Νευρο-Ασαφής Υπολογιστική
Neuro-Fuzzy Computing

Διδάσκων –
Δημήτριος Κατσαρός

@ Τμ. ΗΜΜΥ
Πανεπιστήμιο Θεσσαλίας

Διάλεξη 9η

Batch training in Backpropagation

Batch training in BP

- What we have described in previous lectures is the *online* 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
- For example, if each input occurs with equal probability, the mean square error performance index can be written

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

- The total gradient of this performance index is

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

Batch training in BP

- Therefore, the total gradient of the mean square error is the mean of the gradients of the individual squared errors
- So, in order to implement a batch version of the backpropagation algorithm,
  - we would step through all equations mentioned in slide-27 of lecture-06 for all of the inputs in the training set
  - the individual gradients would be averaged to get the total gradient
  - the update equations for the batch steepest descent algorithm would then be

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

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

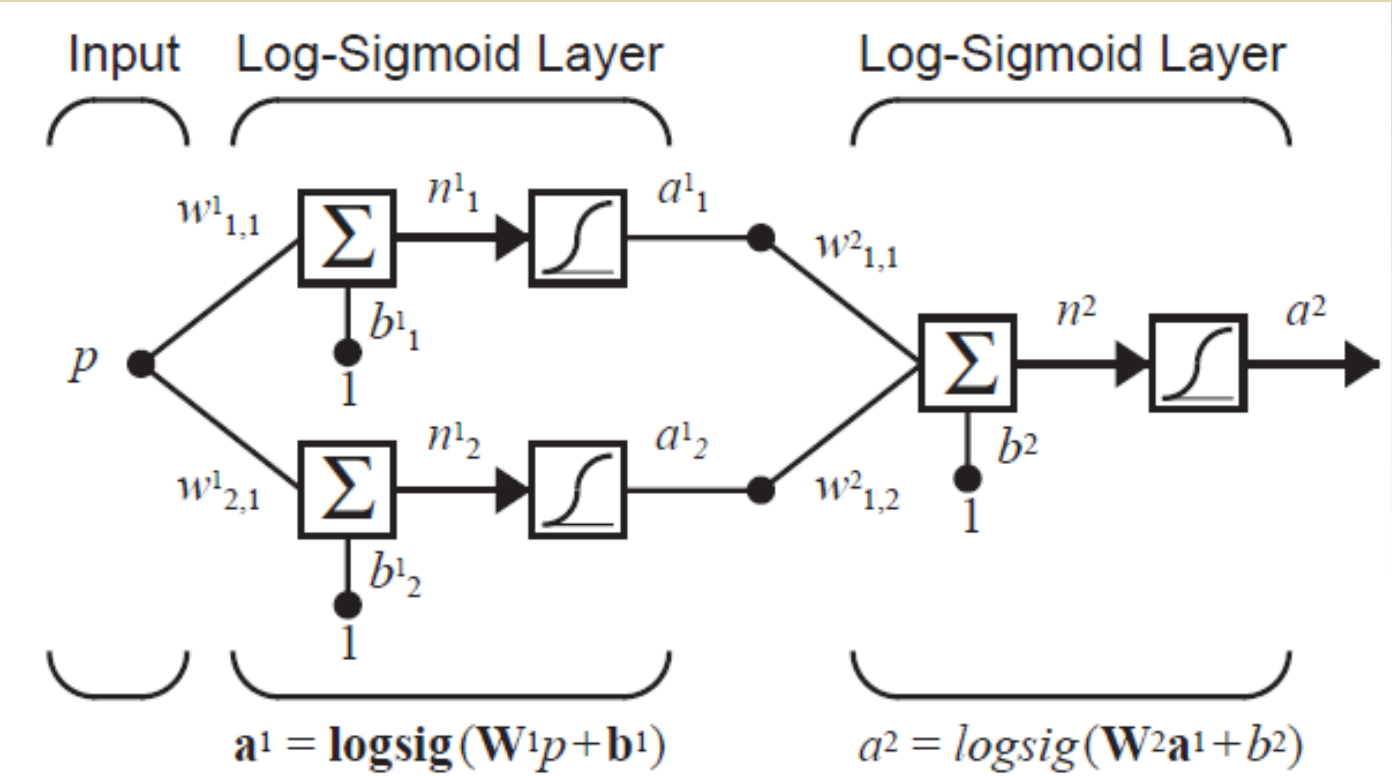
Drawbacks of Backpropagation

Drawbacks of BP

- In this lecture we will refer to the basic backpropagation algorithm as steepest descent backpropagation (SDBP)
- Recall that the LMS algorithm is guaranteed to converge to a solution that minimizes the mean squared error, so long as the learning rate is not too large
  - This is true because the mean squared error for a single-layer linear network is a quadratic function. The quadratic function has only a single stationary point.
- SDBP is a generalization of the LMS algorithm. Like LMS, it is also an approximate steepest descent algorithm for minimizing the mean squared error
  - SDBP is equivalent to the LMS algorithm when used on a single-layer linear network (proved during a class lecture)
- When applied to multilayer networks, the characteristics of SDBP are different
  - This has to do with the differences between the mean squared error performance surfaces of single-layer linear networks and multilayer nonlinear networks
    - While the performance surface for a single-layer linear network has a single minimum point and constant curvature, the performance surface for a multilayer network may have many local minimum points and the curvature can vary widely in different regions

Performance surface example

- To investigate the mean squared error performance surface for multilayer networks we will employ a simple function approximation example
- We will use the 1-2-1 network shown below with log-sigmoid transfer functions in both layers



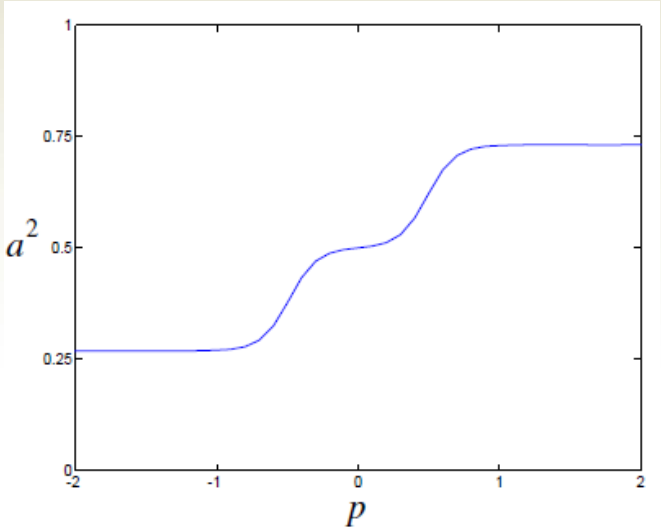
Performance surface example

- In order to simplify our analysis, we will give the network a problem for which we know the optimal solution. The function we will approximate is the response of the same 1-2-1 network, with the following values for the weights and biases:

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

$$w^2_{1,1} = 1, \quad w^2_{1,2} = 1, \quad b^2 = -1$$

- The network response for these parameters is shown below, which plots the network output a^2 as the input p is varied over the range [-2, 2]



Performance surface example

- We want to train our network to approximate the function displayed in the previous slide figure
- The approximation will be exact when the network parameters are set to the values given earlier, i.e.,

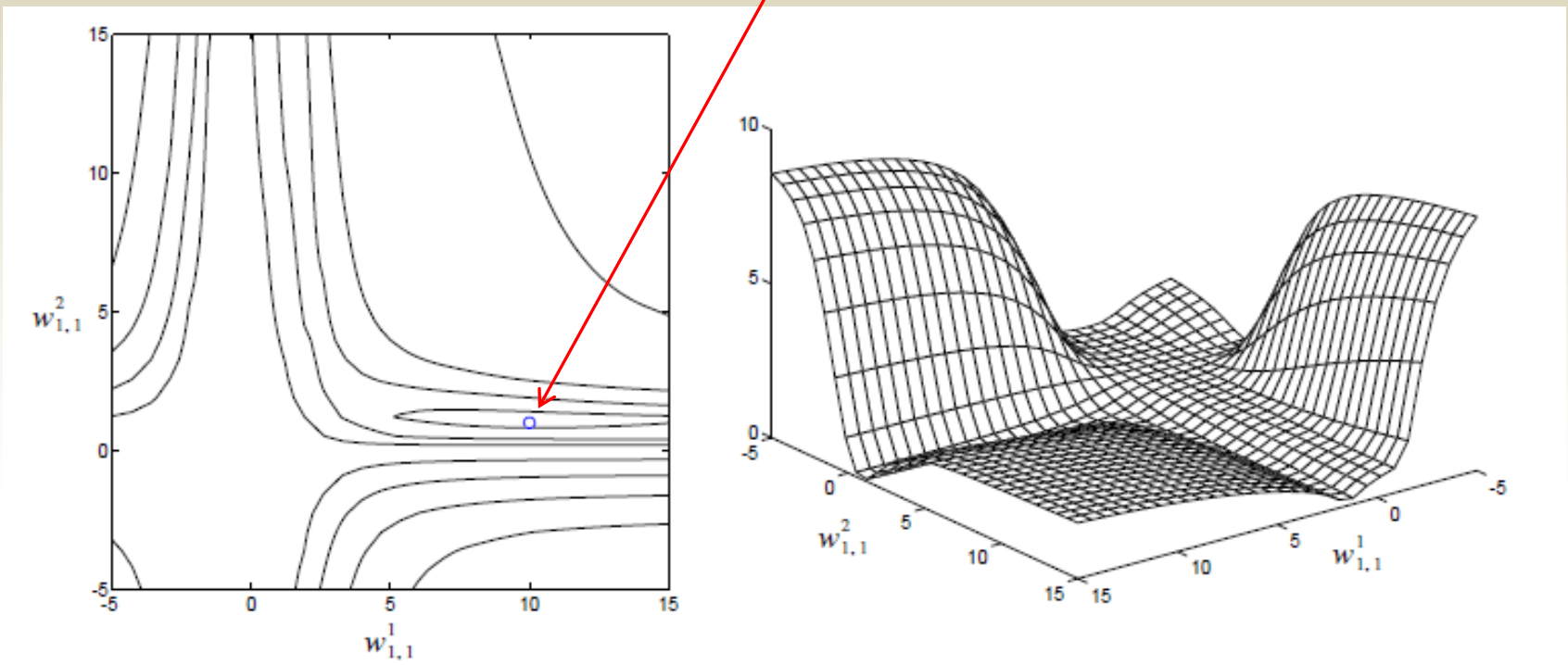
$$w^1_{1,1} = 10, \quad w^1_{2,1} = 10, \quad b^1_1 = -5, \quad b^1_2 = 5$$

$$w^2_{1,1} = 1, \quad w^2_{1,2} = 1, \quad b^2 = -1$$

- This is a very contrived problem, but it illustrates some important concepts
- Let's consider the performance index for our problem. We assume that the function is sampled at the equiprobable values: p = –2, –1.9, –1.8, …1.9, 2
- The performance index will be the sum of the squared errors at these 41 points

Performance surface example

- We will vary only two parameters at a time (to be able to draw)
- The figure illustrates the squared error when only $w^1_{1,1}$ and $w^2_{1,1}$ are being adjusted, while the other parameters are set to their optimal values (from previous slide). Note that the minimum error will be zero, and it will occur when $w^1_{1,1}=10$ and $w^2_{1,1}=1$, as indicated by the open blue circle in the figure



Performance surface example

- There are several features to notice about this error surface
  - A first feature of this error surface is that 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

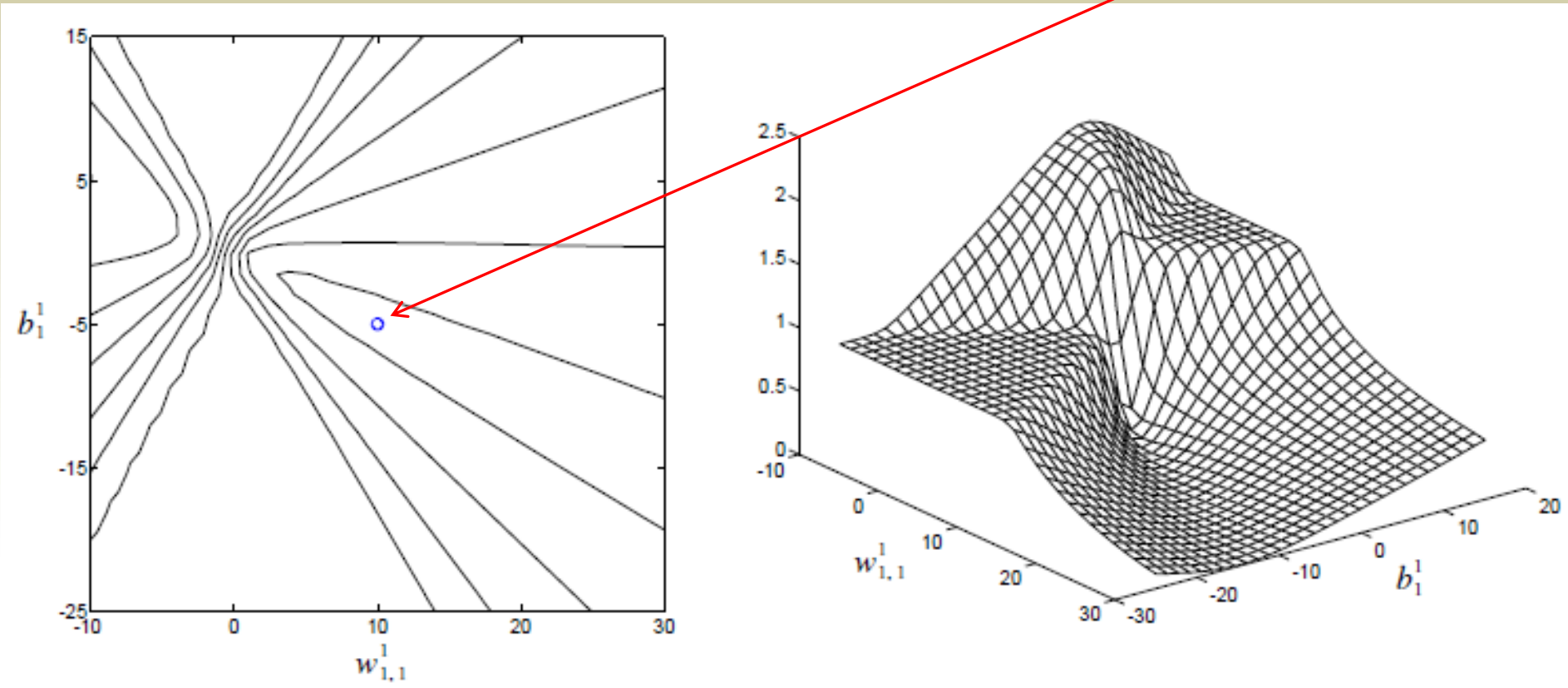
  - It should be noted that the flat regions of the performance surface should not be unexpected, given the sigmoid transfer functions used by the network
    - The sigmoid is very flat for large inputs

Performance surface example

- Continued
  - 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
    - There is also a local minimum, which is located in the valley that runs parallel to the $w^2_{1,1}$ axis. (This local minimum is actually off the graph at $w^1_{1,1}=0.88$ and $w^2_{1,1}=38.6$)

Performance surface example

- This figure illustrates the squared error when $w^1_{1,1}$ and b^1_1 are being adjusted (the other parameters are set to their optimal values)
- The minimum error will be zero, and it will occur when $w^1_{1,1}=10$ and $b^1_1=-5$, as indicated by the open blue circle

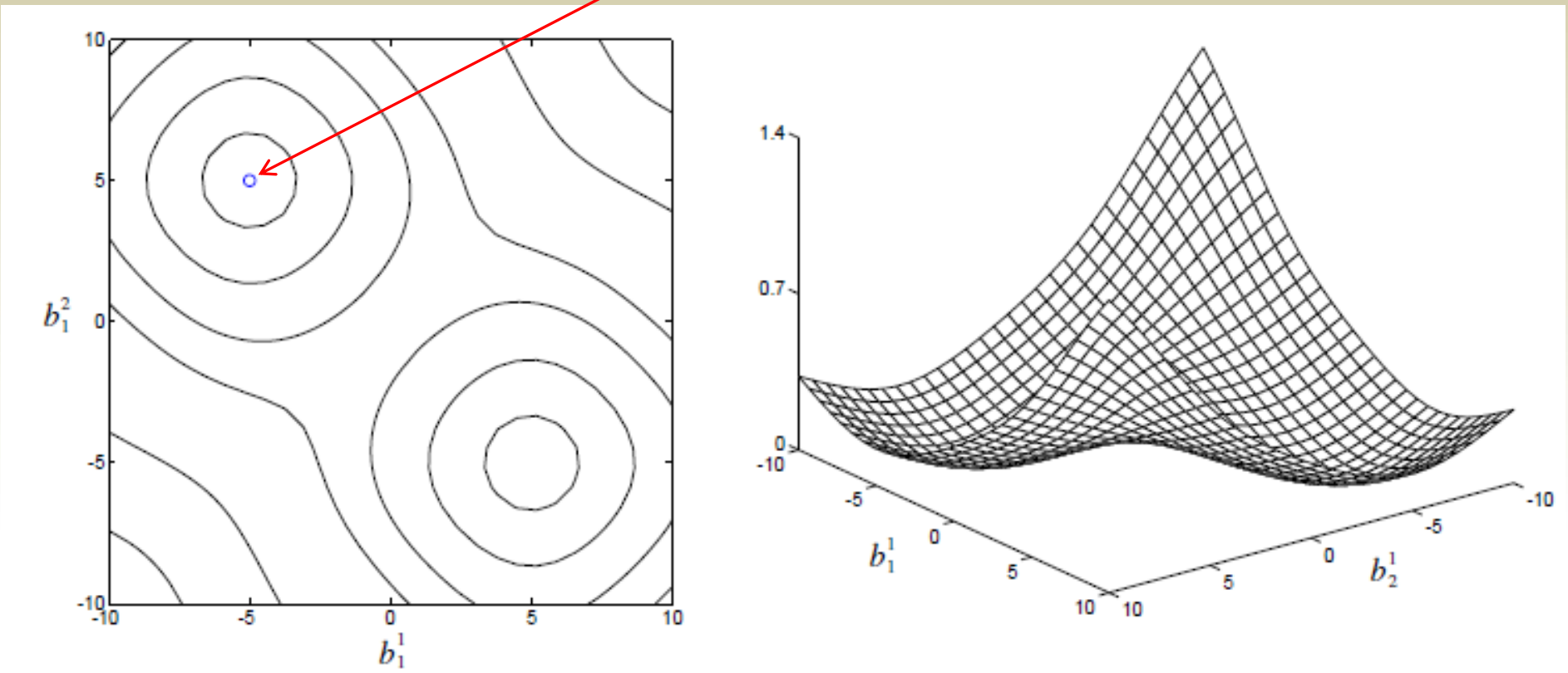


Performance surface example

- We find that the surface has a very contorted shape, steep in some regions and very flat in others.
- The standard steepest 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 steepest descent algorithm would effectively stop, even though it is not close to a local minimum point

Performance surface example

- This figure illustrates the squared error when b^1_1 and b^1_2 are being adjusted (the other parameters are set to their optimal values)
- The minimum error is located at $b^1_1 = -5$ and $b^1_2 = 5$, as indicated by the open blue circle



Performance surface example

- This surface illustrates an important property of multilayer networks: **they have a symmetry to them**
- Here we see that there are two local minimum points and they both have the same value of squared error.
- The second solution corresponds to the same network being turned upside down (i.e., 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 performance surface

Take away lessons

How to set the initial guess for the SDBP algorithm:

1. We do not want to set the initial parameters to zero
   - This is because the origin of the parameter space tends to be a saddle point for the performance surface
2. We do not want to set the initial parameters to large values
   - This is because the performance surface tends to have very flat regions as we move far away from the optimum point
   - Typically we choose the initial weights and biases to be small random values. In this way we stay away from a possible saddle point at the origin without moving out to the very flat regions of the performance surface
   - As we will see in subsequent lectures, it is also useful to try several different initial guesses, in order to be sure that the algorithm converges to a global minimum point

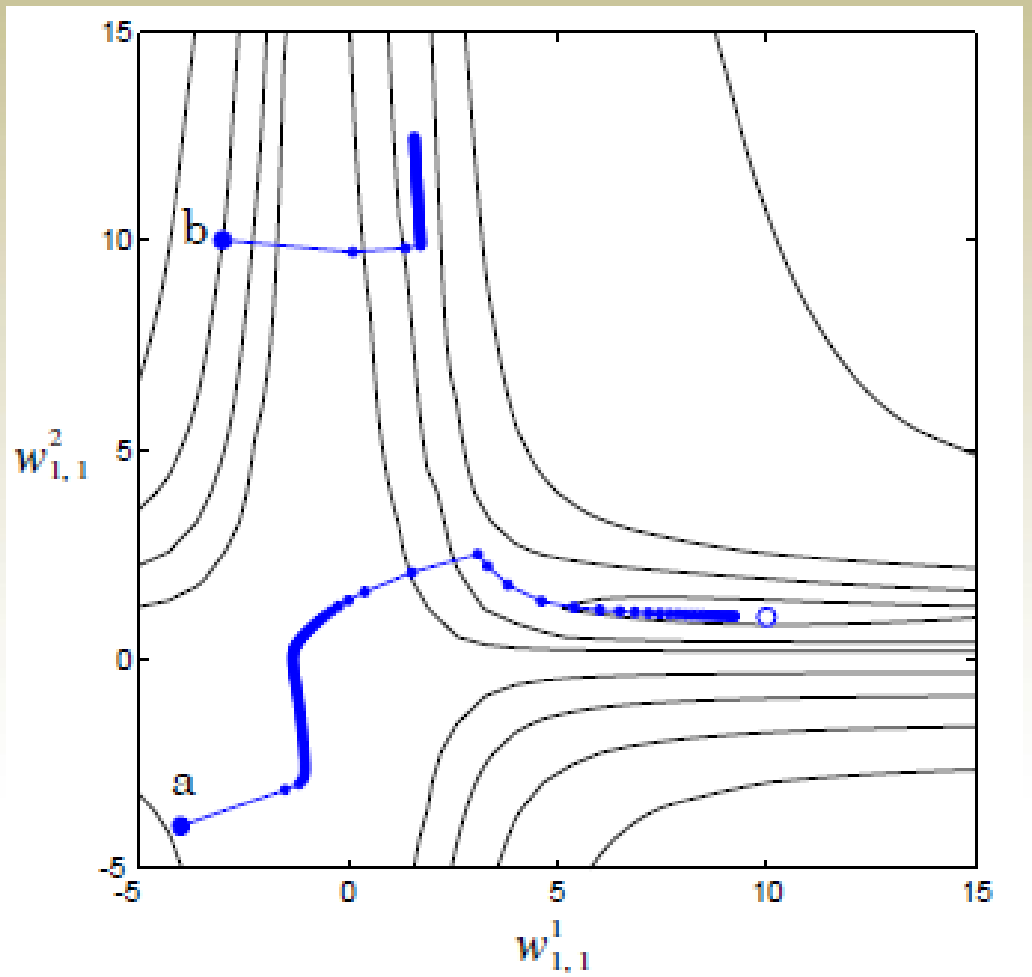
Convergence example

We will now investigate the performance of SDBP. Suppose that only $w^1_{1,1}$ and $w^2_{1,1}$ are being adjusted

For the initial condition labeled "a" the algorithm does eventually converge to the optimal solution, but the convergence is slow

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

If we were to increase the learning rate, the algorithm would converge faster while passing over the initial flat surface, but would become unstable when falling into the valley, as we will see immediately

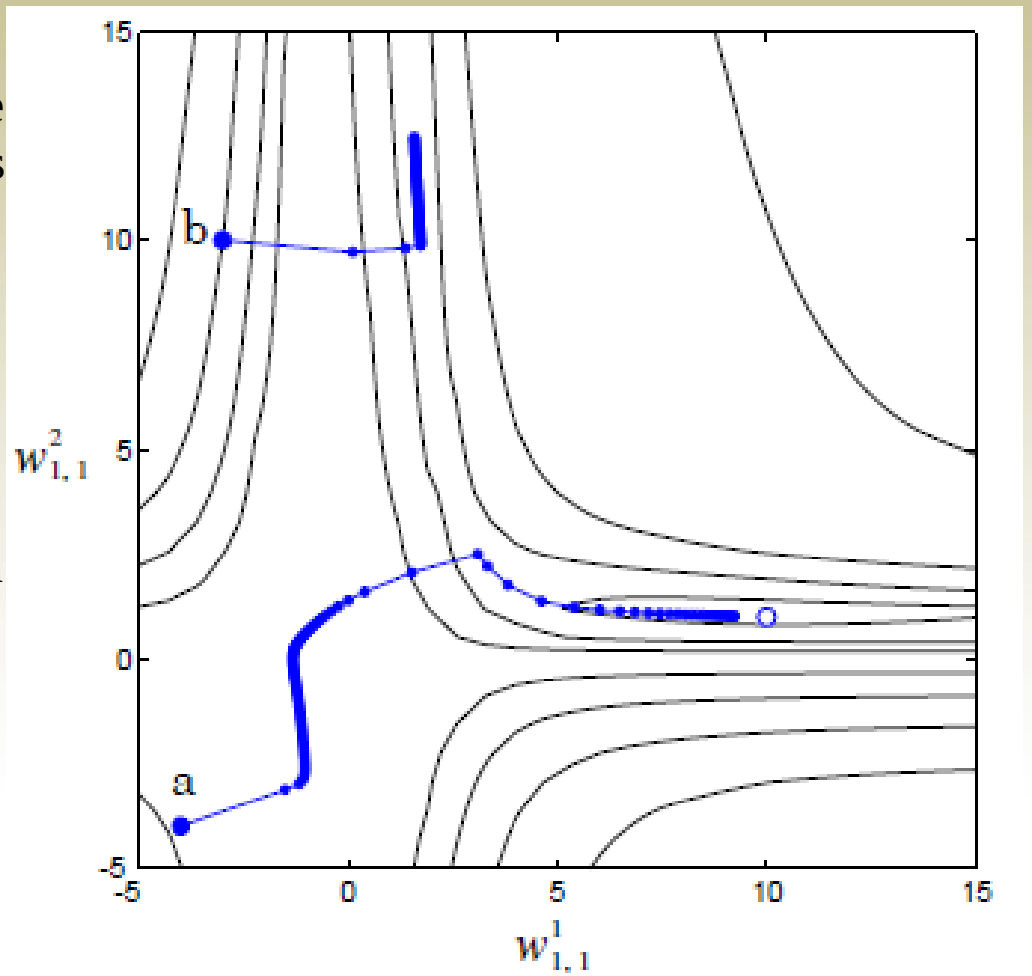


Convergence example

We will now investigate the performance of SDBP. Suppose that only $w^1_{1,1}$ and $w^2_{1,1}$ are being adjusted

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. If allowed to continue the trajectory converges to $w^1_{1,1}$=0.88 and $w^2_{1,1}$=38.6

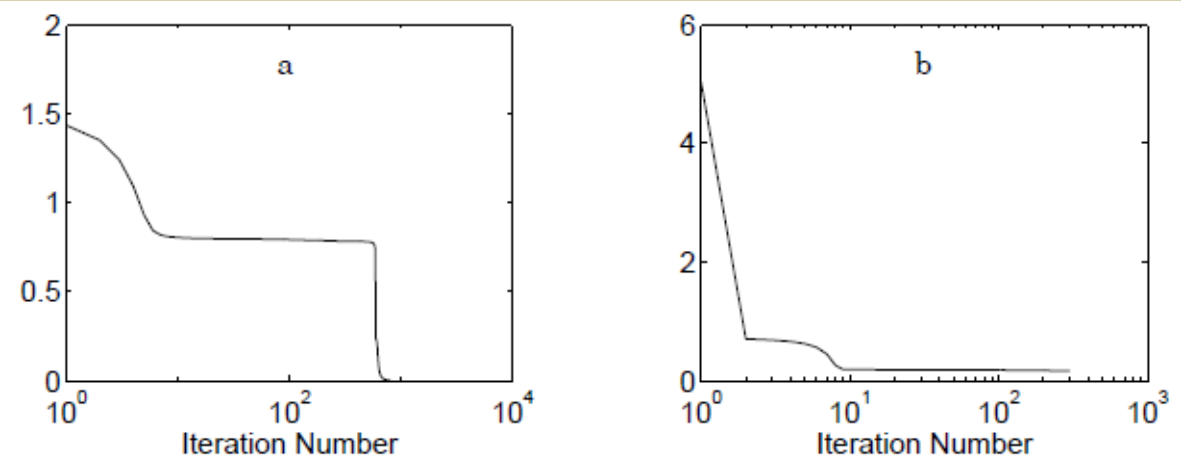
The existence of multiple local minimum points is typical of the performance surface of multilayer networks. For this reason it is best to try several different initial guesses in order to ensure that a global minimum has been obtained. (Some of the local minimum points may have the same value of squared error, so we would not expect the algorithm to converge to the same parameter values for each initial guess.)



Convergence example

The progress of the algorithm can also be seen below where it is shown the squared error versus the iteration number. The curve on the left corresponds to trajectory "a" and the curve on the right corresponds to trajectory "b."

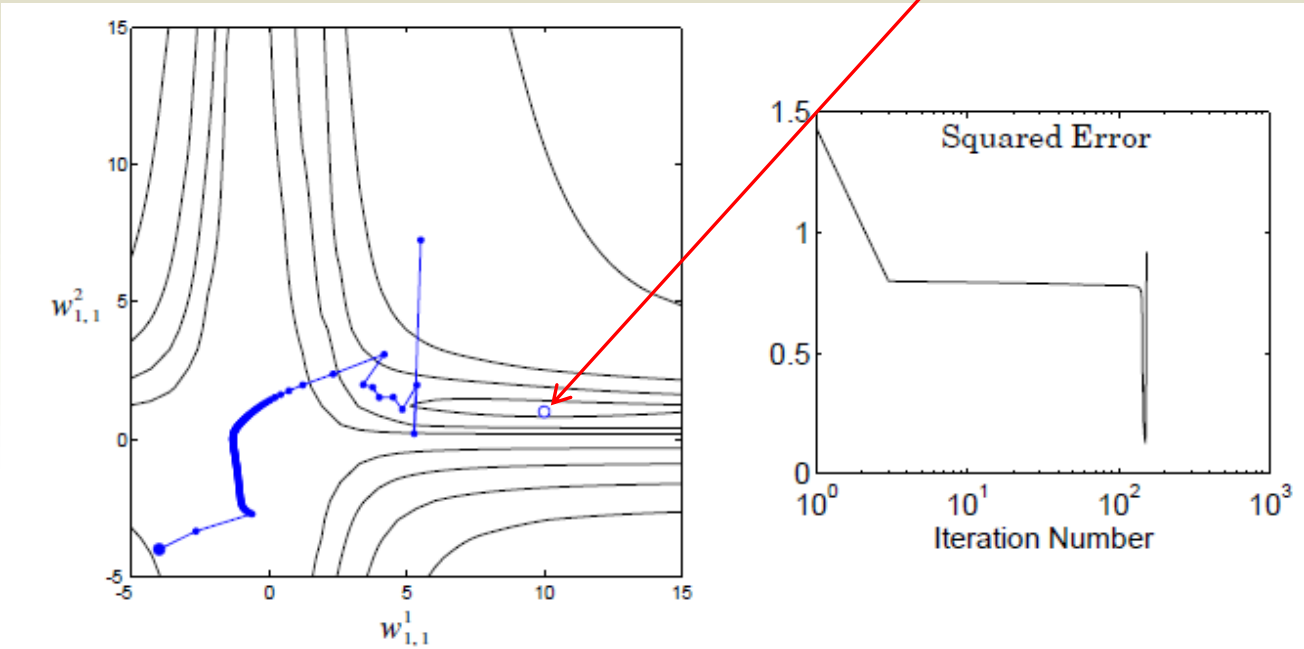
These curves are typical of SDBP, with long periods of little progress and then short periods of rapid advance



- We can see that the flat sections correspond to times when the algorithm is traversing a flat section of the performance surface
  - During these periods 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

Convergence example: lessons learned

- This effect is illustrated here [The trajectory shown here corresponds to trajectory "a" except that a larger learning rate was used]
- The algorithm converges faster at first, but when the trajectory reaches the narrow valley that contains the minimum point the algorithm begins to diverge
- This suggests that it would be useful to vary the learning rate
  - We could increase the learning rate on flat surfaces and then decrease the learning rate as the slope increased



Convergence example: lessons learned

- Another way to improve convergence would be to smooth out the trajectory
- Note in previous 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

Heuristic modifications of Backpropagation: Momentum

The effect of low-pass filtering

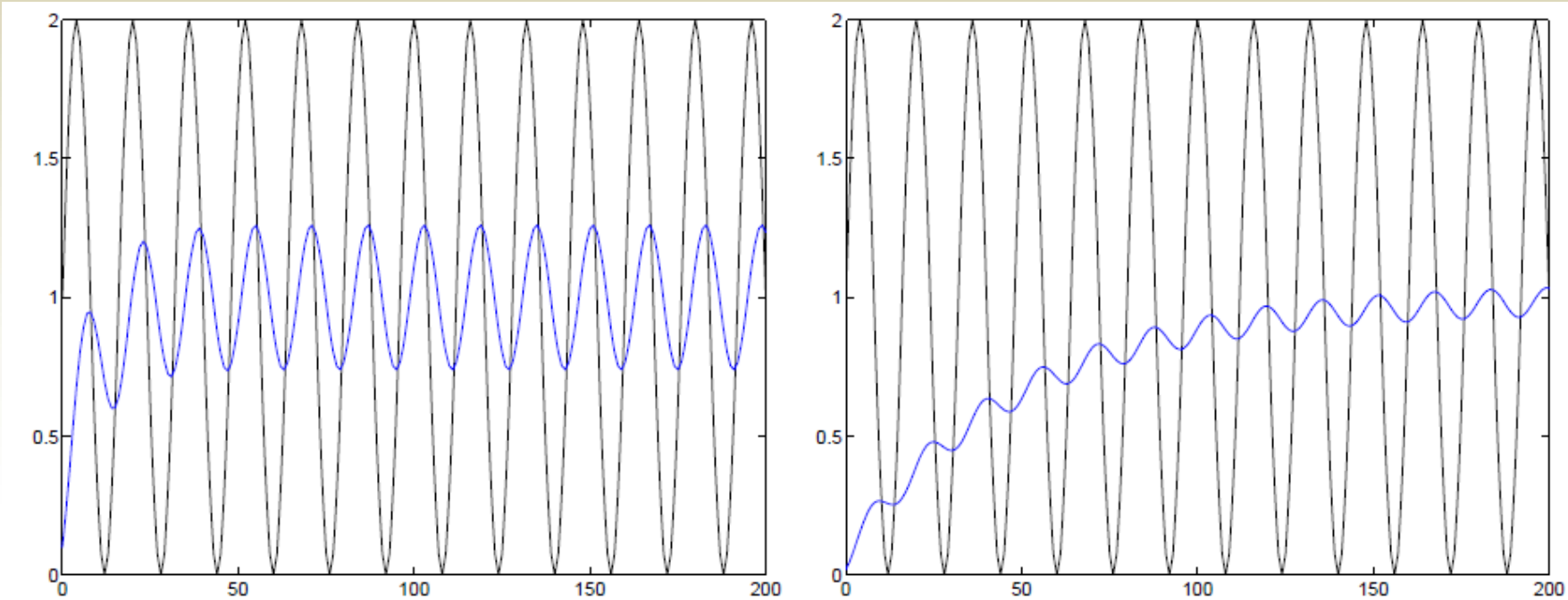
- This is a modification based on our previous observation that convergence might be improved if we could smooth out the oscillations in the trajectory
- We can do this with a low-pass filter
- Before we apply momentum to a neural network application, let's investigate a simple example to illustrate the smoothing effect. Consider the following first-order filter:

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

where w(k) is the input to the filter y(k) is the output of the filter, and γ is the momentum coefficient that must satisfy: $0 \leq \gamma < 1$

The effect of low-pass filtering

- The effect of this filter is shown below. [The input to the filter was taken to be the sine wave: $w(k)= 1+\sin(2\pi k/16)$]
- The oscillation of the output is less than the oscillation in the input
- As γ is increased, the oscillation in the filter output is reduced (but output is slower to respond). The average filter output is the same as the average filter input



a) $\gamma = 0.9$

b) $\gamma = 0.98$

Momentum Backpropagation (MOBP)

- Recall the parameter updates for SDBP:

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

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

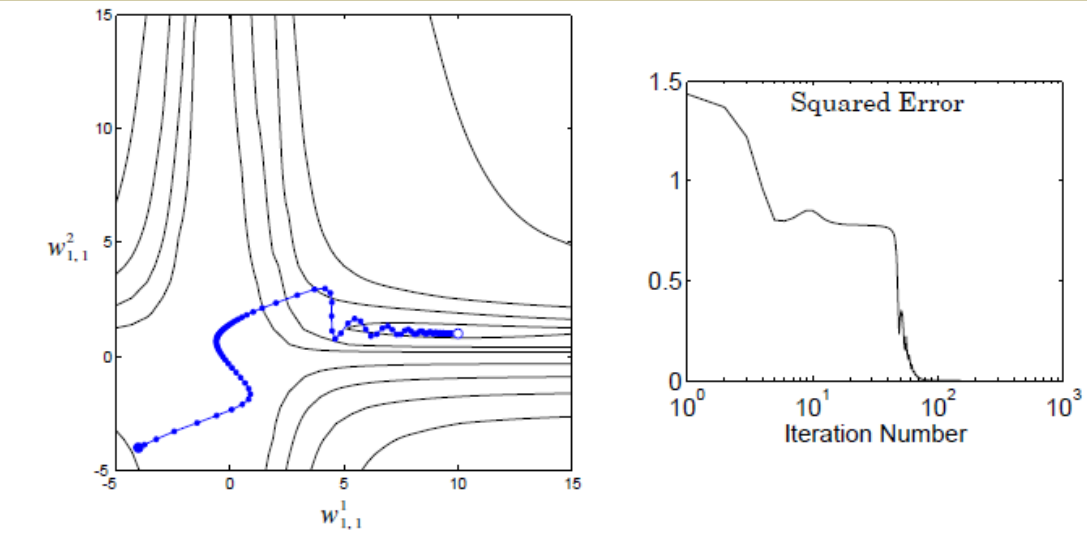
- When the momentum filter is added to the parameter changes, we obtain the following equations for the momentum modification to backpropagation (MOBP):

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

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

Momentum Backpropagation (MOBP)

- If we now apply these equations to the example at the beginning of this lecture, we obtain the results shown below
- This trajectory corresponds to the same initial condition and learning rate as in figure of slide 21, but with a momentum coefficient of 0.8



If you look carefully at the trajectory, you can see why the procedure is given the name *momentum*.
It tends to make the trajectory continue in the same direction.
The larger the value of γ, the more "momentum" the trajectory has.

- 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
- Another feature of momentum is that it tends to accelerate convergence when the trajectory is moving in a consistent direction

Heuristic modifications of Backpropagation: Nestorov's Momentum

Nestorov's Accelerated Gradient Descent

- First, we define the following sequences:

$$\lambda_0 = 0$$

$$\lambda_s = \frac{1 + \sqrt{1 + 4\lambda_{s-1}^2}}{2}$$

$$\gamma_s = \frac{1 - \lambda_s}{\lambda_s + 1}$$

- Apparently $\gamma_s \leq 0$.
- The algorithm is defined by the following equations with arbitrary initial point $x_1 = y_1$:

$$y_{s+1} = x_s - \frac{1}{\beta}\nabla f(x_s)$$

$$x_{s+1} = (1 - \gamma_s)y_{s+1} + \gamma_s y_s$$

Heuristic modifications of Backpropagation: Variable learning rate

Variable learning rate Backpropagation

- We suggested earlier that we might be able to speed up convergence if we increase the learning rate on flat surfaces and then decrease the learning rate when the slope increases
- Recall that the mean squared error performance surface for single-layer linear networks is always a quadratic function, and the Hessian matrix is therefore constant. The maximum stable learning rate for the steepest descent algorithm is two divided by the maximum eigenvalue of the Hessian matrix
- The error surface for the multilayer network is not a quadratic function. The shape of the surface can be very different in different regions of the parameter space
- Perhaps we can speed up convergence by adjusting the learning rate during the course of training
- The trick will be to determine when to change the learning rate and by how much

Variable learning rate Backpropagation

- There are different approaches for varying the learning rate. We will describe a straightforward batching procedure

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 ζ (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
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 has been previously set to zero, it is reset to its original value
3. If the squared error increases by less than ζ, 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

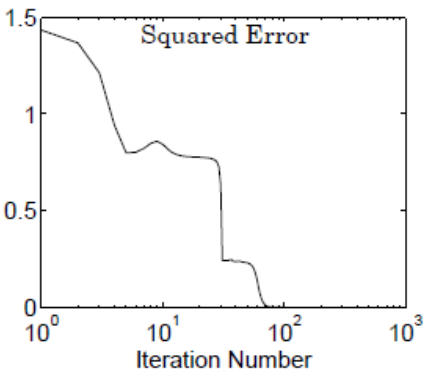
VLBP example

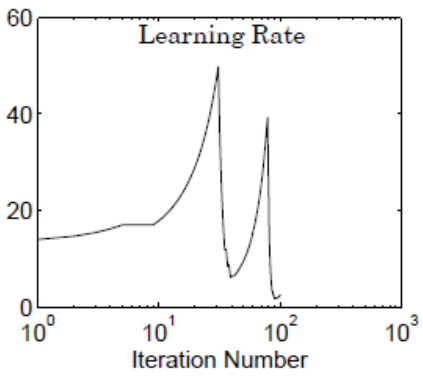
- We apply VLBP to the previous function approximation problem. The figure below displays the trajectory for the algorithm using the same initial guess, initial learning rate and momentum coefficient. The new parameters were assigned the values

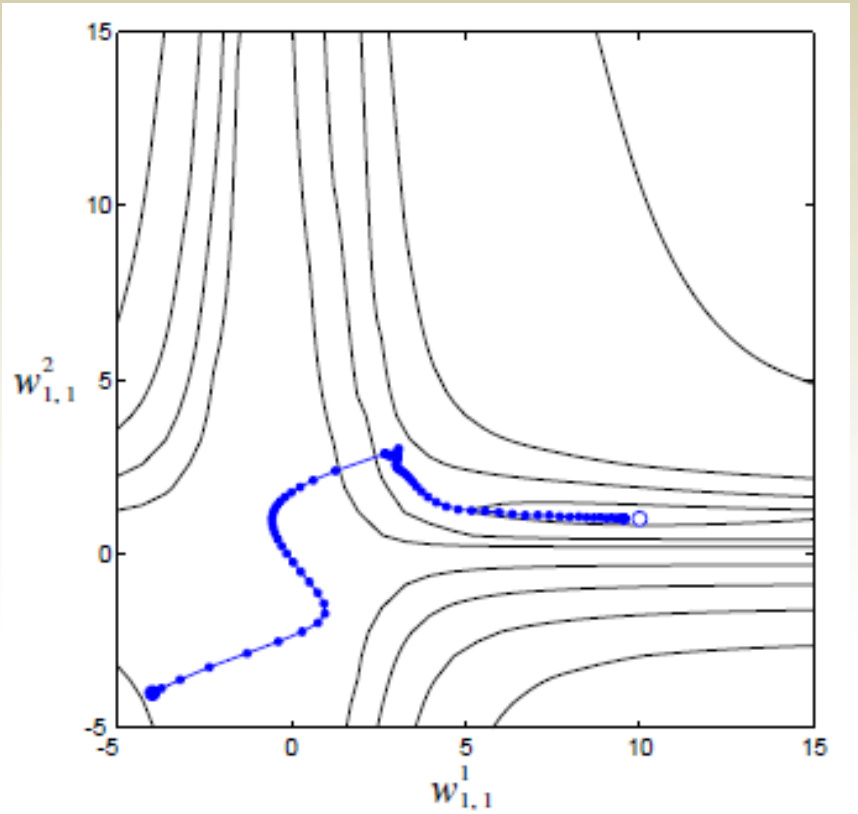
$$\eta = 1.05 \qquad \rho = 0.7 \qquad \zeta = 4\%$$

Notice how the learning rate, and therefore the step size, tends to increase when the trajectory is traveling in a straight line with constantly decreasing error

This effect can also be seen below, which shows the squared error and the learning rate versus iteration number







VLBP example

- When the trajectory reaches a narrow valley, the learning rate is rapidly decreased
  - Otherwise the trajectory would have become oscillatory, and the error would have increased dramatically
- For each potential step where the error would have increased by more than 4% 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 then increases again, which accelerates the convergence
- The learning rate is reduced again when the trajectory overshoots the minimum point when the algorithm has almost converged
- This process is typical of a VLBP trajectory

Variations on VLBP

There are variations on this variable learning rate algorithm

- Jacobs proposed the *delta-bar-delta* learning rule according to which:
  - Each network parameter (weight or bias) has its own learning rate
  - The algorithm increases the learning rate for a network parameter if the parameter change has been in the same direction for several iterations
  - If the direction of the parameter change alternates, then the learning rate is reduced
- The *SuperSAB* algorithm of Tollenaere is similar to the delta-bar-delta rule
  - it has more complex rules for adjusting the learning rates
- Another heuristic modification to SDBP is the *Quickprop* algorithm of Fahlman
  - It assumes that the error surface is parabolic and concave upward around the minimum point and that the effect of each weight can be considered independently

Variations on VLBP: Bibliography

- R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Networks*, vol. 1, no. 4, pp. 295–308, 1988.
- T. Tollenaere, "SuperSAB: Fast adaptive back propagation with good scaling properties," *Neural Networks*, vol. 3, no. 5, pp. 561–573, 1990.
- S. E. Fahlman, "Faster-learning variations on back-propagation: An empirical study," In D. Touretsky, G. Hinton & T. Sejnowski, eds., *Proceedings of the 1988 Connectionist Models Summer School*, San Mateo, CA: Morgan Kaufmann, pp. 38–51, 1988.

Drawbacks of heuristic modifications to SDBP

- The heuristic modifications to SDBP can often provide much faster convergence for some problems. However, there are two main drawbacks to these methods:
- The first is that the modifications require that several parameters be set (e.g., ζ, ρ, and γ), while the only parameter required for SDBP is the learning rate. Some of the more complex heuristic modifications can have five or six parameters to be selected. Often the performance of the algorithm is sensitive to changes in these parameters. The choice of parameters is also problem dependent
- The second drawback to these modifications to SDBP is that they can sometimes fail to converge on problems for which SDBP will eventually find a solution
- Both of these drawbacks tend to occur more often when using the more complex algorithms