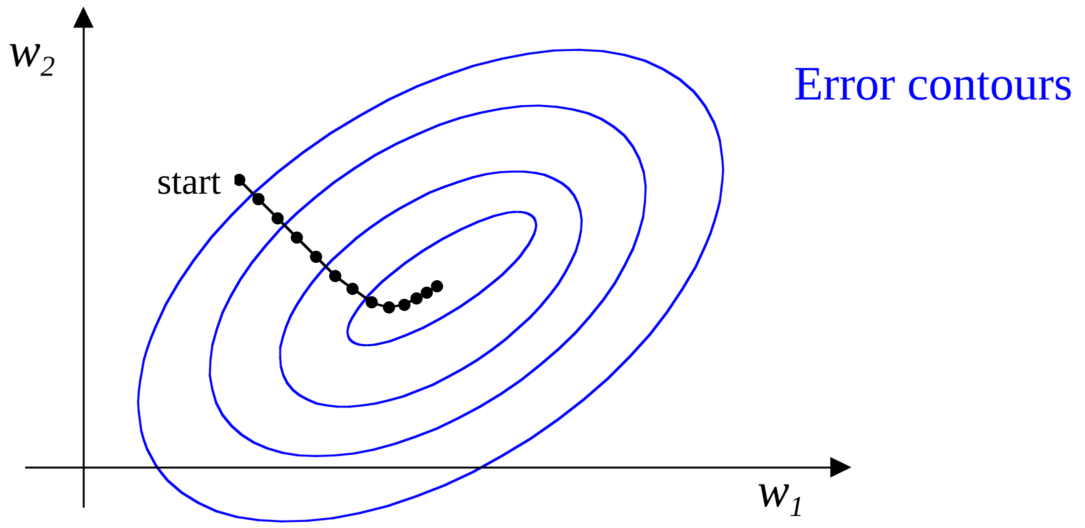
Learning with Momentum, Conjugate Gradient Learning

Introduction to Neural Networks : Lecture 8

Visualising Learning

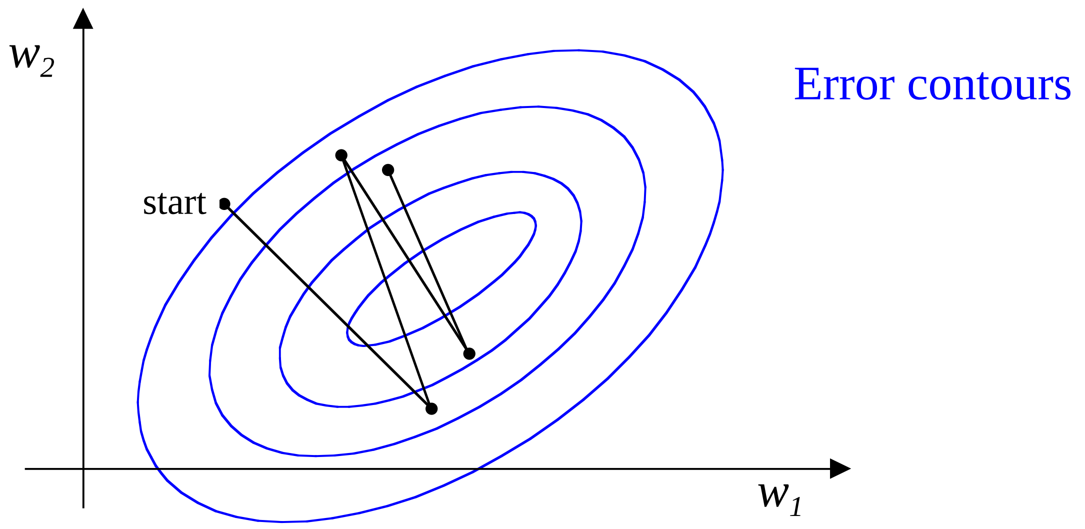
Visualising neural network learning is difficult because there are so many weights being updated at once. However, we can plot error function contours for pairs of weights to get some idea of what is happening. The weight update equations will produce a series of steps in weight space from the starting position to the error minimum:



True gradient descent produces a smooth curve perpendicular to the contours. Weight updates with a small step size η will result in the good approximation to it shown.

Step Size Too Large

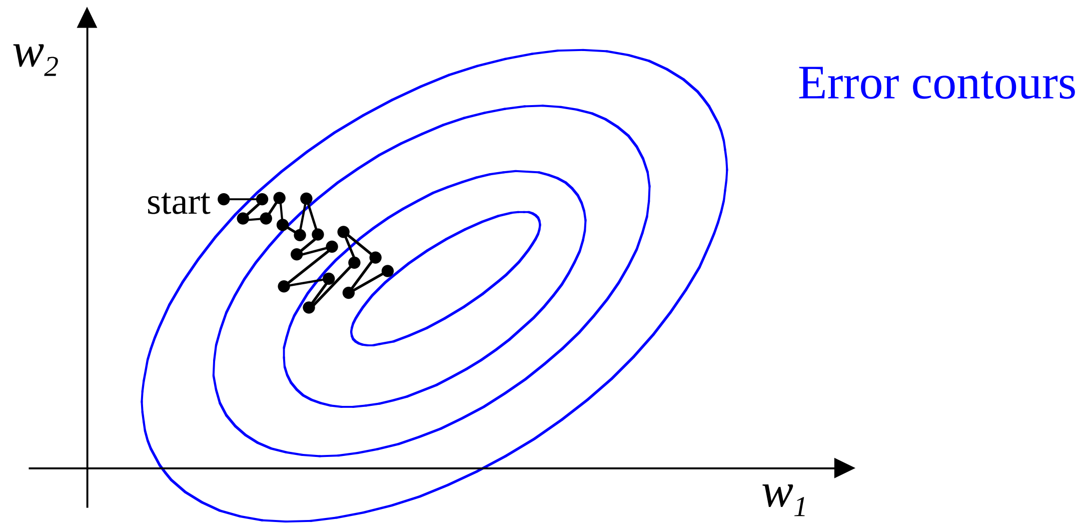
If the our learning rate step size is set too large, the approximation to true gradient descent will be poor, and we will end up with overshoots, or even divergence:



In practice, we don't need to plot error contours to see if this is happening. It is obvious that, if the error function is fluctuating, we should reduce the step size. It is also worth checking individual weights and output activations as well. On the other hand, if everything is smooth, it is worth trying to increase η and seeing if it stays smooth.

On-line Learning

If we update the weights after each training pattern, rather than adding up the weight changes for all the patterns before applying them, the learning algorithm is no longer true gradient descent, and the weight changes will not be perpendicular to the contours:



If we keep the step sizes small enough, the erratic behaviour of the weight updates will not be too much of a problem, and the increased number of weight changes will still get us to the minimum quicker than true gradient descent (i.e. batch learning).

Learning with Momentum

A compromise that will smooth out the erratic behaviour of the on-line updates, without slowing down the learning too much, is to update the weights with the moving average of the individual weight changes corresponding to single training patterns.

If we label everything by the time t (which we can conveniently measure in weight update steps), then implementing a moving average is easy:

$$\Delta w_{hl}^{(n)}(t) = \eta \sum_p delta_l^{(n)}(t).out_h^{(n-1)}(t) + \alpha.\Delta w_{hl}^{(n)}(t-1)$$

We simply add a ***momentum*** term $\alpha.\Delta w_{hl}^{(n)}(t-1)$ which is the weight change of the previous step times a momentum parameter α. If α is zero, then we have the standard on-line training algorithm used before. As we increase α towards one, each step includes increasing contributions from previous training patterns. Obviously it makes no sense to have α less than zero, or greater than one. Good sizes of α depend on the size of the training data set and how variable it is. Usually, we will need to decrease η as we increase α, so that the total step sizes don't get too large.

Learning with Line Searches

Learning algorithms which work by taking a sequence of steps in weight space all have two basic components: the ***step size*** $size(t)$ and the ***direction*** $dir_{hl}^{(n)}(t)$ such that

$$\Delta w_{hl}^{(n)}(t) = size(t).dir_{hl}^{(n)}(t)$$

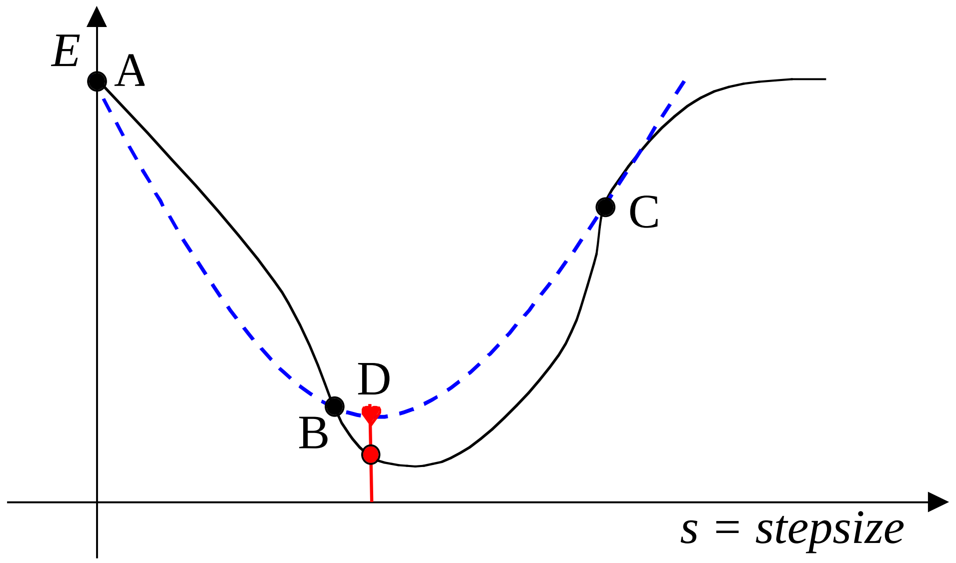
For gradient descent algorithms, such as Back-Propagation, the chosen direction is given by the partial derivatives of the error function $dir_{hl}^{(n)}(t) = -\partial E(w_{jk}^{(m)}) / \partial w_{hl}^{(n)}$, and the step size is simply the small constant learning rate parameter *size(t)*=η.

A better procedure might be to carry on along in the chosen direction until the error starts rising again. This involves performing a ***line search*** to determine the step size.

The simplest procedure for performing a line search would be to take a series of small steps along the chosen direction until the error increases, and then go back one step. However, this is not likely to be much more efficient than standard gradient descent. Fortunately, there are many better procedures for performing line searches.

Parabolic Interpolation

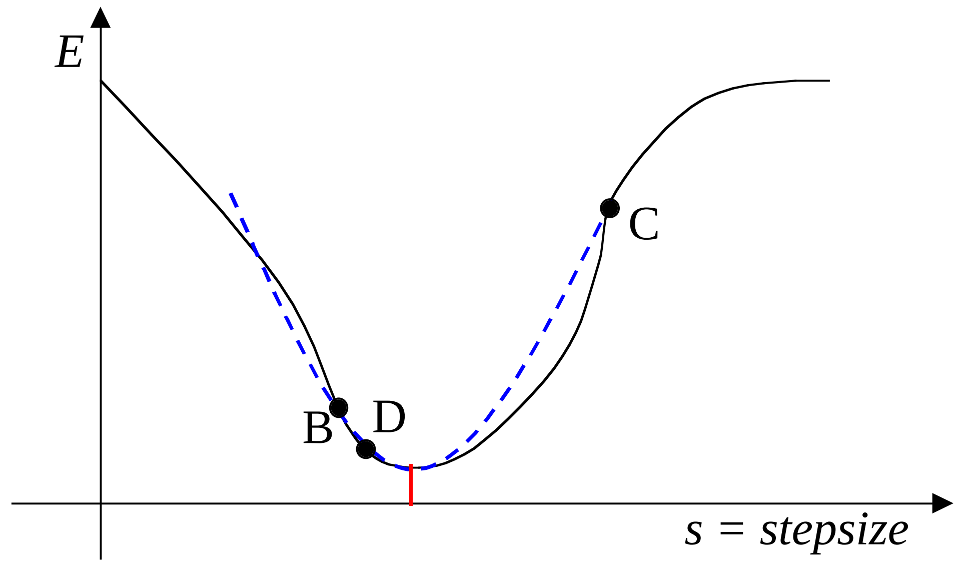
We can plot the variation of the error function *E(s)* as we increase the step *s* taken along our chosen direction. Suppose we can find three points A, B, C such that *E*(A) > *E*(B) and *E*(B) < *E*(C). If A is where we are, and C is far away, this should not be difficult.



The three points are sufficient to define the parabola that passes through them, and we can then easily compute the minimum point D of that parabola. The step size corresponding to that point D is a good guess for the appropriate step size for the actual error function.

Repeated Parabolic Interpolation

The process of parabolic interpolation can easily be repeated. We take our guess of the appropriate step size given by point D, together with the two original points of lowest error (i.e. B and C) to make up a new set of three points, and repeat the procedure:



Each iteration of this process brings us closer to the minimum. However, the gradients change as we take each step, so it is not computationally efficient to get each step size too accurately. Often it is better to get it roughly right and move on to the next direction.

Problems using Gradient Descent with Line Search

We have seen how we can determine appropriate step sizes by doing line searches. It is not obvious, though, that using the gradient descent direction is still the best thing to do. Remember that we chose our step size *s*(*t*–1) to minimise the new error

$$E\left(w_{ij}(t)\right) = E\left(w_{ij}(t-1) - s(t-1)\frac{\partial E(w_{ij}(t-1))}{\partial w_{ij}(t-1)} \right)$$

But we know the derivative of this with respect to *s*(*t*–1) must be zero at the minimum, and we can use the chain rule for derivatives to compute that derivative, so we have

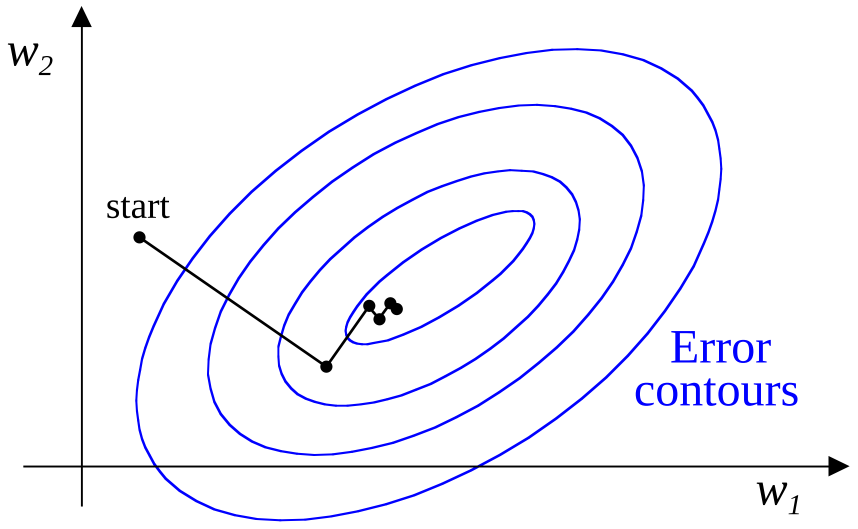
$$\frac{\partial E(w_{ij}(t))}{\partial s(t-1)} = \sum_{i,j} \frac{\partial E(w_{ij}(t))}{\partial w_{ij}(t)} \cdot \frac{\partial E(w_{ij}(t-1))}{\partial w_{ij}(t-1)} = 0$$

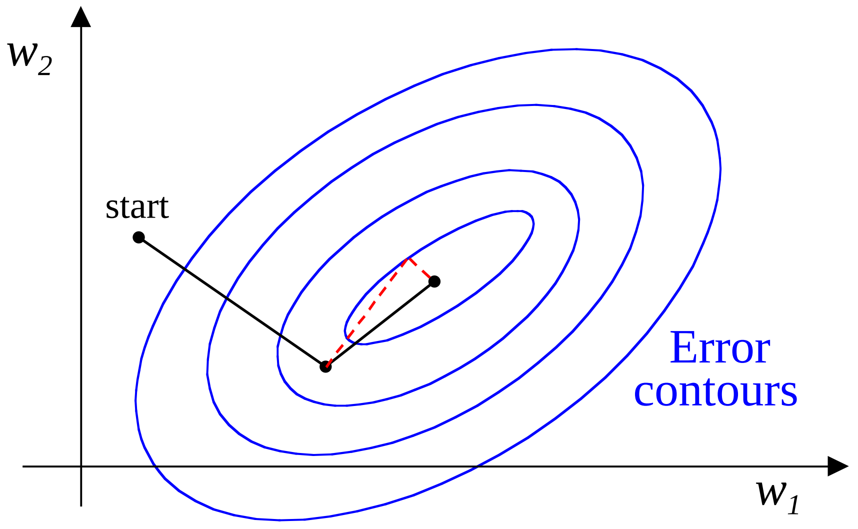
The vector product of the old direction $-\partial E(w_{ij}(t-1))/\partial w_{ij}(t-1)$ and the new direction $-\partial E(w_{ij}(t))/\partial w_{ij}(t)$ is zero, which means the directions are orthogonal (perpendicular). This will result in a less than optimal zig-zag path through weight space.

Finding a Better Search Direction

An obvious approach to avoid the zig-zagging that occurs when we use line searches and gradient directions is to make the new step direction $dir_{ij}(t)$ a compromise between the new gradient direction $-\partial E(w_{ij}(t))/\partial w_{ij}(t)$ and the previous step direction $dir_{ij}(t-1)$:

$$dir_{ij}(t) = -\frac{\partial E(w_{ij}(t))}{\partial w_{ij}(t)} + \beta . dir_{ij}(t-1)$$





Conjugate Gradient Learning

The basis of ***Conjugate Gradient Learning*** is to find a value for β in the last equation so that each new search direction spoils as little as possible the minimisation achieved by the previous one. We thus want to find the new direction $dir_{ij}(t)$ such that the gradient $-\partial E(w_{ij}(t))/\partial w_{ij}(t)$ at the new point $w_{ij}(t)+s.dir_{ij}(t)$ in the old direction is zero, i.e.

$$\sum_{i,j} dir_{ij}(t-1).\frac{\partial E(w_{ij}(t)+s.dir_{ij}(t))}{\partial w_{ij}(t)} = 0$$

An appropriate value of β that satisfies this is given by the ***Polak-Ribiere rule***

$$\beta = \frac{\sum_{i,j}\left(\frac{\partial E(w_{ij}(t))}{\partial w_{ij}(t)} - \frac{\partial E(w_{ij}(t-1))}{\partial w_{ij}(t-1)}\right).\frac{\partial E(w_{ij}(t))}{\partial w_{ij}(t)}}{\sum_{i,j}\frac{\partial E(w_{ij}(t-1))}{\partial w_{ij}(t-1)}.\frac{\partial E(w_{ij}(t-1))}{\partial w_{ij}(t-1)}}$$

For most practical applications this is the fastest way to train our neural networks.

Overview and Reading

1. We began by looking at how we can visualise the process of stepping through weight space to find the error minimum.
2. We saw how adding a momentum term to the gradient descent weight update equations can smooth out and speed up on-line training.
3. We then looked at using line searches as an alternative to constant gradient descent step sizes.
4. We ended with a discussion of Conjugate Gradient Learning.

Reading

1. Bishop: Sections 4.8, 7.5, 7.6, 7.7
2. Hertz, Krogh & Palmer: Sections 6.1, 6.2
3. Haykin: Sections 4.3, 4.4, 4.16, 4.17, 4.18
4. Gurney: Sections 5.4, 6.5