs that will produce a network output of 1.

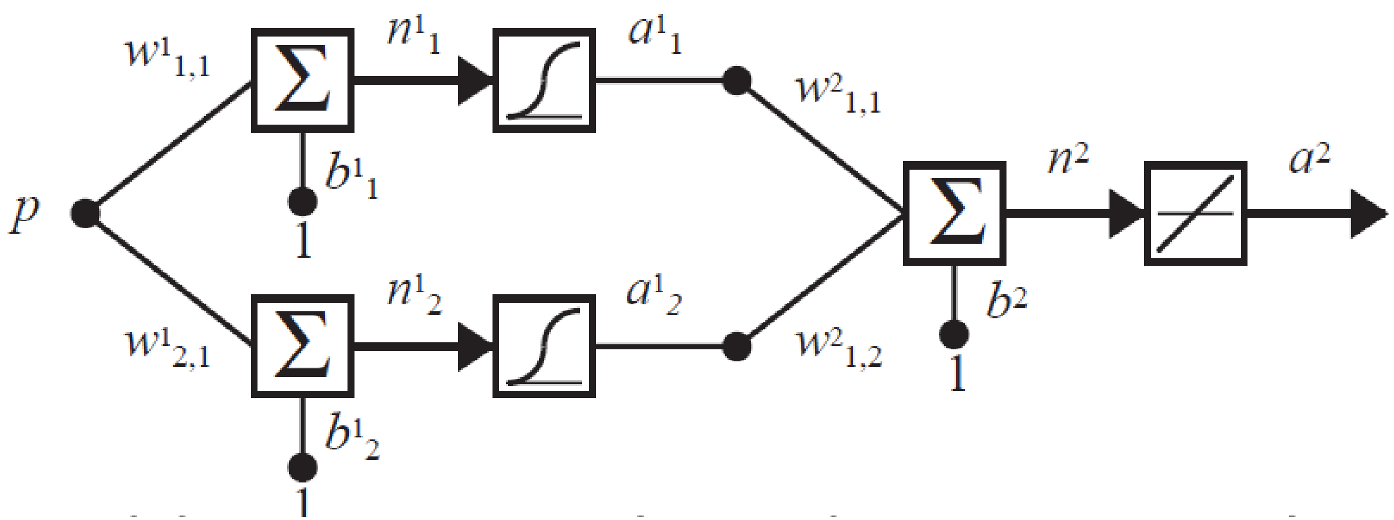


- The resulting two-layer 2-2-1 network:



Multilayer Perceptron

- **MLP for Regression:** Consider a two-layer, 1-2-1 network. For this example the activation function for the first layer is Sigmoid and the transfer function for the second layer is Linear.

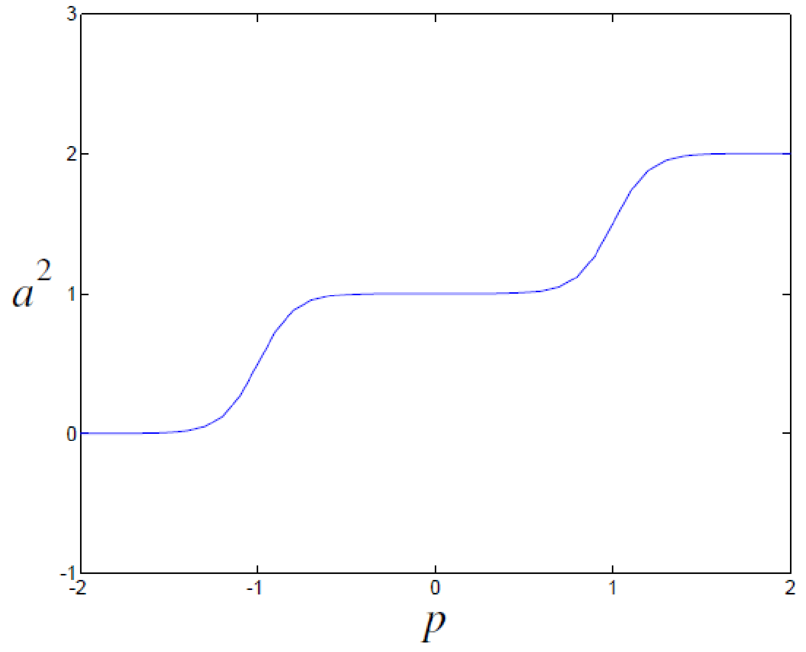


Multilayer Perceptron

- Suppose that the nominal values of the parameters for this network are:

$$w^1_{1,1} = 10, \; w^1_{2,1} = 10, \; b^1_1 = -10, \; b^1_2 = 10 \quad w^2_{1,1} = 1, \; w^2_{1,2} = 1, \; b^2 = 0$$

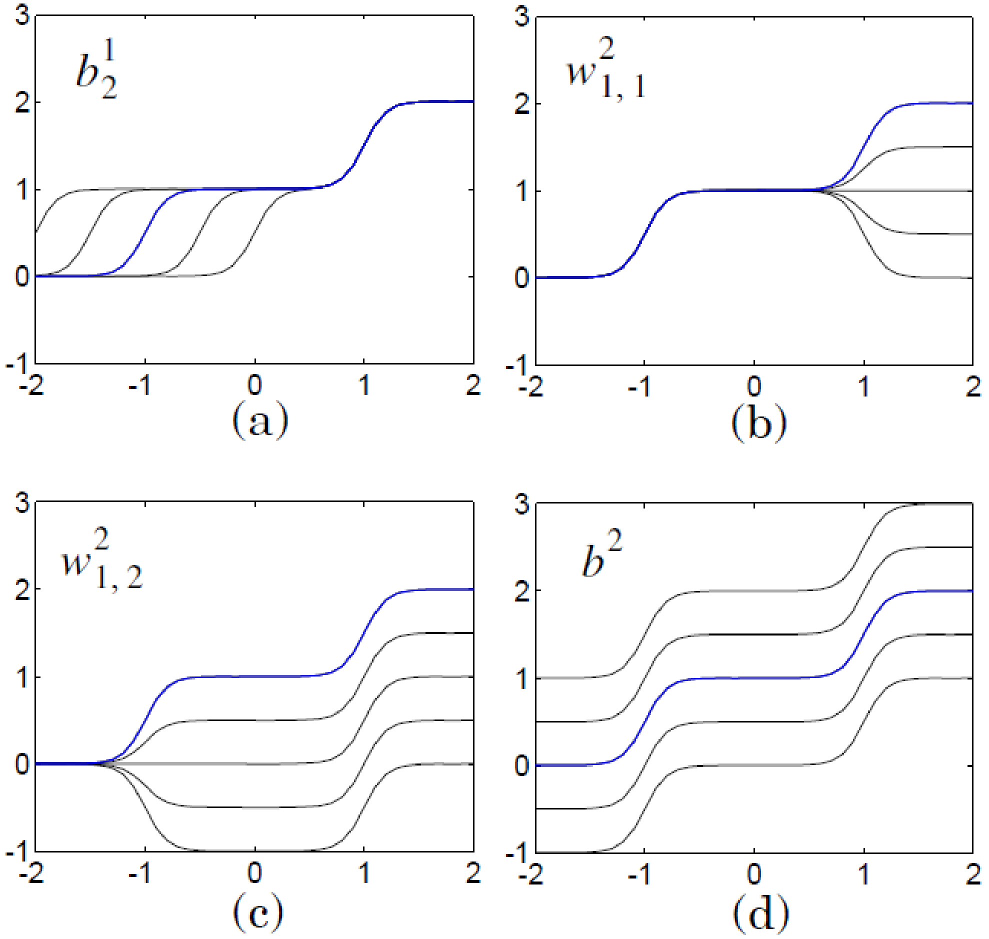
- With these parameters, the response of the network in terms of the input will be:



Multilayer Perceptron

- By adjusting the network parameters, we can change the shape of the response.

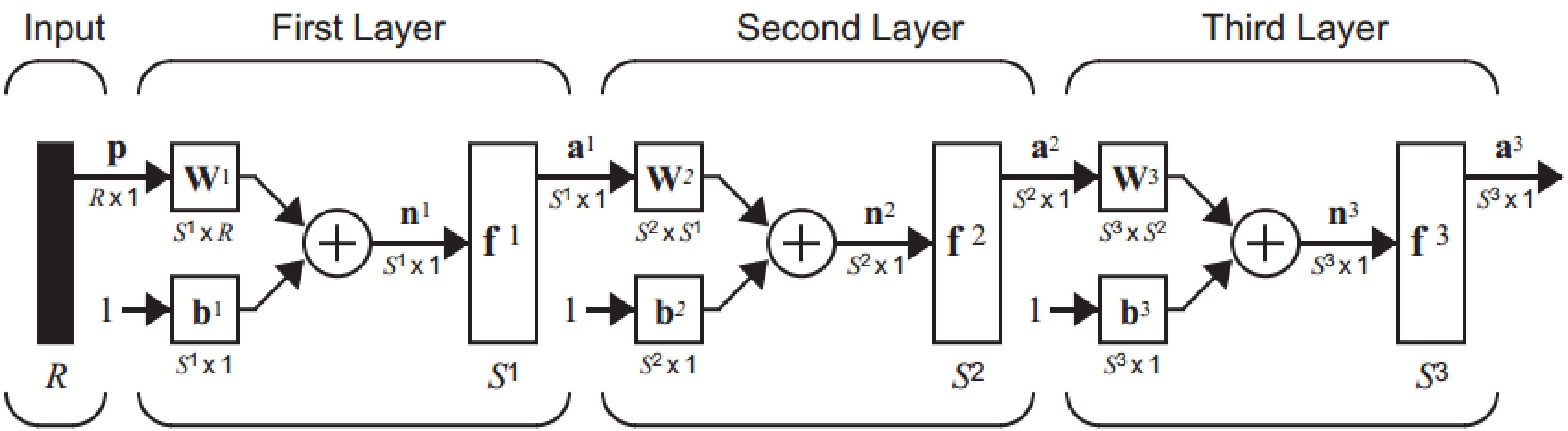
a) The effect of changing b_2^1 on the response.
b) The effect of changing $w_{1,1}^2$ on the response.
c) The effect of changing $w_{1,2}^2$ on the response.
d) The effect of changing b^2 on the response.



Multilayer Perceptron

- We can see how flexible the multilayer network is.
- It would appear that we could use such networks to approximate almost any function, if we had a sufficient number of neurons in the hidden layer.
- In order to train MLPs on some training data we should use a learning rule with the purpose of minimizing the loss function.
- The most important algorithm to this end is called **Backpropagation**. We will see that in the next section.

Multilayer Perceptron



$$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{W}^1\mathbf{p} + \mathbf{b}^1) \qquad \mathbf{a}^2 = \mathbf{f}^2(\mathbf{W}^2\mathbf{a}^1 + \mathbf{b}^2) \qquad \mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{a}^2 + \mathbf{b}^3)$$

$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{p} + \mathbf{b}^1) + \mathbf{b}^2) + \mathbf{b}^3)$$

Three-Layer Network, Abbreviated Notation

Backpropagation Algorithm

- **Forward propagation:** for multilayer networks the output of one layer becomes the input to the following layer:

$$\mathbf{a}^{0} = \mathbf{p},$$

$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^{m} + \mathbf{b}^{m+1}) \text{ for } m = 0, 1, \ldots, M-1$$

where M is the number of layers in the network.

- The outputs of the neurons in the last layer are considered the network outputs:

$$\mathbf{a} = \mathbf{a}^{M}$$

Backpropagation Algorithm

❑ **Mean Square Error:** The BP algorithm should adjust the network parameters in order to minimize the mean square error:

$$F(\mathbf{x}) = E[e^2] = E[(t - a)^2]$$

- where $\mathbf{x}$ is the vector of network weights and biases. If the network has multiple outputs this generalizes to:

$$F(\mathbf{x}) = E[\mathbf{e}^T\mathbf{e}] = E[(\mathbf{t} - \mathbf{a})^T(\mathbf{t} - \mathbf{a})]$$

- we will approximate the mean square error by:

$$\hat{F}(\mathbf{x}) = (\mathbf{t}(k) - \mathbf{a}(k))^T(\mathbf{t}(k) - \mathbf{a}(k)) = \mathbf{e}^T(k)\mathbf{e}(k)$$

Backpropagation Algorithm

- In fact, the expectation of the squared error has been replaced by the squared error at iteration k.

- The Gradient Descent algorithm for the approximate mean square error (stochastic gradient descent) is:

$$w_{i,j}^{m}(k+1) = w_{i,j}^{m}(k) - \alpha \frac{\partial \hat{F}}{\partial w_{i,j}^{m}} \qquad b_{i}^{m}(k+1) = b_{i}^{m}(k) - \alpha \frac{\partial \hat{F}}{\partial b_{i}^{m}}$$

- Therefore, the partial derivatives should be calculated.

Backpropagation Algorithm

- **Chain Rule:** For a single-layer linear network the partial derivatives are conveniently computed.

- But for a multilayer network the error is not an explicit function of the weights in the hidden layers, therefore these derivatives are not computed so easily.

- Because the error is an indirect function of the weights in the hidden layers, we will use the chain rule of calculus to calculate the derivatives.

Backpropagation Algorithm

- Suppose that we have a function f that is an explicit function only of the variable n.

- We want to take the derivative of f with respect to a third variable w. The chain rule is:

$$\frac{df(n(w))}{dw} = \frac{df(n)}{dn} \times \frac{dn(w)}{dw}$$

- For example:

$$f(n) = e^{n} \text{ and } n = 2w \text{, so that } f(n(w)) = e^{2w}$$

$$\frac{df(n(w))}{dw} = \frac{df(n)}{dn} \times \frac{dn(w)}{dw} = (e^{n})(2)$$

Backpropagation Algorithm

- We will use the chain rule concept to find the derivatives:

$$\frac{\partial \hat{F}}{\partial w_{i,j}^m} = \frac{\partial \hat{F}}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial w_{i,j}^m} \qquad \frac{\partial \hat{F}}{\partial b_i^m} = \frac{\partial \hat{F}}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial b_i^m}.$$

- The second term in each of these equations can be easily computed, since the net input to layer m is an explicit function of the weights and bias in that layer:

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

Backpropagation Algorithm

- Then the derivative of the net input to layer m with respect to the weights and bias will be:

$$\frac{\partial n_i^m}{\partial w_{i,j}^m} = a_j^{m-1}, \quad \frac{\partial n_i^m}{\partial b_i^m} = 1$$

- **Sensitivity:** We define the *sensitivity* of $\hat{F}$ to changes in the i^{th} element of the net input at layer m:

$$s_i^m \equiv \frac{\partial \hat{F}}{\partial n_i^m}$$

Now we can say:

$$\frac{\partial \hat{F}}{\partial w_{i,j}^m} = s_i^m a_j^{m-1} \qquad \frac{\partial \hat{F}}{\partial b_i^m} = s_i^m$$

Backpropagation Algorithm

- The Stochastic Gradient Descent (SGD) algorithm can be expressed as:

$$w_{i,j}^{m}(k+1) = w_{i,j}^{m}(k) - \alpha s_i^m a_j^{m-1}$$

$$b_i^m(k+1) = b_i^m(k) - \alpha s_i^m .$$

- SGD in matrix form:

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha \mathbf{s}^m (\mathbf{a}^{m-1})^T$$

$$\mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha \mathbf{s}^m ,$$

$$\mathbf{s}^m \equiv \frac{\partial \hat{F}}{\partial \mathbf{n}^m} = \begin{bmatrix} \frac{\partial \hat{F}}{\partial n_1^m} \\ \frac{\partial \hat{F}}{\partial n_2^m} \\ \vdots \\ \frac{\partial \hat{F}}{\partial n_{S^m}^m} \end{bmatrix}$$

Backpropagation Algorithm

- It now remains for us to compute the sensitivities $\boldsymbol{s}^m$, which requires another application of the chain rule.

- It is this process that gives us the term *backpropagation*, because it describes a recurrence relationship in which the sensitivity at layer m is computed from the sensitivity at layer $m+1$.

Backpropagation Algorithm

- **Backpropagating the sensitivities:** the sensitivity at layer m is computed from the sensitivity at layer $m+1$.

- To derive the recurrence relationship for the sensitivities, we will use the Jacobian matrix:

$$\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^{m}} \equiv \begin{bmatrix} \frac{\partial n_1^{m+1}}{\partial n_1^{m}} & \frac{\partial n_1^{m+1}}{\partial n_2^{m}} & \cdots & \frac{\partial n_1^{m+1}}{\partial n_{S^m}^{m}} \\ \frac{\partial n_2^{m+1}}{\partial n_1^{m}} & \frac{\partial n_2^{m+1}}{\partial n_2^{m}} & \cdots & \frac{\partial n_2^{m+1}}{\partial n_{S^m}^{m}} \\ \vdots & \vdots & & \vdots \\ \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_1^{m}} & \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_2^{m}} & \cdots & \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_{S^m}^{m}} \end{bmatrix}$$

Backpropagation Algorithm

- The i , j element of the Jacobian matrix will be:

$$\frac{\partial n_i^{m+1}}{\partial n_j^m} = \frac{\partial\left(\sum_{l=1}^{S^m} w_{i,l}^{m+1} a_l^m + b_i^{m+1}\right)}{\partial n_j^m} = w_{i,j}^{m+1} \frac{\partial a_j^m}{\partial n_j^m} = w_{i,j}^{m+1} \frac{\partial f^m(n_j^m)}{\partial n_j^m} = w_{i,j}^{m+1} \dot{f}^m(n_j^m)$$

Where:

$$\dot{f}^m(n_j^m) = \frac{\partial f^m(n_j^m)}{\partial n_j^m}$$

Backpropagation Algorithm

- Then the Jacobian matrix can be written as:

$$\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^{m}} = \mathbf{W}^{m+1}\dot{\mathbf{F}}^{m}(\mathbf{n}^{m})$$

where:

$$\dot{\mathbf{F}}^{m}(\mathbf{n}^{m}) = \begin{bmatrix} \dot{f}^{m}(n_1^m) & 0 & \dots & 0 \\ 0 & \dot{f}^{m}(n_2^m) & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & \dot{f}^{m}(n_{S^m}^m) \end{bmatrix}$$

Backpropagation Algorithm

- Now we can write the recurrence relation for the sensitivity by using the chain rule in matrix form:

$$\mathbf{s}^m = \frac{\partial \hat{F}}{\partial \mathbf{n}^m} = \left(\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^m}\right)^T \frac{\partial \hat{F}}{\partial \mathbf{n}^{m+1}} = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T \frac{\partial \hat{F}}{\partial \mathbf{n}^{m+1}} = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T \mathbf{s}^{m+1}$$

- We can see that the sensitivities are propagated backward through the network from the last layer to the first layer:

$$\mathbf{s}^M \rightarrow \mathbf{s}^{M-1} \rightarrow \ldots \rightarrow \mathbf{s}^2 \rightarrow \mathbf{s}^1$$

Backpropagation Algorithm

- To complete the formulation of BP algorithm, we also need the starting point $\mathbf{s}^M$, for the recurrence relation.

$$s_i^M = \frac{\partial \hat{F}}{\partial n_i^M} = \frac{\partial (\mathbf{t}-\mathbf{a})^T(\mathbf{t}-\mathbf{a})}{\partial n_i^M} = \frac{\partial \sum_{j=1}^{S^M} (t_j - a_j)^2}{\partial n_i^M} = -2(t_i - a_i)\frac{\partial a_i}{\partial n_i^M}$$

- We know that:

$$\frac{\partial a_i}{\partial n_i^M} = \frac{\partial a_i^M}{\partial n_i^M} = \frac{\partial f^M(n_i^M)}{\partial n_i^M} = \dot{f}^M(n_i^M)$$

Backpropagation Algorithm

- Therefore the sensitivity of the last layer will be:

$$s_i^M = -2(t_i - a_i)\dot{f}^M(n_i^M)$$

- And in matrix form it can be written as:

$$\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t} - \mathbf{a})$$

✓ The beauty of backpropagation is that we have a very efficient implementation of the chain rule.

Backpropagation Algorithm

- **Summary:** The Backpropagation Algorithm has 3 steps:

1) Forward Propagation:

$$\mathbf{a}^0 = \mathbf{p}$$

$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^m + \mathbf{b}^{m+1}) \text{ for } m = 0, 1, \ldots, M-1$$

2) Backpropagating the sensitivities:

$$\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t}-\mathbf{a})$$

$$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T\mathbf{s}^{m+1}, \text{ for } m = M-1, \ldots, 2, 1$$

3) Updating the parameters:

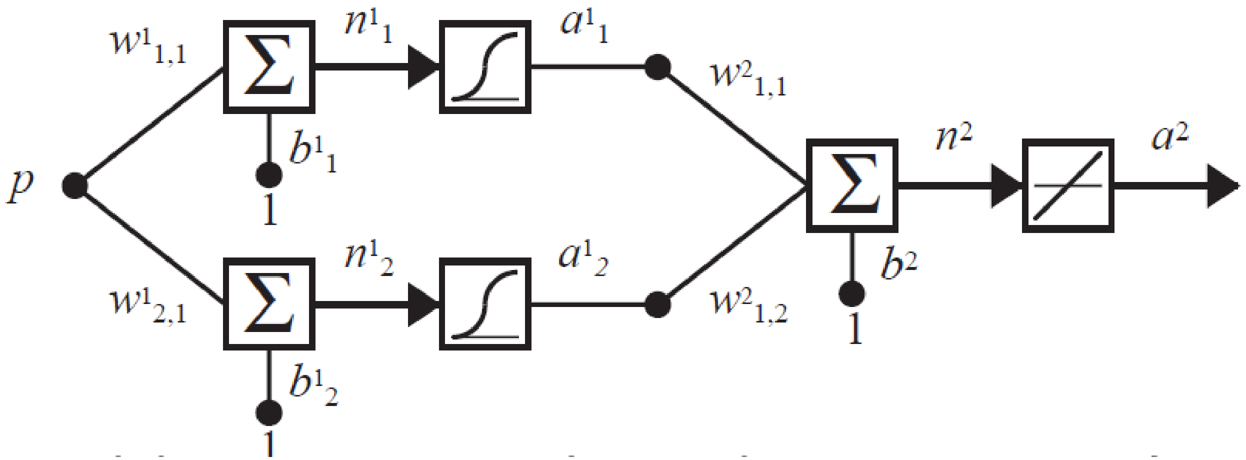
$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha\mathbf{s}^m(\mathbf{a}^{m-1})^T$$

$$\mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha\mathbf{s}^m.$$

Backpropagation Algorithm

1

- **Example:** Consider a 1-2-1 network with Sigmoid activation function in the hidden layer and Linear activation function in the output layer.



- We want to use this network to approximate the function:

$$g(p) = 1 + \sin\left(\frac{\pi}{4}p\right) \text{ for } -2 \le p \le 2$$

Backpropagation Algorithm

2

- To obtain our training set we will evaluate this function at several values of p.

- First we initialize the network weights & biases randomly. For example:

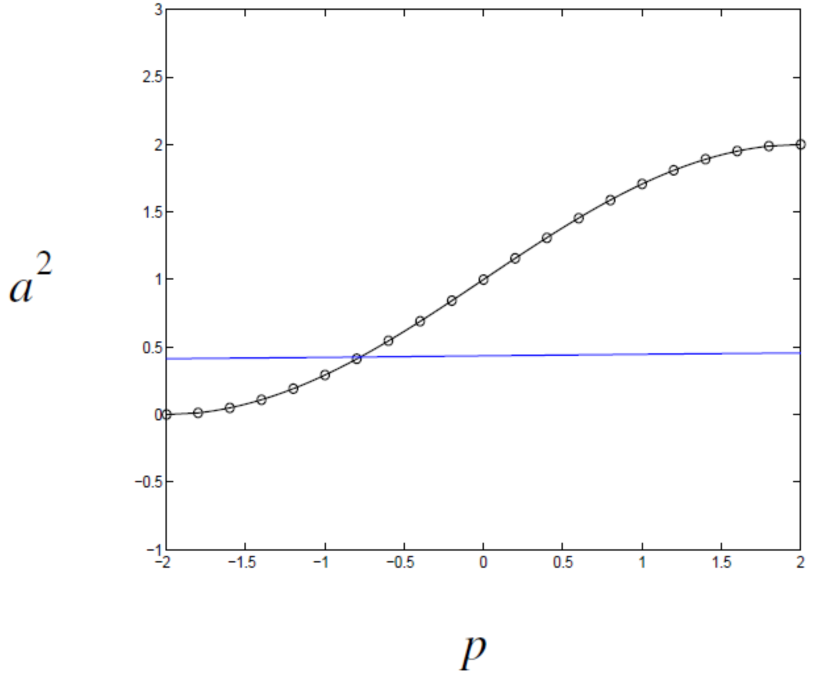
$$\mathbf{W}^1(0) = \begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix},\ \mathbf{b}^1(0) = \begin{bmatrix} -0.48 \\ -0.13 \end{bmatrix},\ \mathbf{W}^2(0) = \begin{bmatrix} 0.09 & -0.17 \end{bmatrix},\ \mathbf{b}^2(0) = \begin{bmatrix} 0.48 \end{bmatrix}$$

- Next, we need to select a training set $\{p_1, t_1\}, \{p_2, t_2\}, \ldots, \{p_Q, t_Q\}$. In this case we will sample the function at 21 points in the range [-2,2] at equally spaced intervals of 0.2.

Backpropagation Algorithm

3

- The response of the network for these initial values is:



- Now we are ready to start the algorithm.

Backpropagation Algorithm

1) Forward Propagation

4

- The training points can be presented in any order, but they are often chosen randomly. For our initial input we will choose $p = 1$, which is the 16th training point:

- The output of the first layer will be:

$$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{W}^1\mathbf{a}^0 + \mathbf{b}^1) = \mathbf{sig}\left(\begin{bmatrix}-0.27\\-0.41\end{bmatrix}\begin{bmatrix}1\end{bmatrix} + \begin{bmatrix}-0.48\\-0.13\end{bmatrix}\right) = \mathbf{sig}\left(\begin{bmatrix}-0.75\\-0.54\end{bmatrix}\right) = \begin{bmatrix}\dfrac{1}{1+e^{0.75}}\\[2ex]\dfrac{1}{1+e^{0.54}}\end{bmatrix} = \begin{bmatrix}0.321\\0.368\end{bmatrix}$$

Backpropagation Algorithm

5

- The output of the second layer will be:

$$a^2 = f^2(\mathbf{W}^2\mathbf{a}^1 + \mathbf{b}^2) = purelin\left(\begin{bmatrix} 0.09 & -0.17 \end{bmatrix}\begin{bmatrix} 0.321 \\ 0.368 \end{bmatrix} + \begin{bmatrix} 0.48 \end{bmatrix}\right) = \begin{bmatrix} 0.446 \end{bmatrix}$$

- Then the error would be:

$$e = t - a = \left\{1 + \sin\left(\frac{\pi}{4}p\right)\right\} - a^2 = \left\{1 + \sin\left(\frac{\pi}{4}1\right)\right\} - 0.446 = 1.261$$

Backpropagation Algorithm

6

2) Backpropagating the sensitivities

- We need the derivatives of the activation functions in both layers:

$$\dot{f}^{1}(n) = \frac{d}{dn}\left(\frac{1}{1+e^{-n}}\right) = \frac{e^{-n}}{(1+e^{-n})^{2}} = \left(1-\frac{1}{1+e^{-n}}\right)\left(\frac{1}{1+e^{-n}}\right) = (1-a^{1})(a^{1})$$

$$\dot{f}^{2}(n) = \frac{d}{dn}(n) = 1$$

- The second layer sensitivity is:

$$\mathbf{s}^{2} = -2\dot{\mathbf{F}}^{2}(\mathbf{n}^{2})(\mathbf{t}-\mathbf{a}) = -2\left[\dot{f}^{2}(n^{2})\right](1.261) = -2\left[1\right](1.261) = -2.522$$

Backpropagation Algorithm

7

- The first layer sensitivity is:

$$\mathbf{s}^1 = \dot{\mathbf{F}}^1(\mathbf{n}^1)(\mathbf{W}^2)^T\mathbf{s}^2 = \begin{bmatrix}(1-a_1^1)(a_1^1) & 0 \\ 0 & (1-a_2^1)(a_2^1)\end{bmatrix}\begin{bmatrix}0.09 \\ -0.17\end{bmatrix}\begin{bmatrix}-2.522\end{bmatrix}$$

$$= \begin{bmatrix}(1-0.321)(0.321) & 0 \\ 0 & (1-0.368)(0.368)\end{bmatrix}\begin{bmatrix}0.09 \\ -0.17\end{bmatrix}\begin{bmatrix}-2.522\end{bmatrix}$$

$$= \begin{bmatrix}0.218 & 0 \\ 0 & 0.233\end{bmatrix}\begin{bmatrix}-0.227 \\ 0.429\end{bmatrix} = \begin{bmatrix}-0.0495 \\ 0.0997\end{bmatrix}.$$

Backpropagation Algorithm

8

3) Updating the parameters

- We use a learning rate $\alpha = 0.1$

- The parameters of the second layer will be changed to:

$$\mathbf{W}^2(1) = \mathbf{W}^2(0) - \alpha \mathbf{s}^2(\mathbf{a}^1)^T = \begin{bmatrix} 0.09 & -0.17 \end{bmatrix} - 0.1 \begin{bmatrix} -2.522 \end{bmatrix} \begin{bmatrix} 0.321 & 0.368 \end{bmatrix}$$

$$= \begin{bmatrix} 0.171 & -0.0772 \end{bmatrix},$$

$$\mathbf{b}^2(1) = \mathbf{b}^2(0) - \alpha \mathbf{s}^2 = \begin{bmatrix} 0.48 \end{bmatrix} - 0.1 \begin{bmatrix} -2.522 \end{bmatrix} = \begin{bmatrix} 0.732 \end{bmatrix},$$

Backpropagation Algorithm

9

- And the parameters of the first layer will be:

$$\mathbf{W}^1(1) = \mathbf{W}^1(0) - \alpha \mathbf{s}^1 (\mathbf{a}^0)^T = \begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix} - 0.1 \begin{bmatrix} -0.0495 \\ 0.0997 \end{bmatrix} \begin{bmatrix} 1 \end{bmatrix} = \begin{bmatrix} -0.265 \\ -0.420 \end{bmatrix},$$

$$\mathbf{b}^1(1) = \mathbf{b}^1(0) - \alpha \mathbf{s}^1 = \begin{bmatrix} -0.48 \\ -0.13 \end{bmatrix} - 0.1 \begin{bmatrix} -0.0495 \\ 0.0997 \end{bmatrix} = \begin{bmatrix} -0.475 \\ -0.140 \end{bmatrix}.$$

- This completes the first iteration of the backpropagation algorithm. We next choose another input randomly from the training set and perform another iteration of the algorithm (an iteration includes all 3 steps).
- We continue this process several epochs until the difference between the network response and the target function reaches some acceptable level.

Batch vs. Incremental Learning

- The BP algorithm described is the stochastic gradient descent algorithm, which involves on-line or incremental training, in which the network weights and biases are updated after each input is presented.

- It is also possible to perform batch training, in which the complete gradient is computed (after all inputs are applied to the network) before the weights and biases are updated.

- If each input occurs with equal probability, the MSE loss can be written

$$F(\mathbf{x}) = E[\mathbf{e}^T\mathbf{e}] = E[(\mathbf{t}-\mathbf{a})^T(\mathbf{t}-\mathbf{a})] = \frac{1}{Q}\sum_{q=1}^{Q}(\mathbf{t}_q-\mathbf{a}_q)^T(\mathbf{t}_q-\mathbf{a}_q)$$

Batch vs. Incremental Learning

- The total gradient of this loss function is:

$$\nabla F(\mathbf{x}) = \nabla\left\{\frac{1}{Q}\sum_{q=1}^{Q}(\mathbf{t}_q - \mathbf{a}_q)^T(\mathbf{t}_q - \mathbf{a}_q)\right\} = \frac{1}{Q}\sum_{q=1}^{Q}\nabla\{(\mathbf{t}_q - \mathbf{a}_q)^T(\mathbf{t}_q - \mathbf{a}_q)\}$$

✓ The total gradient of the mean square error is the mean of the gradients of the individual squared errors.

- Therefore, to implement a batch version of the backpropagation algorithm, we should perform the first and the second steps of the BP algorithm for all of the inputs in the training set.

Batch vs. Incremental Learning

- Then, the individual gradients should be averaged to get the total gradient. The update equations for the batch gradient descent algorithm would then be:

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \frac{\alpha}{Q}\sum_{q=1}^{Q}\mathbf{s}_q^m(\mathbf{a}_q^{m-1})^T$$

$$\mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \frac{\alpha}{Q}\sum_{q=1}^{Q}\mathbf{s}_q^m .$$

Choice of Network Architecture

- As we discussed earlier, multilayer networks can be used to approximate almost any function, if we have enough neurons in the hidden layer.

- We cannot say, in general, how many layers or how many neurons are necessary for adequate performance.

- Let's assume that we want to approximate the following functions:

$$g(p) = 1 + \sin\left(\frac{i\pi}{4}p\right) \text{ for } -2 \le p \le 2$$

where i takes on the values 1, 2, 4 and 8.

Choice of Network Architecture

- As i is increased, the function becomes more complex, because we will have more periods of the sine wave over the interval $-2 \leq p \leq 2$.

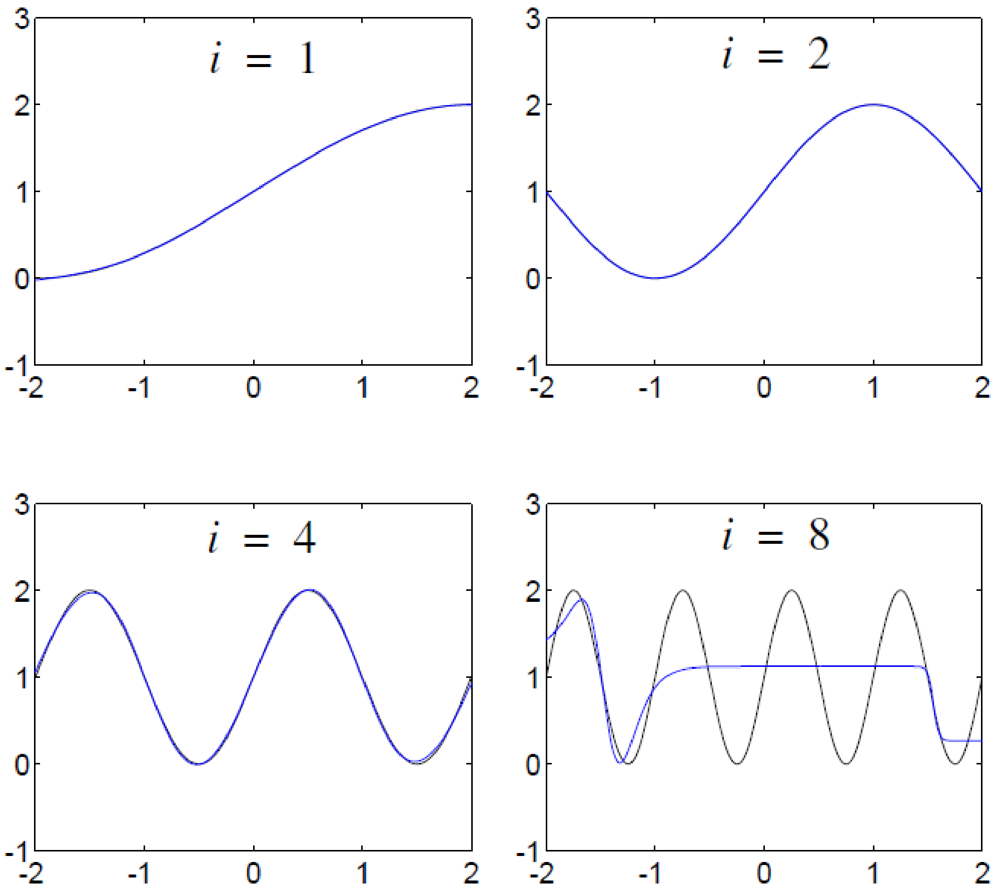
- It will be more difficult for a neural network with a fixed number of neurons in the hidden layers to approximate $g(p)$ as i is increased.

❑ **Case 1:** we will use a 1-3-1 network to approximate these functions where the activation function for the first and the second layer are Sigmoid and Linear, respectively.

Choice of Network Architecture

- This type of two-layer network can produce a response that is a sum of three Sigmoid functions (due to the three neurons in the hidden layer).

- The final response (after training) of this 1-3-1 network for approximating each of $g(p)$ functions will be:



Choice of Network Architecture

- We can see that for $i = 4$ the 1-3-1 network reaches its maximum capability. When $i > 4$ the network is not capable of producing an accurate approximation of $\mathrm{g}(p)$.

- The 1-3-1 network attempts to approximate $\mathrm{g}(p)$ for $i = 8$. Although the MSE loss between the network response and $\mathrm{g}(p)$ is minimized, the network response is only able to match a small part of the function.

Choice of Network Architecture

- **Case 2:** This time we will pick one function $g(p)$ and then use larger and larger networks until we are able to accurately represent the function. For $g(p)$ we will use:

$$g(p) = 1 + \sin\left(\frac{6\pi}{4}p\right) \text{ for } -2 \le p \le 2$$

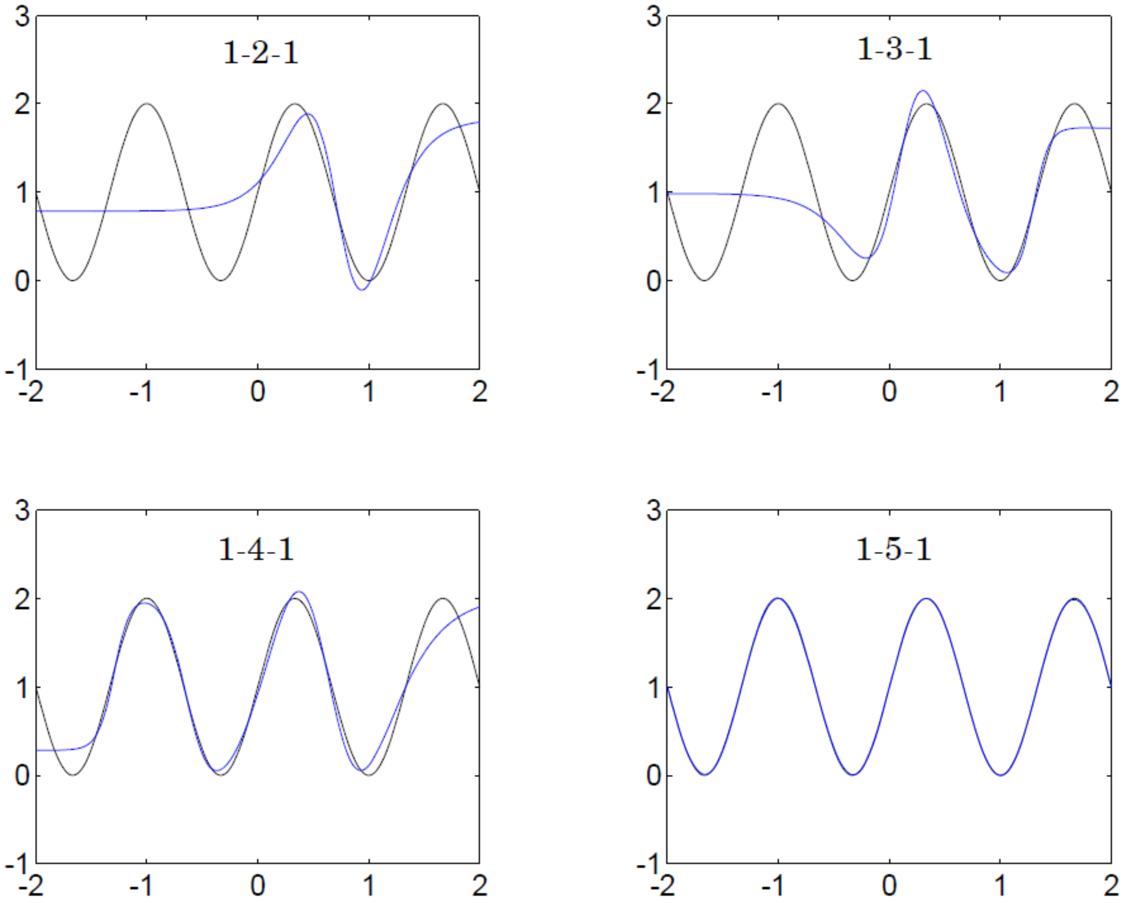
- To approximate this function we will use a two-layer network where the activation function for the first and the second layer are Sigmoid and Linear, respectively (1-S^1-1 network).

- The response of this network is a superposition of S^1 sigmoid functions

Choice of Network Architecture

- This figure illustrates the network response as the number of neurons in the first layer (hidden layer) is increased.

- Unless there are at least five neurons in the hidden layer the network cannot accurately represent $g(p)$.



Choice of Network Architecture

❖ To summarize these results:

- A $1\text{-}S^1\text{-}1$ network, with Sigmoid neurons in the hidden layer and Linear neuron in the output layer, can produce a response that is a superposition of sigmoid functions.

- If we want to approximate a function that has a large number of inflection points, we will need to have a large number of neurons in the hidden layer.

Drawbacks of SDBP

- The Backpropagation algorithm that has been discussed so far, was purely based on the Steepest (Gradient) descent algorithm. Therefore we will refer to that as SDBP algorithm.

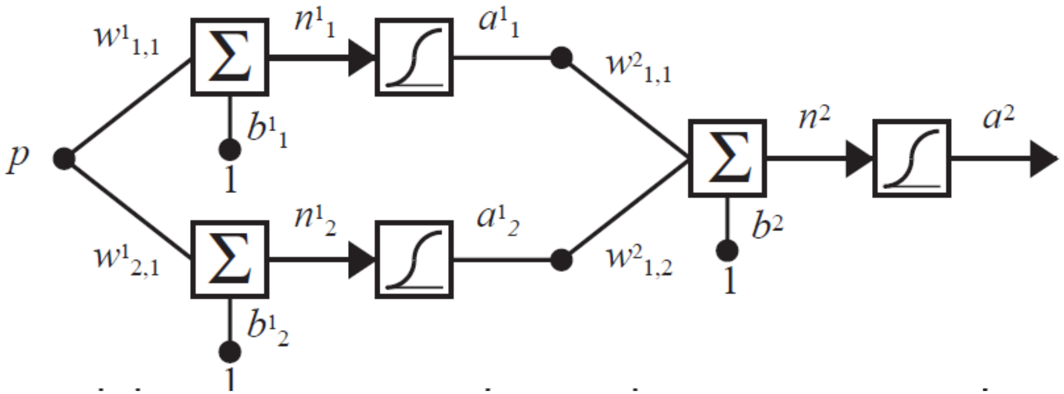
- SDBP algorithm for single-layer linear networks is guaranteed to converge to a solution that minimizes the mean squared error, because the MSE for a single-layer linear network is a quadratic function (has only a single stationary point).

- Additionally, the Hessian matrix of a quadratic function is constant, therefore the curvature of the MSE function in a given direction does not change, and the function contours are elliptical.

Drawbacks of SDBP

✓ As opposed to single-layer linear networks, the MSE loss function for a multilayer network may have many local minimum points, and the curvature can vary widely in different regions of the parameter space.

❑ **Example:** we will use a simple example to explain why SDBP has problems with convergence: Consider a 1-2-1 network with Sigmoid activation functions in both layers for function approximation.

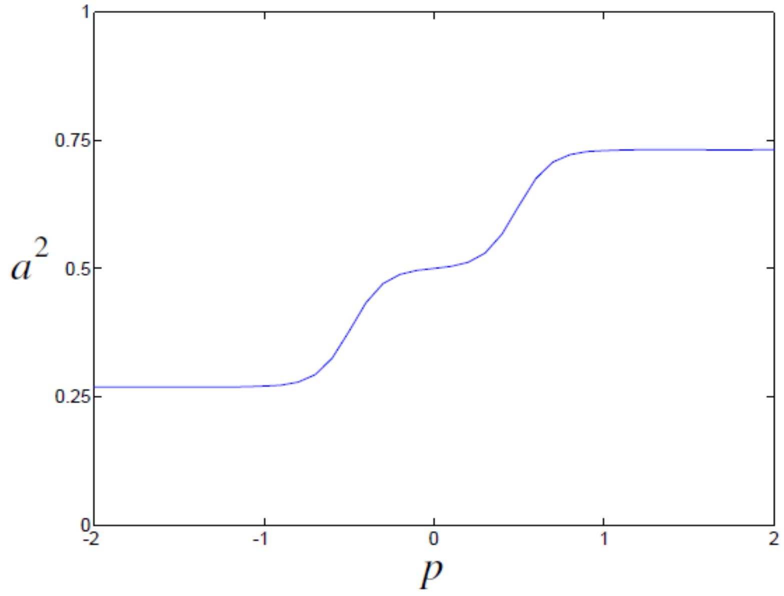


Drawbacks of SDBP

- Suppose that The function we will approximate corresponds to the response of this network to the following values for the weights and biases (The optimal parameters):

$$w^1_{1,1} = 10\,,\; w^1_{2,1} = 10\,,\; b^1_1 = -5\,,\; b^1_2 = 5 \qquad w^2_{1,1} = 1\,,\; w^2_{1,2} = 1\,,\; b^2 = -1$$

- The function we want to approximate (the network response in [-2,2] for these parameters) is:



- We want to train the 1-2-1 network to approximate this function.

Drawbacks of SDBP

- We assume that the function is sampled at the values:

$$p = -2, -1.9, -1.8, \dots, 1.9, 2$$

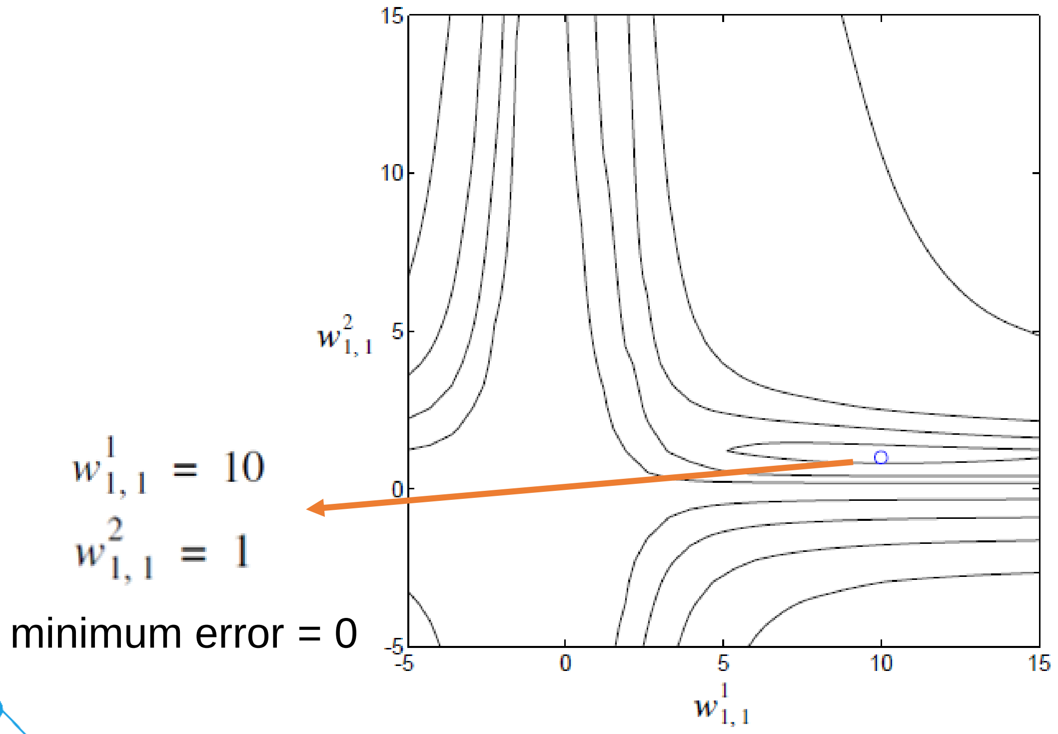
  with equal probability to form the training set.

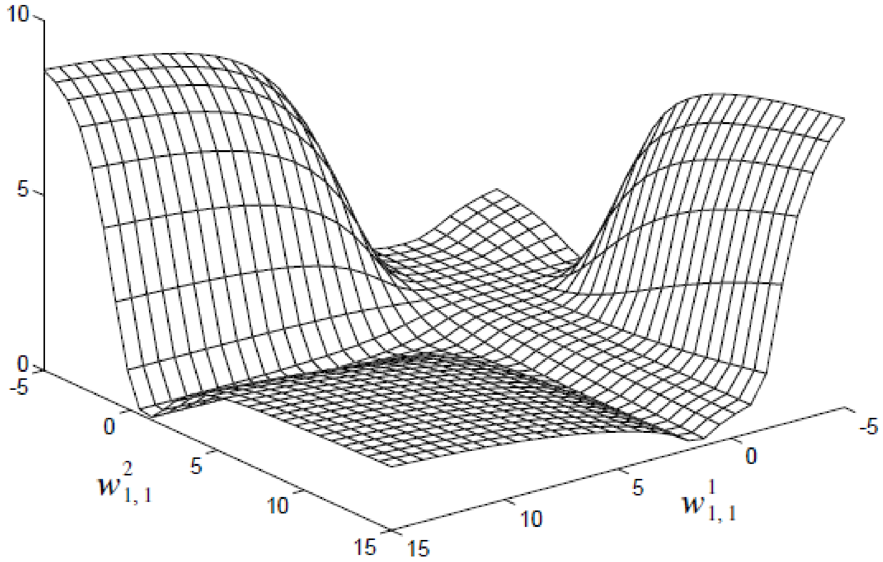
- The loss function will be MSE at these 41 points.

- In order to be able to graph the loss function, we will vary only two parameters at a time and the other parameters are set to their optimal values.

Drawbacks of SDBP

- **Case 1:** $w^1_{1,1}$ and $w^2_{1,1}$ are being adjusted, while the other parameters are set to their optimal values. The MSE surface will be:



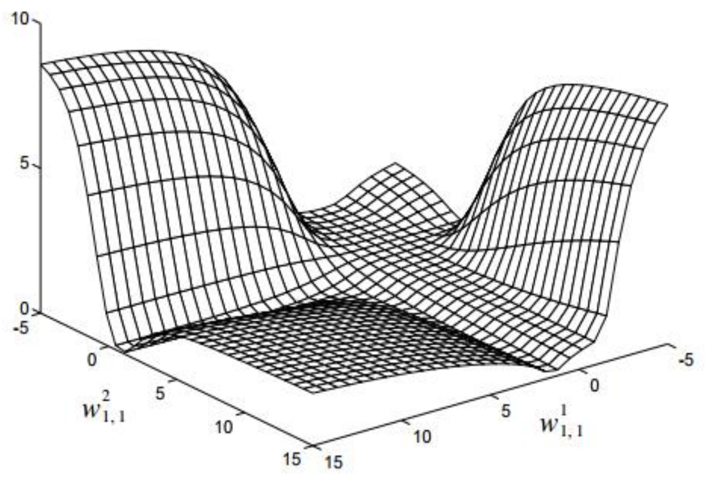


Squared Error Surface Versus and $w^1_{1,1}$ and $w^2_{1,1}$

Drawbacks of SDBP

Some Notes:

- It is clearly not a quadratic function.
- The curvature varies drastically over the parameter space. For this reason it will be difficult to choose an appropriate learning rate for the steepest descent algorithm.
  - In some regions the surface is very flat, which would allow a large learning rate, while in other regions the curvature is high, which would require a small learning rate.
- The flat regions of the loss surface is made by the Sigmoid activation function (Sigmoid output is flat for large inputs).

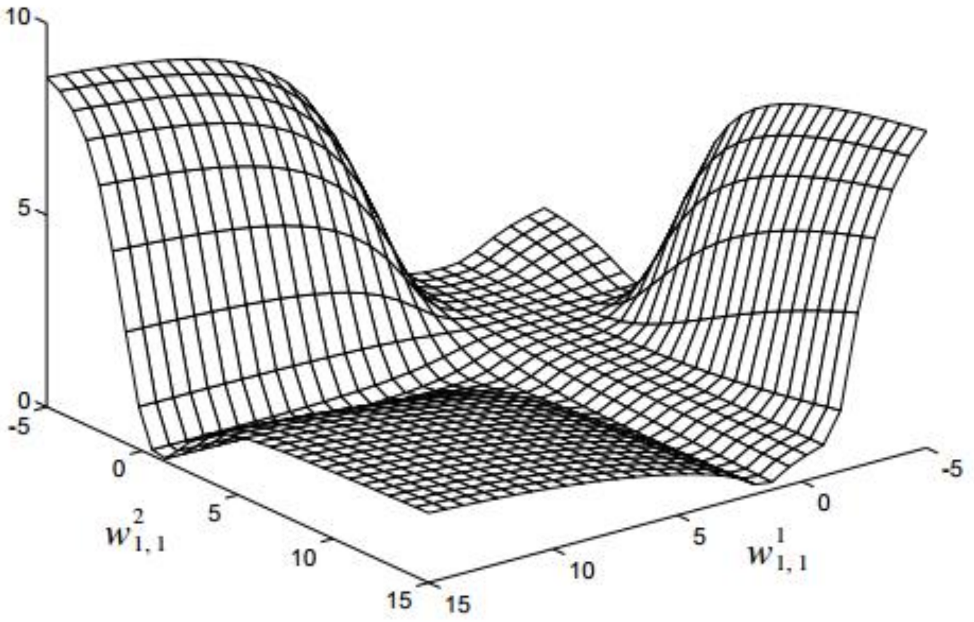


Drawbacks of SDBP

- A second feature of this error surface is the existence of more than one local minimum point.

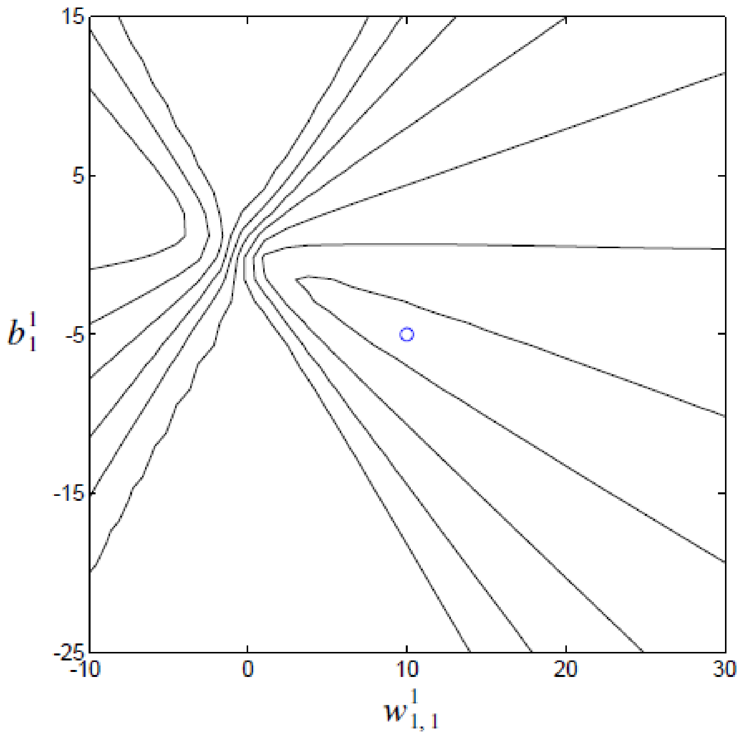
- The global minimum point is located at $w^1_{1,1}$=10 and $w^2_{1,1}$=1 , along the valley that runs parallel to the $w^1_{1,1}$ axis.

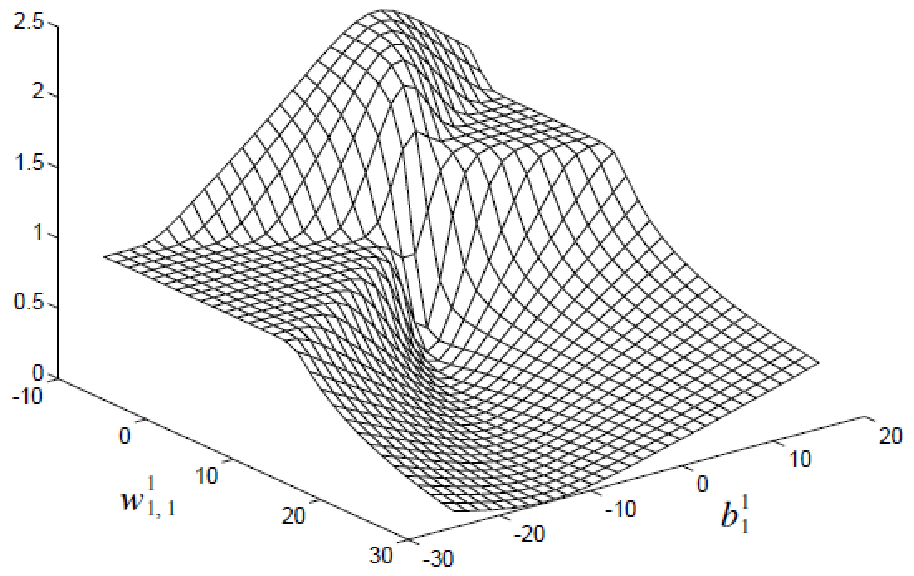
- However, there is also a local minimum, which is located in the valley that runs parallel to the $w^2_{1,1}$ axis.



Drawbacks of SDBP

- **Case 2:** $w^1_{1,1}$ and b^1_1 are being adjusted, while the other parameters are set to their optimal values. The MSE surface will be:





Drawbacks of SDBP

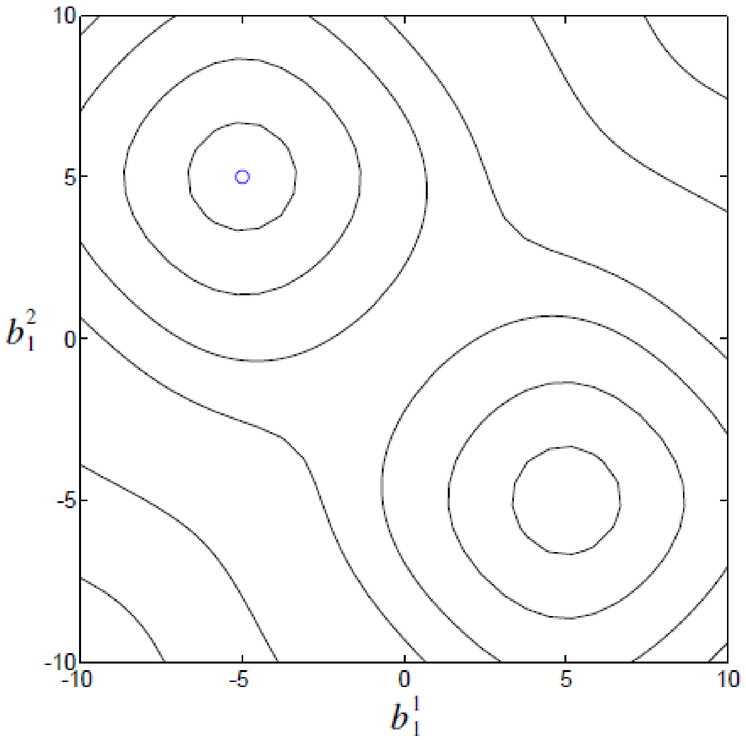
- The minimum error will be zero and it will occur when $w_{1,1}^{1} = 10$ and $b_{1}^{1} = -5$ indicated by the open blue circle in the figure.

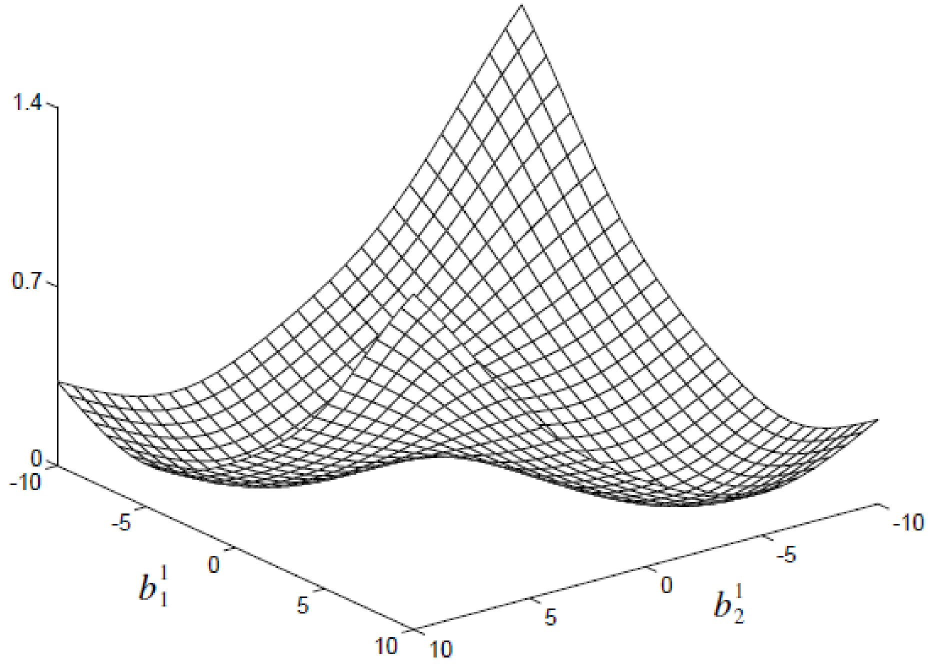
- The loss surface has a very contorted shape, steep in some regions and very flat in others. Therefore the standard Gradient Descent algorithm will have some trouble with this surface.

- For example, if we have an initial guess of $w_{1,1}^{1} = 0$ and $b_{1}^{1} = -10$, the gradient will be very close to zero, and the Gradient Descent algorithm will stop, although it is not close to a local minimum point.

Drawbacks of SDBP

- **Case 3:** b_1^1 and b_2^1 are being adjusted, while the other parameters are set to their optimal values. The MSE surface will be:





Drawbacks of SDBP

- The minimum error will be zero and it will occur when $b_1^1 = -5$ and $b_2^1 = 5$ indicated by the open blue circle in the figure.

- There are two local minimum points and they both have the same value of MSE. The second solution corresponds to the same network being turned upside down (the top neuron in the first layer is exchanged with the bottom neuron).

- It is because of this characteristic of neural networks that we do not set the initial weights and biases to zero. The symmetry causes zero to be a saddle point of the loss surface.

Drawbacks of SDBP

- The study of these 3 cases about loss surfaces for multilayer networks gives us some hints as to how to set the initial guess for the SDBP algorithm:

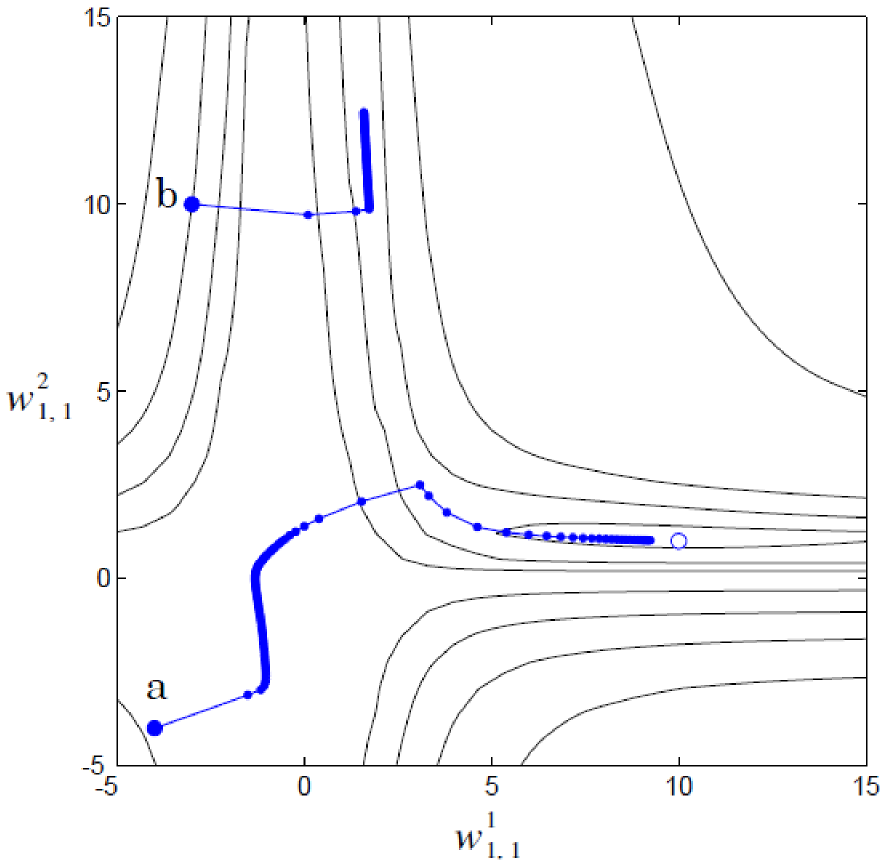
1. We do not set the initial parameters to zero. This is because the origin of the parameter space tends to be a saddle point for the loss surface.

2. We do not set the initial parameters to large values. This is because the loss surface tends to have very flat regions as we move far away from the optimum point.

3. We choose the initial weights and biases to be small random values.

Drawbacks of SDBP

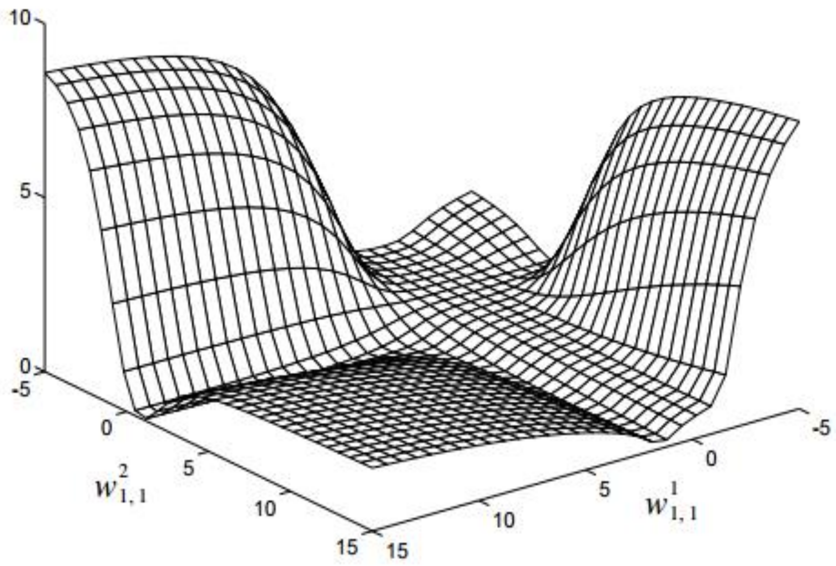
- Consider again the case 1 when $w^{1}_{1,1}$ and $w^{2}_{1,1}$ were the learning parameters. Two trajectories (correspond to different initializations) of SDBP are shown in the figure, labeled "a" and "b".

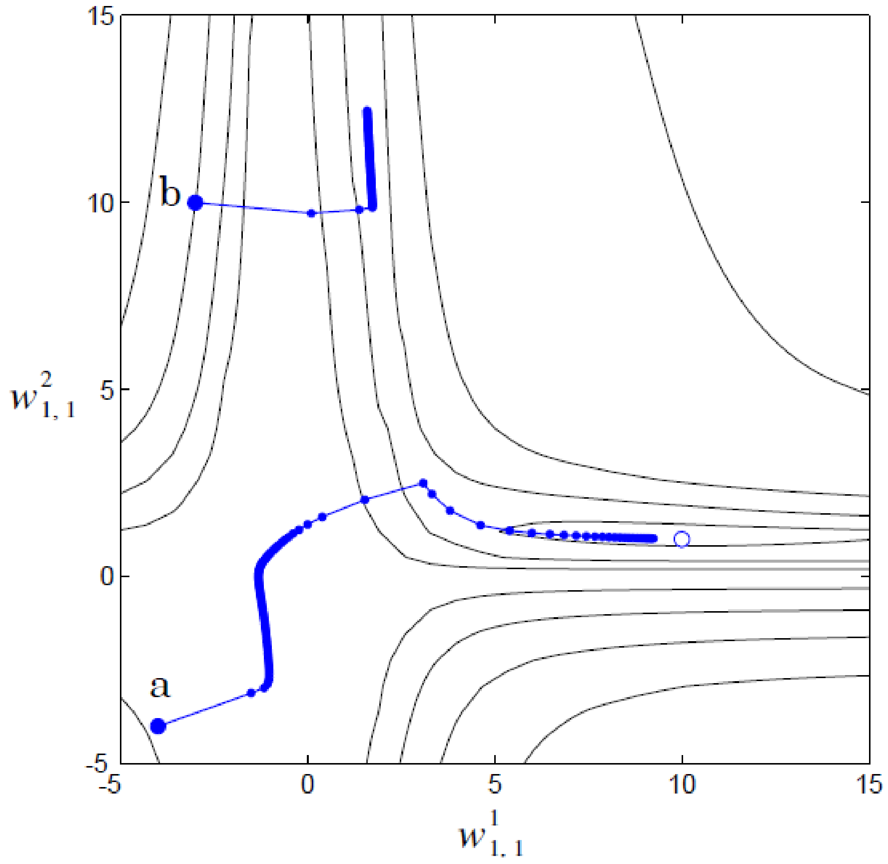


- **a)** For the initial condition labeled "a" the algorithm converges to the optimal solution, but the convergence is slow.

Drawbacks of SDBP

- The reason for the slow convergence is the change in curvature of the surface over the path of the trajectory.
- After an initial moderate slope, the trajectory passes over a very flat surface, until it falls into a very gently sloping valley.

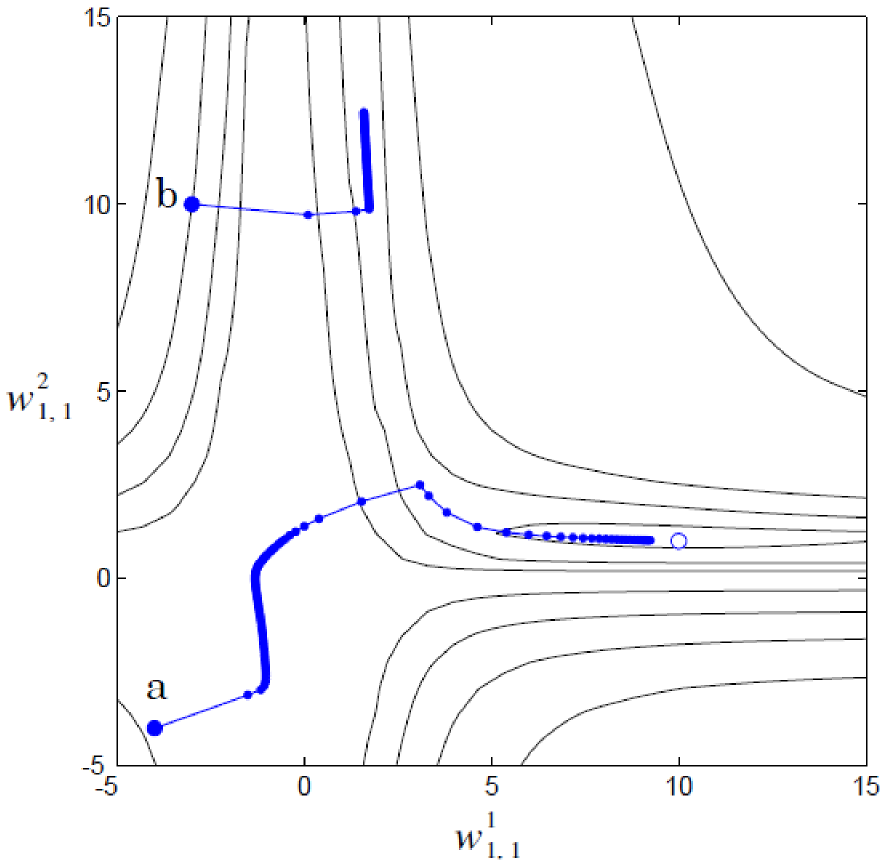




Drawbacks of SDBP

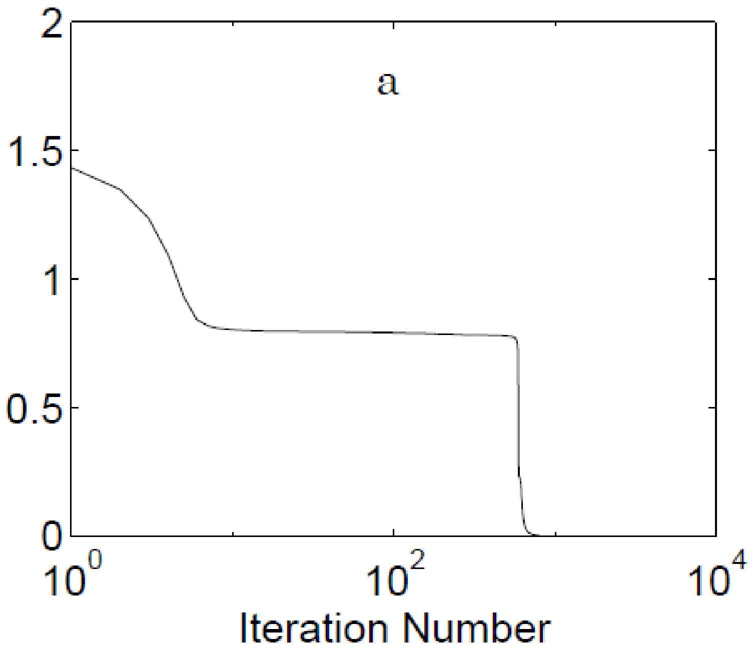
- **b)** Trajectory "b" illustrates how the algorithm can converge to a local minimum point. The trajectory is trapped in a valley and diverges from the optimal solution.

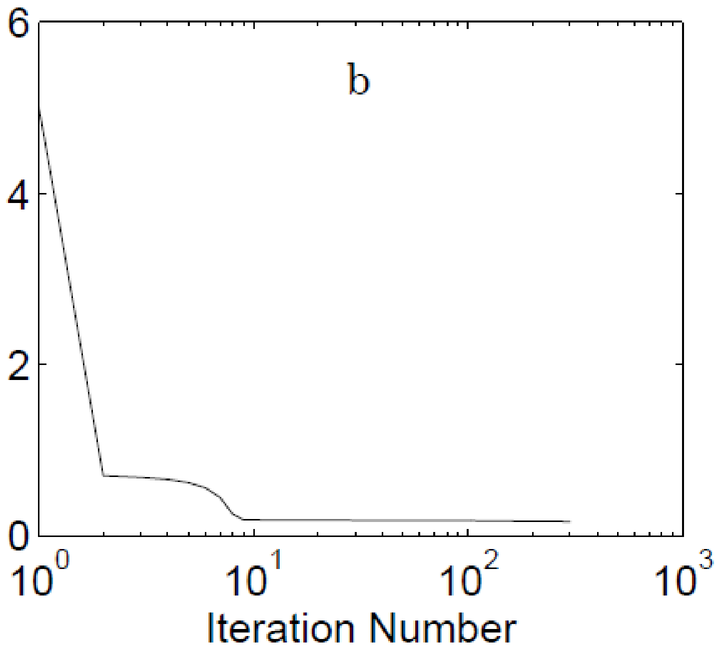
- The existence of multiple local minimum points is typical of the performance surface of multilayer networks. For this reason it is best to try several different initial guesses in order to ensure that a global minimum has been obtained.



Drawbacks of SDBP

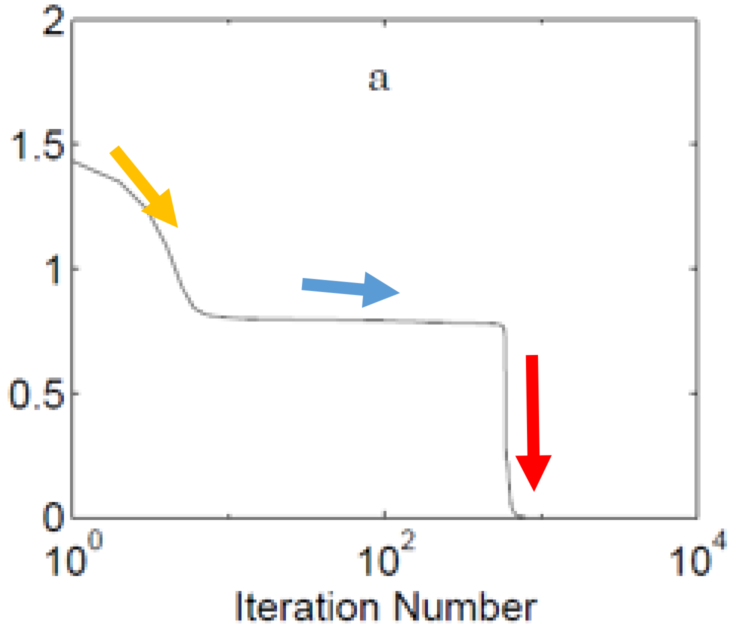
❖ The progress of the algorithm can also be seen for trajectory “a” and “b”, which shows the MSE versus the iteration number.

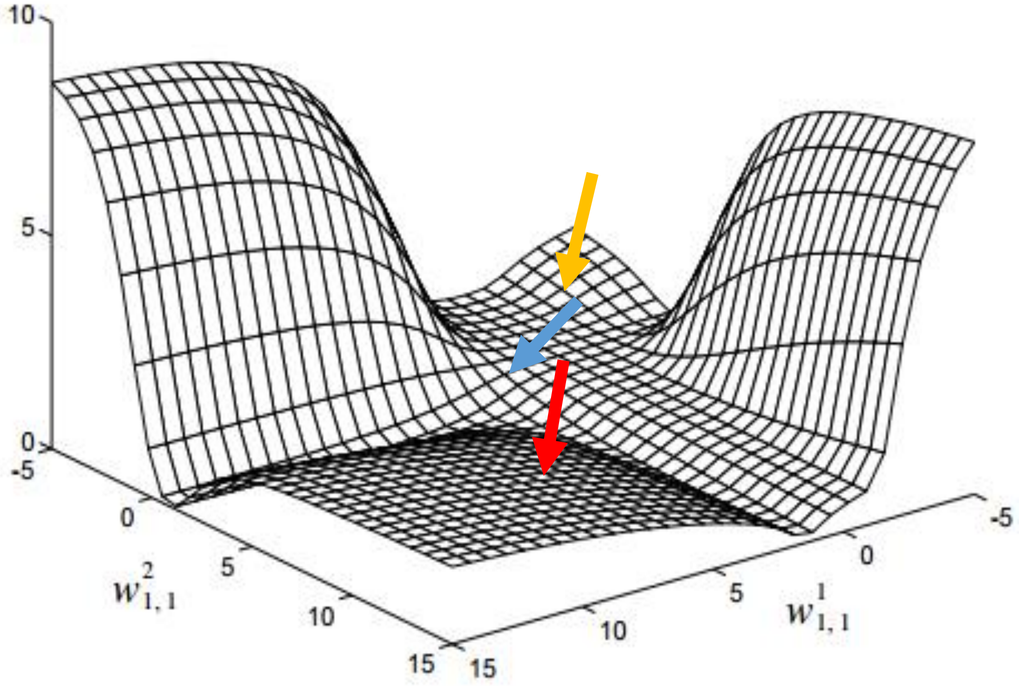




Drawbacks of SDBP

- We can see that the flat sections in the progress curvature correspond to times when the algorithm is traversing a flat section of the loss surface.

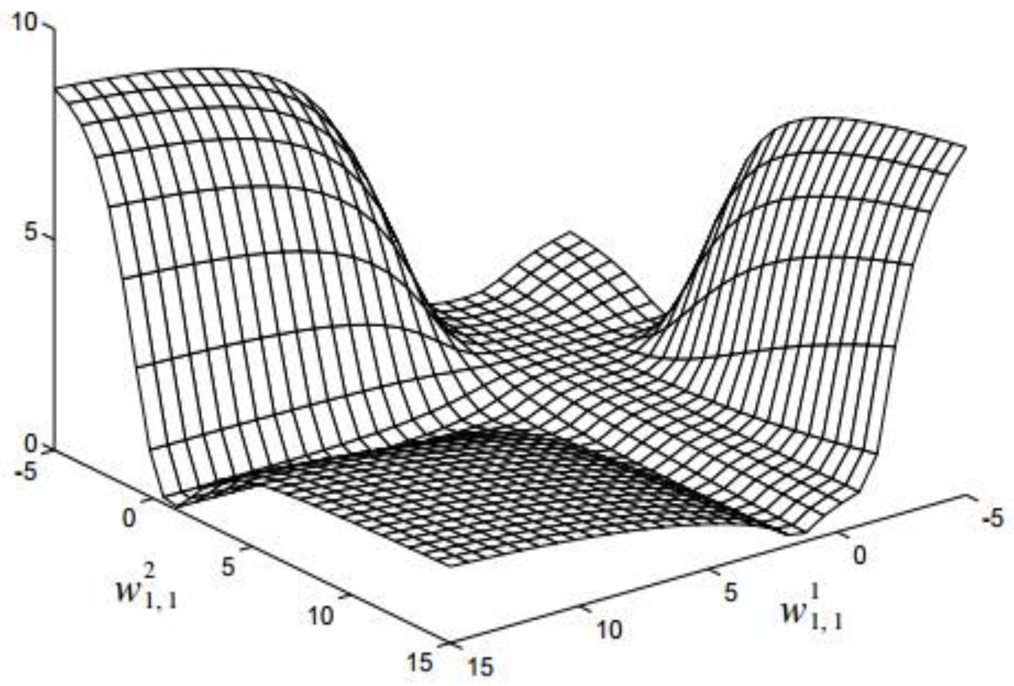




Drawbacks of SDBP

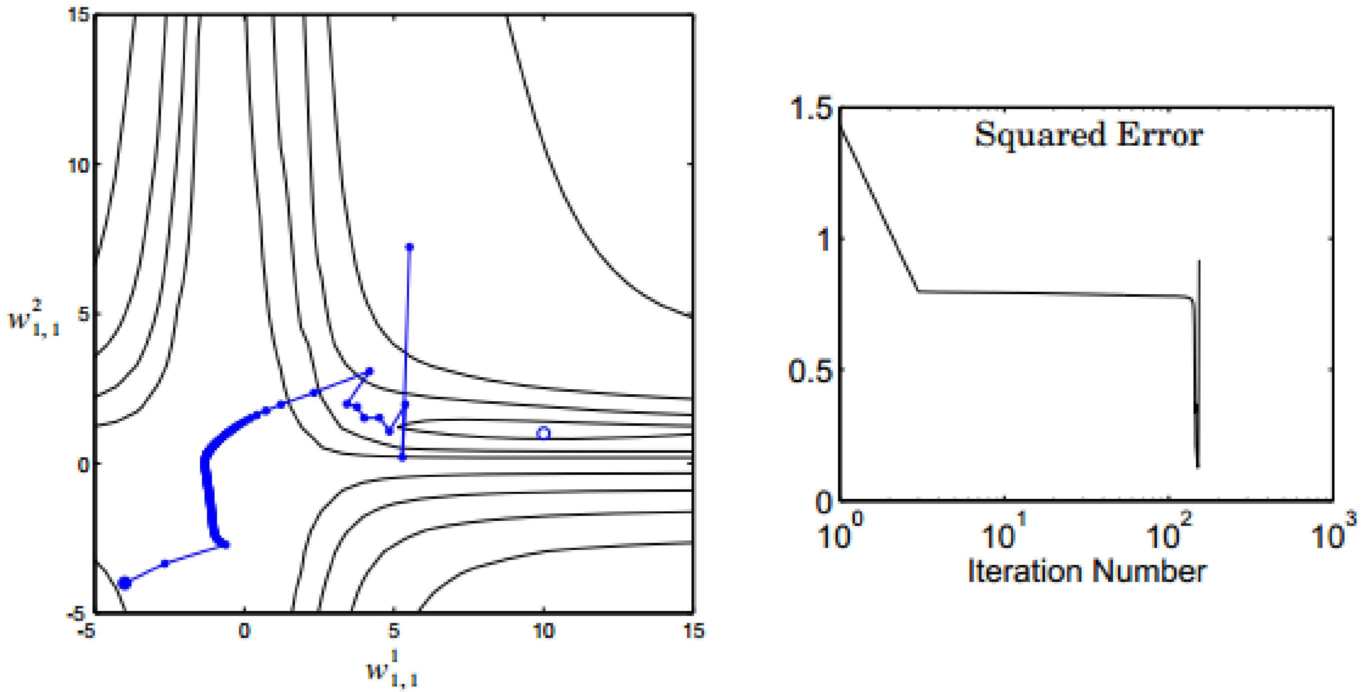
- During these periods (i.e. flat sections) we would like to increase the learning rate, in order to speed up convergence.

- However, if we increase the learning rate the algorithm will become unstable when it reaches steeper portions of the performance surface.



Drawbacks of SDBP

- By using a larger learning rate, the algorithm converges faster at first, but when the trajectory reaches the narrow valley that contains the minimum point the algorithm begins to diverge.



Trajectory with Learning Rate Too Large

Drawbacks of SDBP

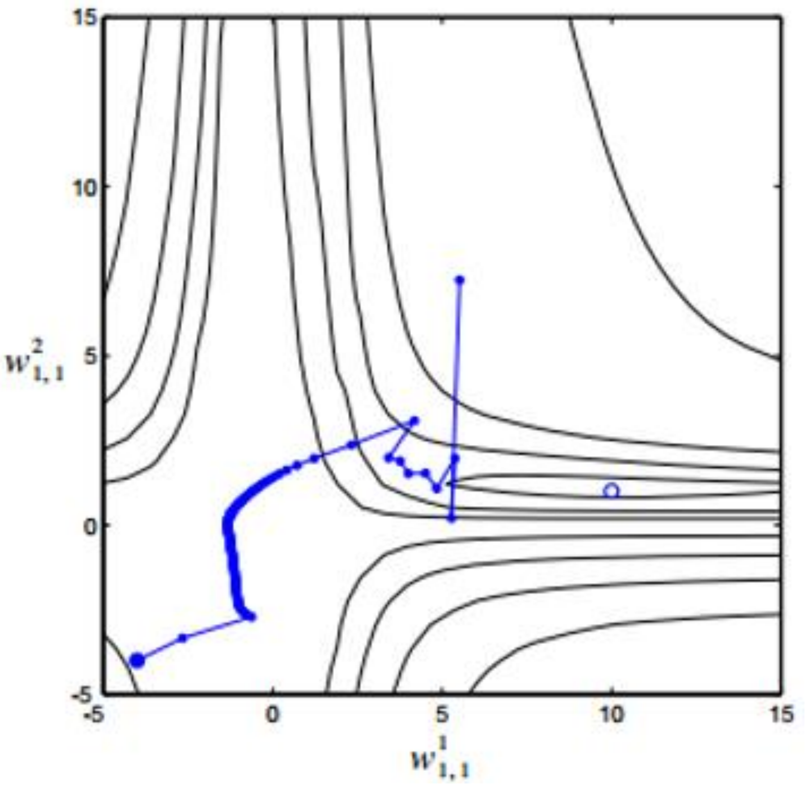
- This suggests that it would be useful to vary the learning rate:
  - increase the learning rate on flat surfaces and,
  - then decrease the learning rate as the slope increased.

- **The question is: "How will the algorithm know when it is on a flat surface?"**

✓ We will see the answer to this question in the future.

Drawbacks of SDBP

- Another way to improve convergence would be to smooth out the trajectory.
- Note in the figure that when the algorithm begins to diverge it is oscillating back and forth across a narrow valley.
- If we could filter the trajectory, by averaging the updates to the parameters, this might smooth out the oscillations and produce a stable trajectory.
- We will discuss this procedure in the next section.



Drawbacks of SDBP

- This example shows that the SDBP algorithm has problems with convergence and might be trapped in local minimum points of loss surface and may not converge to the globally optimal solution.

✓ Due to these behaviors of the SDBP (vanilla backpropagation), some variations of BP has been introduced.

✓ We will see two such variations in the future.

Momentum

- One of the procedures for improving the SDBP algorithm is the use of momentum.

- The basic idea behind this concept is that convergence might be improved if we could smooth out the oscillations in the trajectory.

- The SDBP does not care about what the earlier gradients were, but the momentum optimization does.

- As the optimization procedure goes forward, momentum pays less attention to earlier gradients.

Momentum

- To apply the momentum, we should use a low-pass filter like this:

$$y(k) = \gamma y(k-1) + (1-\gamma)w(k)$$

$w(k)$:the input to the filter

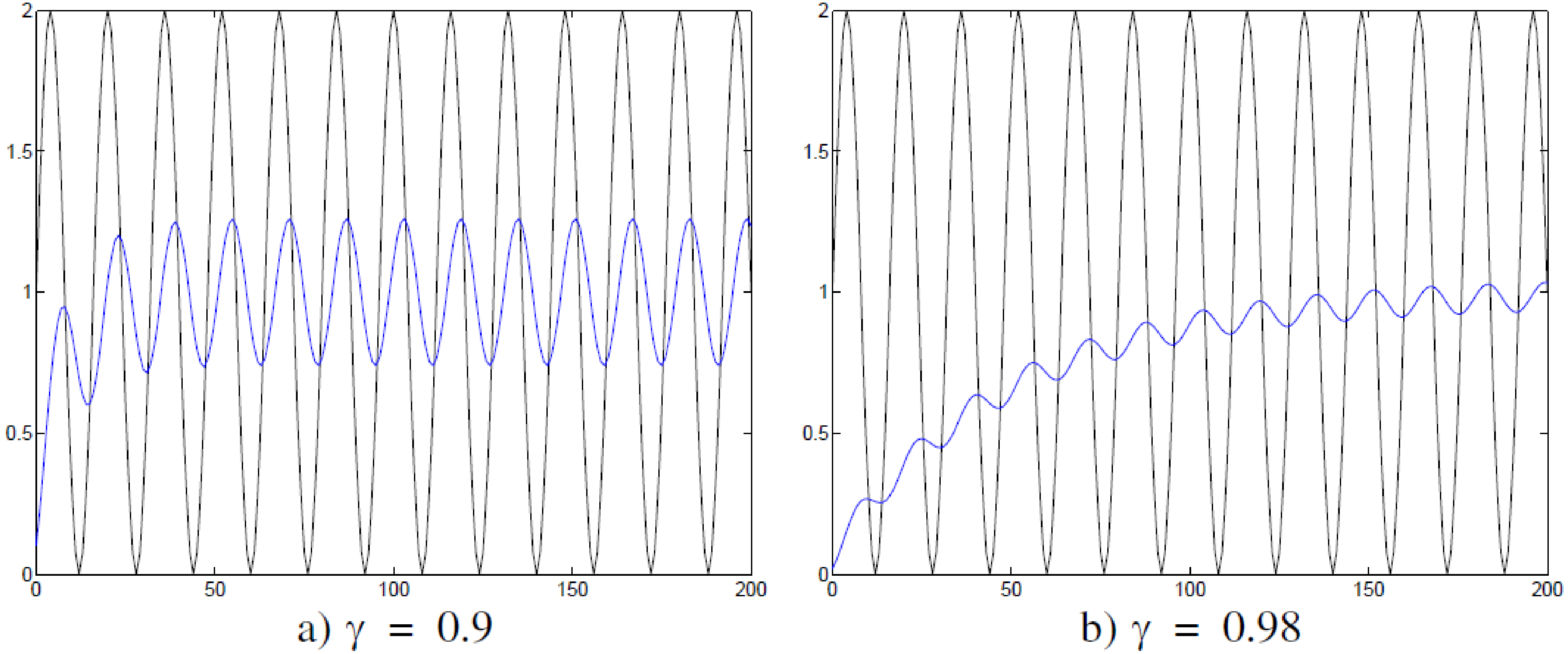
$y(k)$:the output of the filter

γ :the momentum coefficient $0 \leq \gamma < 1$

- The typical momentum coefficient is 0.9

Momentum

- For the input $w(k) = 1 + \sin(\frac{2\pi k}{16})$ (the black lines), the effect of this filter (the blue lines) will be:



Smoothing Effect of Momentum

Momentum

- The oscillation of the filter output is less than the oscillation in the filter input.

- As γ is increased the oscillation in the filter output is reduced.

- The average filter output is the same as the average filter input, although as γ is increased the filter output is slower to respond.

- In the optimization procedure of neural networks, Momentum tends to make the trajectory continue in the same direction. The larger the value of γ, the more momentum the trajectory has.

Momentum

- Let's see how momentum works on the optimization procedure in neural networks. The parameter updates for SDBP (without momentum) was:

$$\Delta \mathbf{W}^m(k) = -\alpha \mathbf{s}^m (\mathbf{a}^{m-1})^T$$

$$\Delta \mathbf{b}^m(k) = -\alpha \mathbf{s}^m .$$

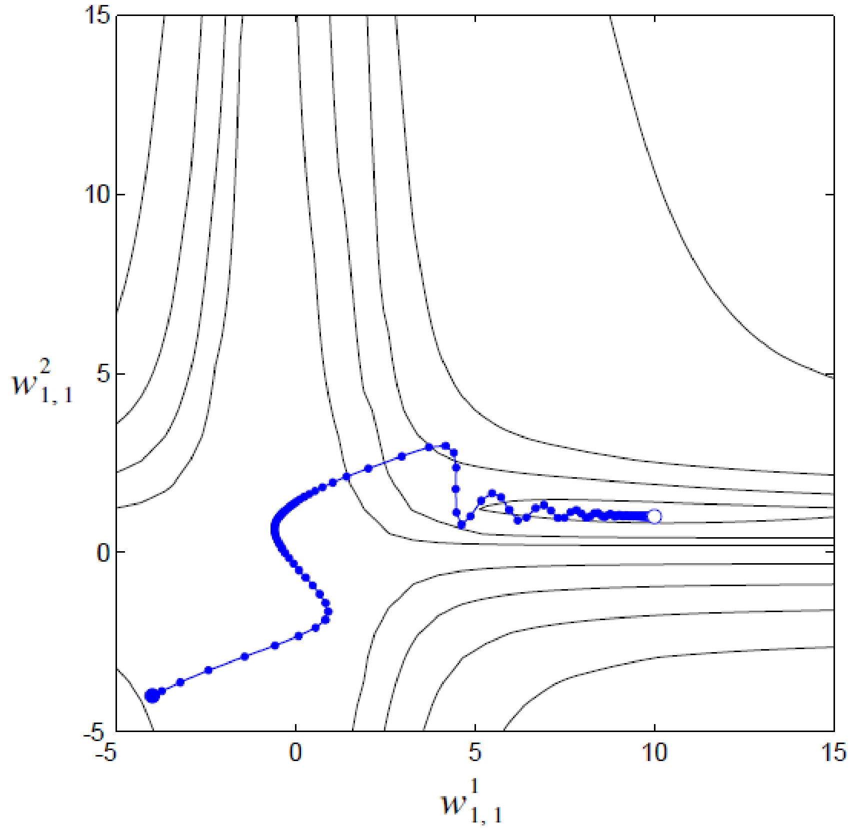
- When the momentum filter is added to the parameter changes, we obtain the following equations for the momentum modification to backpropagation (MOBP).

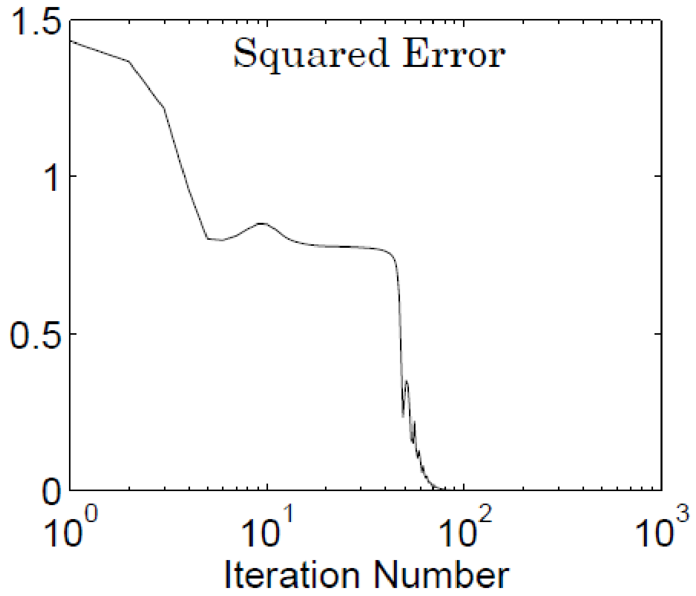
$$\Delta \mathbf{W}^m(k) = \gamma \Delta \mathbf{W}^m(k-1) - (1-\gamma)\alpha \mathbf{s}^m (\mathbf{a}^{m-1})^T$$

$$\Delta \mathbf{b}^m(k) = \gamma \Delta \mathbf{b}^m(k-1) - (1-\gamma)\alpha \mathbf{s}^m .$$

Momentum

- If we now apply the batching form of MOBP with $\gamma = 0.8$ to the previous example (Figure at slide 68), we obtain this result:





Momentum

- Although the initial condition and the learning rate are the same as before, We can see that the algorithm is now stable.

- By the use of momentum we have been able to use a larger learning rate, while maintaining the stability of the algorithm.

- Momentum tends to accelerate convergence when the trajectory is moving in a consistent direction.

Variable Learning Rate

- If we increase the learning rate on flat surfaces and then decrease it when the slope increases, we might be able to speed up convergence.

- For single-layer linear networks, the MSE function is always quadratic and the Hessian matrix is constant. Therefore The maximum stable learning rate for the gradient descent is $\frac{2}{\lambda_{\max}}$.

- But for multilayer networks, The MSE loss is not a quadratic function. The shape of the surface can be very different in different regions of the parameter space.

Variable Learning Rate

- There are many different approaches for varying the learning rate. One of the most popular procedures is where the learning rate is varied according to the performance of the algorithm.

- The rules of the Variable Learning rate Backpropagation algorithm (VLBP) are:

1) If the squared error (over the entire training set) increases by more than some set percentage ξ (typically one to five percent) after a weight update, then the weight update is discarded, the learning rate is multiplied by some factor $0 < \rho < 1$, and the momentum coefficient γ (if it is used) is set to zero.

Variable Learning Rate

2) If the squared error decreases after a weight update, then the weight update is accepted and the learning rate is multiplied by some factor $\eta > 1$. If γ has been previously set to zero, it is reset to its original value.

3) If the squared error increases by less than ξ, then the weight update is accepted but the learning rate is unchanged. If γ has been previously set to zero, it is reset to its original value.

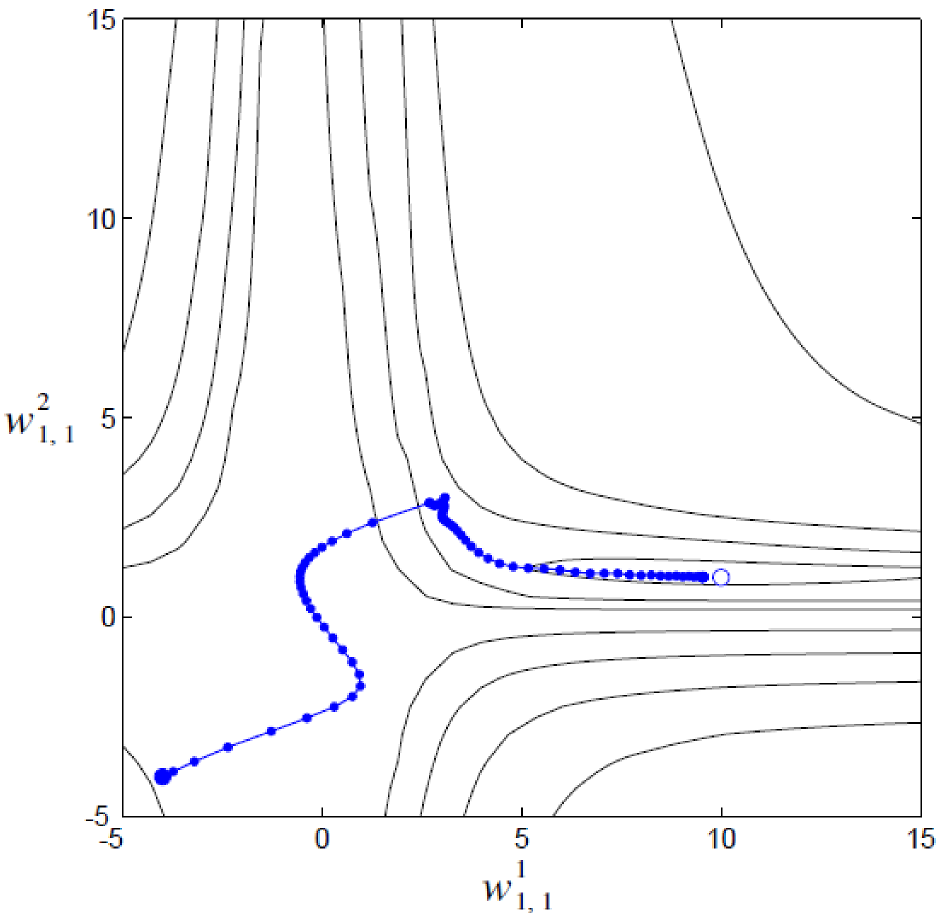
Variable Learning Rate

- Let's apply the VLBP to the function approximation problem (Case 1 in the previous section):

- The initial guess, initial learning rate and the momentum coefficient are the same as before. The new parameters are assigned to:

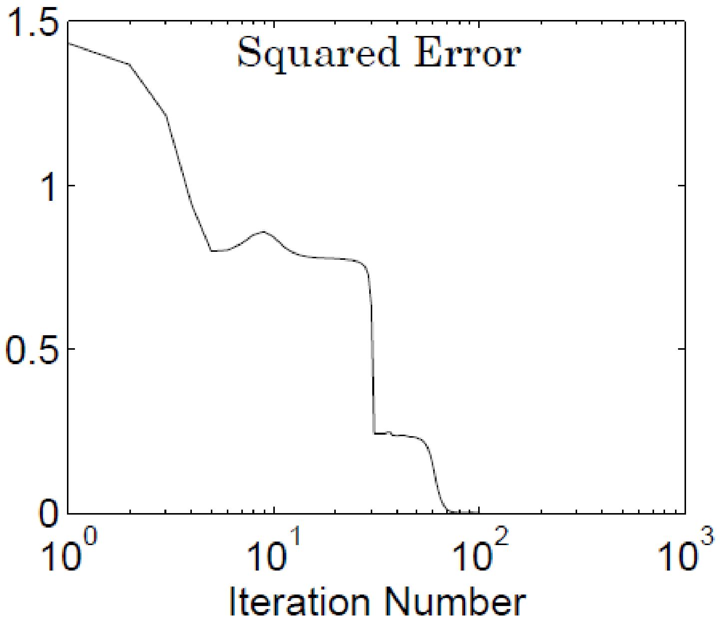
$$\eta = 1.05 \qquad \rho = 0.7 \qquad \xi = 4\%$$

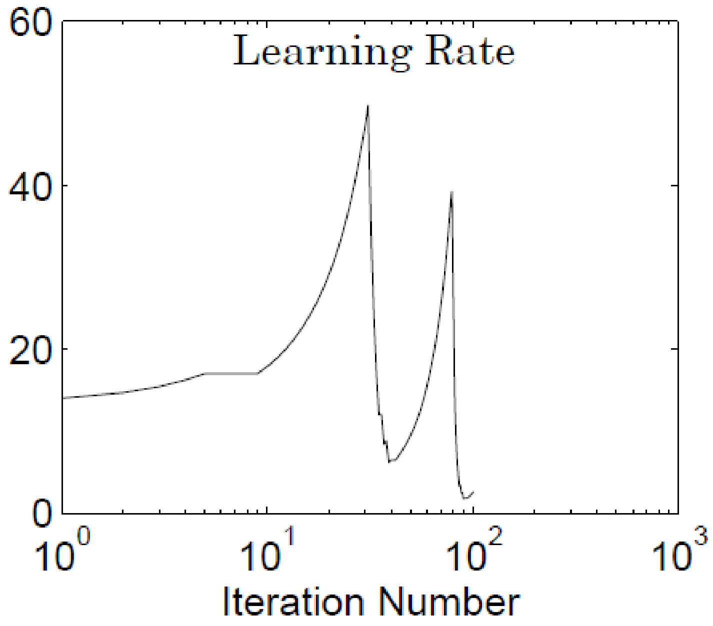
- The trajectory of the VLBP will be:



Variable Learning Rate

- The learning rate and therefore the step size, tends to increase when the trajectory is traveling in a straight line with constantly decreasing loss.





Variable Learning Rate

- **Explanation:** When the trajectory reaches a narrow valley, the learning rate is rapidly decreased. Otherwise the trajectory would have become oscillatory, and the loss would have increased dramatically.

- For each potential step where the error would have increased by more than 4% (ξ) the learning rate is reduced and the momentum is eliminated, which allows the trajectory to make the quick turn to follow the valley toward the minimum point.

- The learning rate increases again, which accelerates the convergence. when the trajectory overshoots the minimum point and the algorithm has almost converged, the learning rate is reduced. This process is typical of a VLBP trajectory.

Variable Learning Rate

- The modifications to SDBP (MOBP, VLBP) can often provide much faster convergence for some problems. However, there are two main drawbacks to these methods:

1. These modifications require that several parameters be set (e.g. ξ, ρ and γ), while the only parameter required for SDBP is the learning rate. Also the performance of the algorithm is often sensitive to changes in these parameters and besides the choice of parameters is problem-dependent.

2. These modifications can sometimes fail to converge on problems for which SDBP will eventually find a solution.

Thanks for your attention

End of chapter 6

Hamidreza Baradaran Kashani