6 Learning Rate and Momentum

6.1 Learning Rate

Recalling (5.25), the back-propagation weight update with momentum is

$$\Delta \mathbf{w}(t) = -\eta \frac{\partial E}{\partial \mathbf{w}}(t) + \alpha \Delta \mathbf{w}(t-1).$$

The learning rate and momentum parameters η and α have an obvious direct effect on the training process so it is desirable to choose good values. Typical ranges are $0.05 \leq \eta \leq 0.75$ and $0 \leq \alpha \leq 0.9$ [330]. Many other factors also affect training, so these should be treated as reasonable guidelines rather than inviolable limits.

Although $\eta = 0.1$ is often suggested, this is basically an arbitrary constant, a "magic number,"—which *is not* appropriate for all problems or even at all times during training. It may work well for many problems, but much larger or smaller values may work better on particular problems. For one thing, the SSE error function is sensitive to the size of the training set. In the familiar two-input XOR problem, for example, there are four training patterns. If all training patterns are duplicated K times, the form of the target function is unchanged but E_{SSE} becomes K times as large, as do the partial derivatives and weight changes. If K is large, say 1000, the weights could become very large in a single step which would almost certainly drive the sigmoid nonlinearities into hard saturation and essentially halt learning. In effect, redundancy in the training set can amplify the effective learning rate to an unreasonably large value. This example is contrived, but similar things can happen when data is heavily clustered. If the clusters are well separated, each can be viewed as many repetitions of a single point with a small amount of noise.

One remedy for this problem would be to use the mean squared error

$$E_{MSE} = \frac{1}{M} \sum_{m=1}^{M} (t_m - y_m)^2$$

to normalize for the size M of the training set. This is reasonable, but it is common practice to use the unnormalized sum-of-squares error. An equivalent solution is to scale the learning rate by the size of the training set.

Unfortunately, there are other problems this does not fix. Redundant input or output nodes (e.g., multiple nodes with very similar training values) as well as internal redundancy (e.g., multiple hidden nodes computing similar functions) can have a similar amplifying effect. Less obvious redundancy in the form of linear dependencies also causes problems.

Other factors affecting the choice of learning rate include

- The distribution of the training data.

• Momentum. Momentum amplifies the effective learning rate to $\eta_{eff} = \eta/(1-\alpha)$ so large momentum values call for smaller η values, in general.

• Weight magnitudes. If the weights are initialized to very small values, the calculated derivatives will be small and larger initial learning rates will be needed to achieve the same training speeds. But then as the weights grow during training, it may be necessary to reduce the learning rate to avoid instability and saturation.

• The shape of the error surface $E(\mathbf{w})$. The gradient $\partial E/\partial \mathbf{w}$ can change dramatically as $\mathbf{w}$ changes. Large learning rates are useful to traverse flat areas quickly, but small values may be needed to avoid instability in "hilly" areas. A single learning rate may not be optimal in all parts of the space.

• Stochastic training. In on-line training, where randomness of pattern selection acts as noise, it is usually necessary to gradually reduce η to zero in later learning stages to obtain convergence to stable weights.

• The difficulty of the problem. In vague terms, difficult problems seem to call for fine distinctions and small learning rates. Often this can be interpreted as ill-conditioning of the Hessian matrix.

Many of these factors are related, of course. The point is that, in general, it is not possible to calculate a best learning rate a priori.

The same learning rate may not even be appropriate in all parts of the network. The general fact that the error is more sensitive to changes in some weights than in others makes it useful to assign different learning rates to each weight. The problem of delta-attenuation (section 6.1.8), where input weight layers receive smaller back-propagated deltas and thus tend to learn much more slowly than output weight layers, makes it useful to use larger learning rates in the early layers. If a single global value must be used, compromises must be made.

Because poor training parameters can cause poor performance it is often necessary to do a mini-search for a good set of training parameters as part of the search for a good set of weights. This is a common criticism of back-propagation as an optimization algorithm. (Sometimes the algorithm is blamed for failure due to an unreasonable choice of parameters, however.)

The difficulty of choosing a good learning rate a priori is one of the reasons adaptive learning rate methods are useful and popular. A good adaptive algorithm will usually converge much faster than simple back-propagation with a poorly chosen fixed learning rate. The speed advantage may be small compared to training with an optimal constant learning rate, but even then adaptive methods relieve the user of the need to search for

the optimal value. Most methods adapt the rate dynamically as appropriate for changing conditions and many assign different rates appropriate for different parts of the network. A few methods are summarized in chapter 9.

6.1.1 An Example

The following example is used to illustrate how different choices of learning rate and momentum affect the training process. The illustrations that follow do not show how to calculate good parameters, but they may help in recognizing when parameter changes are needed. It should be emphasized that the curves in the figures are unique to this example and different curves will be obtained from different networks, data sets, initialization conditions, and so forth. The example is intended to be representative in that most problems will show qualitative similarities, but quantitative differences should be expected. Similar illustrations can be found in [67, 194, 372].

Training Details A 4/4/1 network was trained to learn the 4-bit parity problem using plain batch back-propagation. This problem was chosen because it is well understood and simple enough to repeat many times but not completely trivial. Input values were ± 1 and target values were ± 0.9. Tanh nonlinearities were used at the hidden and output nodes. Initial weights were uniformly distributed in $[-0.5, +0.5]$. A single nonadaptive learning rate was used for the entire network.

Note that *mean-squared-error* was used rather than sum-of-squares error. Use of MSE with learning rate η_o is equivalent to use of SSE with a learning rate η_o/M where M is the number of training patterns, $M = 16$ here. The difference is minor but affects the interpretation of the results.

A variety of learning rates from 0.001 to 10 (37 values) and momentums from 0 to 0.99 (14 values) were tested. One hundred trials were run for each parameter pair; 51,800 networks were trained in all, each with different random initial weights. Each network was allowed 5000 training epochs. Learning was considered successful (converged) if $E_{MSE} < 0.001$ or if every pattern was classified correctly with error less than 0.2. An attempt was made to detect stalled networks: if the change in E_{MSE} between epochs was less than 10^{-12} or if the magnitude of the gradient was less than 10^{-10}, the network was considered stuck and training was halted early. This saved training time but may have skewed the convergence probability estimates downward.

The average convergence time for each parameter pair was calculated as the sum of the times for the converged networks divided by the number of networks that converged

$$T_{avg} = \frac{\sum_i T_i}{N_{converged}}.$$

If no networks converged for a particular set of parameters, T_{avg} was set to 5000 for graphing purposes. The probability of convergence was estimated as the number of networks that converged divided by the number trained (100). Results are summarized in the following sections.

6.1.2 Training Time versus Learning Rate

Figure 6.1 shows typical plots of expected training time and probability of convergence versus learning rate η. The exact shape and scale of the curves will differ depending on details of the particular problem, but the shapes are characteristic. In general, the training time versus learning rate curve has a bathtub or "U" shape. At very small learning rates, training times are high simply because each weight change is so small; the convergence time is controlled mainly by the small step size and decreases as the learning rate increases. Beyond a certain critical value, however, the average training time increases sharply and the probability of convergence falls to zero. In the figure, the critical point shifts to lower learning rates as momentum increases.

The curves in figure 6.1 were obtained by averaging over many trials. Figure 6.1.2 shows the distribution of the data points behind the averages. At very high learning rates, most networks either do not converge or converge to poor solutions and become stuck. Of the networks that do converge, however, most do so very quickly. These probably depend on lucky initializations. Beyond the critical point, the average training time T_{avg} (excluding networks that do not converge) continues to decrease as learning rate increases but the probability of lucky initializations decreases faster. At some point the decreased likelihood of convergence outweighs training time improvements and the plotted training time increases sharply. ($T_{avg} = 5000$ was plotted when none of the 100 trials converged.)

A loose analogy can be made with driving a car. The time to reach your destination depends on situation-specific details such as the make of the car, weather and traffic conditions, and the general terrain along your route. For a given set of conditions, however, there is presumably an optimum speed with the lowest average travel time. The time depends directly on your driving speed but that also affects the probability of mishaps. At very low speeds, the risk of accident is negligible and you can almost always reduce travel time by driving faster. At increasing speeds, travel time decreases further but the risk of accident rises. At some point depending on particular conditions the probability of driving off the road and spending the morning at the repair shop approaches 1 and the expected travel time soars.

6.1.3 Interaction of Learning Rate and Momentum

Figure 6.1 illustrates that when the learning rate is small, large momentum values $\alpha \rightarrow 1$ generally increase the speed and probability of convergence. In the simulations, most cases

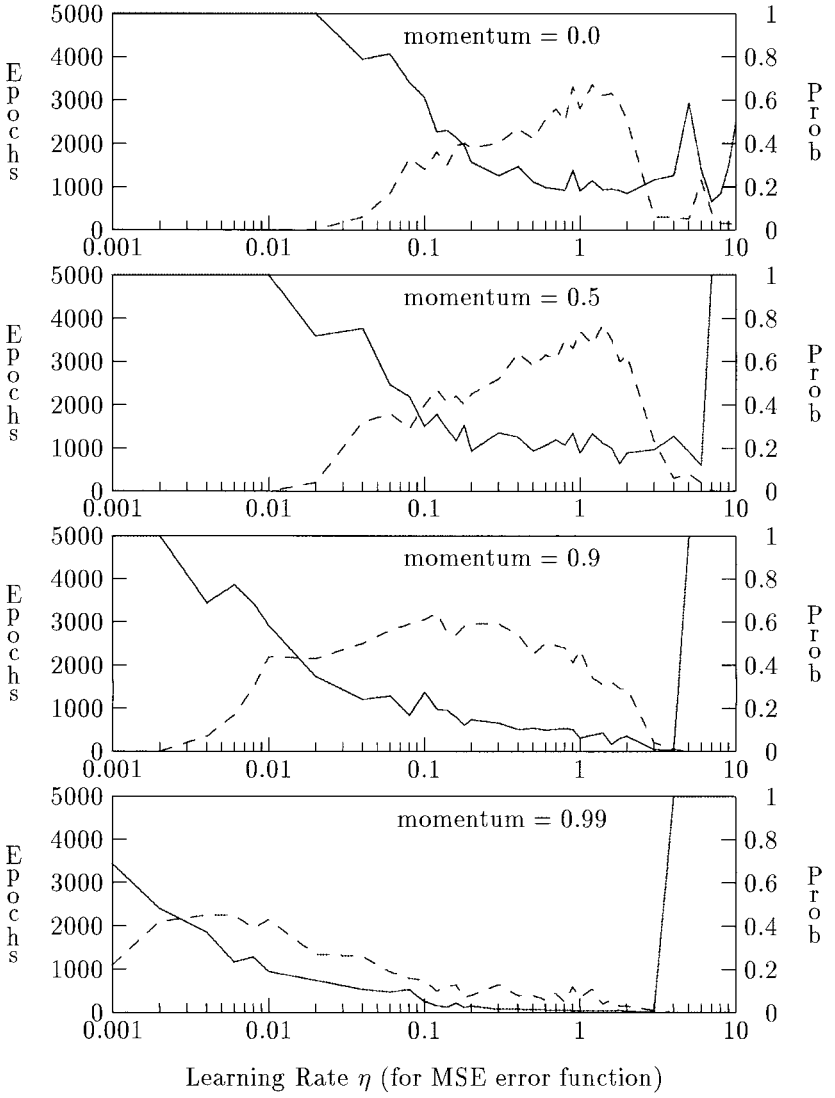
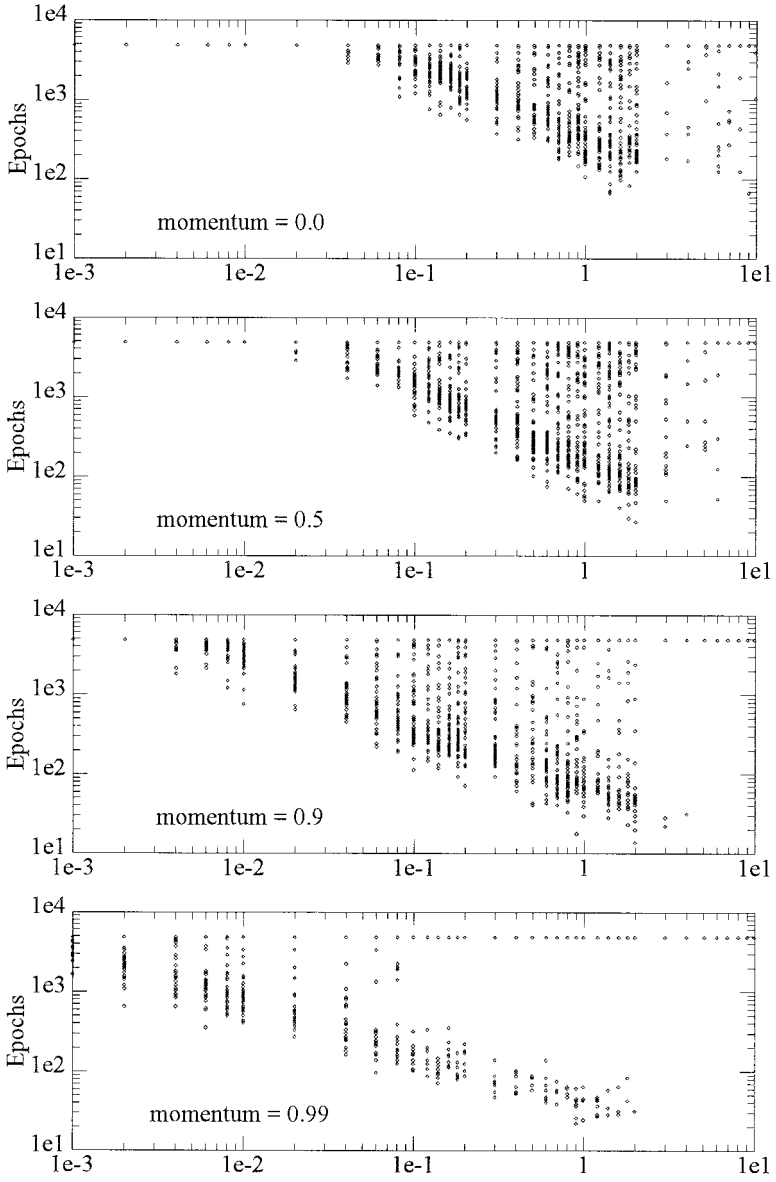


Figure 6.1
Average training time (solid) and convergence probability (dashed) versus learning rate for $\alpha = 0, 0.5, 0.9, 0.99$. A 4/4/1 network was trained on the 4-bit parity problem with batch back-propagation and the MSE error function. Each point is the average of 100 runs with different random initial weights. Note: Use of the MSE error function rather than SSE normalizes the learning rate by the size of the training set. See section 6.1.1 for simulation details.


1e4
1e3
1e2
1e1
Epochs
momentum = 0.0
momentum = 0.5
momentum = 0.9
momentum = 0.99
1e-3
1e-2
1e-1
1
1e1
Learning rate η (for MSE error function)

with very small learning rates didn't converge within the allotted time unless α was large. (Of course, they might have converged eventually if more time were allowed.) In the figure, at $\eta = 0.01$ the highest probability of convergence was obtained for $\alpha = 0.99$. For large learning rates, the situation is reversed and very few cases converge unless the momentum is small. In terms of momentum, the figure shows the peak of the probability of convergence density function shifting to lower learning rates as α increases.

Section 6.2 shows that for small perturbations the *effective learning rate* with momentum is $\eta' = \eta/(1 - \alpha)$. That is, momentum has an amplifying effect on the learning rate. For $\alpha \to 1^-$, the denominator is small and the effective learning rate is large; at $\alpha = 0.99$, for example, $\eta' = 100\eta$. Figure 6.3 illustrates this, showing the expected training time and probability of convergence *vs. effective* learning rate for selected momentum values. Division by $1 - \alpha$ has the effect of sliding the curves horizontally so they overlay one another; there are random deviations, but the curves appear to follow the same basic trend.

The same effective learning rate can obtained from different combinations of η and α. The curves show that these yield roughly the same average training time but differ in the location of the critical point where convergence becomes unlikely. In this example it appears that higher effective learning rates can be obtained with small η and large α rather than vice versa. That is, convergence is obtained at $\eta' = 100$, for example, with $\alpha = 0.99$ but not with the lower momentum values.

6.1.4 Typical *E(t)* Curves

Batch Mode Figures 6.4 and 6.5 show typical RMS error versus training time curves for various learning rates with constant momentum $\alpha = 0.5$. All curves were generated from the same initial weight vector. The training problem, network, and initialization parameters described in section 6.1.1 were used. Note that the learning rates in these figures are for the SSE error function; in Figures 6.1 through 6.3 the corresponding learning rates for the MSE function are 16 times larger.

Although the exact shape of the $E(t)$ curve is unique to each problem and choice of parameters (including the random starting point), several general things can be observed. At very low learning rates, the $E(t)$ curve is smooth, but training is very slow. The error almost always decreases with time, rarely if ever increasing, and there are long flat runs followed by relatively steep drop-offs. In this range, increased learning rates lead to faster

◄ **Figure 6.2**
Actual training times versus learning rate for various momentum values. At high learning rate and momentum, most networks either do not converge or converge to poor solutions and become stuck. Of the networks that do converge, however, most do so very quickly. Each vertical strip shows points for 100 networks initialized with different random weights. Note: Use of the MSE error function rather than SSE scales the learning rate by the size of the training set. See section 6.1.1 for details of the simulation.

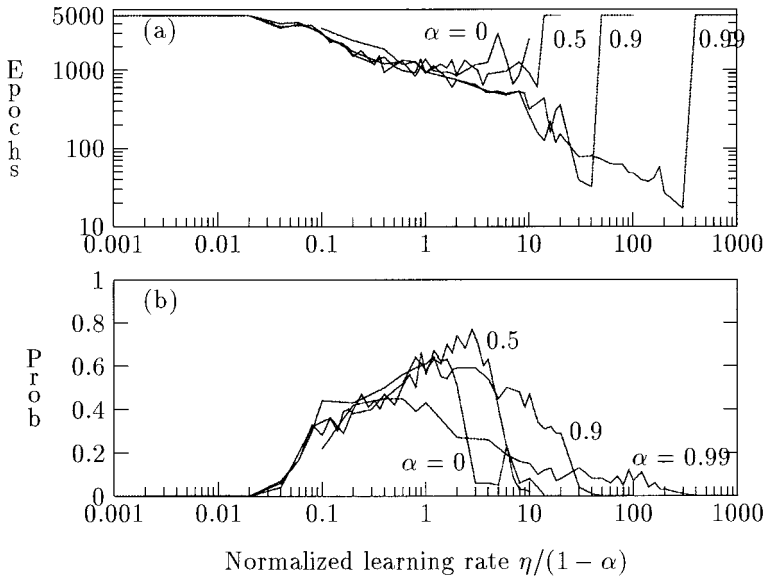


Figure 6.3
Average training time (a) and convergence probability (b) versus normalized learning rate for $\alpha = 0, 0.5, 0.9, 0.99$. With momentum, the effective learning rate is amplified to $\eta' = \eta/(1-\alpha)$. The same effective learning rate can obtained from different combinations of η and α, which yield roughly the same average training time but differ in the location of the critical point where convergence becomes unlikely. In this example, it appears that small learning rates and large momentums are more stable. Note: Use of the MSE error function rather than SSE normalizes the learning rate by the size of the training set. Simulation details are described in section 6.1.1.

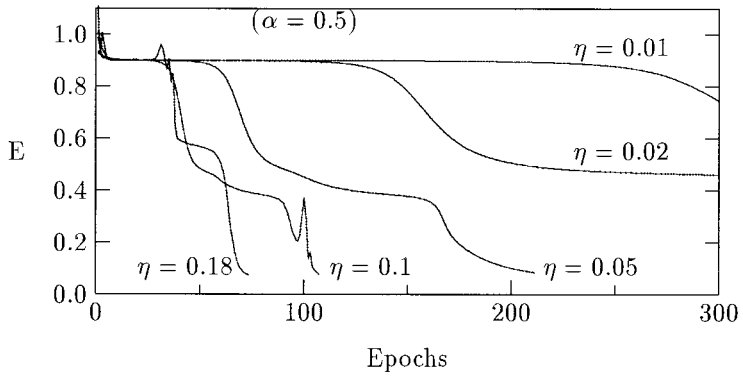


Figure 6.4
$E(t)$ curves for small SSE learning rates. At low learning rates, the $E(t)$ curves are smooth, but convergence is slow. As η increases, convergence time decreases but convergence is less reliable with occasional jumps in error. All networks were initialized with the same random weights.

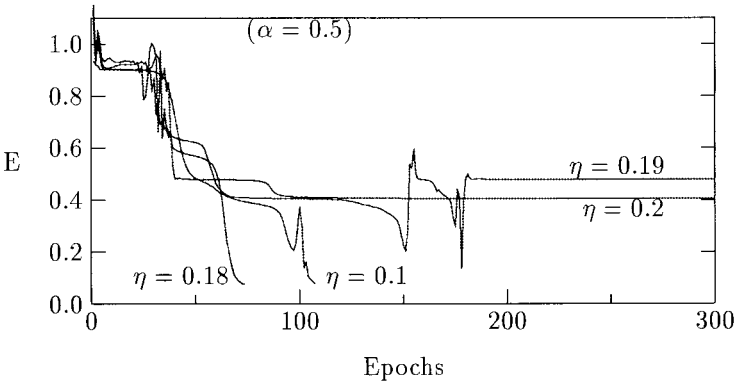


Figure 6.5
$E(t)$ curves for near-critical SSE learning rates. Below some critical learning rate, convergence time decreases as the learning rate η increases. At some point though the system becomes unstable and fails to converge. Note that the change is abrupt; at $\eta = 0.18$ it converges quickly but at $\eta = 0.19$ it overshoots the minimum and appears to get trapped on a plateau. The same network and initial weight vector were used in figure 6.4.

convergence. At the high end of the range, the system makes occasional "mistakes" leading to momentary jumps in error, but it usually recovers quickly.

Figure 6.1 showed that there is a critical learning rate above which convergence becomes very unlikely. Figure 6.4 shows $E(t)$ curves for near-critical learning rates. Just below the critical point, convergence becomes less reliable as η increases; mistakes become more common and the error may show transient chaotic oscillations. Finally, at even higher η values, the system becomes unstable and fails to converge. Note that the change is abrupt; at $\eta = 0.18$ the network converges quickly, at $\eta = 0.19$ it overshoots the minimum and appears to get trapped on a plateau, and at $\eta = 0.2$ it gets stuck sooner and never approaches the minimum. (The value $\eta = 0.2$ in this figure corresponds to $\eta = 3.2$ in figures 6.1 through 6.3.) In the analogy of a marble rolling down a hill, a similar abrupt change in outcomes may occur when a small change in the relative forces makes the difference between the marble rolling into one of two different valleys. In one case it has just enough energy to roll over a ridge and so ends up in valley A; in another case it just fails and rolls back down into valley B instead.

Overall, an aggressive (but subcritical) learning rate may produce faster learning even though it allows the error to increase occasionally. If the increases are mild and not too frequent, the benefits may outweigh the losses so the overall result is faster convergence. A small amount of chaos may also help the system explore more of the weight space. Beyond the critical value, however, the losses outweigh the benefits and the state either wanders aimlessly or gets stuck at high errors.

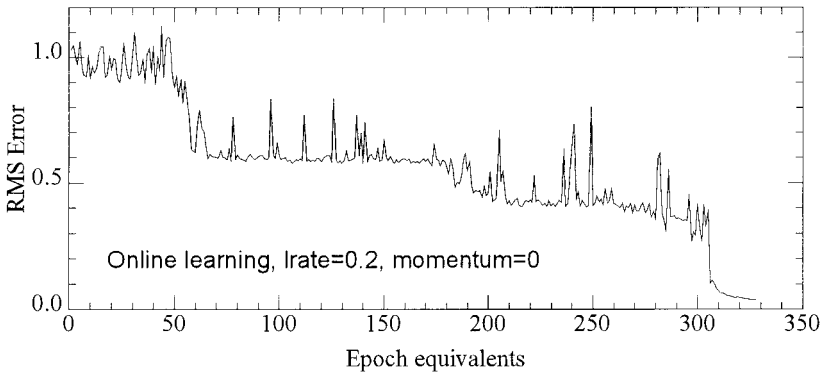


Figure 6.6
With on-line learning, patterns are chosen randomly and the weights are updated after each pattern presentation. This introduces noise into the training process, leading to noisy $E(t)$ curves. In general, larger learning rates cause higher noise levels.

On-Line Mode With on-line learning, jitter in the $E(t)$ graph is normal. Figure 6.6 shows a typical curve. Because weights are updated after each pattern presentation, there is a tendency for the error to be lower on the most recently presented patterns. Presentation of the patterns in random order avoids a bias toward patterns near the end of the training set, but the randomness introduces noise that shows up in the $E(t)$ curves. When things work well, the curve has a downward trend but is usually overlaid with noise whose amplitude is related to the learning rate. If the noise level is high, the network may never settle to a stable minimum so it is common to reduce the learning rate to zero gradually as training progresses.

Because of this randomness, different on-line learning trials with identical initial weights and training parameters will not follow the same trajectory. Figure 6.7 shows four different training runs started with the same initial weights. One of the networks converges quickly while the other three appear to get stuck in local minima. Cases (a) and (d) appear to follow the same general path but the local deviations are different.

As in batch learning, there are interactions between the learning rate and momentum in on-line learning. Momentum causes the same learning rate amplification effect when learning rates are small. Figure 6.8 shows two on-line training trajectories for networks with the same initial weights and learning rates but different momentum values. In the figure the weight changes are small and convergence is slow for a small learning rate, $\eta = 0.01$, and no momentum. With the same learning rate and $\alpha = 0.95$, the weight changes are larger and this example converges quickly. The random properties of the trajectories make it impossible to say that the $\alpha = 0.95$ case will always converge faster though.

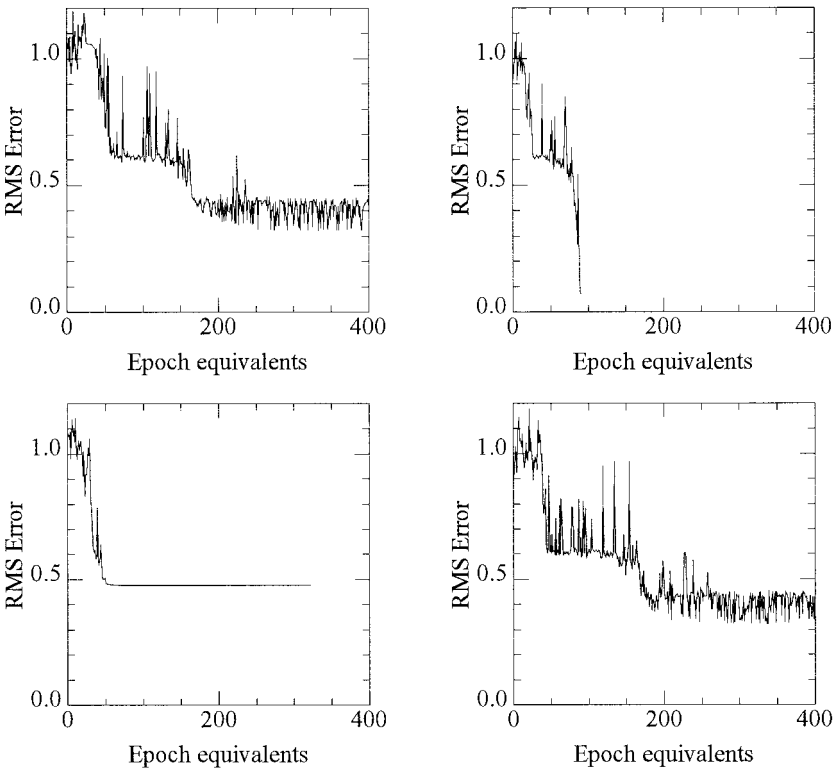


Figure 6.7
$E(t)$ curves for on-line learning. On-line learning is stochastic so different runs with identical initial weights and training parameters will follow different trajectories. Shown are four different trials of on-line back-propagation applied to the 4-input parity problem with $\eta = 0.3$, $\alpha = 0$. All networks started with the same initial random weights. This learning rate appears high for this example—the noise level is high and three of the nets get stuck in a local minimum.

6.1.5 Learning Rate Selection

The preceding examples illustrate some effects of learning rate and momentum variations. In general, there is a trade-off between convergence speed and reliability of convergence. It should be clear that there is no fixed learning rate that is best for all problems.

6.1.6 Selection from Trace(H)

It is well-known that the optimal global learning rate for gradient descent on a linear problem is $\eta = 1/\lambda_{max}$ where λ_{max} is the largest positive eigenvalue of the Hessian matrix

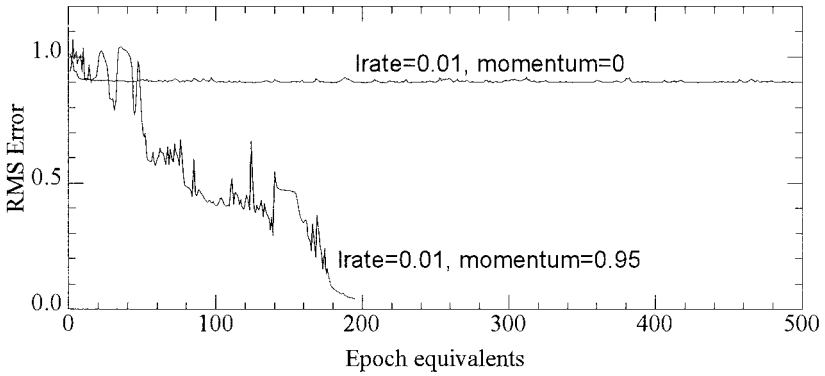


Figure 6.8
On-line learning $E(t)$ curves with and without momentum. Momentum also has a learning rate amplification effect in on-line learning. With a small learning rate, $\eta = 0.01$, (using SSE) and no momentum, the weight changes are small and convergence is slow. With the same learning rate and $\alpha = 0.95$, the weight changes are larger and this example converges quickly. Both examples were initialized with the same random weights.

H of second derivatives of the error with respect to the weights. Since λ_{max} is unknown unless **H** is analyzed, an estimate must be used. Assuming **H** is nonnegative-definite, all eigenvalues are nonnegative $\lambda_i \geq 0$ and λ_{max} can be bounded by

$$\lambda_{max} \leq \sum_i \lambda_i = \text{trace}(\mathbf{H}) \tag{6.1}$$

where $\text{trace}(\mathbf{H}) = \sum_i h_{ii}$. Estimates of the h_{ii} components can be obtained efficiently from the diagonal Hessian approximation (section 8.6.3). Of course, this overestimates λ_{max} so the resulting η value may be smaller than necessary but it is a reasonable starting point for an adaptive method.

A problem is that **H** will not be nonnegative-definite in general. At an arbitrary point on the $E(\mathbf{w})$ surface far from a minima, the matrix is likely to have both positive and negative eigenvalues since E curves up in some directions and down in others. There are, however, standard methods to make the approximation nonnegative.

6.1.7 Selection by On-Line Eigenvalue Estimation

LeCun, Simmard and Pearlmutter [94] describe a method for choosing the learning rate from an on-line estimate of the principal eigenvalue of the Hessian. The recipe is

1. Pick a random unit-length vector Ψ and two small positive constants α and γ, for example, $\alpha = 0.01$ and $\gamma = 0.01$.

2. Pick a training pattern X^p, do the forward and backward propagations and store the resulting gradient vector $G_1 = \nabla E^p(\mathbf{w})$.

3. Add a perturbation $\alpha\mathcal{N}(\Psi)$ to the current weight vector $\mathbf{w}$, where $\mathcal{N}(\cdot)$ denotes vector normalization: $\mathcal{N}(\mathbf{v}) = \mathbf{v}/\|\mathbf{v}\|$.

4. Perform a forward and backward propagation on the same training pattern using the perturbed weight vector and store the resulting gradient $G_2 = \nabla E^p(\mathbf{w} + \alpha\mathcal{N}(\Psi))$.

5. Update Ψ with the running average $\Psi \leftarrow (1-\gamma)\Psi + \frac{\gamma}{\alpha}(G_2 - G_1)$.

6. Restore the weight vector to its original value.

7. Go to step 2 and repeat until $\|\Psi\|$ stabilizes.

8. Set the learning rate $\eta = \|\Psi\|^{-1}$ and continue with regular training.

The constant α controls the size of the perturbation; small values give better estimates but increase the chance of numerical errors. The constant γ smooths the estimate; small values give more accurate estimates but increase the convergence time. In [94] the authors recommend starting with a relatively large value, for example, $\gamma = 0.1$, and decreasing it until the fluctuations in $\|\Psi\|$ are less than about 10%. The point is to obtain an estimate of the proper order of magnitude for Ψ, not to calculate a precise value. Typically, $\|\Psi\|$ will converge in a few hundred iterations. This is an on-line estimate so each iteration has a cost comparable to two pattern evaluations and back-propagations. For large problems, the cost is small compared to the time needed to do a single training epoch.

The iteration is similar to the power method for calculating the largest positive eigenvalue of a matrix. A random vector Ψ can be expressed as a combination of the eigenvectors $\{\mathbf{v}_i\}$ of a given matrix (the Hessian $\mathbf{H}$ in this case)

$$\Psi = \sum_i a_i \mathbf{v}_i. \tag{6.2}$$

$\mathbf{H}$ is assumed to have full rank here. When Ψ is multiplied by $\mathbf{H}$, each eigenvector component grows by an amount proportional to its corresponding eigenvalue

$$\mathbf{H}\Psi = \sum_i a_i \lambda_i \mathbf{v}_i. \tag{6.3}$$

The component parallel to the principle eigenvector (call it $\mathbf{v}_1$) grows the most because its eigenvalue is largest. With iteration and renormalization

$$\Psi \leftarrow \mathbf{H}\mathcal{N}(\Psi),$$

the $\mathbf{v}_1$ component eventually dominates and Ψ approaches $\lambda_1\mathbf{v}_1$, giving $\lambda_{max} \approx \|\Psi\|$. In the recipe just given, the product $\mathbf{H}\Psi$ is approximated by a finite-difference

$$\mathbf{H}\Psi = \frac{\nabla E(\mathbf{w} + \alpha\Psi) - \nabla E(\mathbf{w})}{\alpha} + O(\alpha^2). \tag{6.4}$$

Exact calculation would require the equivalent of two training epochs—one for the original $\mathbf{w}$ and one for the perturbation. The preceding recipe is a further approximation using a running average of on-line estimates to approximate $E(\mathbf{w})$ and $E(\mathbf{w} + \Delta\mathbf{w})$.

Pearlmutter [297] describes an exact method for finding a product of $\mathbf{H}$ with a vector without evaluation of $\mathbf{H}$. This could be used instead of the finite difference approximation in equation 6.4.

6.1.8 Delta Attenuation in Layered Networks

It has been noted that the first layers of layered networks often learn very slowly because the error derivatives are attenuated as they propagate back from the output layer toward the input. For a node with value y, the sigmoid derivative is $y(1 - y)$ and takes values between 0 and 1/4; the tanh derivative is $1 - y^2$ and takes values between 0 and 1. Each node nonlinearity contributes a derivative factor that is normally less than 1, so the derivative may become very small after passing through several layers. This assumes the weights are $O(1)$ magnitude. When the weights are smaller, as they usually are just after random initialization, there is additional attenuation. The result is that $|\partial E/\partial w|$ tends to be very small for weights close to the inputs and so they change very slowly. Deep networks with many layers have been avoided for this reason because almost no learning occurs in the initial layers.

Because the partial derivatives are so small, larger learning rates may be appropriate for hidden units. Values as high as 10 can sometimes be used without causing instability [168]. If no other information is available it might be assumed that the node outputs y are uniformly distributed on [0, 1] in which case the expected attenuation due to each sigmoid derivative is $E[y(1 - y)] = 1/6$. Rigler et al. [316] suggest rescaling the back-propagated derivatives by 6 to compensate. This would be equivalent to increasing the learning rate by 6 for weights into the last layer, by 36 for weights into the second-to-last layer, $6^3 = 216$ for weights 3 layers back from the output, and so on. These are only heuristics, however; it is not a necessary fact that partial derivatives are smaller for weights farther from the outputs. These are based on assumptions about the node nonlinearities and how the weights are initialized.

It seems reasonable to use larger learning rates in earlier layers to compensate for delta attenuation, but the values will depend on the particular training data and network used. The methods described attempt to rescale the learning rates relative to some global value so that each layer sees derivatives of roughly the same size, but they do not say how to control the global value. Without an automatic control algorithm it may be difficult to find

a set of parameters that are both stable and efficient. A fixed learning rate is not necessarily appropriate anyway because conditions change as learning progresses. Adaptive learning rate techniques have the advantage of being able to adjust the rate dynamically to match current conditions. Rprop (section 9.6) seems to work better than some other methods in this case because the learning rate adjustments and weight updates depend only on the signs of the derivatives, not their magnitudes. Appropriate values can be found for each layer so early layers learn faster than they would otherwise and deep networks are not as difficult to train.

6.1.9 Learning Rate Fan-In Scaling

Scaling of the learning rate based on the number of inputs to each node was suggested by Plaut, Nowlan, and Hinton [299]. In [332], back-propagated δ values are scaled by the fan-in, which has the same effect. Some justification for this can be based on an eigenvalue analysis of the Hessian (see Appendix A.2).

An intuitive explanation of why this might help is that immediately after initialization with small random weights all of the h hidden units compute approximately linear functions and their combined effect on the following layer is approximately that of a single linear "virtual" unit. If there is an optimum learning rate, say η^*, for the output weight of this virtual unit, then the learning rate for the output weights of the h actual units should be η^*/h so that they sum to η^*. This suggests relative values for learning rates in different layers, but it does not say how to choose η^*. Also, the same conditions may not hold later in learning.

In [168], a separate learning rate is calculated for each hidden layer based on the number of weights into and out of each node. The initial value is then adapted based on the magnitudes of the back-propagated δ's.

It is not clear if this heuristic has much effect unless there are large differences in layer sizes [9]. The heuristic of scaling the weight initialization range based on fan-in may achieve the same results without committing the system to a learning rate that may be inappropriate at later stages of learning. Algorithms that adjust a separate learning rate for each weight (e.g., Rprop) may be less troublesome; they generally achieve the same result in cases where this heuristic would help without causing problems in cases where it would not.

6.2 Momentum

Back-propagation with momentum can be viewed as gradient descent with smoothing. The idea is to stabilize the weight trajectory by making the weight change a combination of the

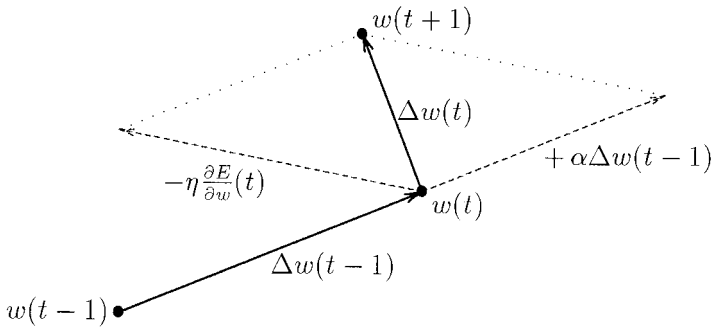


Figure 6.9
With momentum, the current weight change $\Delta\mathbf{w}(t)$ is a combination of a step down the negative gradient, $-\eta\frac{\partial E}{\partial \mathbf{w}}(t)$, plus a fraction $0 \leq \alpha < 1$ of the previous weight change. For $\alpha \approx 1$, opposing weight change components (horizontal) approximately cancel while complementary components (vertical) sum, producing a smoothing effect on the weight trajectory.

gradient-decreasing term in equation 5.23 plus a fraction of the previous weight change. The weight update rule is [330]

$$\Delta\mathbf{w}(t) = -\eta\,\frac{\partial E}{\partial \mathbf{w}}(t) + \alpha\Delta\mathbf{w}(t-1). \tag{6.5}$$

This is sometimes called the *generalized delta rule*. The weight change $\Delta\mathbf{w}(t)$ is a combination of a step down the negative gradient, $-\eta\frac{\partial E}{\partial \mathbf{w}}(t)$, plus a fraction $0 \leq \alpha < 1$ of the previous weight change (figure 6.9). For $\alpha \approx 1$, opposing weight change components approximately cancel while complementary components sum, leading to a smoothing effect on the weight trajectory. When successive gradients point in the same direction, the terms reinforce each other, leading to accelerated learning.

The smoothing effect of momentum can be illustrated by expansion of (6.5)

$$\begin{aligned}
\Delta\mathbf{w}(t) &= -\eta\,\frac{\partial E}{\partial \mathbf{w}}(t) + \alpha\Delta\mathbf{w}(t-1) \\
&= -\eta\,\frac{\partial E}{\partial \mathbf{w}}(t) + \alpha\left(-\eta\frac{\partial E}{\partial \mathbf{w}}(t-1) + \alpha\Delta\mathbf{w}(t-2)\right) \\
&= -\eta\,\frac{\partial E}{\partial \mathbf{w}}(t) + \alpha\left(-\eta\frac{\partial E}{\partial \mathbf{w}}(t-1) + \alpha\left(-\eta\frac{\partial E}{\partial \mathbf{w}}(t-2) + \ldots\right)\right) \\
&= -\eta\sum_{k=0}^{\infty}\alpha^k\frac{\partial E}{\partial \mathbf{w}}(t-k).
\end{aligned} \tag{6.6}$$

That is, with momentum the weight update is an exponential average of *all* the previous gradient terms rather than just the most recent term. Because $\alpha < 1$, the contribution from earlier derivative terms decays with each time step and the sum is dominated by the more recent terms. The time-constant of the system is controlled by α. For small α, the coefficients decay quickly as k increases so the system "forgets" earlier terms quickly. For large $\alpha \to 1^-$, however, the coefficients decay very slowly and the system has a long memory; the system will be stable but slow to react to changes in the error term.

The learning accelerating effect of momentum can be illustrated by considering the case where the derivative is constant, $\partial E/\partial \mathbf{w}(t) = J$. This is a reasonable approximation when η is very small so $\mathbf{w}$ does not change much with each step. It is also reasonable on flat areas of $E(\mathbf{w})$ where the gradient is small. Then

$$\begin{aligned}\Delta \mathbf{w}(t) &= -\eta \sum_{k=0}^{\infty} \alpha^k \frac{\partial E}{\partial \mathbf{w}}(t-k) \\ &= -\eta J \sum_{k=0}^{\infty} \alpha^k \\ &= \frac{-\eta J}{1-\alpha} \end{aligned} \tag{6.7}$$

where the identity $\sum_{k=0}^{\infty} \alpha^k = 1/(1-\alpha)$ (for $|\alpha| < 1$) is used in the last step. Without momentum, $\Delta \mathbf{w}(t)$ would be $-\eta J$. With momentum, however, $\Delta \mathbf{w}(t) = -\eta J/(1-\alpha)$. Momentum thus has the effect of amplifying the learning rate from η to the effective value $\eta' = \eta/(1-\alpha)$.

As $\alpha \to 1$, the effective learning rate can become very large, but the time constant also becomes large. Weight changes are affected by error information from many past cycles, which may make it difficult for the system to respond quickly to new conditions in the $E(w)$ surface. The weight trajectory may coast over a minimum but be unable to stop because of the continuing effects of earlier weight changes.

6.2.1 Effects of Momentum

As noted earlier, batch-mode back-propagation with a small learning rate is an approximation of gradient descent. Two problems with gradient descent are (1) when the learning rate is small, progress may be very slow (figure 6.10a) and (2) when the learning rate is too large and the error surface contains "ravines," the weight vector may oscillate wildly from one side of the valley to the other while creeping slowly along the length of the valley to the minimum, an effect sometimes called cross-stitching (figure 6.10b). Upon reaching the neighborhood of a minimum, it may overshoot many times before settling down.

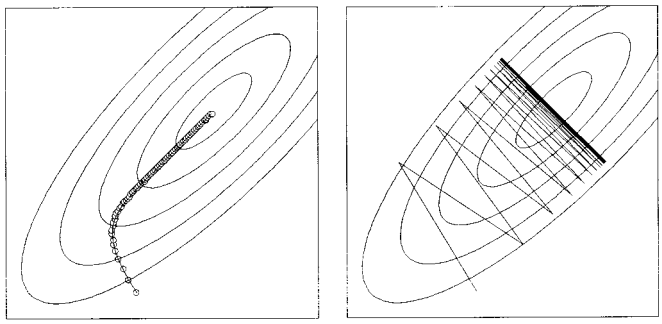


Figure 6.10
Gradient descent trajectories: (a) with a small step size (0.01), pure gradient descent follows a smooth trajectory, but progress may be very slow; (b) "cross-stitching"; When the step size is too big (0.1) and the error surface has "valleys," the trajectory may oscillate wildly from one side to the other while creeping slowly along the length of the valley to the minimum. Upon reaching the neighborhood of a minimum, it may overshoot many times before settling down.

Briefly, momentum has the following effects:

- It smooths weight changes by filtering out high frequency variations. When the learning rate is too high, momentum tends to suppress cross-stitching because consecutive opposing weight changes tend to cancel. The side to side oscillations across the valley damp out leaving only the components along the axis of the valley, which add up.
- When a long sequence of weight changes are all in the same direction, momentum tends to amplify the effective learning rate to $\eta' = \eta/(1-\alpha)$, leading to faster convergence.
- Momentum may sometimes help the system escape small local minima by giving the state vector enough inertia to coast over small bumps in the error surface.

Cross-stitching is a problem for gradient descent when the learning rate is too large and error surface has steep-sided ravines that have a shallow slope along the axis. (This can be stated more technically in terms of the eigenvalues of the Hessian matrix, see section A.2.) Without momentum, the network has only the gradient information to guide its path. Because the gradient on one side of a steep valley points almost directly across the valley and has only a very small component along it, the weight vector tends to jump back and forth across the valley and progress along the axis of the valley is slow relative to the size of the weight changes. Also, upon reaching the neighborhood of a minimum, the weight vector may overshoot many times before settling down. With the momentum term, side to side oscillations tend to cancel but steps along the axis sum so progress along the

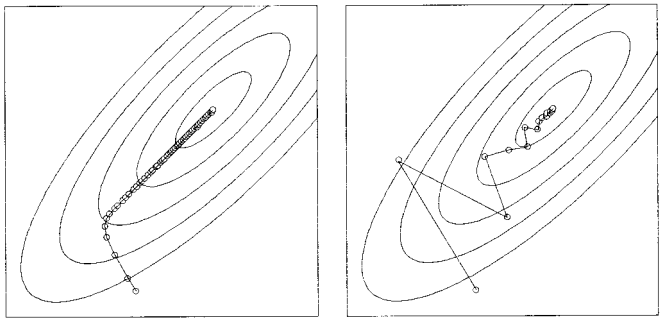


Figure 6.11
Gradient descent trajectories with momentum. With momentum, opposite (side to side) changes tend to cancel while complementary changes (along the length of the valley) tend to sum. The overall effect is to stabilize the oscillations and accelerate convergence. (a) When the step size is small, momentum acts to accelerate convergence (step size 0.01 and momentum 0.99, cf. figure 6.10a). (b) Small amounts of momentum also help to damp oscillations when the step size is too large (step size 0.1 and momentum 0.2, cf. figure 6.10b).

valley is faster. The network can follow the path of the ravine better, so the learning rate can often be increased, leading to faster training times. Learning acceleration and oscillation dampening effects of momentum can be seen in comparing Figures 6.10 and 6.11.

Too Much Momentum With momentum, the state vector has a tendency to keep moving in the same direction. Weight changes are affected by error information from many past cycles—the larger the momentum, the stronger the lingering influence of previous changes. In effect, momentum gives the weight state inertia, which "keeps the marble rolling," allowing it to coast over flat spots and perhaps out of small local minima.

A little inertia is useful for stabilization but too much may make the system sluggish; it may overshoot good minima or be unable to follow a curved valley in the error surface. The system may coast past minima and out onto high plateaus where it becomes stuck (section 6.2.2).

Interaction with Learning Rate Many studies have claimed that momentum tends to make choice of learning rate η less critical. When η is too small, successive weight updates tend to be in the same direction and momentum effectively amplifies η to $\eta/(1-\alpha)$. When η is too large, successive updates tend to be in nearly opposite directions and momentum causes them to cancel out, effectively reducing the learning rate. Some support is seen in figure 6.1 where the probability of convergence density function is wider (on a logarithmic scale) for $\alpha = 0.9$ than for $\alpha = 0$. On a linear scale, however, the width of the density

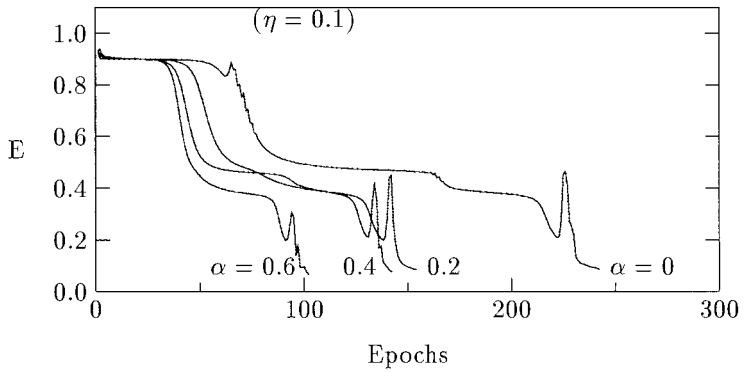


Figure 6.12
At the low end of the momentum range, α increases generally lead to faster convergence. Here the $E(t)$ curves are smooth. Occasional error spikes may occur but the system recovers quickly. All trajectories start from the same random weights.

function decreases in agreement with results that show that momentum reduces the stable range of learning rates for the LMS algorithm [343, 326].

6.2.2 Typical *E(t)* Curves with Momentum

Figures 6.12 and 6.13 show $E(t)$ curves for various momentum values and fixed learning rate. All curves were generated from the same initial weight vector as in figures 6.4 and 6.5. Simulation details are described in section 6.1.1.

At low momentum values, the $E(t)$ curves are smooth and larger values of α lead to faster convergence (assuming reasonable learning rates). Occasional spikes may occur but the system recovers quickly. The curves in figure 6.12 are all qualitatively similar so the increased convergence speed may be due mostly to amplification of the effective learning rate. That is, the system appears to be following the same basic trajectory at varying speeds.

As α increases past a certain point, however, convergence becomes unreliable (figure 6.13). At $\alpha = 0.6$ the system converges quickly, but at larger values it becomes stuck in a poor minimum. For $\alpha = 0.99$, $E(t)$ oscillates strongly and the system jumps from a poor minimum to a worse one at about $t = 100$.

6.2.3 Small-Signal Analysis, Momentum Only

The acceleration and smoothing effects of momentum can be explained in terms of small-signal analyses. The weight update equation with momentum is

$$\Delta w(t) = -\eta \frac{\partial E}{\partial w}(t) + \alpha \Delta w(t-1). \tag{6.8}$$

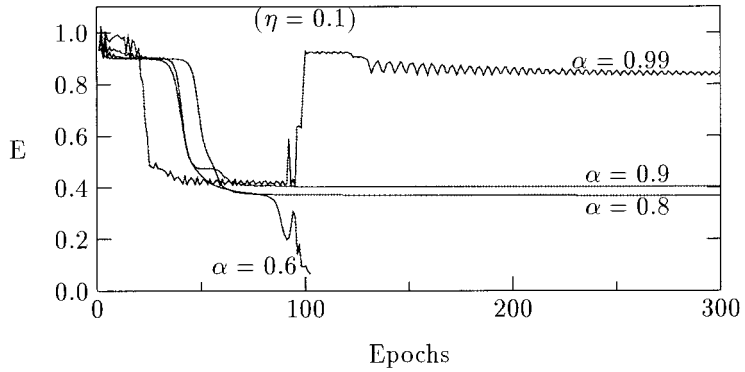


Figure 6.13
$E(t)$ curves for large momentum values. Convergence becomes unreliable when the momentum is too large for the learning rate η. This system converges quickly with $\alpha = 0.6$ but not with higher values. With $\alpha = 0.99$, $E(t)$ oscillates strongly and the system jumps from a poor minimum to a worse one at about $t = 100$. All trajectories start from the same random weights.

Convert this discrete-time iteration to a continuous-time system by the approximation $\dot{w} \equiv \frac{\partial w}{\partial t} \approx \frac{\Delta w}{\Delta t}$ and assume $\Delta t = 1$. Then

$$\dot{w}(t) = -\eta J + \alpha \dot{w}(t-1) \tag{6.9}$$

where $J = \partial E / \partial w$. Another discrete-time to continuous-time approximation for the second derivative gives

$$\dot{w}(t) - \dot{w}(t-1) \approx \ddot{w}(t)\Delta t \tag{6.10}$$

and

$$\alpha \ddot{w}(t) + (1-\alpha)\dot{w}(t) = -\eta J. \tag{6.11}$$

This second-order differential equation is easily solved for certain special cases. Laplace transforms (e.g., [292]) are used in the following discussions.

Impulse Response Assume J is an impulse at 0, that is, $J(t) = J_o \delta(t)$. This approximates the case of encountering a "cliff" in the $E(\mathbf{w})$ surface, where J is large, and then coasting on a flat plateau where $J \approx 0$. Taking Laplace transforms gives,

$$s^2 W + \frac{1-\alpha}{\alpha} s W = -\frac{\eta}{\alpha} J_o$$

$$W = -\frac{1}{s(s+\frac{1}{\tau})} \frac{\eta J_o}{\alpha} \tag{6.12}$$

where $\tau = \alpha/(1-\alpha)$. For $0 < \alpha < 1$, this has the solution

$$w(t) = w(0) - \frac{\eta J_o}{1-\alpha}\left(1 - e^{-t/\tau}\right). \tag{6.13}$$

For $\alpha \geq 1$, the solution is unstable. Otherwise $w(t)$ asymptotically approaches a final value $w(\infty) = w(0) - \eta J_o/(1-\alpha)$. Instead of taking a single step ηJ_o at $t = 0$, it takes many steps that asymptotically add up to a value $1/(1-\alpha)$ times as large.

Momentum thus has the effect of amplifying the learning rate from η to the effective value $\eta' = \eta/(1-\alpha)$. For $\alpha \to 1$, the effective learning rate can become very large, but the time constant τ also becomes large, which may make it difficult for the system to respond quickly to new conditions in the $E(w)$ surface.

Step Response Assume J is a step function at $t = 0$, that is, $J(t) = J_o u(t)$ where $u(t)$ is the unit step function

$$u(t) = \begin{cases} 0 & t < 0 \\ 1 & t \geq 0. \end{cases}$$

This is reasonable when η is small and the local error surface is nearly flat; the gradient changes very little in one iteration and can be approximated by a constant. Laplace transforms give

$$s^2 W + \frac{1-\alpha}{\alpha} s W = -\frac{\eta}{\alpha}\frac{J_o}{s} \tag{6.14}$$

$$W = -\frac{1}{s^2(s + \frac{1}{\tau})}\frac{\eta J_o}{\alpha} \tag{6.15}$$

where $\tau = \alpha/(1-\alpha)$ again. The solution is unstable for $\alpha \geq 1$. For $0 < \alpha < 1$, this has the solution

$$w(t) = w(0) - \frac{\eta J_o}{1-\alpha} t - \eta J_o \frac{\alpha}{(1-\alpha)^2}\left(1 - e^{-t/\tau}\right). \tag{6.16}$$

This approximates a ramp function —a linear rise with t plus a transient term similar to (6.13). For $t \gg \tau$,

$$w(t) \approx w(0) - \frac{\eta J_o}{1-\alpha}\left(t - \frac{\alpha}{(1-\alpha)^2}\right) \tag{6.17}$$

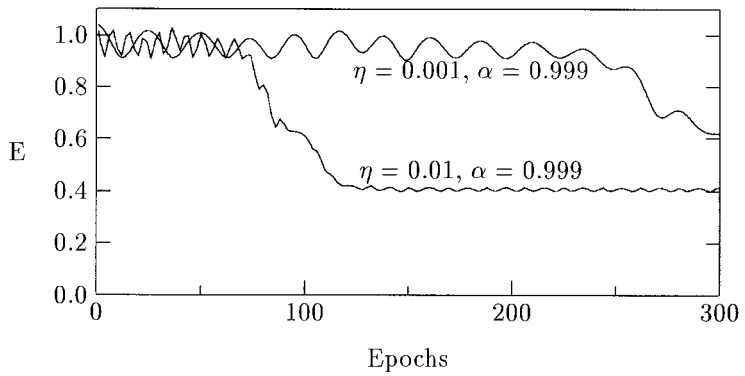


Figure 6.14
Oscillation in $E(t)$ due to momentum. With momentum, the weight vector has inertia, which allows it to "coast up hillsides" in the $E(w)$ surface. The larger the momentum and the smaller the learning rate, the farther it can rise and the longer it takes to stop. This, in combination with the $E(w)$ surface, can lead to oscillation. Mathematically, smaller η and larger α give the dynamic system a longer time-constant, visible here in a lower oscillation frequency. (These values were chosen to exaggerate the effect; they are not necessarily recommended.)

Frequency Response The common term

$$W(s) \propto \frac{1}{s + \frac{1}{\tau}}$$

corresponds to a leaky integrator and has a low-pass frequency response. (A leaky integrator is an exponentially weighted averager.) This is another justification for the statement that momentum helps filter out high frequency oscillations in the weight changes. Recall that $\tau = \alpha/(1-\alpha)$ becomes large as $\alpha \to 1$. The time-domain impulse response is

$$h(t) \propto e^{-t/\tau}.$$

Convolution of a signal $J(t)$ with $h(t)$ yields an exponentially weighted average $\overline{J(t)}$ which weights recent J values more heavily than older values.

Generally, smaller η and larger α values give the dynamic system a longer time-constant, visible in figure 6.14 as a lower frequency of oscillation.

6.2.4 Small-Signal Analysis, Momentum, and Weight Decay

It is relatively easy to extend these linear small-signal analyses to include weight decay terms. The new weight update equation is

$$\Delta w(t) = -\rho w(t) - \eta \frac{\partial E}{\partial w}(t) + \alpha \Delta w(t-1) \tag{6.18}$$

where $0 \leq \rho \ll 1$ is the weight decay parameter. The same discrete-time to continuous-time approximations give

$$\dot{w}(t) = -\rho w(t) - \eta J + \alpha \dot{w}(t-1),$$

$$\dot{w}(t) - \dot{w}(t-1) \approx \ddot{w}(t)\Delta t,$$

and

$$\alpha \ddot{w}(t) + (1-\alpha)\dot{w}(t) + \rho w = -\eta J. \tag{6.19}$$

Impulse Response When J is an impulse at 0, that is, $J(t) = J_o\delta(t)$, Laplace transforms give

$$W = -\frac{\eta}{\alpha} \frac{J_o}{(s^2 + \frac{1-\alpha}{\alpha}s + \frac{\rho}{\alpha})}. \tag{6.20}$$

The denominator has roots

$$s = \frac{1}{2\alpha}\left(-\beta \pm \sqrt{\beta^2 - 4\rho\alpha}\right) \tag{6.21}$$

where $\beta = 1 - \alpha$. For $\beta^2 - 4\rho\alpha > 0$, both roots are real. Critical damping occurs at

$$\rho_o = (1-\alpha)^2/(4\alpha). \tag{6.22}$$

This is a decreasing function of α. As $\alpha \to 1$, ρ must approach 0 to prevent oscillation. For $\rho < \rho_o$, both roots are real. For larger values, the roots are complex and the solution is an exponentially decaying sinusoid. Convergence of these types of systems is usually fastest when the system is slightly underdamped.

Step Response When J is a step function at $t = 0$, that is, $J(t) = J_o u(t)$, Laplace transforms give

$$W = -\frac{\eta}{\alpha} \frac{J_o}{s(s^2 + \frac{1-\alpha}{\alpha}s + \frac{\rho}{\alpha})}. \tag{6.23}$$

Similar arguments are made by Bailey [15] using the damped oscillator equation

$$\ddot{w} + b\dot{w} + kw = 0. \tag{6.24}$$

Here $b = (1-\alpha)/\alpha$ and $k = \rho/\alpha$. They require $b < 1/N$ in order to average over all N patterns in the training set and choose $b = 1/(2N)$ as a reasonable value. This corresponds to $\alpha = 2N/(2N+1)$. Critical damping occurs when $k = (b/2)^2$, which corresponds to the value in equation 6.22.

6.3 Remarks

The preceding sections illustrate some effects that learning rate and momentum have on training and list some hints for recognizing when parameter changes are needed. It should be emphasized that the illustrations are based on a single small classification problem. The actual curves are unique to the example and different curves will be obtained from different networks with different training sets and initialization conditions. The examples are intended to be representative in that many problems will show qualitative similarities; quantitative differences should be expected however.

The main benefit of good parameters is faster training and the prevention of divergence to avoidable bad solutions. As in the more general case of tuning an optimization method, serious training difficulty may be an indication of problems more basic than optimization parameters. That is, if careful tuning of learning rate and momentum are needed to obtain good solutions, the effort might be better spent in reconsidering more fundamental things like the choice of representation or network architecture.

The difficulty of choosing a good learning rate *a priori* is one of the reasons adaptive learning rate methods are useful and popular. Most adapt parameters dynamically as appropriate for changing conditions and many assign different rates appropriate for different parts of the network. A few methods are summarized in chapter 9.