1

LINEAR REGRESSION FOR FUNCTION APPROXIMATION

Vasant Honavar

Artificial Intelligence Research Laboratory
Pennsylvania State University
University Park, PA 16823

1 INTRODUCTION

Many practical problems call for function approximation. This requires a real-valued function $\Re^n \rightarrow \Re$. In this chapter, we introduce a simple *linear* neuron and a learning algorithm for linear function approximation. We will later extend this approach to non-linear function approximation using multi-layer neural networks. This class of algorithms relies on gradient-based error minimization techniques. The derivation of such learning algorithms requires the use of elementary concepts from multi-variate calculus. Therefore, this chapter also includes a brief review of the relevant mathematics.

2 FUNCTION APPROXIMATION

Given a set of training examples $\{[\mathbf{X}, f(\mathbf{X})]\}$ of an unknown function $f : \Re^n \rightarrow \Re$. We want a network that learns $\phi(\mathbf{X})$ which is a *good* approximation of $f(\mathbf{X})$. We will make this more precise later.

In the simplest case, this is simply the familiar curve fitting problem where given the values of a function $f(x)$ of a single scalar variable x for some finite set of values of its argument x, one tries to *fit* a curve or a function $\phi(x)$ that approximates $f(x)$. Here $f : \Re \rightarrow \Re$. Typically, one does this by assuming

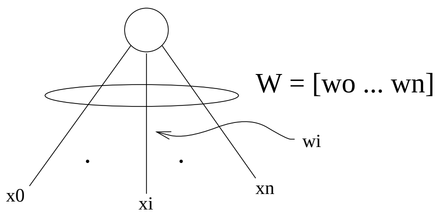

Figure 1 Linear function with weights $\mathbf{W} = [w_0 \ldots w_n]$.

that $\phi(x)$ has a certain form (e.g., linear, quadratic, etc.) and selecting the parameters that define $\phi(x)$ (e.g., the slope and intercept if $\phi(x)$ is assumed to vary linearly with x).

We are interested in a somewhat more general version of this problem wherein f is a function of several arguments represented by $\mathbf{X} = [x_0 \cdots x_n]$. To start with, let us assume that $f(\mathbf{X})$ is a *linear* function. Then we can approximate $f(\mathbf{X})$ by a linear neuron whose output for a given input pattern $\mathbf{X}_p$ is given by:

$$o_p = \mathbf{W} \cdot \mathbf{X}_p$$

This function is determined by its weight vector $\mathbf{W}$. We can generalize this later, to allow more complex (non-linear) functions.

Given the desired output for the function, y_p, we define the error of the function for input pattern $\mathbf{X_p}$ as:

$$e_p = (y_p - o_p)$$

We wish to minimize the magnitude of this error irrespective of the sign, we will use the squared error:

$$e_p^2 = (y_p - o_p)^2$$

Because we are really interested in the error over all $\mathbf{X_k}$, the *Mean Squared Error* is useful:

$$\mathcal{E}' = \frac{1}{P} \sum_{p=1}^{P} (y_p - o_p)^2$$

Where P is the number of patterns $\mathbf{X_k}$. Since P is a constant, we can ignore the $\frac{1}{P}$ coefficient. We will use

$$\mathcal{E} = \frac{1}{2} \sum_{p=1}^{P} (y_p - o_p)^2$$

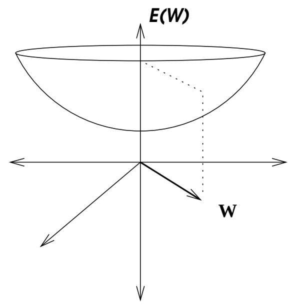

Figure 2 $\mathcal{E}(\mathbf{W})$ defines an error surface. We wish to find a $\mathbf{W}^\star$ that minimizes $\mathcal{E}(\mathbf{W})$.

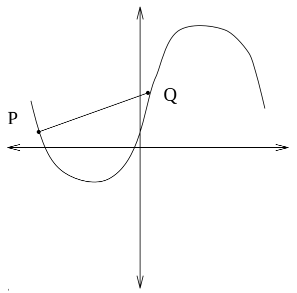

Figure 3 A function that is convex between points P and Q.

as our definition of the error of the approximation.

Once we fix a training set, it is easy to see that $\mathcal{E}$ is a function of the weight vector $\mathbf{W}$. In particular, in the case of a linear neuron, $\mathcal{E}(\mathbf{W})$ is a quadratic function of $\mathbf{W}$. The learning problem reduces to searching the $(n+1)$-dimensional weight space for a weight vector $\mathbf{W}^*$ that minimizes $\mathcal{E}(\mathbf{W})$ (figure 2).

3 A LITTLE MATH

Consider $f(x)$, a function of a scalar variable x over the domain $\mathcal{D}_x$. ($\mathcal{D}_x$ is the set of possible values that x can assume).

- is *non-decreasing* if, $x_1 < x_2 \Rightarrow f(x_1) < f(x_2) \forall x_1, x_2 \in \mathcal{D}_x$.
- is *convex* over some subdomain $\mathcal{D}$ if $\forall x_1, x_2 \in \mathcal{D}$, the chord joining the points $P = f(x_1)$ and $Q = f(x_2)$ on the graph of the function $f(x)$ lies above the graph of $f(x)$. It is *concave* if the chord lies below the graph.

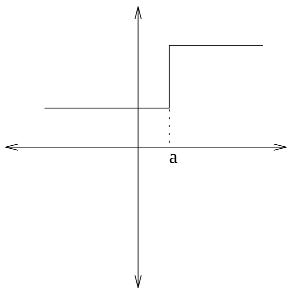

Figure 4 The function $f_1(x)$ is not continuous at $x = a$.

- has a *local minimum* at $x = a$ if $\exists$ a neighborhood $\mathcal{U} \subset \mathcal{D}_x$ around a such that $f(x) > f(a)\ \forall x \in \mathcal{U}$. A *local maximum* requires that $f(x) < f(a)$.

We say that $\lim_{x\to a} f(x) = A$ if, for any $\varepsilon > 0$, $\exists \delta > 0$ such that $\mid f(x) - A \mid < \varepsilon$, $\forall x$ such that $\mid x - a \mid < \delta$. Note that the value of the limit may be different if we approach a along the x-axis in the positive as opposed to the negative direction.

A function is said to be *continuous* (figure 8) at $x = a$ if:

$$\lim_{x\to a^+} f(x) = \lim_{x\to a^-} f(x)$$

This means that the limit of $f(x)$ is the same when we approach a from the positive as well as the negative direction along the x axis. In particular, we will be interested in continuity of error functions (because continuous error functions lend themselves to minimization using techniques borrowed from multi-variate calculus).

The *derivative* of a function $f(x)$ is defined as follows:

$$\frac{df}{dx} = \lim_{\Delta x\to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

Thus, the derivative of $f(x)$ evaluated at $x = a$ (written as $f(x)\mid_{x=a}$) gives the *rate of change* of $f(x)$ with respect to x at $x = a$.

The second derivative of a function $f(x)$ is defined as:

$$\frac{d^2 f}{dx^2} = \frac{d}{dx}\frac{df}{dx}$$

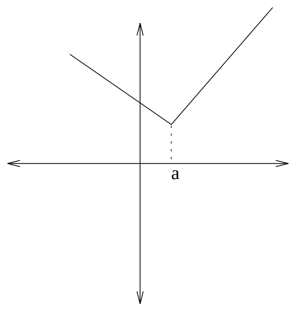

Figure 5 The function $f_2(x)$ is continuous at $x = a$.

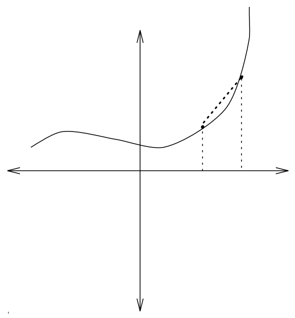

Figure 6 The derivative specifies the rate of change of $f(x)$.

.

Clearly, it represents the rate of change of the derivative of $f(x)$.

The following properties of derivatives are easy to derive from first principles using the definition of the derivative. Let u and v be functions of x. Then:

$$\frac{d(u+v)}{dx} = \frac{du}{dx} + \frac{dv}{dx}$$

$$\frac{d\frac{u}{v}}{dx} = \frac{v\frac{du}{dx} - u\frac{dv}{dx}}{v^2}$$

$$\frac{d(uv)}{dx} = u\frac{dv}{dx} + v\frac{du}{dx}$$

If $\frac{df}{dx} > 0$ then $f(x)$ is *strictly increasing.*
If $\frac{df}{dx} < 0$ then $f(x)$ is *strictly decreasing.*
If $\frac{df}{dx} = 0$ then $f(x)$ is *strictly constant.*

The derivative of $f(x)$ at $x = x_0$ is zero if x_0 is a local minimum or a local maximum of $f(x)$ (or if $f(x) =$ constant):

$$\frac{df}{dx} \mid_{x=x_0} = 0$$

We can determine whether x_0 is a minimum or maximum by evaluating $f(x)$ in the neighborhood of x_0.

Suppose $f(x)$ is differentiable (i.e., its derivatives $\frac{df}{dx}, \frac{d^2f}{dx^2}, \cdots \frac{d^nf}{dx^n}$ exist) and $f(x)$ is continuous in the neighborhood of x_0. Then the *Taylor Series* expansion of a function $f(x)$ around $x = x_0$ is given by:

$$f(x) = f(x) \mid_{x=x_0} + \frac{df}{dx} \mid_{x=x_0} (x - x_0) + \frac{1}{2}\frac{d^2f}{dx^2} \mid_{x=x_0} (x - x_0)^2 + \ldots + \frac{1}{n!}\frac{d^nf}{dx^n} (x - x_0)$$

Taylor series expansion is useful in approximating the value of a function $f(x)$ over small neighborhoods in its domain $\mathcal{D}_x$, when we are given (of can somehow measure or estimate) the value of the function and its derivatives at a point x_0 in the neighborhood. If we consider only the first two terms of the expansion, we get a first order approximation (or linear approximation) of $f(x)$ in the neighborhood of x_0. Considering additional terms involving higher order derivatives results in successively higher order (e.g., quadratic, cubic, etc.) approximations.

Suppose we know that $f(1) = 1$; and $\frac{df}{dx} \mid_{x=1} = 0.1$; Then, the first order Taylor series approximation of $f(x)$ at $x = 1.01$ is given by $f(1 + 0.01 = f(1) + 0.1 \times 0.01 = 1.001$.

3.1 Functions of several variables

The concepts introduced above extend quite naturally to the case of multivariate functions (i.e., functions of several variables). Consider a multivariate function $f(\mathbf{X}) = f(x_0, \ldots, x_n))$. Now we have *partial derivatives* that represent the rate of change of $f(\mathbf{X})$ with respect to each variable x_i. A partial derivative with respect to x_i is computed by taking the derivative of $f(x_0, \ldots x_n)$ by treating $\forall j \neq i$, x_j as though it were a constant.

Consider the following example:

$$y = \phi(x_1, x_2) = 4x_1^2 + x_2^2 + 2x_1x_2$$
$$\frac{\partial \phi}{\partial x_1} = 8x_1 + 2x_2$$
$$\frac{\partial \phi}{\partial x_2} = 2x_2 + 2x_1$$
$$\frac{\partial^2 \phi}{\partial x_1^2} = 8$$
$$\frac{\partial^2 \phi}{\partial x_2^2} = 2$$

The **chain rule** is often useful in simplifying the calculation of partial derivatives. Consider the following:

$$z = \phi(u, v)$$
$$u = f_1(x, y)$$
$$v = f_2(x, y)$$

Then, chain rule allows us to compute the partial derivative of z with respect to x as follows:

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial u}\frac{\partial u}{\partial x} + \frac{\partial z}{\partial v}\frac{\partial v}{\partial x}$$

As an example, consider

$$z = u + v^2$$
$$u = x^2$$
$$v = xy$$

Then

$$\frac{\partial z}{\partial x} = 1(2x) + (2v)(y) = 2x + 2(xy)y = 2x + 2xy^2$$

It is left as an exercise to verify that direct computation of the partial derivative $\frac{\partial z}{\partial z}$ by first substituting for u and v in the expression for z yields the same result.

More generally, consider:

$$z = \phi(u_1, u_2 \cdots u_m)$$

where

$$u_i = f_i(x_1, x_2 \cdots x_n$$

Then, we have from the chain rule, $\forall i = 1 \cdots n$:

$$\frac{\partial z}{\partial x_i} = \sum_{j=1}^{m} \frac{\partial z}{\partial u_j} \frac{\partial u_j}{\partial x_i}$$

Taylor Series can be used to approximate a function of several variables in a neighborhood where the function is continuous and differentiable. For example, the Taylor Series expansion for the function $\phi(x_1, x_2)$ around $\mathbf{X_0} = (x_{01}, x_{02})$ is given by:

$$\phi(\mathbf{X_0}) + \tfrac{\partial \phi}{\partial x_1} \mid_{\mathbf{X}=\mathbf{X}_0} (x_1 - x_{01}) + \tfrac{\partial \phi}{\partial x_2} \mid_{\mathbf{X}=\mathbf{X}_0} (x_2 - x_{02}) + \\ \tfrac{1}{2} \tfrac{\partial^2 \phi}{\partial x_1^2} \mid_{\mathbf{X}=\mathbf{X}_0} (x_1 - x_{01})^2 + \tfrac{1}{2} \tfrac{\partial^2 \phi}{\partial x_2^2} \mid_{\mathbf{X}=\mathbf{X}_0} (x_2 - x_{02})^2 + \ldots$$

This can be extended to functions of several variables. Usually, we will only be interested in the first order terms of the expansion.

4 MINIMIZATION OF FUNCTIONS

Recall that approximation of functions from examples essentially involves finding a set of weights that minimize a suitably defined error function. This entails finding the values of arguments of the error function which yield a minimum value of the error.

To further develop our intuition about this problem, consider a function $f(z)$ of one variable, namely, z. Suppose we want to minimize $f(z)$. It seems reasonable to start with some initial value for z (say $z = z_0$) and then take small steps along the z-axis so that at each step, we obtain a smaller value for $f(z)$ than that which we had at the previous step. If $f(z)$ is a convex function, then it is easy to see that such a procedure is guaranteed to lead us to a value of z that minimizes $f(z)$. Consider the move from z_0 to z_1 where $\Delta z = z_1 - z_0$. We

would like to ensure that $f(z_1) \leq f(z_0)$.
Using Taylor series expansion of $f(z)$ in the neighborhood of $z = z_0$, we have:

$$f(z_1) = f(z_0 + \Delta z) = f(z_0) + \frac{\partial f}{\partial z} |_{z=z_0} (\Delta z) + \cdots$$

Thus, the change in value of $f(z)$ as one moves from z_0 to z_1 (that is, by an amount Δz) from z_0 along the z axis, is given by:

$$\Delta f = f(z_1) - f(z_0) = \frac{\partial f}{\partial z} |_{z=z_0} \Delta z$$

We want to make sure that Δf is negative so that the value of $f(z_1)$ is smaller than $f(z_0)$. Suppose we choose:

$$\Delta z = -\eta \frac{\partial f}{\partial z} |_{z=z_0}$$

where $\eta > 0$ is a suitable learning rate. We see that:

$$\Delta f = -\eta (\frac{\partial f}{\partial z})^2 |_{z=z_0} \leq 0$$

Note that we assumed that

1. f is differentiable at $z = z_0$
2. $f(z)$ is continuous in the neighborhood of $z = z_0$
3. Δz will be small (ideally infinitesimally small) so that the first order Taylor series approximation is reasonably accurate in the neighborhod of $z = z_0$. (This requires η to be infinitesimally small. In practice, we will use as large a value for η as we can get away with. More on this later).

In essence, we perform a *gradient descent* on $f(z)$. This technique is naturally extended to minimization of multivariate functions as we shall see shortly.

5 MEAN SQUARED ERROR MINIMIZING LEARNING RULE FOR LINEAR FUNCTION APPROXIMATION

Going back to the problem of approximating an unknown function from examples, we have o_p = actual output of a linear function for pattern $x_p = [x_{0p} \cdots x_{np}]$.

For a linear function we have: $o_p = \Sigma_i = 0^n w_i \cdot x_{ip}$ and y_p = the desired output of the function for input pattern x_p

Our goal is to find $\mathbf{W}^\star$ that minimizes

$$\mathcal{E}(\mathbf{W}) = \frac{1}{2}\Sigma_{p=0}^{p}(y_p - o_p)^2 = \frac{1}{2}\Sigma_p(e_p)^2$$

We start with an initial weight vector (typically chosen at random) and we change $\mathbf{W}$ in the direction of the negative gradient of $\mathcal{E}(\mathbf{W})$ at each step, with respect to each of the variables w_i where $0 \leq i \leq n$.

Thus,

$$w_i \leftarrow w_i - \eta\frac{\partial \mathcal{E}}{\partial w_i}$$

The task reduces to computing

$$\frac{\partial \mathcal{E}}{\partial w_i}$$

in terms of known or observed quantities.

Consider:

$$\frac{\partial \mathcal{E}}{\partial w_i} = \frac{1}{2} \cdot \frac{\partial}{\partial w_i} \cdot (\sum_p e_p^2)$$

$$= \frac{1}{2}\sum_p \frac{\partial e_p^2}{\partial w_i}$$

$$= \frac{1}{2}\sum_p 2e_p \cdot \frac{\partial e_{jp}}{\partial_i}$$

$$= \frac{\partial e_p}{\partial w_i} = \frac{\partial e_p}{\partial o_p} \cdot \frac{\partial o_p}{\partial w_i}$$

$$= (-1) \cdot \frac{\partial}{\partial w_i} \cdot [\sum_k w_k \cdot x_{kp}]$$

$$= (-1) \cdot \frac{\partial}{\partial w_i} \cdot [w_i \cdot x_{ip} + \sum_{k \neq i} (w_k \cdot x_{kp})]$$

$$= -x_{ip}$$

Thus:

$$w_i \leftarrow w_i + \eta \cdot \sum_p (y_p - o_p) \cdot x_{ip}$$

6 BATCH VERSUS PER-PATTERN WEIGHT UPDATE

The version of the learning rule in which weights are updated each time after a complete pass through the training set is called the batch update rule.

In the case of the linear function, the batch update rule is given by:
$w_i \leftarrow w_i + \eta \cdot \sum_p (y_p - o_p) x_{ip}$

Alternatively, this can be approximated by a per-pattern update (where weights are changed after each pattern presentation.

Eg. $w_i \leftarrow w_i + \eta(y_p - o_p)x_{ip}$, in the case of linear function.

In either case, the learning algorithm involves an iterative procedure which starts with an initial weight vector (typically chosen at random) and makes several passes through the training set, modifying the weights using either the batch or per-pattern weight update until the error falls below a desired value ϵ.

7 VECTOR-VALUED FUNCTION APPROXIMATION

The algorithms derived above can be easily extended to deal with vector valued functions (i.e., where there are multiple outputs to be computed for a given input).

Let w_{ji} = weight from input x_{ip} to jth output and let $\mathcal{E}_j$ denote the corresponding error. Since the outputs are independent, we can use the update equation that we derived for the single output case for each of the outputs.

$$w_{ji} \leftarrow w_{ji} - \eta \cdot \frac{\partial \mathcal{E}_j}{\partial w_{ji}}$$

Note: We cannot do this in general unless o_j's are mutually independent. In general, we will have to consider the partial derivative of the error with respect to each of the parameters of interest. We will see examples of this later when we develop the generalized delta rule for multi-layer networks.

8 CHOICE OF LEARNING RATE

Taylor series approximation requires that the learning rate η be small (ideally infinitesimally small - for the first order approximation to be accurate) But in practice if η is too small, learning will take forever. If η is too large, we may overshoot the desired minimum of $\mathcal{E}$ and oscillate. This raises the following question: Is it possible to establish a useful practical range for η ?

A useful bound on η can be derived as follows:

Let $\mathbf{W}$ be the weight vector and $o_p = \mathbf{W} \cdot \mathbf{X}_p$ the output for pattern $\mathbf{X}_p$

$\mathcal{E}_p(\mathbf{W}) = (y_p - o_p)^2$ (Error on pattern $\mathbf{X}_p$ using the current weight vector $\mathbf{W}$).

Suppose we use <u>per-pattern</u> version, a LMS rule to obtain an updated weight vector $\mathbf{W}_{new}$:

$$\mathbf{W}_{new} = \mathbf{W} + \eta(y_p - o_p)\mathbf{X}_p$$

$$\begin{aligned}\mathcal{E}_p(\mathbf{W}_{new}) &= (y_p - \mathbf{W}_{new}\cdot\mathbf{X}_p)^2\\ &= (y_p - (\mathbf{W} + \eta(y_p - \mathbf{W}\cdot\mathbf{X}_p)\mathbf{x_p})\mathbf{X}_p)^2\\ &= ((y_p - \mathbf{W}\cdot\mathbf{X}_p) - \eta(y_p - \mathbf{W}\cdot\mathbf{X}_p)\mathbf{X}_p\cdot\mathbf{X}_p)^2\\ &= (y_p - \mathbf{W}\cdot\mathbf{X}_p)^2 + (\eta)^2(y_p - \mathbf{W}\cdot\mathbf{X}_p)^2(\|\mathbf{X}_p\|)^4 - 2\cdot\eta(y_p - \mathbf{W}\cdot\mathbf{X}_p)^2(\|\mathbf{X}_p\|)^2\\ &= \mathcal{E}_p(\mathbf{W}) - 2\cdot\eta\cdot E_p(\mathbf{W})(\|\mathbf{X}_p\|)^2 + \eta^2\cdot E_p(\mathbf{W})(\|\mathbf{X}_p\|)^4\end{aligned}$$

In order for the the error after update to be lower than the error before the weight update, we need:

$$\mathcal{E}_p(\mathbf{W}_{new}) \leq E_p(\mathbf{W})$$

$$\mathcal{E}_p(\mathbf{W}_{new}) - \mathcal{E}_p(\mathbf{W}) \leq 0$$

or

$$\mathcal{E}_p(\mathbf{W}) - \mathcal{E}_p(\mathbf{W}_{new}) \geq 0$$

$$\mathcal{E}_p(\mathbf{W}) - \mathcal{E}_p(\mathbf{W}_{new}) = 2\cdot\eta\cdot\mathcal{E}_p(\mathbf{W})(\|\mathbf{X}_p\|)^2 - \eta^2\cdot\mathcal{E}_p(\mathbf{W})(\|\mathbf{X}_p\|)^4$$

We know that $\mathcal{E}_p(\mathbf{W}) \geq 0$ because it is $(y_p - o_p)^2$. Thus, we have:

$$\mathcal{E}_p(\mathbf{W}) - \mathcal{E}_p(\mathbf{W}_{new}) \geq 0$$

if:

$$2\cdot\eta\|\mathbf{X}_p\|^2 - \eta^2\cdot\|\mathbf{X}_p\|^4 \geq 0$$

$$= 2\cdot\eta - \eta\cdot\|\mathbf{X}_p\|^2 \geq 0$$

Since $\eta > 0$, this means:

$$2 - \eta\cdot\|\mathbf{X}_p\|^2 \geq 0$$

$$0 < \eta \leq \frac{2}{\|\mathbf{X}_p\|^2}$$

Widrow suggests the following rule of thumb:

$$\frac{0.1}{\|\mathbf{X}_p\|^2} < \eta < \frac{1}{\|\mathbf{X}_p\|^2}$$

Note that the bound derived above is not very rigorous because it is based on per-pattern update and does not make use of the true gradient of the error function. A more rigorous set of bounds on η can be derived assuming batch version of LMS rule:

$$0 < \eta < \frac{1}{\lambda_{max}}$$

where λ_{max} is the largest eigenvalue of the Hessian (matrix of 2nd order partial derivatives) of the error function $\mathcal{E}$ (Eigenvalues of matrix $\mathbf{A}$ are given by the solution of $|\mathbf{A} - \mathbf{\Lambda I}| = 0$ where $\mathbf{\Lambda}$ is a vector of eigenvalues and $\mathbf{I}$ is a matrix with 1s along the diagonal and zeros everywhere else.

Even though this gives a mathematically rigorous approach to choosing an appropriate learning rate, it incurs the additional overhead of computing the Hessian of the error function.

In practice, there is a simpler method for adaptively changing η that works reasonably well.

Momentum Modification

This modification to the weight update rule is based on the following physical intuition: If one is running down a hill one's velocity increases as one gains momentum , so one will move faster and faster until the gradient changes direction and causes one to slow down.

Use of Momentum term to speed up LMS Learning:

Let

$\triangle w_i(t)$ denote the change in the ith component of the weight vector as dictated by the LMS rule. Suppose we choose

$$w_i(t+1) = w_i(t) + \triangle w_i(t)$$

where

$$\triangle w_i(t) = -\eta \frac{\partial \mathcal{E}}{\partial w_i} + \alpha \triangle w_i(t-1)$$

$$\triangle w_i(0) = -\eta \cdot \frac{\partial \mathcal{E}}{\partial w_i}|_{w_i = w_i(t)}$$

$$\triangle w_i(1) = \alpha \triangle w_i(0) - \eta \cdot \frac{\partial \mathcal{E}}{\partial w_i}|_{w_i = w_i(1)}$$

$$\triangle w_i(2) = \alpha \triangle w_i(1) - \eta \cdot \frac{\partial \mathcal{E}}{\partial w_i}|_{w_i = w_i(2)}$$

Thus we have:

$$\triangle w_i(t) = -\eta \sum_{\tau=o}^{t} \alpha^{t-\tau} \cdot \frac{\partial \mathcal{E}}{\partial w_i}|_{w_i = w_i(\tau)}$$

It can be shown that this series converges for values of α that satisfy $0 \leq \alpha \leq 1$. In practice, α is typically set to 0.9 or so. Note that the momentum modification allows us to *effectively* vary the learning rate in the LMS learning rule by taking large steps in the weight space when the gradient of the error function is more or less constant while at the same time, forcing small steps when the gradient is changing.

9 SUMMARY

In this chapter, we have developed simple gradient-based learning algorithms for linear function approximation. Linearity of the approximation limits the class of functions that can be approximated. Later, we will extend these concepts to deal with multi-layer non-linear function approximators and pattern classifiers. Nevertheless, even the simple models that we have looked at find a number of practical applications.