Table of Contents

CHAPTER I - DATA FITTING WITH LINEAR MODELS 4
1. INTRODUCTION 5
2. LINEAR MODELS 11
3. LEAST SQUARES 15
4. ADAPTIVE LINEAR SYSTEMS 20
5. ESTIMATION OF THE GRADIENT - THE LMS ALGORITHM 28
6. A METHODOLOGY FOR STABLE ADAPTATION 36
7. REGRESSION FOR MULTIPLE VARIABLES 48
8. NEWTON'S METHOD 64
9. ANALYTIC VERSUS ITERATIVE SOLUTIONS 67
10. THE LINEAR REGRESSION MODEL 68
11. CONCLUSIONS 74
END OF CHAPTER 1 78
LEAST SQUARES DERIVATION 78
VARIANCE 79
DERIVATION OF CORRELATION COEFFICIENT 80
COMPUTATION OF CORRELATION COEFFICIENT 81
BATCH VERSUS ONLINE LEARNING 82
MORE DERIVATION OF PERFORMANCE SURFACE 82
MORE ON DERIVATION OF LARGEST STEPSIZE 83
DERIVATION OF THE TIME CONSTANT OF ADAPTATION 84
MORE ON SCHEDULING STEPSIZES 85
DERIVATION OF NORMAL EQUATIONS 85
PERFORMANCE SURFACE PROPERTIES 87
MULTIPLE VARIABLE CORRELATION COEFFICIENT 89
CONVERGENCE FOR MULTIPLE WEIGHTS CASE 90
ESTIMATION OF EIGENVALUE SPREAD 91
CASTI REFERENCE 91
PROCESSING ELEMENT 91
EPOCH 91
LINEAR REGRESSION 92
MEAN SQUARE ERROR 92
LEAST SQUARES 92
CORRELATION COEFFICIENT 92
ADAPTIVE SYSTEMS 92
PERFORMANCE SURFACE 92
SUPERVISED LEARNING 93
UNSUPERVISED LEARNING 93
GRADIENT 93
STEEPEST DESCENT 93
LEAST MEAN SQUARE 93
STEP SIZE 93
ON-LINE TRAINING 94
EPOCH 94
BATCH TRAINING 94
TRAINING SET 94
TEST SET 94
LEARNING CURVE 94
WEIGHT TRACK 94
GEOMETRIC RATIO 95
TIME CONSTANT OF ADAPTATION 95
RATTLING 95
MISADJUSTMENT 95
LEARNING RATE SCHEDULING 95

EIGENVALUE SPREAD 95
NORMALIZED LMS 95
BIG O NOTATION 96
ADALINE 96
EQ. 4 96
EQ. 14 96
EQ.6 96
EQ.9 97
EQ.3 97
EQ.10 97
EQ.12 97
EQ.11 97
EQ.13 97
EQ.54 97
EQ.16 98
EQ.17 98
EQ.19 98
EQ.25 98
EQ.28 98
EQ.31 98
EQ.68 98
EQ.69 99
EQ.65 99
EQ.27 99
EQ.36 99
EQ.21 99
EQ.33 99
EQ.32 100
EQ.41 100
EQ.5 100
EQ.45 100
EQ.53 100
EQ.52 101
EQ.55 101
EQ.30 101
EQ.26 101
EQ.63 101
EQ.40 101
WIDROW 102
EQ.7 102
EQ.15 102
WIDROW AND STEARNS 102
LINEAR MODELS 102
EQ.49 102
OUTLIER 102
RLS 103
GAUSS 103
COVARIANCE 103
STANDARD DEVIATION 103
ESTIMATION THEORY 103
AUTOCORRELATION 104
CROSSCORRELATION 104
DERIVATION OF SOLUTION 104
CONTOUR 104
EIGENVALUES 105
Z SCORES 105

NEWTON'S DERIVATION 105
ILL-CONDITIONED 105
GRADIENT DEFINITION AND CONSTRUCTION 105
DEVELOPMENT OF THE PHONE SYSTEM 107
MARS' PATHFINDER MISSION 108

Chapter I - Data Fitting with Linear Models

version 2.0

This Chapter is Part of:

Neural and Adaptive Systems: Fundamentals Through Simulation© by

Jose C. Principe
Neil R. Euliano
W. Curt Lefebvre

The goal of this chapter is to introduce the concepts of:

- Data fitting and the derivation of the best linear (regression) model.
- Iterative solution of the regression model.
- Steepest descent methods.
- The LMS (least mean square) estimator for the gradient.
- The trade-off between speed of adaptation and solution accuracy.
- Examples using NeuroSolutions.

- 1. Experimental Model Building
- 2. Linear Models
- 3. Least Squares
- 4. Least squares as a search for the parameters of a linear system
- 5. Estimation of the gradient - the LMS algorithm
- 6. Getting a grip on adaptation
- 7. Regression for multiple variables
- 8. Newton's method
- 9. Analytic versus Iterative solutions
- 10. The linear Regression Model
- 11. Conclusions

Go to next section
Go to the Appendix

1. Introduction

Engineering is a discipline that builds physical systems from human dreams, re-inventing the physical world around us. In this respect it transcends physics that has a passive role of explaining the world, and also mathematics that stops at the edge of the physical reality. Engineering design is just like a gigantic Lego, where each piece is a subsystem grounded in its physical or mathematical principles. The role of the engineer is to first develop the blue print of the “dream” through specifications, and then look for the pieces that fit the blue print. Obviously the pieces can not be put together at random since each has its own principles attached. So it is mandatory that the engineer first learns the principles attached to each piece and specifies the interface. Normally this study is done using the scientific method. When the system is physical we use the principles of physics, and when it is software we use the principles of mathematics. development of the phone system This method has been highly successful, but let us evaluate it in broad terms.

First, engineering design requires the availability of a model for each subsystem. Second, when the number of pieces increase the interactions among the subsystems increase exponentially. Fundamental research will continue to provide a steady flux of new physical and mathematical principles (provided the present trend of federal funding for fundamental science is reversed) but the exponential growth of interactions required for larger and more sophisticated systems is harder to control. In fact at this point in time, we simply do not have a clear vision how to handle complexity in the long term. But there are two more factors that present big challenges. They are the autonomous interaction of systems with the environment and the optimality of the design. We will discuss these below.

Humans have traditionally mediated the interaction of engineering systems with the external world. After all humans use technology to enhance their physical constraints so we have been in control of the machines we build. Since the invention of the digital computer there is a trend to create machines that interact directly with the external world without the human in the loop. This brings the complexity of the external world directly into engineering design. We are not yet totally prepared for this, because our mathematical and physical theories about the external world are mere approximations: very good approximations in some cases, but rather poor in others. This disturbs the order of engineering design, and creates performance problems (the worse subsystem tends to limit the performance of the full system) Mars' pathfinder mission .

System optimality is also a rising concern to save resources and augment the performance/price ratio. We could think that designing optimally each sub-system would bring global optimality, but this is not always true. So optimal design of complex systems is a difficult problem that has also to take into consideration the particular type of system function, that is, the complexity of the environment is once again present. We can conclude that the current challenges faced in engineering are the complexity of the systems, the need for optimal performance, and the autonomous interaction with the environment that will require some form of intelligence. These are the challenges for XXI century (and beyond) engineering.

Whenever there is a challenge, we should look elsewhere for answers. Quite often the difficulty of a task is also linked to the particular method we are using to find the solution. Is building machines by specification the only way to proceed?

Let us look at living creatures from an engineering systems perspective. The cell is the ultimate optimal factory building directly from the environment at the fundamental molecular level what it needs to carry out its function. The animals we observe today interact efficiently with the environment (otherwise they would not have survived), they work very close to optimality in terms of resources (otherwise they would have been replaced in their niche by more efficient animals), and they sure are complex. Biology has in fact conquered already some of the challenges we face in building engineering systems, so it is worthwhile to investigate what are the principles at work

Biology has found a set of *inductive* principles that are particularly well tuned to the interaction with a complex and unpredictable environment. These principles are not known explicitly, but are being intensively studied in biology, computational neurosciences, statistics, computer science and engineering. They involve extraction of information from sensor data (feature extraction), efficient learning from data, creation of invariants and representations, and decision making under uncertainty. In a global sense autonomous agents have to build and fit models to data through their daily experience, they have to store these models, choose which shall be applied in each circumstance, and assess the likelihood of success for a given task. An implicit optimization principle is at play, since the goal is to do the best with the available information and resources.

From a scientific perspective, biology uses *adaptation* to build optimal system functionality. The anatomical organization of the animal (the wetware) is specified in the long term by the environment (through evolution), and in the short-term it is used as a constraint to extract in real time the information that the animal needs to secure survivability. At the nervous system level, it is well accepted that the interaction with the environment molds the wetware using a learning from examples metaphor.

1.1. Neural and Adaptive systems

Neural and adaptive are a unique and growing interdisciplinary field that studies adaptive, distributed, and mostly nonlinear systems, three of the ingredient found in biology. We believe that neural and adaptive systems should be considered another tool in the scientist/engineers toolbox. They will complement effectively the present engineering design principles and help build the preprocessors to interface with the real world, and the optimality needed in complex systems. When applied correctly the performance of a neural or adaptive system may considerably outperform other methods.

Neural and adaptive systems are used in many important engineering applications such as, signal enhancement, noise cancellation, classification of input patterns, system identification, prediction, and control. They are used in many commercial products such as: modems, image processing and recognition systems, speech recognition, frontend signal processors, biomedical instrumentation, etc. We expect that the list we will grow exponentially in the near future.

The leading characteristic of neural and adaptive systems is their adaptivity, which brings a totally new system design style (Figure 1). Instead of being built *a priori* from specification, neural and adaptive systems use external data to automatically set their parameters. This means that neural systems are parametric. It also means that they are made "aware" of their output through a performance feedback loop that includes a cost function. The performance feedback is utilized directly to change the parameters through systematic procedures called learning or training rules, such that the system output improves with respect to the desired goal (i.e. that the error decreases through training).

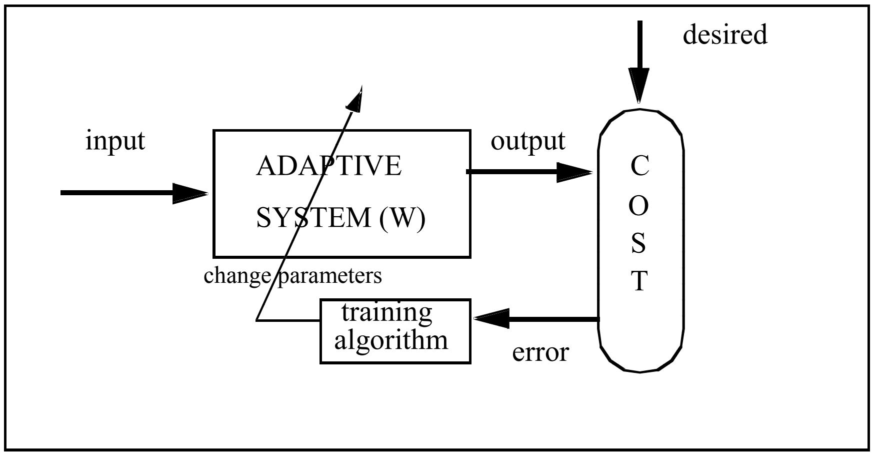


Figure 1. Adaptive system's design methodology

The system designer has to specify just a few but crucial steps in the overall process: he/she has to decide the system topology, to choose a performance criterion, to design the adaptive algorithms. In neural systems the systems parameters are modified in a selected set of data called the training set, and fixed during operation. So the designer has to know how to specify the input and desired response data and when to stop the training phase. In adaptive systems the system parameters are continuously adapted during operation with the current data. We are at a very exciting stage in neural and adaptive system development because:

- We now know some powerful topologies that are able to create universal input-output mappings.
- We also know how to design general adaptive algorithms to extract information from data and adapt the parameters of the mappers.
- We are also starting to understand the pre-requisites for generalization, i.e. to guarantee that the performance in the training set can be extended to the data found during system operation.

Therefore we are in a position to design effective adaptive solutions to moderately difficult real world problems. Due to the practicality derived from these advances we believe the time is right to teach adaptive systems in undergraduate engineering and science curricula.

Throughout this textbook we will be explaining the principles that are necessary to make judicious choices about the design options for neural and adaptive systems. The discussion is slanted towards engineering, both in terminology and in perspective. We are very much interested in the engineering model-based approach, and in explaining the mathematical principles at work. We center the explanation on concepts from adaptive signal processing, which are rooted in statistics, pattern recognition and digital signal processing. Moreover, our study will be restricted to model building from data.

1.2 Experimental Model Building

The problem of data fitting is one of the oldest in experimental science. The real world tends to be very complex, unpredictable, and the exact mechanisms that generate the data are often unknown. Moreover, when we collect physical variables the sensors are not ideal (finite precision, noisy, constraint bandwidth, etc.) so the measurements do not represent exactly the real phenomena. One of the quests in science is to estimate the underlying data model.

The importance of inferring a model from the data is to apply mathematical reasoning to the problem. The major advantage of a mathematical model is the ability to understand, explain, predict and control outcomes in the natural system [Casti]. Figure 2 illustrates the data modeling process. The most important advantage of the existence of a formal equivalent model is the ability to predict the natural system behavior at a future time and to control its outputs by applying appropriate inputs.

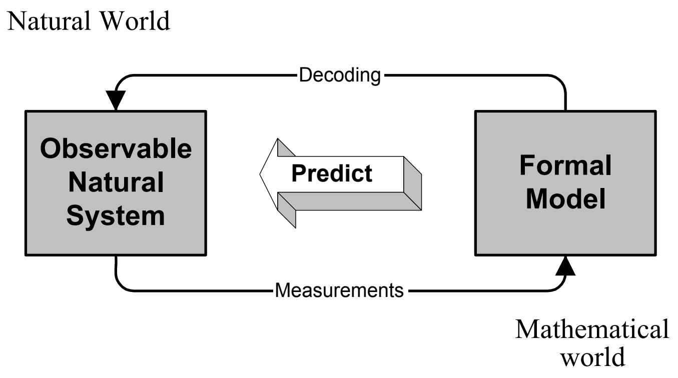


Figure 2. Natural systems and formal models

In this chapter we will address the issues of fitting data with linear models, which is called the *linear regression* problem. Notice that we have not specified what the data is, because it is really immaterial. We are seeking relationships between the values of the external (observable) variables of the natural system in Figure 1. So this methodology can be applied either to meteorological data, biological data, financial data, marketing data, engineering data, etc.

1.2 Data Collection

The data collection phase must be carefully planned to ensure that:

- data will be sufficient,
- data will capture the fundamental principles at work,
- data is as free as possible from observation noise.

X	D
1	**1.72**
2	**1.90**
3	**1.57**
4	**1.83**
5	**2.13**
6	**1.66**
7	**2.05**
8	**2.23**
9	**2.89**
10	**3.04**
11	**2.72**
12	**3.18**

Table 1 - Regression Data

Table I presents a data example with two variables *x*, *d* in tabular form. The measurement *x* is assumed error free, and *d* is contaminated by noise. By observing Table I very little can be said about the data, except that there is a trend, i.e. when *x* increases *d* also increases. Our brain is somehow able to extract much more information from figures than numbers, so data should be first plotted before performing data analysis. Plotting the data allows verification, ensures the researcher that the data was collected correctly and provides a "feel" for the relationships that exist in the data (e.g. natural trends, etc.).

Go to the Next Section

2. Linear models

From the simple observation of Figure 3, it is obvious that the relationship between the two variables *x* and *d* is complex, if one assumes that no noise is present. However there is an approximate linear trend in the data. The deviation from the straight line could be produced by noise, and underlying the apparent complexity could be a very simple (possibly linear) relationship between *x* and *d*, i.e.

$d \approx wx + b$ **Equation 1**

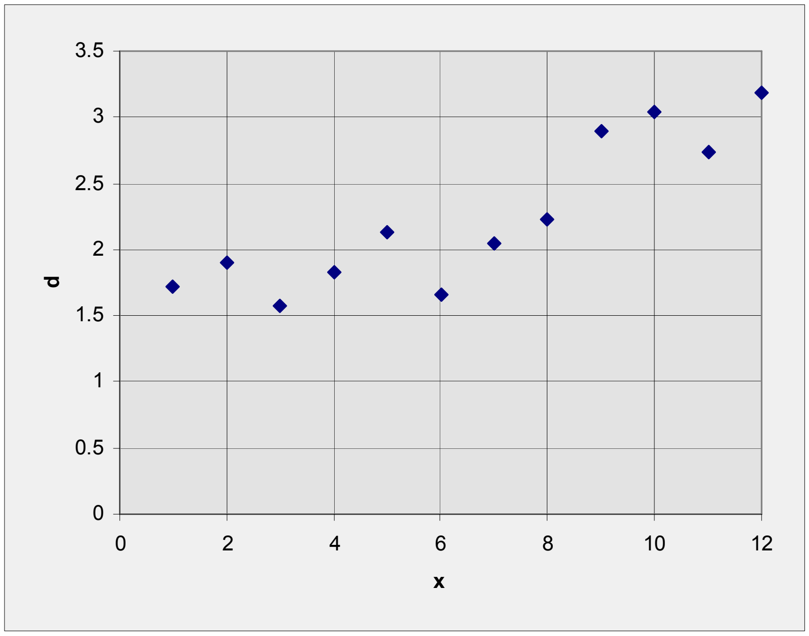


Figure 3. Plot of x versus d

or more specifically,

$$d_i = wx_i + b + \varepsilon_i = y_i + \varepsilon_i$$ **Equation 2**

where ε_i is the instantaneous error that is added to y_i (the linearly fitted value), *w* is the slope and *b* is the *y* intersect (or bias). Assuming a linear relationship between *x* and *d* has the appeal of simplicity. The data fitting problem can be solved by a linear system with only two free parameters, the slope *w* and the bias *b*.

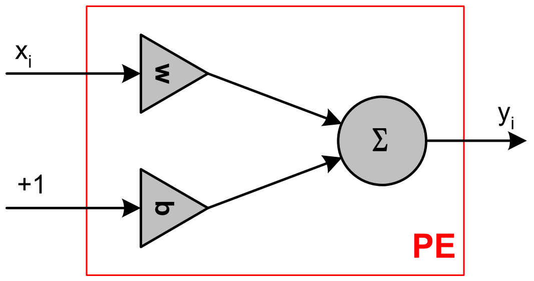


Figure 4 - Linear Regression Processing Element

The system of Figure 4 will be called the linear processing element (PE), or ADALINE (for adaptive linear element) and it is very simple. It is built from two multipliers and one adder. The multiplier *w scales the input*, and the multiplier *b is a bias*, which can also be thought of as an extra input connected to the value +1.The parameters *(b, w)* have different functions in the solution. We will be particularly interested in studying the dependence of the solution on the parameter(s) that multiply the input x_i.

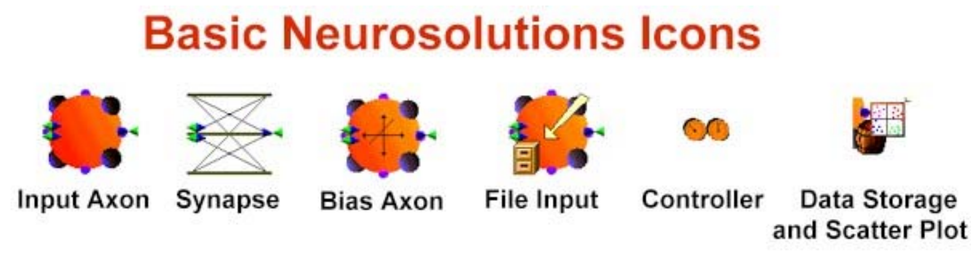
NeuroSolutions 1

1.1 The Linear Processing Element in NeuroSolutions

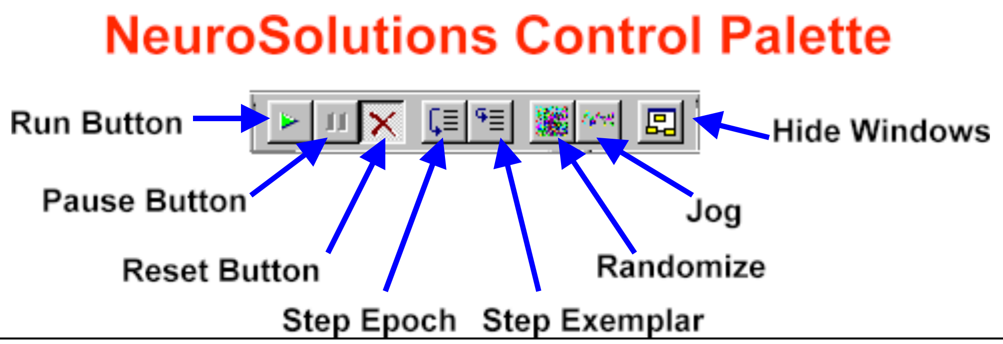
The goal of this book is to demonstrate as many concepts as possible through demonstrations and simulations. Neurosolutions is a very powerful Neural Network/Adaptive System design and simulation package which we will use for the demonstrations. Neurosolutions constructs adaptive systems in a Lego style, i.e. component by component. The components are chosen from palettes, selected with the mouse and dropped in the large window called the breadboard. This object oriented methodology allows for the simple creation of adaptive systems by simply "dragging and dropping" components, connecting them, and then adjusting their parameters. Particularly in the early chapters, we will automatically create the adaptive systems for you through a set of "macros". This will shield you from the details of Neurosolutions until you have a better grasp of the fundamentals of adaptive systems and the use of Neurosolutions.

In this first example, we introduce a few simple components. The first component required in any simulation is an input component, which belongs to the Axon family. Its function is to receive data from the computer file system or from signal generators within the package. In this case, we will add a file input component to the input axon to read in the data from Table 1. The linear PE shown in figure 3 can be constructed with a Synapse and a BiasAxon. The Synapse implements a sum of products and the BiasAxon adds the bias. The output of such system is exactly Eq. 2. The "controller" manages the system and controls the firing of data

through the system. Since Table 1 has 12 data points, the controller is configured to send 12 points through the system.



The purpose of this example is to display the output of the linear PE, which is a line, and modify its location in the space by entering different slope and bias values. To display the input and regression line, we use the DataStorage component (stores 12 samples) and the Scatter Plot component. The Scatter Plot component allows us to plot the input (x axis) versus the system response (y axis). We also add two edit boxes to allow you to change the values of the two parameters, the weight (slope) and bias (y-intercept). After changing these parameters, you use the "control palette" to run the network.



The Run Button is the green triangle and tells the controller to send the data through the network. The other buttons are not important now, but will be used and explained later. Now run the NeuroSolutions Example by clicking on the yellow NeuroSolutions icon below. It will walk you through the creation of the breadboard and allow you to see how the regression line changes as you change the weight (slope) and bias (y intercept).

<u>**NeuroSolutions Example**</u>

We face a problem when trying to fit a straight line to the noisy observations of Table I. A single line will fit any two observations (two points define a line), but it is unlikely that all points will fall on exactly the same line. Since no single line will fit every point, a global property of the points is needed to find the best fit. The problem of fitting a line to noisy data can be formulated as follows: what is the best choice of *(w, b)* such that the fitted line passes the "closest" to all the points?

Goto Next Section

3. Least Squares

Least squares solves the problem by finding the line for which the sum of the square deviations (or residuals) in the *d* direction (the noisy variable direction) are minimized.

The fitted points in the line will be denoted by $\tilde{d}_i = b + wx_i$. The residuals are defined as $\varepsilon_i = d_i - \tilde{d}_i$. The fitted points $\tilde{d}_i$ can also be interpreted as approximated values of *di* estimated by a linear model when the input xi is known,

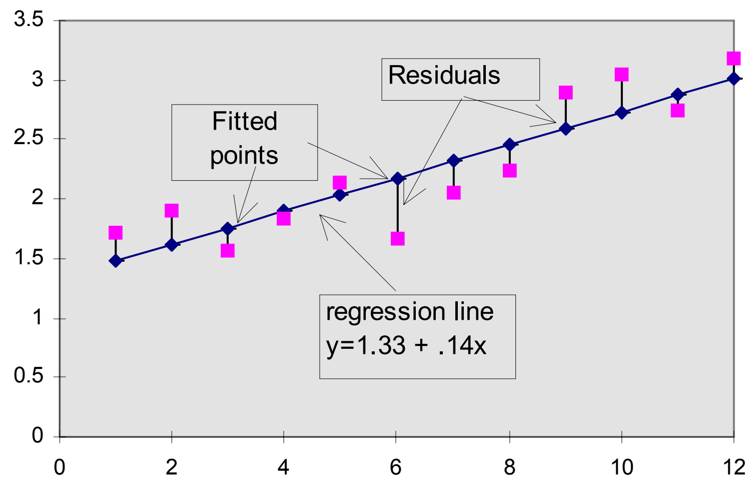


Figure 5. Regression line showing the deviations.

$$d_i - (b + wx_i) = d_i - \tilde{d}_i = \varepsilon_i$$ **Equation 3**

This linear model will be called the linear regressor. Estimated quantities will be denoted by the tilda ~ throughout the book. The outputs of the linear system of Figure 4 are the fitted points, i.e. $\tilde{d}_i = y_i$ in Figure 5. In order to pick the line which best fits the data, we need a criterion to determine which linear estimator is the "best". The average sum of square errors *J* (also called the mean square error (MSE) (MSE)) is a widely utilized performance criterion given by

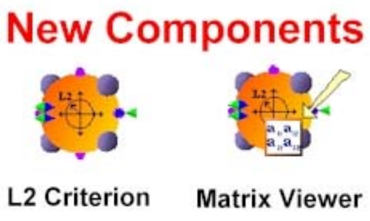
$$\boxed{J = \frac{1}{2N}\sum_{i=1}^{N}\varepsilon_i^2} \qquad \textbf{Equation 4}$$

where N in the number of observations.

NeuroSolutions 2

1.2 Computing the MSE for the linear PE

In order to create a simulation that displays the MSE, we have to add a new component to the breadboard, the L2Criterion. The L2Criterion implements the mean square error Eq. 4 . The L2Criterion requires two inputs to compute the MSE – the system output and the desired response. We will attach the L2Criterion to the output of the linear PE (system output) and attach a file input component to the L2Criterion to load in the value of the desired response from Table 1. In order to visualize the MSE, we will place a MatrixViewer probe over the L2 criterion (cost access point). This MatrixViewer simply displays the data from the component that it resides over – in this case, the mean square error.



Run the demonstration and try to set the slope and bias to minimize the mean square error. Compute by hand the error according to Eq. 4 and see if it matches the value displayed.

NeuroSolutions Example

Our goal is to minimize *J* analytically, which according to Gauss can be done by taking its partial derivative with respect to the unknowns and equate the resulting equations to zero, i.e.

$$\begin{cases} \dfrac{\partial J}{\partial b} = 0 \\ \dfrac{\partial J}{\partial w} = 0 \end{cases}$$

Equation 5

which yields after some manipulation Click here for Least Squares Derivations

$$b = \frac{\sum_i x_i^2 \sum_i d_i - \sum_i x_i \sum_i x_i d_i}{N[\sum_i (x_i - \bar{x})^2]} \qquad w = \frac{\sum_i (x_i - \bar{x})(d_i - \bar{d})}{\sum_i (x_i - \bar{x})^2}$$

Equation 6

where the bar represents the variable's mean value $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$.

This procedure to determine the coefficients of the line is called the least square method. If we apply these equations to the data of Table I, we get the regression equation (best line through the data)

$$d = 0.13951x + 1.33818$$

The least square computation for a large data set is time consuming, even with a computer.

NeuroSolutions 3

1.3 Finding the minimum error by trial and error

Enter these values for the slope and bias by typing them in the respective Edit Boxes. Verify that with these values the error is the smallest. Change the values slightly (in either direction) and see that the MSE increases. Enter a negative slope and see how the error increases a lot. For the negative slope, what is the value of

the bias that gives the smallest error? Note that when one of the coefficients is wrong, the value of the other for best performance is also wrong, i.e. they are coupled.

It is important to explore the NeuroSolutions breadboards. The best way to accomplish this is to open the Inspector associated with each icon. Select a component with the mouse. Then press the right mouse button, and select properties. The Inspector will appear in the screen. The Inspector has fields that allow us to configure the NeuroSolutions components, and tell us what are the settings being used. For instance, go to the input Axon and open the inspector. You will see that it has one input and one output and no weights (go to the soma level to look at the weights). If you do the same in the Synapse you will see that it also has a single input and output and one weight which happens to be our slope parameter. The BiasAxon has a single input and a single output and has a single weight that is our bias.

The large barrel on the input Axon is a probe that collects data. Since the barrel is placed on the activity point, it is storing the 12 data samples that are injected into the network. This is exactly what gets displayed in the x axis of the ScatterPlot. The y axis is sent from the L2Criterion by the small barrel (a data transmitter). So the Scatter plot is effectively displaying the pairs of points (xi,di). Likewise it is also displaying the output of the system in blue, i.e. the pairs of points (xi,yi).

If you want to know what the component is and what it does, just go to the control bar, select the arrow with the question mark, and click on the component that you want to know about (this is called context sensitive help).

NeuroSolutions Example

3.1 Correlation Coefficient

We have found a way to compute the regression equation, but we still do not have a measure of how successfully the regression line represents the relationship between *x* and *d*. The size of the Mean Square Error (J) can be used to determine which line best fits the data, but it doesn't necessarily reflect whether a line fits the data at all because the MSE depends upon the magnitude of the data samples. For instance, by simply scaling the data, one can change the MSE without changing how well the data is fit by the regression line. The correlation coefficient ® solves this problem by comparing the variance of the predicted value with the variance of the desired value variance . The value r^2 represents the amount of variance in the data captured by the linear regression:

$$\boxed{r^2 = \frac{\sum_i (y_i - \bar{d})^2}{\sum_i (d_i - \bar{d})^2}}$$

Equation 7

If we substitute y_i by the equation of the regression line and operate, we obtain Derivation of correlation coefficient

$$r = \frac{\frac{\sum_i (x_i - \bar{x})(d_i - \bar{d})}{N}}{\sqrt{\frac{\sum_i (d_i - \bar{d})^2}{N}}\sqrt{\frac{\sum_i (x_i - \bar{x})^2}{N}}}$$

Equation 8

The numerator is the covariance of the two variables (see Appendix), and the denominator is the product of the corresponding standard deviation . The correlation coefficient is confined to the range [-1,1]. When *r =1* there is a perfect positive correlation between *x* and *d*, i.e. they covary which means that they vary by the same amount. When *r=-1*, there is a perfect negative correlation between *x* and *d*, i.e. they vary in opposite ways (i.e. when *x* increases*, y* decreases by the same amount). When *r=0* there is no correlation between *x* and *d*, i.e. the variables are called uncorrelated. Intermediate values describe partial correlations. For our example *r=0.88* which means that the fit of the linear model to the data is reasonably good.

The method of least squares is very powerful. Estimation theory says that the least square estimator is the "best linear unbiased estimator" (BLUE), since it has no bias and has minimal variance among all possible estimators. Least squares can be generalized to higher order polynomial curves such as quadratics, cubics, etc. *(the generalized least squares).* In this case nonlinear regression models are obtained. More coefficients need to be computed but the methodology still applies. Regression can also be extended to multiple variables (7. Regression for multiple variables). The dependent variable *d* in multiple variable regression is a function of a vector $\mathbf{x}=[x_1,...,x_p]^T$, where T means the transpose. In this book vectors are denoted by bold letters. In this case the regression line becomes a *hyperplane* in the space $x_1,x_2,...x_p$. This case will be studied later in the chapter.

Go to Next Section

4. Adaptive Linear Systems

4.1. Least squares as a search for the parameters of a linear system

The purpose of least squares is to find parameters *(b, w)* that minimize the difference between the system output y_i and the desired response d_i. *So, regression is effectively computing the optimal parameters of an interpolating system* (linear in this case) which predicts the value of *d* from the value of *x*.

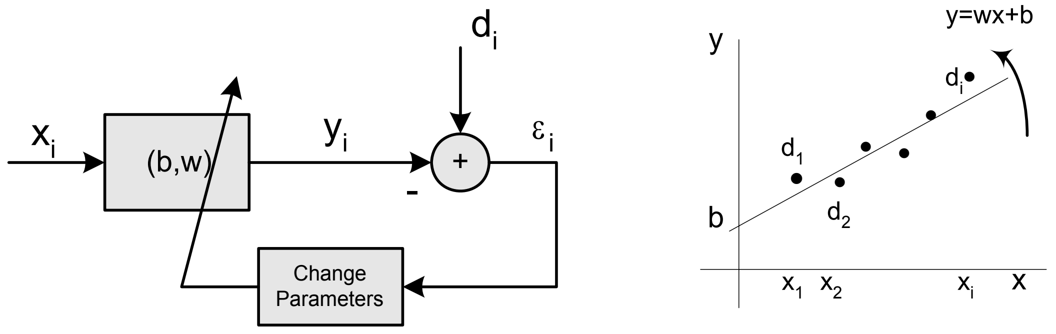


Figure 6. Regression as a linear system design problem

Figure 6 shows graphically the operation of adapting the parameters of the linear system. The system output *y* is always a linear combination of the input *x* with the bias, so it has to lie on a straight line of equation *y=wx+b*. Changing *b* modifies the *y* intersect, while changing *w* modifies the slope. Therefore we conclude that the goal of linear regression is to adjust the position of the line such that the average square difference between the *y* values (on the line) and the cloud of points d_i i.e. the criterion *J* is minimized.

The key point is to recognize that the error contains information that can be used to optimally place the line. Figure 6 shows this by including a subsystem that accepts the error as input and modifies the parameters of the system. Thus, the *error ε_i is fed back to the system* and indirectly affects the output through a change in the parameters *(b,w)*. Effectively the system is made "aware" of its performance through the error. With the incorporation of the mechanism that automatically modifies the system parameters, a very powerful linear system can be built that will constantly seek optimal parameters. Such systems are called Neural and Adaptive systems, and are the focus of this book.

4.2. Neural and Adaptive systems

Before pursuing the study of adaptive systems, it is important to reflect briefly on the implications of neural and adaptive systems in Engineering design. System design usually begins with specifications. First the problem domain is studied and modeled, specifications are established, and then a system is built to meet the specifications. The key point is that the system is built to meet the current specifications and will always use the designed set of parameters, even if the external conditions change.

Here we are proposing a very different system design approach based on adaptation which has a biological flavor to it. In the beginning the system parameters may be way off, creating a large error. However, through the feedback from the error, the system can change its parameters to decrease the error as much as possible. The system's "experience" with the data designs the best set of parameters. An adaptive system is more complex because it not only has to accomplish the desired task, but also has to be equipped with a subsystem that adapts its parameters. But notice that even if the data changes in the future, this design methodology will modify the system parameters such that the best possible performance is obtained. Additionally, the same system can be used for multiple problems.

There are basically two ways to adapt the system parameters: supervised learning and unsupervised learning. The method described until now belongs to supervised learning because there is a desired response. Later on in the book we will find other methods that also adapt the system parameters, but using only an internal rule. Since there is no desired response these methods are called unsupervised. We will concentrate here on supervised learning methods.

The ingredients to pursue adaptive system design are:

- a system (linear in this case) with adaptive parameters;
- the existence of a desired or target response *d*;
- an optimality criterion (the MSE in this case) to be minimized;
- a method (subsystem) to compute the optimal parameters.

The method of least squares finds the optimal parameters *(b,w)* analytically. Our goal is to find alternate ways of computing the same parameters using a search procedure.

4.3. Analysis of the error in the space of the parameters - The performance surface.

Let us analyze the mean square error (*J*) as we change the parameters of the system (*w* and *b*). Without loss of generality, we are going to assume that *b=0* (or equivalently that the mean of *x* and *d* have been removed), such that *J* becomes a function of the single variable *w*

$$\boxed{J = \frac{1}{2N}\sum_i \left(d_i - wx_i\right)^2 = \frac{1}{2N}\sum_i \left(x_i^2 w^2 - 2d_i x_i w + d_i^2\right)}$$

Equation 9

If *w* is treated as the variable and all other parameters are held constant, one can immediately see that *J* is quadratic on *w* with the coefficient of w^2 (e.g. x_i^2) being always positive. In the space of the possible *w* values, *J* is a *parabola* facing upwards (*J* is always positive since it is a sum of squares). The function *J(w)* is called the Performance surface for the regression problem (Figure 7). The performance surface is an important tool that helps us visualize how the adaptation of the weights affects the mean square error.

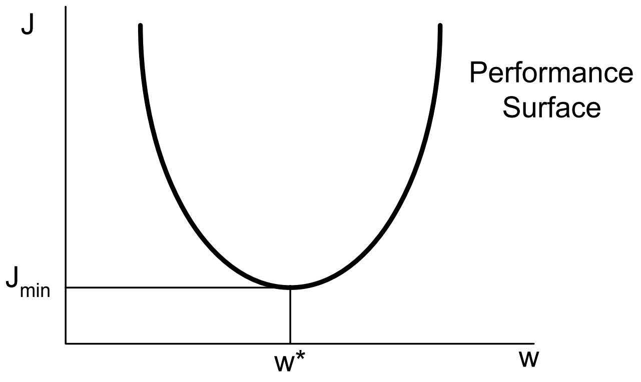


Figure 7. The performance surface for the regression problem

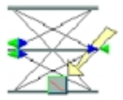
NeuroSolutions 4

1.4 Plotting the performance surface

The performance surface is just a plot of the error criterion (J) versus the value of the weights. So what we will do is to vary the Synapse weight (which corresponds to the slope parameter of the linear regressor) between two appropriate values during the simulation. We can imagine that the error will be minimum at an intermediate value of the weight, and it will increase for both lower values and higher values.

In order to modify incrementally the Synapse weight we will attach a "linear scheduler" to the Synapse, and place the MatrixViewer on it so we can see how the weight is changing. In order to visualize the MSE we will bring another ScatterPlot to the L2 criterion component. This will allow us to plot the cost versus weight (performance surface).

New Components


Linear Scheduler
on the Synapse

Now, run the example and see how the slope parameter of the linear PE affects the mean square error of the linear regressor. As we are going to see the input and desired signals affect tremendously the shape of the performance surface. But can you change the shape of the performance curve without touching the data files? Let us change the L2Criterion. Go to Palettes and open the ErrorCriteria. Click on the Lp criterion and bring the pointer to the breadoard. Notice that the pointer changed to a stamper. If you left click on the L2Criterion component, the L2 is substituted by the new component which computes a cost given by

$$J = \frac{1}{p}\sum_i \varepsilon_i^p$$

By default the norm is p=5. Run the simulation again. What do you see? Does the location of the minimum change appreciably? What about the shape of the

performance surface? Do you understand now better the function of the cost?

NeuroSolutions Example

Using the performance surface, we can develop a geometric method for finding the value of *w,* here denoted by *w**, which minimizes the performance criterion. Previously, we computed *w** by setting to zero the derivative of J with respect to *w*.

T*he* gradient *of the performance surface* is a vector (with the dimension of *w*) which always points towards the direction of maximum *J* change and with a magnitude equal to the slope of the tangent of the performance surface (Figure 8). If you visualize the performance surface as a hill-side, each point on the hill will have a gradient arrow which points in the direction of steepest *ascent* at that point, with larger magnitudes for steeper slopes. Thus, a ball rolling down the hill will always attempt to roll in the opposite direction of the gradient arrow (steepest descent). The slope at the bottom is zero, so the gradient is also zero (that is the reason the ball stops there).

In our special case the gradient has just one component along the weight axis *w*

$\nabla J = \nabla_w J$ given by

$$\nabla_w J = \frac{\partial J}{\partial w}$$ **Equation 10**

gradient definition and construction A graphical way to construct $\nabla_w J$ at a point w_0 is to first find the level curve (curve of constant *J* value) that passes through the point (also called the contour plot). Then take the tangent to the level curve at w_0. The gradient component $\nabla_w J$ is always perpendicular to the contour curve at w_0, with a magnitude given by the partial derivative of *J* with respect to the weight *w* (Eq. 10). For one weight as in Figure 8 (1-Dimensional problem) the construction is simplified and we have to only find the direction of the gradient on the axis.

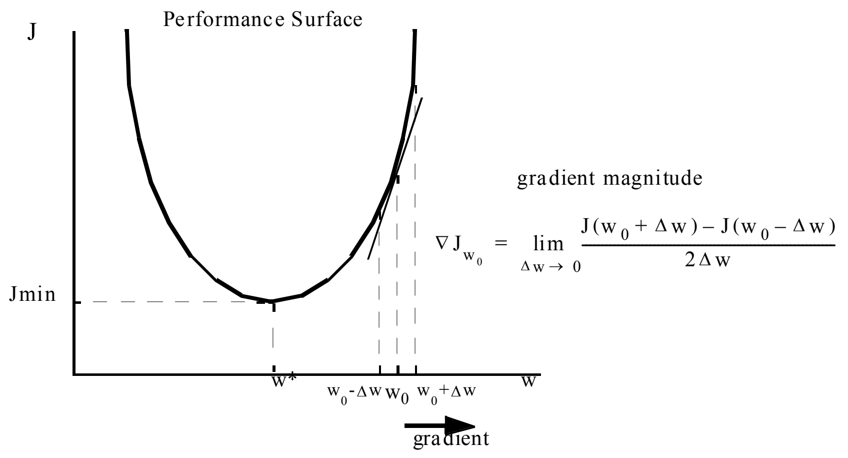

Performance Surface
J
Jmin
gradient magnitude
$\nabla J_{w_0} = \lim_{\Delta w \to 0} \frac{J(w_0 + \Delta w) - J(w_0 - \Delta w)}{2\Delta w}$
w*
$w_0 - \Delta w$
w_0
$w_0 + \Delta w$
w
gradient

Figure 8. Performance surface and its gradient

At the bottom of the bowl, the gradient is zero, because the parabola has slope 0 at the vertex. So, for a parabolic performance surface, computing the gradient and equating it to zero finds the value of the coefficients that minimize the cost, just as we did in Eq.6 . The important observation is that the analytical solution found by the least squares coincides with the minimum of the performance surface. Substituting the value of *w** into Eq.9 , the minimum value of the error (J_{min}) can be computed.

more derivation of performance surface

NeuroSolutions 5

1.5 Comparison of performance curves for different data sets

In this example, we will provide two sets of input files and two sets of output files. By changing the input data we will find that the minimum error, its location in the weight space (a weight line in this 1D example), as well as the shape of the performance surface changes. On the other hand, if we change the desired signal, only the minimum value of the performance and its location changes, but the overall shape remains de same.

NeuroSolutions Example

4.4. Search of the performance surface with steepest descent

Since the performance surface is a paraboloid which has a single minimum, an alternate procedure to find the best value of the coefficient *w* is to search the performance surface instead of computing the best coefficient analytically by Eq.6 . The search for the minimum of a function can be done efficiently using a broad class of methods based on *gradient information*. The gradient has two main advantages for search.

- The gradient can be computed locally.
- The gradient always points in the direction of maximum change.

If the goal is to reach the minimum, the search must be in the direction opposite to the gradient. So, the overall method of search can be stated in the following way:

Start the search with an *arbitrary initial weight w(0),* where the *iteration* is denoted by the index in parenthesis. Then compute the gradient of the performance surface at *w(0),* and modify the initial weight proportionally to the negative of the gradient at *w(0).* This changes the operating point to *w(1).* Then compute the gradient at the new position *w(1),* and apply the same procedure again, i.e.

$$w(k+1) = w(k) - \eta \nabla J(k)$$ **Equation 11**

where η is a small constant and ∇J denotes the gradient of the performance surface at the kth iteration. η is used to maintain stability in the search by ensuring that the operating point does not move too far along the performance surface. This search procedure is called the steepest descent method. Figure 9 illustrates the search procedure

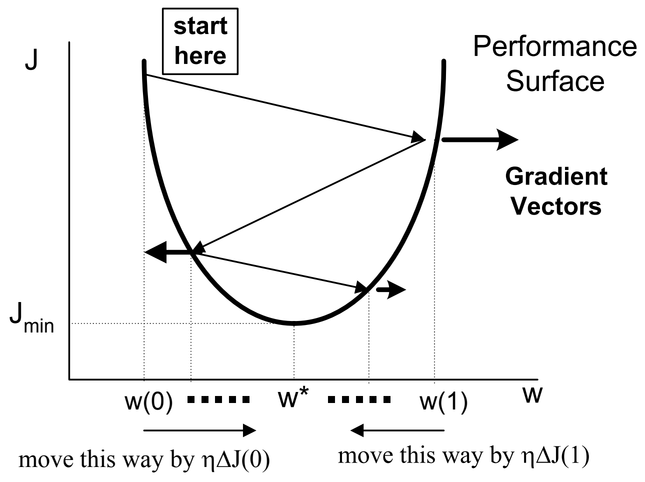


Figure 9. The search using the gradient information

If one traces the path of the weights from iteration to iteration, intuitively we see that if the constant η is small, eventually the best value for the coefficient *w** will be found. Whenever *w>w**, we decrease *w*, and whenever *w<w**, we increase *w*.

Goto Next Section

5. Estimation of the gradient - the LMS algorithm

An adaptive system can use the gradient to optimize its parameters. The gradient, however, is usually not known analytically, and thus must be estimated. Traditionally, the difference operator estimated the derivative as outlined in Figure 8. A good estimate, however, requires many small perturbations to the operating point to obtain a robust estimation through averaging. The method is straight forward but not very practical.

In the late 1960's Widrow , proposed an extremely elegant algorithm to estimate the gradient that revolutionized the application of gradient descent procedures. His idea is very simple*: Use the instantaneous value of the gradient as the estimator for the true quantity*. This means to drop the summation in Eq.9 , and define the gradient estimate at step k as its instantaneous value. Substituting Eq. 4 into Eq.10 , removing the summation, and then taking the derivative with respect to *w* yields

$$\nabla J(k) = \frac{\partial}{\partial w} J(k) = \frac{\partial}{\partial w} \frac{1}{2N} \sum \varepsilon^2 \approx \frac{1}{2} \frac{\partial}{\partial w} \left(\varepsilon^2 (k) \right) = -\varepsilon(k) x(k)$$ **Equation 12**

What Eq. 12 tells us is that an *instantaneous estimate of the gradient is simply the product of the input to the weight times the error at iteration k*. The amazing thing is that the gradient can be estimated with one multiplication per weight. This is the gradient estimate that led *to the famous Least Means Square (LMS) algorithm* (or LMS rule). The estimate will be noisy, however, since the algorithm uses the error from a single sample instead of summing the error for each point in the data set (e.g. the MSE is estimated by the error for the current sample). But remember that the adaptation process does not find the minimum in one step. Normally many iterations are required to find the minimum of the performance surface, and during this process the noise in the gradient is being averaged (or *filtered*) out.

If the estimator of Eq.12 is substituted in Eq.11 , the steepest descent equation becomes

$$w(k+1) = w(k) + \eta \varepsilon(k) x(k)$$ **Equation 13**

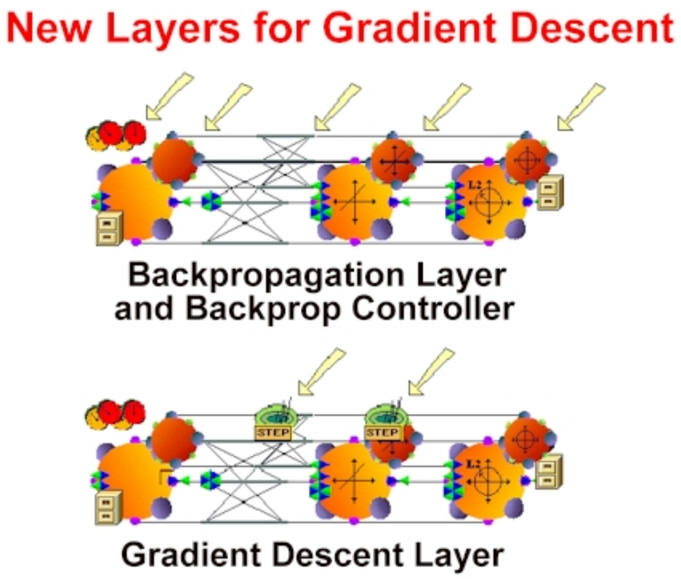
This equation is the *LMS algorithm*. So, with the LMS rule one does not need to worry about perturbation and averaging to properly estimate the gradient at each iteration, it is the iterative process that is improving the gradient estimator. The small constant η is called the step size *or the learning rate*.

NeuroSolutions 6

1.6 Adapting the linear PE with LMS

Several things have to be added to the previous breadboard of the linear PE to make it learn automatically using the LMS algorithm. The methodology will be explained in more detail later. However, the technique used in NeuroSolutions is called "backpropagation". In short, the algorithm passes the input data forward through the network and the error (desired - output) backwards through another network. The error is propagated through a second layer which can be obtained

from the first with minor and well established modifications (more about this later). So at every component there is a local activity (the x) and a local error (the ε) such that the weights of the network can be modified by Eq. 13. NeuroSolutions implements this technique by adding two additional layers to the network: the backpropagation layer and the gradient search layer. These two layers can be automatically added to the breadboard. The backpropagation layer looks like a small version of the network which sits on top of the original network (in red instead of orange). The gradient search layer sits on top of the backpropagation layer and uses the gradient search method to adjust the weights. In our case, the gradient search layer is a simple "step" layer which implements the gradient descent rule Eq.13 . Notice that only the components which have adjustable weights (the synapse (w) and bias axon (bias)) have gradient search components.



In addition to adding the two layers, we need an additional controller to manage the backpropagation layer. This controller sits above the yellow controller from before. The backprop controller is where we set parameters like whether we use batch or on-line learning. In this example, we will use batch learning, i.e. the system will compute all the weight updates for the training set add them up, and at

the end of the epoch (one presentation of all the training data) update the weights according to Eq. 13. The initial value of the step size will be set at 0.01. Now we can click on the start button of the controller to initiate the simulation.

When you run the network, watch the regression line move towards the optimum value in the ScatterPlot. When the network has finished, notice that the weight is approximately .139, the bias is approximately 1.33 and the error is approximately 0.033 – all in excellent agreement with the optimal values we computed analytically.

You should explore this breadboard by entering several values of the stepsize, and opening the Inspector to see how each component is configured.

NeuroSolutions Example

5.1. Batch and sample by sample learning

The LMS algorithm was presented in a form where the weight updates are computed for each input sample, and the weights modified after each sample. This procedure is called sample by sample learning or on-line training . As we have mentioned, the estimate of the gradient is going to be noisy, i.e. the direction towards the minimum is going to zigzag around the gradient direction.

An alternative solution is to compute the weight update for each input sample, but store these values during one pass through the training set which is called *an (epoch)* . At the end of the epoch all the contributions are added, and only then the weights will be updated with the composite value. This method adapts the weights with a cumulative weight update, so it will follow the gradient more closely. It is called the *batch training mode or batch learning*. Batch learning is also an implementation of the steepest descent procedure. In fact, it provides an estimator for the gradient that is smoother than the LMS. We will see that the agreement between the analytical quantities that describe adaptation and the ones obtained experimentally is excellent with the batch update.batch versus online learning

In order to visualize the differences between these two update methods, we will plot the decrease of the error during adaptation (the learning curve) with both of them.

NeuroSolutions 7

1.7 Batch versus online adaptation

It is important to visualize the differences in adaptation for on-line and batch learning. Up to now we have been using the batch mode. In this example we will set the Backprop Controller to use on-line training. To display the learning curve, we have to introduce one new component – the Megascope. The Megascope is a probe, similar to the Scatter Plot. The Megascope acts just like an oscilloscope – it plots a continuous stream of inputs, using the iteration number as the x-axis. To create the learning curve, we simply place a data barrel over the L2 Criterion and then place a Megascope on top of the data barrel.

We will see that the learning curve is not smooth anymore because we are updating the weights after each example. Since the individual errors vary from sample to sample, our updates will make the learning curve noisy. The learning curve will have a periodic component superimposed on a decaying exponential. The exponential tells us that we are approaching a better overall solution. The periodic features show the error obtained for each input sample. So the envelope is related to the learning curve for the batch mode. Note that the weights never

stabilize, otherwise the performance curve should be smooth and converge to a single final value. Since there is more noise in on-line learning, we must decrease the step size to get smoother adaptation. But the price paid is a longer adaptation time, i.e. the system needs more iterations to get to a predefined final error. Experiment with the learning rates to observe this behavior.

NeuroSolutions Example

5.2. Robustness and system testing

One of the interesting aspects of the LMS solution is its robustness. From the picture given, no matter what is the initial condition for the weights, the solution always converges to basically the same value. We can even add some noise to the desired response and find out that the linear regressor parameters are basically unchanged. This robustness is rather important for real world problems, where noise is omnipresent.

The group of input samples and desired responses (shown in Table I) used to train the system are called collectively the training set for obvious reasons. It is with their information that the system parameters were adapted. But once the optimal parameters are found, the parameters can be fixed and the system can be utilized in new inputs never encountered before. It will produce for each input a response based on the parameters obtained during training which should resemble the value of the desired response for that particular input value.

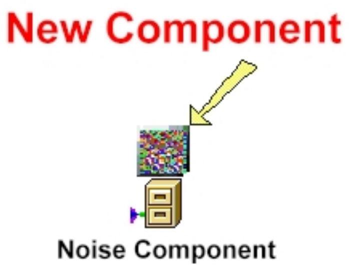
So we see that the system has the ability to extrapolate responses for new data. This is an important feature since in general the system will be deployed and one wishes that the performance obtained in the training set will also apply (generalize) to the new data. But due to the methodology utilized to derive the parameter values one can never be exactly sure of how well the system will respond to new data.

For this reason it is a good methodology to use a test set to verify the system performance before deploying it to the real world application. The test set consists of new data not used for training, but for which we still know the desired response. It is kind of the final rehearsal before the play's inauguration. One should also compute the correlation coefficient in the test set. Normally we will find a slight decrease in performance from the training set. If the performance in the test set is not acceptable one has to go back to the drawing board. When this happens in regression the most common source is lack of data in the training or not an exhaustive coverage of experimental conditions. This point will be addressed in more depth in the following chapters.

NeuroSolutions 8

1.8 Robustness of LMS to noise

The LMS algorithm is very robust. It will work from any arbitrary location and even work well with noise added to the desired data. In order to demonstrate that the system works well even with noisy data, we will add one additional component to the breadboard from the previous example – the noise component. The noise component allows uniform, Gaussian, or "user defined" noise to be added to the input or desired signals. We will add the noise component to the desired signal and watch as the system moves close to the optimum location even with the noisy data.



<u>NeuroSolutions Example</u>

5.3. Computing the correlation coefficient in adaptive systems

The correlation coefficient, *r*, tells how much of the variance of *d* is captured by a linear regression on the independent variable *x*. As such, *r* is a very powerful quantifier of the modeling result. It has a great advantage with respect to the MSE (mean square error) because it is automatically normalized, while the MSE is not. However, the correlation coefficient is "blind" to differences in means because it is a ratio of variances (see Eq.7), that is, as long as the desired and output co-vary *r* will be small, in spite of the fact that they may be far apart in actual value. So one effectively needs both quantities (*r* and MSE) when testing the results of regression.

Eq. 7 presents a simple way of computing the correlation coefficient requiring only knowledge of *y* and *d*. Note, however, that *y* changes during adaptation so one should wait until the system adapts to read the final correlation coefficient. During adaptation the numerator of Eq. 7 can be larger than the denominator giving a value for *r* larger than 1, which is meaningless. So we propose to compute a new parameter *g* that is a reasonable proxy to the correlation coefficient even during adaptation. We subtract a term from the numerator of Eq. 7 that becomes zero at the optimal setting (i.e. $g \to r \quad when \quad w \to w^*$) but limits *g* such that its value is always between -1 and 1 even during adaptation. We can write computation of correlation coefficient

$$g = \frac{\sqrt{\sum_i (y_i - \bar{d})^2} - \dfrac{\sum_i \varepsilon_i (y_i - \bar{d})}{\sqrt{\sum_i (y_i - \bar{d})^2}}}{\sqrt{\sum_i \left(d_i - \bar{d}\right)^2}}$$

Equation 14

Note that all these quantities can be computed on line with the information of the error, the output and the desired response. Remember however that Eq. 14 measures the correlation coefficient only when the adaline has been totally adapted to the data.

NeuroSolutions 9

1.9 Estimating the correlation coefficient during learning

NeuroSolutions does not include a component to compute the correlation coefficient. It does, however, allow you to write your own components. These custom components are called DLLs. A custom component looks just like the component it takes the place of, except that its icon has "DLL" printed on it. In this example, we include a custom component to compute the correlation coefficient. This component looks exactly like an L2 component except it has "DLL" printed on it.

Plug in the values of the optimal weights and verify that the formula Eq.14 gives the correct correlation coefficient. Slightly modify w to 0.120 and verify that the correlation coefficient decreases. If you plug in values for w and b that are very far away from the fitted regression, this estimation of r using Eq. 14 becomes less accurate, but still bound by -1 and 1. The example also uses LMS to adapt the coefficients. Observe that the correlation coefficient is always between -1 and 1 during adaptation and that the final value corresponds to the computed one.

NeuroSolutions Example

Goto Next Section

6. A Methodology for Stable Adaptation

During adaptation, the learning algorithm automatically changes the system parameters by Eq.13 . This adaptation algorithm has one parameter (e.g. the step size) that must be user selected. In order to appropriately set the parameter, the user should have a good understanding of what is happening inside the system. In this section, we will quantify the adaptation process and develop visualization tools that will help understand how well the system is learning.

6.1. Learning curve

As is readily apparent from Figure 9, when the weights approach the optimum value, the values of *J(w(k))* (the MSE at iteration k) will also decrease, approaching its minimum value J_{min}. One of the best ways to monitor the convergence of the adaptation process is to plot the error at each iteration. The plot of the MSE across iterations is called the learning curve (Figure 10). The learning curve is as important for adaptive systems as the thermometer is to check your health. It is an *external, scalar, easy to compute indication* of how well the system is learning. But similar to body temperature, it is unspecific, i.e. when the system is not learning it does not tell us why.

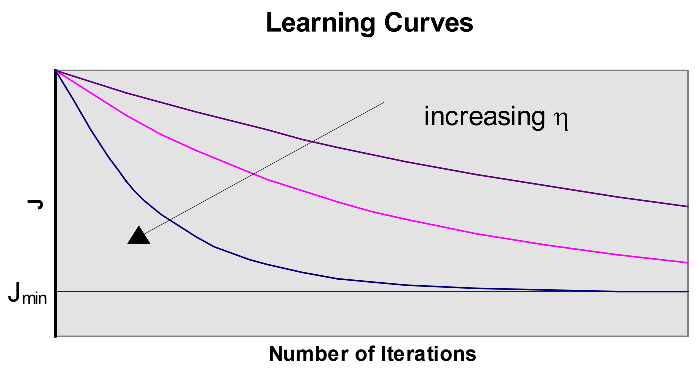


Figure 10. The learning curve

Notice that the error approaches the minimum in an one sided manner (i.e. always larger than J_{min}). As one can expect, the rate of decrease of the error depends on the value of the step size η. Larger step sizes will take less iterations to reach the neighborhood of the minimum provided the adaptation converges. However, too large a step size creates a divergent iterative process and the optimal solution is not obtained. It is interesting to note that we would like as large a step size as possible because this decreases the convergence time. However, if the step size is increased too much divergence will result.

So we must seek a way to find the largest possible step size that guarantees convergence.

NeuroSolutions 10

1.10 The learning curve

The goal of this example is to display the learning curve and show how the learning rate affects its shape. This will plot the Mean Squared Error over time which is the Learning Curve, the thermometer of learning.

When you run the simulation, watch as the regression line moves towards the optimum location how the error moves towards zero. You can also change the learning rates and watch how the regression line moves faster or slower towards the optimum location, thus causing the learning curve to be steeper or shallower. The visualization of the regression line contains more information about what the system is doing, but is very difficult to compute and display in higher dimensions. The learning curve, however, is an external, scalar quantity that can be easily measured with minimal overhead.

NeuroSolutions Example

6.2. Weight tracks

An adaptive system modifies its weights in an effort to find the best solution. The plot of the value of a weight over time is called the weight track . Weight tracks are an important and direct measure of the adaptation process. The problem is that normally our system has many weights and we don’t know what their optimal values are. *Nevertheless the dynamics of learning can be inferred and monitored from the weight tracks*.

In the gradient descent adaptation, adjustments to the weights are governed by two quantities Eq.11 : the *step size* η, and the value of the gradient at the point. Even for a constant step size, the weight adjustments will become smaller and smaller as the adaptation approaches *w**, since the slope of the quadratic performance surface is decreasing near the bottom of the performance surface. Thus, the weights approach their final values asymptotically (Figure 11).

Three cases are depicted in Figure 11. If the step size is small, the weight converges monotonically to *w**, and the number of iterations to reach the bottom of the bowl may be large. If the step size η is increased, the convergence will be faster but still monotonic. After a value called *critically damped*, the weight will approximate *w** in an oscillatory fashion ($\eta_2 > \eta_1$), i.e. it will overshoot and undershoot the final solution. The number of iterations necessary to reach the neighborhood of w* will increase again. If the step size is too large ($\eta_3 > \eta_2$), the iterative process will diverge, i.e. instead of getting closer to the minimum, the search will visit points of larger and larger MSE, until there is a numeric overflow. We say that the learning diverged.

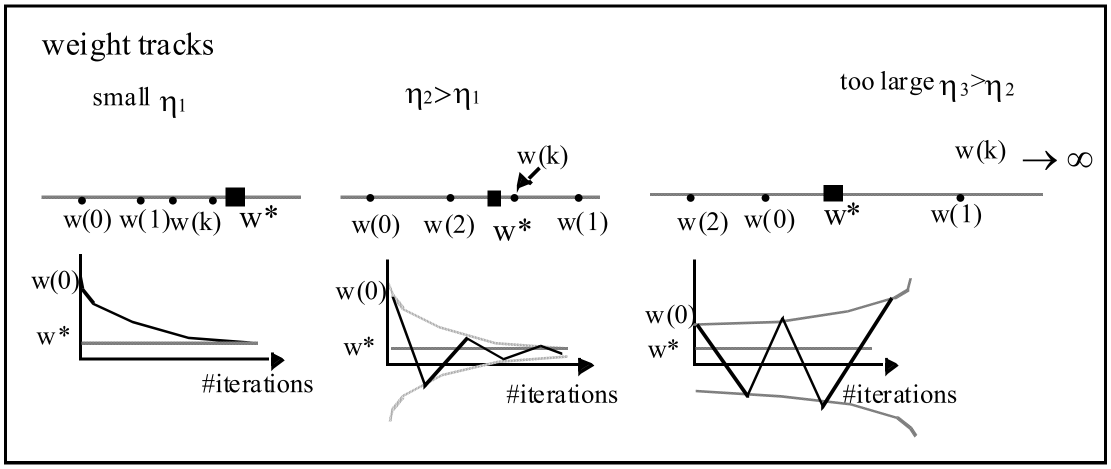


Figure 11. Weight tracks and plots of the weight values across iteration for 3 values of η.

NeuroSolutions 11

1.11 Weight tracks

It is very instructive to observe the linear PE parameters during learning, and how they change as a function of the step size. Let us install a MegaScope over the

Synapse to visualize the slope parameter of the regressor, and over the BiasAxon to visualize the regressor bias. These are called weight tracks. Run the simulation and watch how changing the step sizes affects the way the system approaches its final weights.

The weight tracks are a finer display of how adaptation is progressing, but the problem is that in systems with many weights, it becomes impractical to observe all the weight tracks. Why do we say that weight tracks give us a better handle on the adaptation parameters? Enter 0.02 for the stepsize and see the weight tracks converge monotonically to their minimum value. Now enter 0.035. The weight tracks are oscillating towards the final value which means that the system is already in the underdamped regime (but the learning curve is still monotonically decreasing towards the minimum at a faster rate). We can expect divergence if we increase the weighs further. Try 0.038 and see it happen. Relate this behavior with Figure 11.

NeuroSolutions Example

6.3. Largest step size for convergence

As we have just discussed, the user would like to choose the largest step size possible for fastest convergence without creating an unstable system. Since adjustment to the weights is a product of the step size and the local gradient of the performance surface, it is clear that the largest step size depends upon the shape of the performance surface. We saw already that the shape of the performance surface is controlled by the input data Eq.54 . So we can conclude that the maximum step size will be dictated by the input data. But how?

If we rewrite the equations which produce the weight values in terms of the first weight w(0), derivation of largest stepsize we get

$$w(k+1) = w^* + (1-\eta\lambda)^k (w(0) - w^*)$$ **Equation 15**

where

$$\lambda = \frac{1}{N}\sum_i x_i^2$$ **Equation 16**

Since the term *(1-ηλ)k* is exponential, it must be less than or equal to one to guarantee weight convergence (and less than one to guarantee convergence to 0, giving *w(k+1)=w**). This implies that

$$|\rho| = |1-\eta\lambda| < 1 \Rightarrow \eta < \frac{2}{\lambda}$$ **Equation 17**

where ρ is the geometric ratio of the iterative process. Hence, the value of the step size η must always be smaller than $2/\lambda$. *The fastest convergence is obtained with the critically damped step size of 1/λ*. The closer η is to $1/\lambda$ the faster is the convergence, but faster convergence also means that the iterative process is closer to instability. We can visualize this in Figure 11. When η is increased, a monotonic (overdamped) convergence to w* is substituted by an alternating (underdamped) convergence that finally degenerates into divergence.

There is a slight practical problem that must be solved. During batch learning the weight updates are added together during an epoch to obtain the new weight. This effectively includes a factor of N in the LMS weight update formula Eq.13 . In order to apply the analysis of the largest stepsize Eq.17 one has to use a normalized stepsize

$$\eta_n = \frac{\eta}{N}$$ **Equation 18**

With this modification, even if the number of samples in our experiment changes, the stepsizes do not need to be modified. Note that for on-line learning (N=1) we get the LMS rule again. We will always use normalized stepsizes but to make the notation simpler, we will drop the subscript *n* in the normalized stepsize. An added advantage of using normalized stepsizes is that we can switch between on-line updates and batch updates without having to change the stepsize in the simulations.

This analysis of the largest stepsize Eq.17 also applies *in the mean* to the LMS algorithm. However, since the LMS uses an instantaneous (noisy) estimate of the gradient, even when η obeys Eq.17, instability may occur. When the iterative process diverges, the algorithm "forgets" its location in the performance surface, i.e. the values of the weights will change drastically. This means that all the iterations up to that point were wasted. Hence, with the LMS it is common to include a safety factor of 10 in the largest η ($\eta=0.1/\lambda$), or to use batch training.

NeuroSolutions 12

1.12 Linear regression without bias

The previous example solved the linear regression problem with one weight and one bias. In order to compare the equations given above (which are a function of a single parameter) with the simulations, we have to make a modification in the data set or in the simulation. Shortly we will see how to extend the analysis for multiple weights, but for the time being let us work with the simpler case.

We will substitute the BiasAxon by an Axon, a component that simply adds its inputs, i.e. the regression solution becomes y=wx which has to pass through the origin. With this new breadboard we can compare the numerical results of the simulations directly with all the equations derived in this section since there is only a free parameter. Batch updates will be used throughout.

The optimal value of the slope parameter is computed by Eq.6 , which gives w=0.30009, with an average error of 0.46. This solution is different from the value obtained previously (w= 0.139511) for the bias regressor because the regression line is now constrained to pass through the origin. It turns out that this constrained solution is worse than before as we can see by the error (0.23 versus

0.033). Observing in the scatter plot the output (red points) and the input samples (blue) shows clearly what we are describing.

Computing λ Eq.16 yields 54. So, according to Eq.17 the maximum step size is η=3.6e-2. The critically damped solution is obtained with a step size of 1.8e-2, and adaptation with a stepsize below this value is overdamped. When we run the simulator in the overdamped case, the weights approach the final value monotonically; for the critically damped case, they stabilize quite rapidly; while for the underdamped case they oscillate around the final value, and the convergence takes more iterations. Notice also that the linear regressor "vibrates" around the final position, since the slope parameter is overshooting and undershooting the optimum value.

According to Eq.19 for the critically damped stepsize τ=1, so the solution should stabilize in 4 updates (epochs). This stepsize yields the fastest convergence. Go to the Controller Inspector and use the epoch button to verify the number of samples until convergence.

NeuroSolutions Example

6.4. Time constant of adaptation

An alternative view of the adaptive process is to quantify the convergence of *w(k)* to *w** in terms of an exponential decrease. We know that *w(k)* converges to *w** as a geometric progression (Eq.15). The envelope of the geometric progression of weight values can be approximated by an exponential decay *exp(-t/τ),* where τ is the time constant of weight adaptation. A single iteration can be linked to a time unit. So one may want to approximately know how many iterations are needed until the weights converge. The time constant of weight adaptation can be written:

$$\tau = \frac{1}{\eta\lambda}$$

Equation 19

derivation of the time constant of weight adaptation

which clearly shows that fast adaptation (small time constant τ) requires large step sizes. For all practical purposes the iterative process converges after 4 time constants.

The steps used to derive the time constant of weight adaptation can be applied also to come up with a closed form solution to the decrease of the cost across iterations which is called *the time constant of adaptation*. Eq.15 tells us how the weights converge to w^*. If the equation for the weight recursion is substituted in the equation for the cost (Eq.55) we get

$$J = J_{\min} + \lambda(1-\eta\lambda)^{2k}(w(0)-w^*)^2$$

which means that J also approximates J_{min} in a geometric progression, with a ratio equal to ρ^2. Therefore the time constant of adaptation is

$$\tau_{mse} = \frac{\tau}{2}$$

Since the geometric ratio is always positive, J approximates J_{min} monotonically (i.e. an exponential decrease). The time constant of adaptation describes practically the learning time (in number of iterations) needed to adapt the system. Notice that these expressions assume that the adaptation follows the gradient. With the instantaneous estimate used in the LMS, J may oscillate during adaptation since the estimate is noisy. But even in the LMS, J will approach J_{min} in a one sided way (i.e. always greater than or equal to J_{min}).

6.5. Rattling

Up to now our main focus was the speed of adaptation, i.e. how fast the weights approximate w^*, or equivalently, how fast J approximates J_{min}. Unfortunately, this is only part of the story. For fast convergence we need large step sizes (η). But, when the search is close to the minimum *w**, where the gradient is small but not zero, the iterative process continues to wander around a neighborhood of the minimum solution without ever stabilizing. This phenomenon is called rattling (Figure 12), and the rattling basin increases proportionally to the step size η. This means that when the adaptive process is stopped by an external command (such as the number of iterations through the data), the weights may not be exactly at *w**. We know they are in a neighborhood of this point, but not exactly at the optimum.

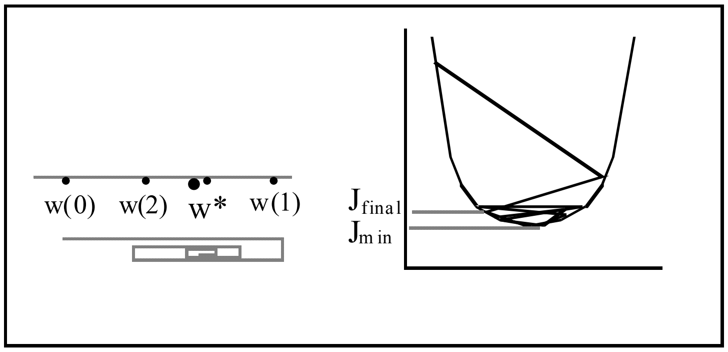


FIGURE 12. **Rattling of the iteration procedure**

If we picture the performance surface (Figure 12), when the final weights are not at *w** there will be a penalty in performance, i.e. the final MSE will be higher than J_{min}. In the theory of adaptation, the difference between the final MSE and the J_{min} (normalized by J_{min}) is called the misadjustment M.

$$M = \frac{J_{final} - J_{\min}}{J_{\min}}$$ **Equation 20**

This means that in search procedures that use gradient descent there is an intrinsic compromise between accuracy of the final solution (small misadjustment) and speed of convergence. The parameter that controls this compromise is the step size η. High η means fast convergence but also large misadjustment, while small η means slow convergence but little misadjustment.

NeuroSolutions 13

1.13 Rattling

We observed in Example 8 how noisy the learning curve became with the on-line update. This is an external indication that the weights were changing from sample to sample even after the system reached the neighborhood of the optimum. The implication of this random movement in the weights is a penalty in the final MSE. In this example we will exactly show and quantify the rattling.

The rattling has important consequences for adaptation, since if one sets the stepsize large for fast convergence we pay a price of inaccurate coefficients, which is translated in an excess MSE. The rule of thumb for LMS is to use a stepsize that is 1/10 of the largest possible stepsize. If this is not done the regressor is basically unusable since the weights ever stabilize. Effectively we do not have a single regressor but a family of systems, each with a different parameter. We can see this in the ScatterPlot since the blue dots are no longer in a straight line. For stepsize close to the largest possible, effectively the MSE for the epoch is smaller than the theoretical minimum, which is impossible. This happens because the parameters are changing so much with each update that the slope is being continuous changed with the present sample. The problem is that when we stop the training we do not know if the final value of the weight is a good approximation to the theoretical regression line.

This shows that for adaptive systems the final MSE is only part of the story. We have to make sure that the system coefficients have stabilized… It is interesting to note that with batch updates there is no rattling, so in the linear case the batch solution is more appropriate. Observe this in the simulations by displaying the MSE for large and small stepsizes. We are just paying a small price of storing the individual weight updates. For nonlinear systems the batch is unfortunately no longer always superior to the on-line update as we will see.

NeuroSolutions Example

This example shows that obtaining a small MSE is a necessary but not sufficient condition for stable adaptation. *Adaptation also requires that the weights of the model settle onto stable values*. This second condition is required because the system can be endlessly changing its parameters to fit the present sample. This will give always a small MSE, but from a modeling point of view it is a useless solution because *no single model to fit the data set was found*.

6.6. Scheduling the step sizes

As we saw in the latest examples, for fast convergence to the neighborhood of the minimum a large step size is desired. However, the solution with a large step size suffers from rattling. One attractive solution is to use a large learning rate in the beginning of training to move quickly towards the location of the optimum weights but then the learning rate should be decreased to obtain good accuracy on the final weight values. This is called learning rate scheduling . This simple idea can be implemented with a variable step size controlled by

$$\eta(n+1) = \eta(n) - \beta \qquad \textbf{Equation 21}$$

where $\eta(0)=\eta_0$ is the initial step size, and β is a small constant. Note that the step size is being linearly decreased at each iteration. If one has control of the number of iterations we can start with a large step size and decrease it to practically zero towards the end of training. The value of β needs to be experimentally determined. Alternatively, one can decrease slowly (in optimization this slow decrease is called annealing) the step size using either a linear, geometric, or logarithmic rule.

more on scheduling stepsizes

NeuroSolutions 14

1.14 Scheduling of stepsizes

In this demonstration we will use the scheduling component we used previously (to vary the weights and show the performance surface) to vary the step size. The scheduler is a component that takes an initial value from the component beneath it and changes according to a predetermined rule. Here we use the linear rule, and since we want to decrease the stepsize the factor β is negative. We should set a maximum and a minimum value just to make sure that the parameters are always within the range we want. β should be set according to the number of iterations and the initial and final values (Initμ -βN=residualμ).

Here the important parameter is the minimum (the residual stepsize is set at 0.001) because after scheduling we may want to let the system fine tune its parameters to the minimum. However, notice that this implies that the parameter is already in its neighborhood, and this depends upon a lot of unknowns. So if the scheduling is not right the adaptation may stall in positions far from the minimum.

You should explore the breadboard by entering other values for b and the final value and see their impact on the final weight value. You can also bring the exponential or the logarithmic schedulers and see how they behave. Which one do you prefer for this case?

NeuroSolutions Example

Goto Next Section

7. Regression for multiple variables

Assume that *d* is now a function of several inputs x_1, x_2,...x_p (independent variables), and the goal is to find the best linear regressor of *d* on all the inputs (Figure 13). For *p=2* this corresponds to fitting a plane through the *N* input samples, or a hyperplane in the general case of *p* dimensions.

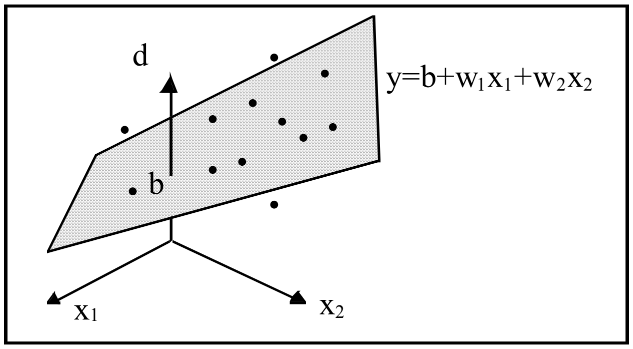


FIGURE 13. **Fitting a regression plane to a set of samples in 2D space.**

As an example, let us assume that we have two variables x_1 (speed) and x_2 (feed rate) that affect the surface roughness (*d*) of a machined workpiece. In abstract units the values of x_1, x_2, *d* for 15 workpieces are presented in Table II.

x1	x2	d
1	2	2
2	5	1
2	3	2
2	2	2
3	4	1
3	5	3
4	6	2
5	5	3
5	6	4
5	7	3
6	8	4
7	6	2
8	4	4
8	9	3
9	8	4

The goal is to find how well one can “explain” the quality of machining by the two variables x_1 and x_2, and which is the most important parameter.

As before, we will assume that the measurements *x* are noise free and that *d* is contaminated by a noise vector ε with some properties (Gaussian distributed with components that are zero mean, equal variance $\sigma 2$ and uncorrelated with the inputs).

The regression equation when p=2 is now:

$$\varepsilon_i = d_i - (b + w_1 x_{i1} + w_2 x_{i2})$$ **Equation 22**

Where x_{i1} is the ith value of x_1 (the ith workpiece in the training set). In the general case, we write the equation as:

$$\varepsilon_i = d_i - \left(b + \sum_{k=i}^{p} w_k x_{ik} \right) = d_i - \sum_{k=0}^{p} w_k x_{ik} \qquad \mathrm{i = 1...N}$$ **Equation 23**

where we made $w_0=b$ and $x_{i0}=1$ (compare with Eq.3). The goal of the regression problem is to find the coefficients *w0, ….wp*. To simplify the notation we will put all these values into a vector **w =** [*w0, ….wp*] that minimizes the MSE of ε_i over the *n* samples. We will use bold letters for vectors. Figure 14 shows that the linear PE now has *p* inputs and one bias.

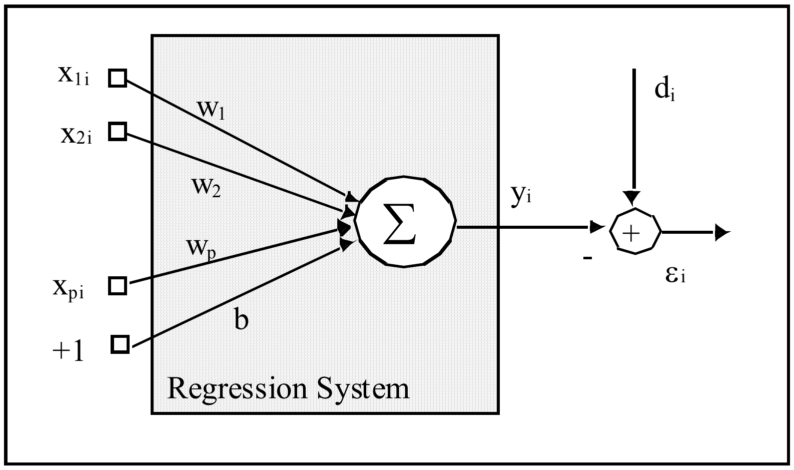


FIGURE 14. **Regression system for multiple inputs**

The mean square error (MSE) becomes for this case

$$J = \frac{1}{2N} \sum_i \left(d_i - \sum_{k=0}^{p} w_{ik} x_{ik} \right)^2$$ **Equation 24**

The solution to the extreme (minimum) of this equation can be found exactly in the same way as before, i.e. by taking the derivatives of *J* with respect to the unknowns (w_k), and equating the result to zero. derivation of normal equations

This solution is the famous *normal matrix equation*

$$\sum_i x_{ij} d_i = \sum_{k=0}^{p} w_k \sum_i x_{ik} x_{ij} \qquad \mathrm{j} = 0,1,\ldots \mathrm{p}$$

Equation 25

The normal equations can be written much more compactly with matrix notation (see the Appendix). Let us define

$$R_{kj} = \frac{1}{N} \sum_i x_{ik} x_{ij}$$

Equation 26

as the autocorrelation of the input samples for indices k, j. As you can see the autocorrelation measures similarity across the samples of the training set. When k=j, R is just the sum of the squares of the input samples (the variance in the data). When k differs from j, R measures the sum of the crossproducts for every possible combination of the indices. As we did for **w**, we will also put all these **R**kj values into a matrix **R**, i.e.

$$\boldsymbol{R} = \begin{bmatrix} \boldsymbol{R}_{00} & \ldots & \boldsymbol{R}_{0p} \\ \ldots & \ldots & \ldots \\ \boldsymbol{R}_{p0} & \ldots & \boldsymbol{R}_{pp} \end{bmatrix}$$

. Thus one obtains pairwise information about the structure of the data set.

Let us call

$$P_j = \frac{1}{N} \sum_i x_{ij} d_i$$

Equation 27

the crosscorrelation of the input *x* for index *j* and desired response *d*, which can be also put into a vector **p** of dimension *p+1*. As we can expect, Pj measures the similarity between the input *x* and the desired response *d* at shift j. Substituting these definitions in Eq.25 , the set of normal equations can be written simply

$$\mathbf{p} = \mathbf{R}\mathbf{w}^* \quad \text{or} \quad \mathbf{w}^* = \mathbf{R}^{-1}\mathbf{p}$$

Equation 28

where **w** is a vector with the *p+1* weights w_i. **w*** represents the value of the p+1 weights for the optimum (minimum) solution. **R**-1 denotes the inverse of the autocorrelation matrix (see Appendix). Eq. 28 states that the solution of the multiple regression problem can be computed analytically as the product of the inverse of the autocorrelation of the input samples multiplied by the crosscorrelation vector of the input and the desired response. The least square solution for this problem yields

$$y = 1.353480 + 0.286191x_1 - 0.004195x_2$$

It is remarkable that we are able to write an equation that describes the relationship between the two variables when only measured data samples were given. This attests the power of linear regression. But as for the single variable case, we still do not know how accurately the equation fits the data, i.e. how much of the variance of the input is actually captured by the regression model. The multiple correlation coefficient r*m* can also be defined in the multiple dimensional case for a single output, as multiple variable correlation coefficient

$$r_m = \sqrt{\frac{\mathbf{w}^{*T}\mathbf{U}_x\mathbf{d} - N\bar{d}^2}{\mathbf{d}^T\mathbf{d} - N\bar{d}^2}}$$

Equation 29

and measures the amount of variation explained by the linear regression, normalized by the variance of **d**. In this expression **d** is the vector built from the desired responses d*i*, and **U** is a matrix whose columns are the input data vectors. For this case r*m*=0.68, so there is a large portion of the variability that is not explained by the linear regression (either the process is nonlinear, or there are more variables involved). We still can approximate the correlation coefficient for the multiple regression case by Eq. 14 after the system has adapted.

NeuroSolutions 15

1.15 Multivariable regression

Moving to multiple dimensional inputs is very simple in NeuroSolutions. You simply change the input and desired files (for the new input data) and change the

input axon to accept two inputs. The rest is automatic. In this example, we will do all this for you using macros. Note that in the two dimensional case the regression line is now a regression plane. There is currently not a good way of showing a plane in three dimensions in NeuroSolutions so we will not have our regression line plot. When we run the network, we will see that the learning curve (one of our only indications of whether the network is training correctly) decreases steadily and that the weights eventually approach the theoretical optimum weights.

The amazing thing about the adaptive system's methodology is that we changed the problem, but the solution did not change that much. It is true that we have to dimension the system properly, choose new values for the stepsize, but the fundamental aspects of the methodology did not change at all....

NeuroSolutions Example

7.1. Setting the problem as a search procedure

All the concepts previously mentioned for linear regression can be extended to the multiple regression case. The performance surface concept can be extended to *p* dimensions, making *J* a paraboloid in *p+1* dimensions, facing upwards (Figure 15 depicts the two weight case). *J* involves now matrix computations, but it remains a scalar qunatity that is a quadratic function of the weights

$$J = \left[0.5\mathbf{w}^T\mathbf{R}\mathbf{w} - \mathbf{p}^T\mathbf{w} + \sum_i \frac{d_i^2}{2N} \right]$$

Equation 30

where the superscript T means the transpose.

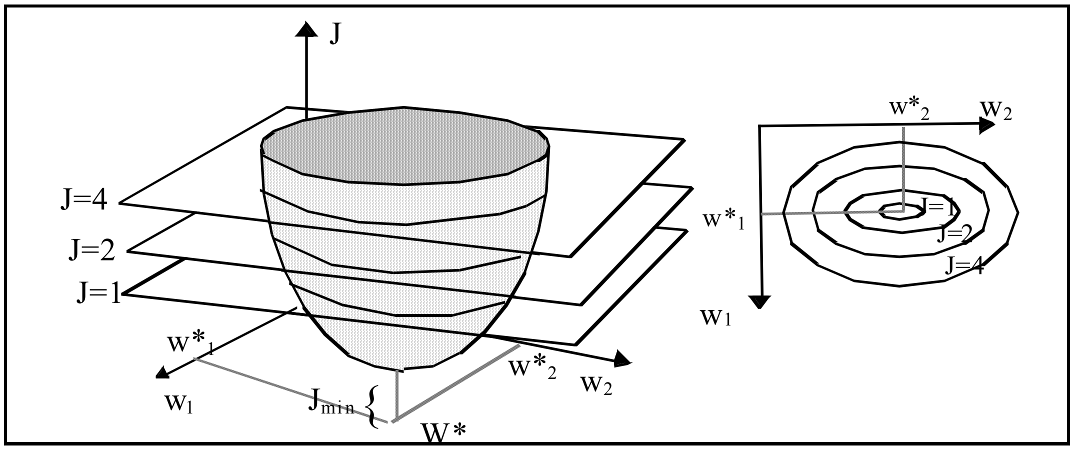


FIGURE 15. **The performance surface for 2 dimensions and its contour plot.**

The coefficients that minimize the solution are

$$\nabla J = 0 = \mathbf{Rw}^* - \mathbf{p} \quad \text{or} \quad \mathbf{w}^* = \mathbf{R}^{-1}\mathbf{p}$$ **Equation 31**

which gives exactly the same solution as Eq.28 Derivation of Optimal Solution . In the space (w_1,w_2), J is a parabola facing upwards. performance surface properties

Summarizing, the autocorrelation of the input ® completely specifies the shape of the performance surface Eq.69 . However, the location of the performance surface in the space of the weights Eq.31 and its minimum value Eq.68 depend also on the desired response.

NeuroSolutions 16

1.16 Checking the LMS solution with the optimal weights

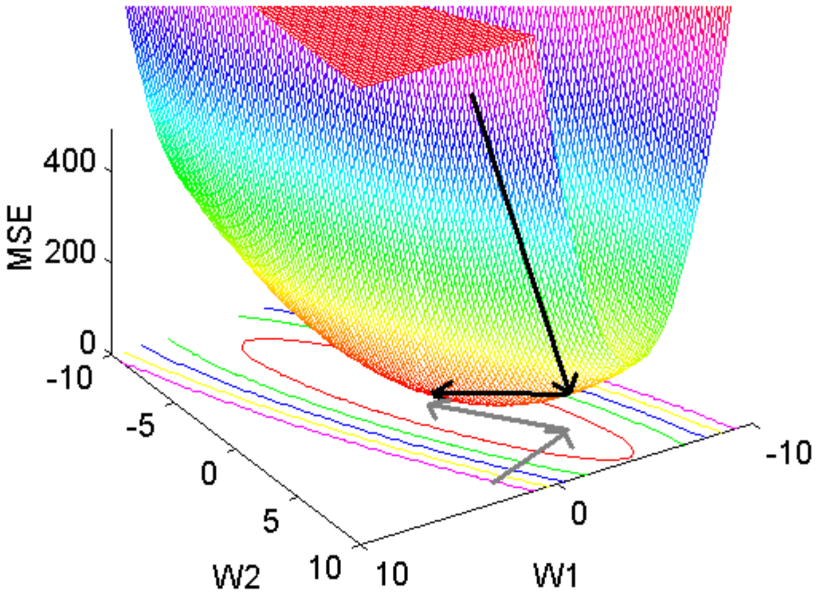
Let us consider first a least square solution with only two weights w_1, and w_2, since we can still compute it easily by hand. For the data set of Table II, the autocorrelation matrix is Eq.26 Eq.63

$$R = \frac{1}{15}\begin{bmatrix} 416 & 429 \\ 429 & 490 \end{bmatrix}$$

To determine the eigenvalues, we solve the equation

$$\det[R - \lambda I] = 0$$

which yields λ_1=59 and λ_2=1.5. From these results we can immediately see that the eigenvalue spread is roughly 40, so the performance surface paraboloid is very skewed (i.e. much narrower in one direction). The performance surface is shown in the following figure. Notice how it is very steep in one direction and very shallow in the other. Thus, if we train the network with gradient descent, we would expect it to move very quickly down the steep slope at first and then move slowly down the valley towards the optimum.



To compute the optimum solution, we first need to compute the crosscorrelation vector Eq.27 , Eq.65 is

$$P = \frac{1}{15}\begin{bmatrix} 212 \\ 229 \end{bmatrix}$$

For the two dimensional case it is still easy to solve for w_1 and w_2, by writing Eq.28

$$\begin{cases} 416w_1 + 429w_2 = 212 \\ 429w_1 + 490w_2 = 229 \end{cases}$$

which gives for optimal weights w_1=0.2848 and w_2=0.2180. The minimum J is 0.390 Eq.68 . When we run the simulator with the BiasAxon substituted by the Axon (no bias), the network weights will eventually approach the optimum.

NeuroSolutions Example

7.2. Steepest descent for multiple weights

Gradient techniques can also be used to find the minimum of the performance surface, but now the gradient is a vector with p+1 components

$$\nabla \mathbf{J} = \left[\frac{\partial J}{\partial w_0}, \ldots, \frac{\partial J}{\partial w_p} \right]^T$$

Equation 32

The extension of Eq.11 is

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \eta \nabla \mathbf{J}(k)$$

Equation 33

where all quantities are vectors, i.e. $\mathbf{w}(k) = [w_0(k), \ldots w_p(k)]^T$. In order to calculate the largest step size η, we again rewrite the update equation in the form of

$$\mathbf{w}(k+1) = (\mathbf{I} - \eta \mathbf{R})\mathbf{w}(k) + \eta \mathbf{R}\mathbf{w}^*$$

Equation 34

where **I** is the identity matrix, **R** is the input autocorrelation matrix and $\mathbf{w}^*(k) = [w_0^*(k), \ldots w_p^*(k)]^T$. The *solution of this equation is cross coupled*, i.e. the way **w** converges to **w*** depends on the behavior of the geometric progression in all the *p+1* directions. So, the simple picture of having **w**(k+1) converge to **w*** with a single geometric ratio as in the unidimensional case has to be modified. One can show that the weights converge with different time constants, each related to an eigenvalue of **R**.

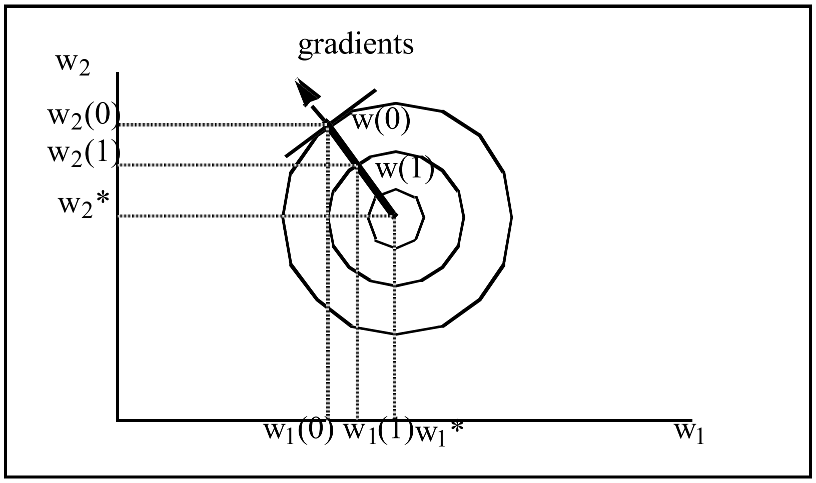
convergence for multiple weights case

7.3 Stepzise Control

As we have seen, the set of values taken by the weight during adaptation is called the weight track. The weight moves in the opposite direction of the gradient at each point, so the weight track depicts the gradient direction at each point of the performance surface visited during adaptation. Therefore, the gradient direction tells us about the performance surface shape. In particular we can construct the contour plot of J since the gradient has to be perpendicular to the lines that link points with the same *J* value. It is important to provide a graphical construction for the gradient at each point assuming we know the contour plot.

Given a point in a contour, we take the tangent of the contour at the point. The gradient is perpendicular to the tangent, so the weights will move along the gradient line and pointing in the opposite direction. Likewise if we run the adaptation algorithm with several initial conditions and we record the value of J at each point, we can determine the contour plots by taking ellipses that pass through the points of equal cost and are perpendicular to the weight tracks.

When the eigenvalues of R are the same (see Appendix), the contour plots are circular and the gradient always points to the center, i.e. to the minimum. In this case the gradient descent only has a single time constant as in the 1-D case. But this is an exceptional condition. In general the eigenvalues of R will be different. When the eigenvalues are different, the weight track bends because it follows the direction of the gradient at each point, which is perpendicular to the contours (Figure 17). So the gradient direction does not point to the minimum, which means that the weight tracks will not be straight lines to the minimum. The adaptation will take longer for two reasons: first a longer path to the minimum will be taken. Secondly, the stepsize must be decreased compared with the circular case. Let us address the stepsize aspect further.



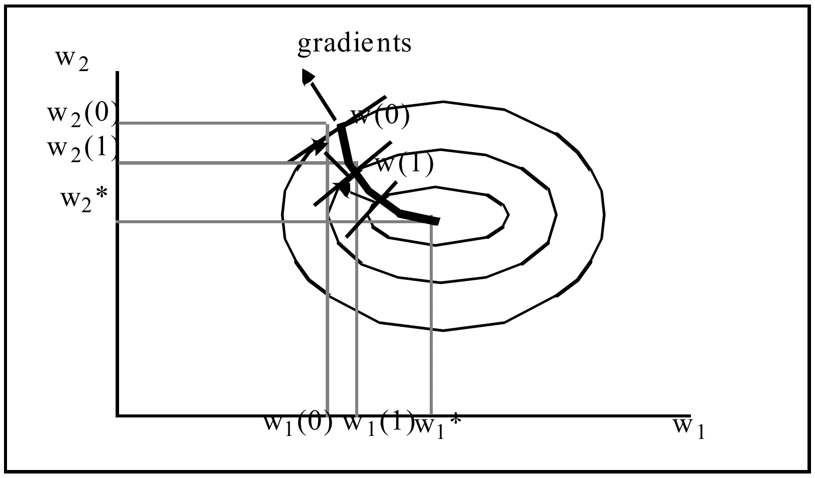


FIGURE 17. **Weight track towards the minimum. First is the case of equal eigenvalues.**

For guaranteed convergence, the learning rate in each principal direction of the performance surface must be

$$0 < \eta < \frac{2}{\lambda_i}$$ **Equation 35**

where λ_i is the corresponding eigenvalue. The worst case condition to guarantee convergence to the optimum **w*** in all directions is therefore,

$$\eta < \frac{2}{\lambda_{\max}}$$ **Equation 36**

i.e., the step size η must be smaller than the inverse of the largest eigenvalue of the correlation matrix. Otherwise the iteration will diverge in one (or more) directions. Since the adaptation is coupled, divergence in one direction will cause the entire system to diverge.

In the early stages of adaptation, the convergence is primarily along the direction of the largest eigenvalue since the weight update along this direction will be bigger. On the other hand, towards the end of adaptation, the algorithm will adapt basically only the weight associated with the smallest eigenvalue (which correspond to the smallest time constant). The time constant of adaptation is therefore

$$\tau = \frac{1}{\eta\lambda_{\min}}$$

Equation 37

An implication of this analysis is that when the eigenvalue spread of **R** is large, there will be very different time constants of adaptation in each direction. This reasoning gives a clear picture of the fundamental constraint of adapting the weights using gradient descent with a single step size η: *the speed of adaptation is controlled by the smallest eigenvalue, while the largest step size is constrained by the inverse of the largest eigenvalue.* This means that if the eigenvalue spread is large, the convergence will be intrinsically slow. There is no way around it when only a single stepsize is used in the steepest descent.

The learning curve will approach J_{min} in a geometric progression as before. However, there will be *many different time constants of adaptation, one per each direction*. Initially the learning curve will decrease at the rate of the largest eigenvalue, but towards the end of adaptation the rate of decrease of *J* is controlled by the time constant of the smallest eigenvalue.estimation of eigenvalue spread

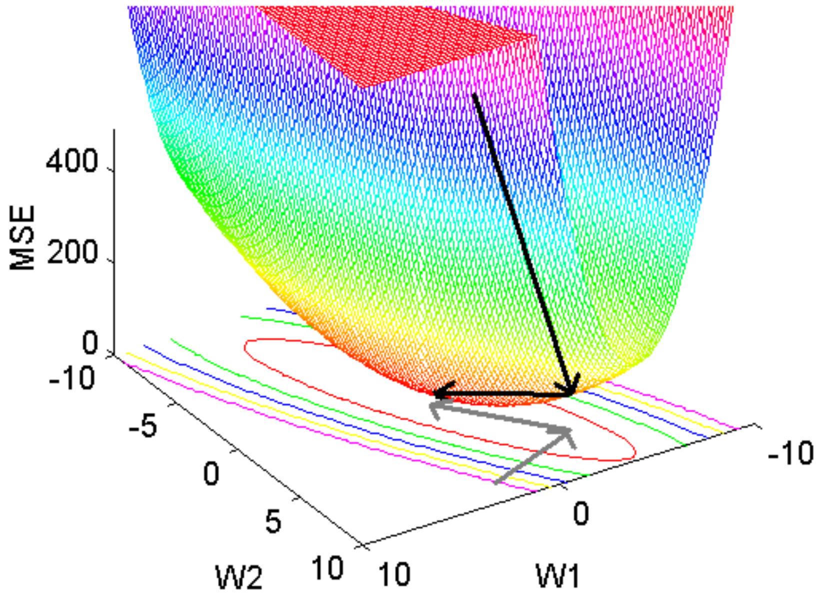
NeuroSolutions 17

1.17 Visualizing the weight tracks and speed of adaptation

According to our previous calculations, the largest stepsize for convergence is Eq.36 3.3e-2. The critically damped mode along the largest eigenvector should be 1.6e-2. The time constant of adaptation for the largest stepsize is around 20

iterations (epochs for batch), i.e. the convergence should take 80 epochs with this stepsize.

When we run the simulations, the algorithm converges first along the direction of the largest eigenvalue (largest eigenvector direction), and then along the direction of the smallest eigenvector. Since the eigenvalue spread is 40, the steps are much bigger along the largest eigenvector direction. If we look at the figure below, we can see that the weights converge perpendicular to the contour plots since this is the steepest descent path. As we will see, there are two distinct regions in the learning curve: in the beginning it is controlled by the geometric ratio along the largest eigenvector, while towards the end it is controlled by the geometric ratio of the smallest eigenvector.



After running this example and observing the weight tracks let us change the input data file such that the eigenvalue spread is smaller. Mouse down on the input file icon and bring up its inspector by clicking the mouse right button. Remove the present input file, and add the file regression2a.asc from the NSBook/chapter1/NS30Examples/1. 17 MR weight tracks folder.

The modification was only made in the variable x2, all the rest is the same. Respond to the panel Associate by clicking on the close button. In the Costumize panel skip the desired signal, and click on close. You have just modified the input data to this example. This new file has a much smaller eigenvalue spread, so we can expect that the weight tracks are basically straight lines to the minimum. Compute the new eigenvalue spread, and adjust the learning rates such that the convergence is as fast as possible.

NeuroSolutions Example

7.4. The LMS algorithm for multiple weights

It is straight forward to extend the gradient estimation given by the LMS algorithm from one dimension to many dimensions. We just apply the instantaneous gradient estimate Eq.12 to each element of Eq.33 . The LMS for multiple dimensions reads

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta\varepsilon(k)\mathbf{x}(k)$$ **Equation 38**

What is interesting is that the LMS *adaptation rule still uses local computations*, i.e. we can write for the ith weight

$$w_i(k+1) = w_i(k) + \eta\varepsilon(k)x_i(k)$$ **Equation 39**

Note that although the analysis of the gradient descent techniques is complex, the LMS algorithm itself is still very simple. This is one reason why the LMS is so widely used. But, since the LMS is a steepest descent algorithm, the analysis and discussions concerning the largest step size for convergence and coupling of modes also apply to the LMS algorithm.

NeuroSolutions 18

1.18 Visualizing weight tracks with on-line learning

In this example, we will switch the backprop controller to on-line learning to implement the LMS algorithm. Notice that the weight tracks follow basically the same path as before, but now the path is much more irregular due to the sample by sample update of the weights. When the eigenvalue spread is very large (the performance surface is very steep in one direction and shallow in others), the problem is difficult for LMS to solve. Any small perturbation in the smallest eigenvector direction gets amplified by the large eigenvalue spread.

NeuroSolutions Example

7.5. Multiple regression with bias

Up to now we have implemented and solved analytically the multiple regression problem without bias. The reason for that is only based on simplicity. With two weights we can still easily solve the multiple regression case by hand, however if the bias is added, we must do the computations with three parameters. The simulations are transparent to these difficulties since one just substitutes the Axon by a BiasAxon. Note that the largest stepsize between the two cases will differ since the input data was effectively changed if one interprets the bias as a weight connected to an extra constant input of one. Hence the autocorrelation function changed, and likewise its eigenvalue spread.

We should state that the use of a bias is called the full least square solution and it is the recommended way to apply least squares. The reason can be understood easily: when a bias is utilized in the PE the regression line is not restricted to pass through the origin of the space, and normally smaller errors are achieved. There are two equivalent ways to set up the full least squares solution for N input variables:

- • The input and desired responses need to be modified such that they become zero mean variables (this is called the deviation or z scores). In this case a N weight regression will effectively solve the original problem. The bias b is computed indirectly by

$$b = \bar{d} - \sum_{i=1}^{N} w_i \bar{x}_i$$

where w*i* are the optimal weights and the bars represent mean values.

- • Alternatively, the input matrix has to be extended with an extra column of 1s (the first column). This transforms **R** into a (N+1)x(N+1) matrix, which introduces an N+1 weight in the solution (the bias).

NeuroSolutions 19

1.19 Linear regression without bias

We will now substitute the a BiasAxon for the Axon in the previous breadboard. This will effectively provide the regression solution without constraining the regression plane to pass through the origin. We see that the weight tracks are very similar in the beginning but that the error continues to drop, and the weights advancing towards the w1=0 line. This means that the optimal solution changed. We now have a better solution than before, but with increased complexity of the performance surface (4 dimensional instead of 3) and an increased number of adjustable parameters in our system (2 weights and a bias).

NeuroSolutions Example

7.6. The LMS algorithm in practice

One can use some rules of thumb to choose the step size in the LMS algorithm. The step size should be normalized by the variance of the input data estimated by the trace of **R**.

$$\eta = \frac{\eta_0}{tr(\mathbf{R})}$$ **Equation 40**

where η_0= 0.5 to 0.01. This normalization by the input variance was the original rule proposed by Widrow to adapt the adaline. We can expect the algorithm to converge in a number of iterations k given by

$$k \approx \frac{1}{4\eta\lambda_{\min}}$$ **Equation 41**

The LMS algorithm has a misadjustment that is basically the trace of **R** times η

$$M = \eta tr(R)$$ **Equation 42**

So with the LMS algorithm, selecting η such that it produces 10% misadjustment means a training duration in iterations of 10 times the number of inputs.

Go to next section

8. Newton's method

If you are familiar with numerical analysis, you may be asking why aren't we using Newton's method for the search? Newton's method is known to find the roots of quadratic equations in one iteration. The minimum of the performance surface can be equated to finding the root of the gradient equation Eq.32 , as is outlined by Eq.31 . Hence Newton's method can also be used in search. The adaptive weight equation using the Newton's method is Newton's Derivation

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \mathbf{R}^{-1}\nabla\mathbf{J}(k)$$ **Equation 43**

Comparing with Eq.33 note that the gradient information is weighted by the inverse of the correlation matrix of the input, and η is equal to one. This means that Newton's method corrects the direction of the search such that it *always points to the minimum*, while the gradient descent points to the maximum direction of change. These two directions may or may not coincide (Figure 18).

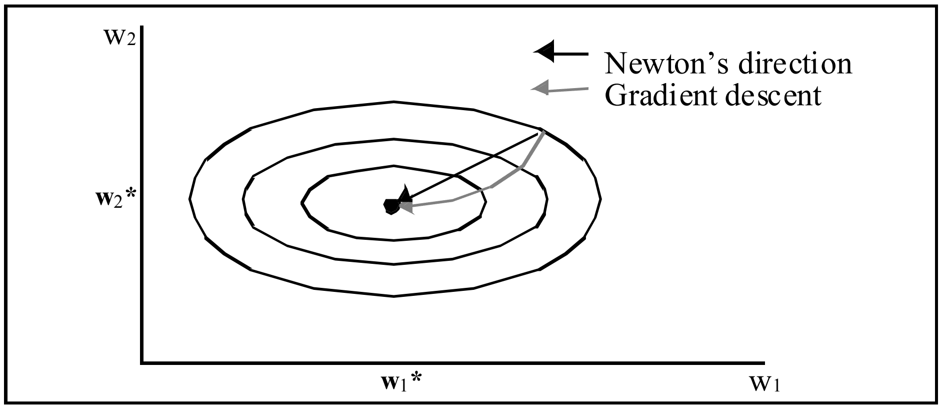


FIGURE 18. **Directions of the steepest descent and newton's method**

They coincide when the contour plots are circles, i.e. when the largest and the smallest eigenvalue of the correlation matrix are the same. When the ratio of the largest to the smallest eigenvalue (*the eigenvalue spread*) increases, the slope of the performance surface in the two direction differs more and more. So for large eigenvalue spreads, the optimization path taken by gradient descent is normally much longer than the path taken by Newton's method. This implies that Newton's method will be faster than LMS when the input data correlation matrix has a large eigenvalue spread.

Another advantage of Newton's method versus the steepest descent is in terms of geometric ratios or time constant of adaptation. When the gradient is multiplied by $\mathbf{R}^{-1}$ not only the direction of the gradient is being changed but also *the different eigenvalues in each direction are being equalized*. What this means is that Newton's method is correcting automatically the time constant of adaptation for each direction such that *all the weights converge at the same rate*. Hence, Newton's method has a *single time constant of adaptation*, unlike the steepest descent method.

These advantages of the Newton's method should not come as a surprise, because Newton's method uses much more information about the performance surface (the curvature). In fact, to implement Newton's method one needs to compute the inverse of the correlation matrix, which takes significantly longer than the single multiplication required by the LMS method and also requires global information. Newton's method is also brittle, i.e. if the surface is not exactly quadratic, the method may diverge. This is the reason Newton's method is normally modified to have also a small step size η instead of using η=1 as in Eq.40 .

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \mathbf{R}^{-1}\varepsilon(k)\mathbf{x}(k)$$ **Equation 44**

Note that **x**(k) is a vector and $\mathbf{R}^{-1}$ is a matrix, so the update for one weight influences all the other inputs in the system. This is the reason the computations are no longer local to each weight. However they are not difficult if one assumes that the inverse of **R** is known a priori. The case where $\mathbf{R}^{-1}$ has to be estimated on-line is much more involved and leads to the recursive least squares (RLS) algorithm.

Alternatively, to improve convergence speed with the LMS, we can implement an orthogonalizing transformation of the input correlation function followed by an equalization of the eigenvalues which is called a whitening transformation. (see Appendix). Since the Newton's method coincides with the steepest descent for performance surfaces that are symmetric, this preprocessing will make the LMS perform as Newton.

NeuroSolutions 20

1.20 Newton's method

In this example, we implement Newton's method with a custom DLL. For this example, we must compute $\mathbf{R}^{-1}$ and apply Eq.41 to the simulator. The autocorrelation function for this example is

$$R = \frac{1}{15}\begin{bmatrix} 416 & 429 \\ 429 & 490 \end{bmatrix}$$

so $\mathbf{R}^{-1}$ becomes (see Appendix)

$$R^{-1} = 15\begin{bmatrix} 0.0247 & -0.0217 \\ -0.0217 & 0.0210 \end{bmatrix}$$

By applying Newton's method to the learning algorithm, we have essentially compensated the eigenvalue spread. This means that the Newton's method behaves as the steepest descent for a circular performance surface where the steepest descent direction always points directly to the optimal value. Thus, although the calculations are more complicated and more demanding (we need to know $\mathbf{R}^{-1}$), the convergence is much faster (in fact, you can converge in one epoch!). When we run the simulator, notice that no matter where we start, we always head directly towards the optimum.

<u>NeuroSolutions Example</u>

.

Go to next section

9. Analytic versus Iterative solutions

Selecting a search procedure to find the optimal weights is a drastic conceptual change from the analytic least square solution, albeit equivalent. In learning systems the iterative solution is the most utilized for several reasons:

When working with learning systems the interest is very often in on-line solutions, i.e. solutions that can be implemented sample by sample. The analytic solution requires data to be available before hand to compute the correlation matrix **R** and crosscorrelation vector **p**. Fast computers are required to crank out the solution (inverse of **R** and product with **p**). The method produces a value that immediately gives the best possible performance. But several problems may surface when applying the analytic approach, because if the matrix **R** is ill-conditioned , the computation of $\mathbf{R}^{-1}$ may not be very accurate. Moreover, the analytic solution also requires lots of computation time (computation of a matrix inverse is proportional to the square of the number of columns N of the matrix. In the big O notation this means $O(N^2)$).

The iterative solution is not free from shortcomings. We already saw that there is no guarantee that the solution is close to the optimal weight **w*** when all the input samples are used by the algorithm. This depends on the data and upon a judicious selection of the step size η. The accuracy of the iterative solution is not directly dependent upon the condition number of **R**, but matrices with large eigenvalue spread produce slow convergence because the gradient descent adaptation is coupled. As we said previously, the slowest mode controls the speed of adaptation, while the largest stepsize is constrained by the largest eigenvalue.

The great appeal of the iterative approach to optimization is that *very efficient algorithms exist* to estimate the gradient (e.g. the LMS algorithm). Only two multiplications per weight are necessary, so the computation scales proportional to the number of weights N (i.e. O(N) time). Moreover, the *method can be readily extended to nonlinear systems*, while the analytic approach for most of the cases of practical relevance can not be computed.

Go to the next section

10. The Linear Regression Model

We started this chapter by pointing out the advantages of building models from experimental data. In the previous sections we developed a set of techniques that adapt the parameters of a linear system (the adaline) to fit as well as possible the relationship between the input (*x*) and the desired data (*d*). This is our first model and it "explains" the relationship *f(x,d)* as a hyperplane that minimizes the square distance of the residuals.

We will have the opportunity to study other (nonlinear) models in later chapters.

It is instructive to stop and ask the question: How can we use the newly developed regression model? One interesting aspect of model building that we mentioned previously is the ability to *predict the behavior of the experimental system*. Basically what this means is that once the adaline is trained, we can "forecast" the value of *d* when *x* is available. We do this by computing the adaline output *y* and assume that the error ε is small (Eq.3). You can now understand why we want to minimize the square of the error, since if the square of the error is small than *d* is going to be close to *y* in the training data. Figure 19 shows a productive way of looking at the input-output pairs that we used to train the adaline.

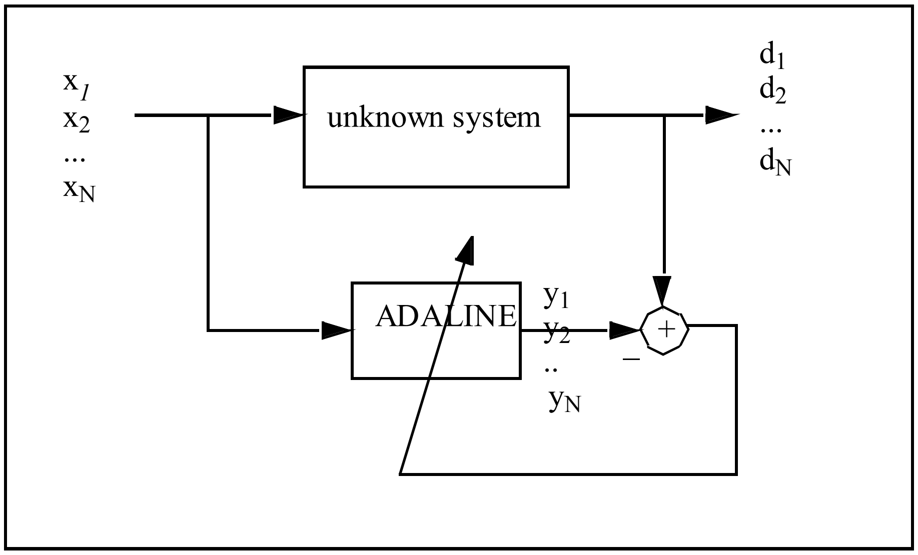


Figure 19. A view of the desired response as the output of an unknown model

We assume that the experimental system produces the desired response *d* for each input *x* according to a rule that we do not know. The purpose of building the model is to approximate as well as possible this hidden relationship.

We expect also that even for *x* values that the system did not use for training, *y* is going to be close to the corresponding unknown value *d*. Our intuition tells us that if:

- the data used for training covered well all the possible cases,
- if we had enough training data,

- and the correlation coefficient is close to one,

then in fact *y* should be close to the unknown value *d*. However, this is an inductive principle, which has no guarantee of being true. The ability to extrapolate the good performance from the training set to the test set is called *generalization*. Generalization is a central issue in the adaptive systems' approach since it is the only guarantee that the model will perform well in the future data that will be presented to the system while in operation.

Remember that in the test mode the system parameters must be kept constant, i.e. the learning algorithm MUST be disabled. In the next section we will familiarize ourselves with training and using the linear model.

10.1 Regression Project

Getting real world data

We will end Chapter I by giving you a flavor of the power of linear regression to solve real life problems. We will go to the World Wide Web and seek real data sets, import them into NeuroSolutions and solve regression problems. We will adopt the breadboard from Example 7.

The first thing is to decide what data we will work with. There are many interesting Web sites to visit in the search for data. We suggest the following sites:

climate data: http://ferret.wrc.noaa.gov/fbin/climate_server

Center for Biomedical Modeling Research (CBMR)

http://www.scs.unr.edu/~cbmr/research/data.html

or Dr. B's WWW Data site

http://seamonkey.ed.asu.edu/~behrens/teach/WWW_data.html

These sites have plenty of data (some duplicated). We assume that you know how to get connected to the Web and how to download data. You should get the data in ASCII and store it in column format with one of the variables (the independent variable) in the input file and the dependent variable in the output file. Alternatively we have provided sample data on the CD-ROM under the Chapter1\data directory. Read the readme file to choose the data sets that interest you.

NeuroSolutions Project

The fundamental question is to find out how well a linear relation “explains” the dependence between the input data and the desired data. We will exemplify the project with a single dimensional set of input data, but the multidimensional case is similar.

The first thing to do is to modify the NeuroSolutions breadboard such that it will be able to work with the data you downloaded. The data should be stored in an ASCII file and formatted in columns. Right click on the input file icon and select properties. The Inspector will appear on the screen. Remove the present file (click the remove button) and click on the add button. The Windows 95 file inspector will appear and you have to open the file that contains the input data, i.e. the input to your linear model.

In NeuroSolutions, the Associate panel appears which you can close (we assume that the input file has ASCII data in column format). The next panel that pops open is the Customize panel. Here you select the columns that you want to use (for those columns that you do not want select the column label and click on the skip button), and then click on the close button. The input file is now open and ready to be used by NeuroSolutions. You should repeat the procedure for the desired file. Make sure that the number of samples of the input and desired files are the same.

Another thing that we should do is to normalize the data. Sometimes the input and desired variables have very different ranges so one should always normalize between 0 (or -1) and 1 both the desired and input data files. To do this go to the Stream page (click on the Stream tab) of the Inspector to access the normalization panel. Click the normalize check box and set the normalization range (don’t forget to go to the DataSet level of the inspector to translate the data again and make the normalization effective).

We always recommend that you visually check the data either with a plotting program or the Scatter plot in NeuroSolutions to ensure that there aren't any outlier present in the data. When outliers exist, they may distort any possible linear relationship that may exist.

Once the data sets are open, we can effectively start the adaptation of the linear regressor. The first important consideration is the largest stepsize that can be used for convergence. When the data is normalized one can always guess an initial value of 0.1. By plotting the learning curve, or the weight tracks (if the problem has few input channels) we can judge how appropriate this value might be. Alternatively we can compute the eigenvalues and find the exact largest possible stepsize, but this is rarely done. The trial and error method is OK for small problems.

If the problem takes a long time to converge and increasing the step size creates instability, then the eigenvalue spread is large, and there is little we can do short of using Newton's method.

After the algorithm converges (the error stabilizes) one should bring the correlation coefficient DLL to estimate the correlation coefficient. Note that it is always possible to pass an hyperplane through some data points, but the real issue is does the hyperplane provide a good model? To answer this question one needs to estimate the correlation coefficient.

For the multiple variable regression case the relative weight magnitude tells us about the relative importance of the each variable in the regression equation. So it is rather important to read the values of the regression weights including the bias. Remember that if the data is normalized, the displayed weight values must be "unnormalized" in order to compare them with the original data. You can find the values NeuroSolutions used to normalize the data by going to the DataSet page of the inspector and opening the normalization file. Neurosolutions multiplies the data set by the first value in the normalization file (range) and adds the second value in the normalization file (offset). To reverse this process, you must subtract the second value and then divide by the first value.

Remember that the parameters of the regression equation can be used to predict desired responses when the input is known. We can do this by testing the system with another data file for which we do not have a desired response. To do this in NeuroSolutions, you should go to the Controller (the yellow dial) Inspector and turn off the “learning” check box (this fixes the weights).

No problem is finalized without a critical assessment of the results obtained. You should start with a hypothesis about the data relationship, and confirm your hypothesis with NeuroSolutions results. If there is a discrepancy between what you expect and the results, you must explain it. This is where the NeuroSolutions probes are very effective. You should verify that the data is being properly read, if the input and output files are synchronized, if the system is converging (weight tracks, learning curve), etc. Computers are great tools, but they are very susceptible to the “garbage-in garbage-out syndrome” so it is the user responsibility to check the inputs and the methodology of data analysis.

NeuroSolutions 21

1.21 Linear regression Project

We will illustrate the project with a regression between two time series, the sea temperature and atmospheric pressure downloaded from the NOAA site (atmospheric data base). We will start with the breadboard from Example 7. We will replace the input with the file containing the sea temperature and desired response data with the file containing the pressure data. NeuroSolutions automatically sets the number of inputs from the file (verify this in the file Inspector), and the number of exemplars in an epoch. Verify this in the Controller Inspector. We also have to decide how many iterations we need. In the Controller Inspector enter 1,000 in the Epochs/Run. This number may be too large, but when the coefficients do not change we can always interrupt the simulation. Experiment with everything we have learned in this chapter.

NeuroSolutions Example

Go to next section

11. Conclusions

In this first Chapter we introduced very important ideas for the rest of the book. Probably the most important was the concept of adaptive systems. Instead of designing the system through specifications, we let the system learn from the input data. In order to achieve this the system has to be augmented with an external cost criterion to measure "goodness of fit" and an algorithm that will adapt the system parameters such the minimum of the cost can be reached. This idea will be with us until the end of the book.

But we covered much more in this chapter. We described an extremely simple and elegant algorithm that is able to minimize the external cost function by using local information available to the system parameters. The principle is to search the performance surface in the opposite direction of the gradient. The name of the algorithm is LMS (least means squares) and in just 2 multiplications per weight and data sample it is able to put the system parameters in the neighborhood of the optimal values. Gradient descent is a powerful concept that we will hear constantly until the end of the book.

When we applied the LMS to the linear network we end up with a system that can fit hyperplanes to data, and which is called the linear regressor. The solution is identical to least squares.

We quantified the properties of the LMS algorithm, and we showed the fundamental trade-off of adaptation: the compromise between speed of adaptation and precision in the final solution. We defined the learning curve, which we called the thermometer of learning. This will also be with us until the end of the book. Therefore, this chapter covers the basic concepts for the intriguing adventure of designing systems that learn directly from data.

We have also provided a project to help you understand the power of adaptive systems. The applications of the adaline are bounded by our imagination and the data we can find to train it. So knowing how to get data from the Web and how to use it in NeuroSolutions is of great value.

NeuroSolutions Examples

1.1 The Linear Processing Element in NeuroSolutions

1.2 Computing the MSE for the linear PE

1.3 Finding the minimum error by trial and error

1.4 Plotting the performance surface

1.5 Comparison of performance curves for different data sets 1.6 Adapting the linear PE with LMS

1.7 Batch versus online adaptation

1.8 Robustness of LMS to noise

1.9 Estimating the correlation coefficient during learning

1.10 The learning curve

1.11 Weight tracks

1.12 Linear regression without bias

1.13 Rattling

1.14 Scheduling of stepsizes

1.15 Multivariable regression

1.16 Checking the LMS solution with the optimal weights

1.17 Visualizing the weight tracks and speed of adaptation

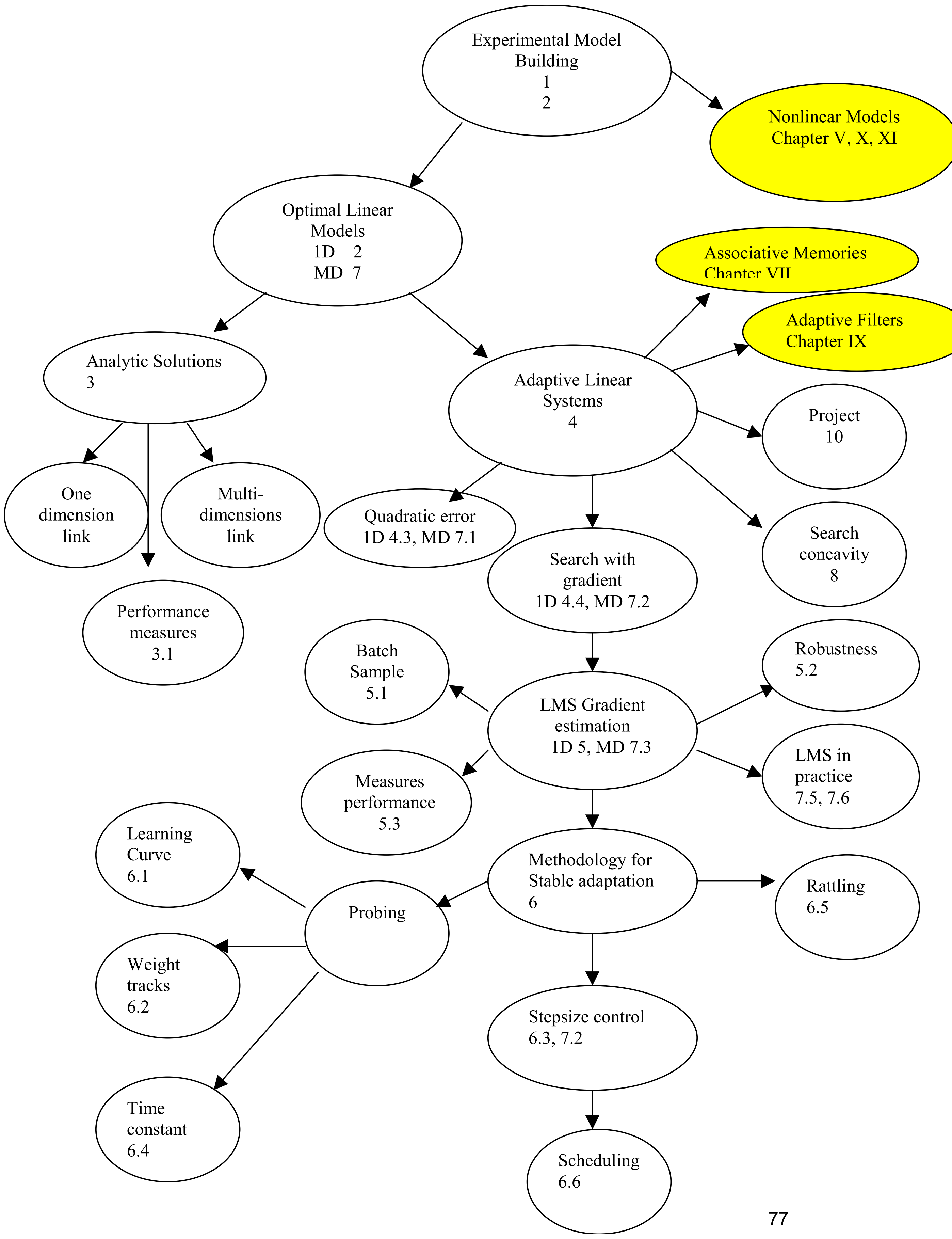
1.18 Visualizing weight tracks with on-line learning

1.19 Linear regression without bias

1.20 Newton's method

1.21 Linear regression Project

Concept Map for Chapter I


Experimental Model Building 1 2
Nonlinear Models Chapter V, X, XI
Optimal Linear Models 1D 2 MD 7
Associative Memories Chapter VII
Adaptive Filters Chapter IX
Analytic Solutions 3
Adaptive Linear Systems 4
Project 10
One dimension link
Multi-dimensions link
Quadratic error 1D 4.3, MD 7.1
Search with gradient 1D 4.4, MD 7.2
Search concavity 8
Performance measures 3.1
Batch Sample 5.1
LMS Gradient estimation 1D 5, MD 7.3
Robustness 5.2
LMS in practice 7.5, 7.6
Measures performance 5.3
Learning Curve 6.1
Methodology for Stable adaptation 6
Probing
Rattling 6.5
Weight tracks 6.2
Stepsize control 6.3, 7.2
Time constant 6.4
Scheduling 6.6

Go to next Chapter

Go to the Table of Contents

Go to the Appendix

End of Chapter 1

This is the End of Chapter 1

Go to the Table of Contents

least squares derivation

From Eq.5 we can work out the derivatives to obtain

$$\sum_{i=1}^{N} d_i = Nb + w\sum_{i=1}^{N} x_i$$
$$\sum_{i=1}^{N} x_i d_i = b\sum_{i=1}^{N} x_i + w\sum_{i=1}^{N} x_i^2$$

Equation 45

We will demonstrate this for the derivative with respect to w, i.e.

$$\frac{\partial J}{\partial w} = \frac{1}{2N}\sum \frac{\partial (d_i - wx_i - b)^2}{\partial w} = \sum \frac{1}{N}(d_i - wx_i - b)x_i = 0$$

Equation 46

which gives the second equation in Eq.6 . The set of Eq.45 is called the normal equations.

The solution of this set of equations is

$$b = \frac{\sum_i x_i^2 \sum_i d_i - \sum_i x_i \sum_i x_i d_i}{\sum_i x_i^2 - \frac{\left(\sum_i x_i\right)^2}{N}} \qquad w = \frac{\sum_i x_i d_i - \frac{\sum_i x_i \sum_i d_i}{N}}{\sum_i x_i^2 - \frac{\left(\sum_i x_i\right)^2}{N}}$$

Equation 47

which provides the coefficients for the *regression line of d on x*. The summations run over the input output data pairs. In order to solve Eq. 45, one just needs to get the value of *b* from the first equation and substitute it in the second equation to obtain *w* as a function of *x* and *d*. Continue by substituting the value of *w* in the first equation to finally obtain *b* as a function of *x* and *d* (variable elimination). It is easy to prove that the regression line passes through the point

$$\left(\frac{\sum_i x_i}{N}, \frac{\sum_i d_i}{N} \right)$$

Equation 48

which is called the centroid of the observations. The denominator of the slope parameter of w and b is the corrected (for the mean) sum of squares of the input.

Return to Text

variance

Data collected from experiments is normally very complex and difficult to describe by few parameters. The mean and the variance are statistical descriptors of data clusters which are normally utilized in such cases.

The mean of N samples is defined as

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

A physical interpretation for the mean is the center of mass of a body made up of samples of the same mass. It is the first moment of the probability density function (pdf).

We can have very different data distributions with the same mean, so the mean is not that powerful descriptor. Another descriptor very often used is the variance, which is defined as

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2$$

The variance is the second moment around the mean and it measures the dispersion of samples around the mean. The square root of the variance is called the standard deviation. Mean and variance are much better descriptors of data clusters. In fact *they define univocally Gaussian distributions*, which are very good models for lots of real world phenomena.

Go back to text

Derivation of correlation coefficient

Note that $d_i - \bar{d} = \left(y_i - \bar{d}\right) + \left(d_i - y_i\right)$ which leads to

$$\sum_i \left(d_i - \bar{d}\right)^2 = \sum_i \left(y_i - \bar{d}\right)^2 + \sum_i \left(d_i - y_i\right)^2$$ **Equation 49**

when the optimal solution is obtained (the cross terms are zero for the optimal solution because the error is orthogonal to the output y).

The first term measures the dispersion (square difference) of the predicted values with respect to the mean, while the second term measures the mismatch between the observed values and the result of the regression. Hence, the first term measures the dispersion contained in the regression model, and the second measures the dispersion that was not modeled by the linear model (the variance of ε). If we normalize the first term of the equation by the variance of d we get an index of how much the variability of d is captured by the regression model,

$$r^2 = \frac{\sum_i (y_i - \bar{d})^2}{\sum_i (d_i - \bar{d})^2}$$

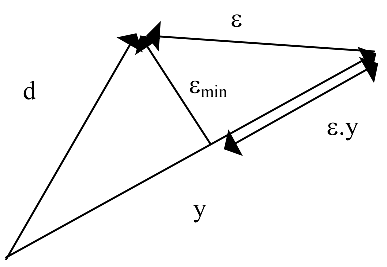
Equation 50

We can now substitute the regression equation for y=w*x and the definition of $\bar{d}$ and w* to obtain the equation in the text.

Return to text

computation of correlation coefficient

The important thing to note is that with optimal coefficients the error samples interpreted as a vector is perpendicular to the adaline output *y*. This condition is called the orthogonality condition. In fact from the figure below it is easy to see that the smallest error is obtained when the projection of *d* on *y* is the orthogonal projection.



During adaptation the error will always be larger than ε_{min}, meaning that *y* can be larger than *d*. So Eq.7 may be larger than one, which is misleading since |r|<1. Using the fact that the minimum error is perpendicular to *y*, one can compute the dot product of ε with *y* and subtract it to the numerator of Eq. 7. We can prove that the numerator is always smaller than *d* and the extra term is zero at the optimal solution, so will not affect the final value of the correlation coefficient. This is exactly what is done in Eq. 14 .

Return to text

batch versus online learning

The on-line and batch modes are equivalent for parabolic performance surfaces.

Note that the number of weight updates of the two methods for the same number of data presentations is very different. The on-line method (LMS) does an update each sample, while batch does an update each epoch, i.e.

LMS updates =(batch updates) x(# samples in training set).

Return to text

more derivation of performance surface

So, for a quadratic performance surface Eq.9 , computing the gradient and equating it to zero finds the value of the coefficients that minimize the cost, i.e.

$$\nabla J = \frac{\partial J}{\partial w} = 0 = \frac{1}{N}\left(-\sum_i d_i x_i + w\sum_i x_i^2 \right)$$

Equation 51

or

$$w^* = \frac{\sum_i x_i d_i}{\sum_i x_i^2}$$

Equation 52

This solution is fundamentally the same as found in Eq.6 (*b=0* is equivalent to assuming that the average value of *x* and *d* are zero). So, the very important observation is that the analytical solution found by the least squares coincides with the minimum of the performance surface. Substituting this value of *w** into Eq.9 , the minimum value of the error becomes

$$J_{\min} = \frac{1}{2N}\left[\sum_i d_i^2 - \frac{\left(\sum_i d_i x_i\right)^2}{\sum_i x_i^2}\right]$$

Equation 53

Eq.9 can be re-written in the form

$$J = J_{\min} + \frac{1}{2N}(w - w^*)\sum_i x_i^2 (w - w^*)$$

Equation 54

To verify this just operate Eq. 54 and substitute Eq. 52 for w* and Eq. 53 for $J_{\min}$. This is another important conclusion. Notice that:

- the minimum value of the error $J_{\min}$ Eq.53 depends on both the input signal (x_i), and the desired signal (d_i)
- the location in coefficient space where the minimum *w** occurs Eq.52 also depends on both x_i, d_i.
- the performance surface shape Eq.54 depends only on the input signal (x_i)

Return to text

more on derivation of largest stepsize

The best way to find the upper bound for η is to write the equation that produces the weight values. Let us rewrite the ideal performance surface Eq.54 as

$$J = J_{\min} + \frac{\lambda}{2}(w - w^*)^2$$

Equation 55

where

$$\lambda = \frac{1}{N}\sum_i x_i^2$$

Equation 56

By computing the gradient of J Eq.55 , we get

$$\nabla J = \lambda(w - w^*)$$ **Equation 57**

so the iteration that produces the weight updates Eq.11 can be written as

$$w(k+1) = (1 - \eta\lambda)w(k) + \eta\lambda w^*$$

Equation 58

This is a first order linear constant coefficient difference equation which can be solved by induction. Start with a solution w(0).

$$w(1) = (1 - \eta\lambda)w(0) + 2\eta\lambda w^*$$
$$w(2) = (1 - \eta\lambda)^2 w(0) + 2\eta\lambda w^*[(1 - \eta\lambda) + 1]$$
$$w(3) = (1 - \eta\lambda)^3 w(0) + 2\eta\lambda w^*[(1 - \eta\lambda)^2 + (1 - \eta\lambda) + 1]$$

which provides by induction the equation

$$w(k) = (1 - \eta\lambda)^k w(0) + \eta\lambda w^* \sum_{n=0}^{k-1} (1 - \eta\lambda)^n = (1 - \eta\lambda)^k w(0) + \eta\lambda w^* \frac{1 - (1 - \eta\lambda)^k}{1 - (1 - \eta\lambda)}$$

This equation can be rewritten as in the text.

Return to text

derivation of the time constant of adaptation

Writing exp(-1/τ)= ρ and expanding the exponential in Taylor series,

$$\rho = \exp(-\frac{1}{\tau}) = 1 - \frac{1}{\tau} + \frac{1}{2!\tau^2} - ....$$

we get approximately ρ~1-1/τ. We saw that geometric ratio of the gradient descent is Eq.17 so we get

$$\tau = \frac{1}{\eta\lambda}$$

Return to text

more on scheduling stepsizes

If the initial value of η_0 is set too high, the learning can diverge. The selection of β can be even tricker than the selection of η because it highly dependent on the performance surface. If β is too large, the weights may not move quickly enough to the minimum and the adaptation may stall. If β is too small, then the search may reach the global minimum quickly and must wait a long time before the learning rate decreases enough to minimize the rattling.   There are other (more automatic) methods for adapting the learning rate which we will discuss later in the book.

Return to text

derivation of normal equations

When the derivative of J with respect to the unknown quantities (the weights) is taken, we end up with a set of *p+1* equations in *p+1* unknowns

$$\frac{\partial J}{\partial w_j} = -\frac{1}{N}\sum_i x_{ij}\left(d_i - \sum_{k=0}^{p} w_k x_{ik}\right) = 0 \qquad \text{for j} = 0\ldots\text{p}$$

Equation 59

Notice that these equations are linear in the unknowns (the w_i), so they can be easily solved. The solution is the famous normal matrix equation

$$\sum_i x_{ij} d_i = \sum_{k=0}^{p} w_k \sum_i x_{ik} x_{ij} \qquad \text{j} = 0,1,\ldots\text{p}$$

Equation 60

or expanding

$$\sum_i x_{i0} d_i = \sum_k w_k \sum_i x_{ik} x_{i0}$$
$$\sum_i x_{i1} d_i = \sum_k w_k \sum_i x_{ik} x_{i1}$$
$$\sum_i x_{ip} d_i = \sum_k w_k \sum_i x_{ik} x_{ip}$$

Equation 61

Let us define

$$R_{kj} = \frac{1}{N} \sum_i x_{ik} x_{ij}$$

Equation 62

as the autocorrelation of the input samples for indices k, j. The autocorrelation matrix **R** of dimension (p+1)(p+1) can be created with entries R_{kj},

$$\mathbf{R} = \begin{bmatrix} R_{00} & R_{01} & \dots & R_{0p} \\ R_{10} & R_{11} & \dots & R_{1p} \\ \dots & \dots & \dots & \dots \\ R_{P0} & R_{p1} & \dots & R_{pp} \end{bmatrix}$$

Equation 63

This matrix is square and symmetric (but not necessarily Toeplitz). Let us call

$$P_j = \frac{1}{N} \sum_i x_{ij} d_i$$

Equation 64

the crosscorrelation of the input *x* for index *j* and desired response *y*, which can be also put into a vector **p** of dimension *p+1*.

$$\mathbf{p} = \begin{bmatrix} P_0 \\ P_1 \\ \dots \\ P_p \end{bmatrix}$$

Equation 65

Substituting these definitions in Eq.25 , the set of normal equations can be written simply

$$\mathbf{p} = \mathbf{R}\mathbf{w}^* \quad \text{or} \quad \mathbf{w}^* = \mathbf{R}^{-1}\mathbf{p}$$

Equation 66

where **w** is a vector with the *p+1* weights w_i.

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \cdots \\ w_p \end{bmatrix}$$

Equation 67

w* represents the value of the vector for the optimum (minimum) solution.

We used here the statistical definition for **R** and **p**. Let us clarify that when estimating these quantities from real data the properties only approximately apply .

Return to text

performance surface properties

The minimum value of the error can be obtained by substituting the optimal weight Eq.31 into the cost equation Eq.30 , yielding

$$J_{\min} = \frac{1}{2}\left[\sum_i \frac{d_i^2}{N} - \mathbf{p}^T\mathbf{w}^*\right]$$

Equation 68

We can re-write the performance surface in terms of its minimum value and **w*** as

$$J = J_{\min} + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T\mathbf{R}(\mathbf{w} - \mathbf{w}^*)$$

Equation 69

For the one dimensional case, this equation is the same as Eq.54 (*R* becomes a scalar equal to the variance of the input). In the space (w_1,w_2) *J* is now a parabola facing upwards. The shape of *J* is again solely dependent upon the input data (through its autocorrelation function). One can show that the principal axes of the performance surface contours (surfaces of equal error) correspond to the *eigenvectors of the input* *correlation matrix* **R**, *(see* Appendix *) while the eigenvalues of* **R** *give the rate of change of the gradient along the principal axis of the surface contours of J* (Figure 15).

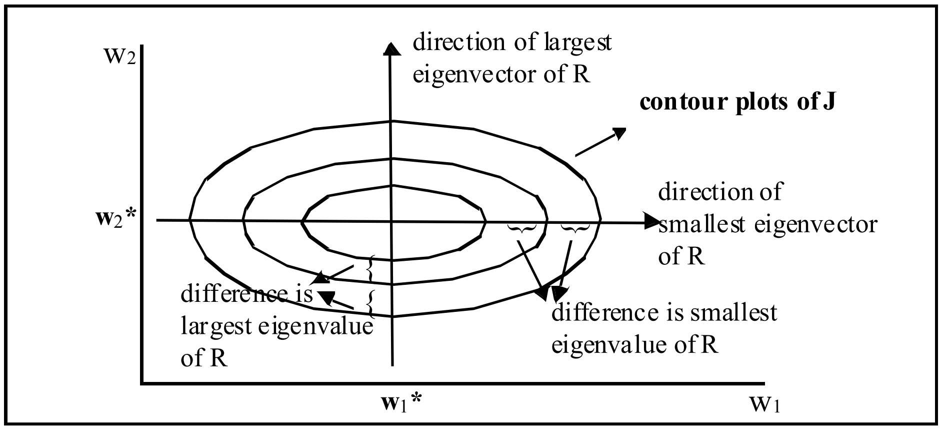

w2
direction of largest eigenvector of R
contour plots of J
w2*
direction of smallest eigenvector of R
difference is largest eigenvalue of R
difference is smallest eigenvalue of R
w1*
w1

FIGURE 15. **Contour plots of the performance surface with two weights**

The eigenvectors and eigenvalues of the input autocorrelation matrix are all what matters to understand convergence of the gradient descent in multiple dimensions. The eigenvectors represent the natural (orthogonal) coordinate system to study the properties of **R**. In fact in this coordinate system the convergence of the algorithm can be studied as a joint adaptation of several (one for each dimension of the space) unidimensional algorithms. Along each eigenvector direction the algorithm behaves just like the one variable case that we studied in the beginning of this chapter. The eigenvalue becomes the projection of the data onto that direction just like λ in Eq.55 is the projection of the data on the weight direction..

The location of the performance surface in weight space depends upon both the input and desired response Eq.31 . The minimum error is also dependent upon both data Eq.68 . Multiple regression finds the location of the minimum of a paraboloid placed in an unknown position in weight space. *The input distribution defines the shape of the performance surface*. The input distribution and its relation with the desired response distribution define both the *value of the minimum of the error and the location in coefficient space where that minimum occurs*.

Return to text

multiple variable correlation coefficient

The idea of the correlation coefficient is the same for 1D or multiple dimensions. The equations get a little more complicated since we are working now with an ensemble of input vectors. So the nice form of Eq.7 has to be modified. An ensemble of vectors is better described as a matrix, so we are going to define a new matrix **U** as

$$\mathbf{U}_X = \begin{bmatrix} x_1^1 & \dots & x_1^N \\ \dots & \dots & \dots \\ x_p^1 & \dots & x_p^N \end{bmatrix}$$

where each column is one of the input samples. Likewise we are going to define a column vector **d** with all the desired responses (this is a vector for the single output regression, otherwise also becomes a matrix)

$$\mathbf{d} = \begin{bmatrix} d_1 \\ \dots \\ d_N \end{bmatrix}$$

The total error variance can be written as

$$\varepsilon^T \varepsilon = \mathbf{d}^T \mathbf{d} - \mathbf{w}^{*T} \mathbf{U}_x \mathbf{d}$$

where **w*** is the set of optimal coefficients. This expression can be easily derived if the output of the regressor is substituted in the left side of the equation Linear Models . The part of the error that is explained by the linear model is the second term. This equation keeps the same form if we express it in terms of variance instead of error variance (just subtract the mean of the desired signal). So if we normalize this equation by the variance of the desired response we get

$$r^2 = \frac{\mathbf{w}^{*T}\mathbf{U}_x\mathbf{d} - N\bar{d}^2}{\mathbf{d}^T\mathbf{d} - N\bar{d}^2}$$

which leads to the correlation coefficient for the multivariate case.

Return to text

convergence for multiple weights case

One can show that the condition to guarantee converge Widrow and Stearns is

$$\lim_{k\to\infty}(\mathbf{I}-\eta\Lambda)^k = 0$$

Equation 70

where Λ is the eigenvalue matrix,

$$\Lambda = \begin{bmatrix} \lambda_0 & \ldots & 0 \\ \ldots & \ldots & \ldots \\ 0 & \ldots & \lambda_p \end{bmatrix}$$

Equation 71

which means that in every principal direction of the performance surface (given by the eigenvectors of the input correlation matrix **R**) one must have

$$0 < \eta < \frac{2}{\lambda_i}$$

Equation 72

where λ_i is the corresponding eigenvalue. This equation also means that with a single η each weight $w_i(k)$ is approaching its optimal value w_i^* with a different time constant ("speed"). So the weight tracks bend and the path is no longer a straight line towards the minimum.

This is the mathematical description that we said earlier that the gradient descent algorithm behaves as many one dimensional univariable algorithms along the eigenvector directions. Notice that Eq. 71 is diagonal so there is no cross-coupling between time constants along the eigenvector directions.

In any other direction of the space, there will be coupling. However, we can still decompose the overall weight tract as a combination of weight tracts along each eigendirection as we did in Figure 16. Eq. 72 shows that he stepsize along each direction obeys the same rule as the unidimensional case (Eq.17).

Return to text

estimation of eigenvalue spread

The eigenvalue spread can be computed by an eigendecomposition of **R**, but this is a time consuming operation and hardly ever is performed. An estimate of the eigenvalue spread is the ratio between the maximum and the minimum of the magnitude of the Fourier transform of the input data.

Alternatively, the simple inspection of the correlation matrix of the input can provide an estimation of the time to find a solution. The best possible case is when **R** is diagonal with equal values in the diagonal, because in this case the eigenvalue spread is 1 and the gradient descent goes in a straight line to the minimum. One can not have a faster convergence than this even when second order methods (such as the Newton's method studied later is used). When **R** is diagonal but with different values, the ratio of the largest number over the smallest is a good approximation to the eigenvalue spread. When **R** is fully populated, the analysis becomes much more difficult. However, if the non-diagonal terms have values comparable to the diagonal terms, one can expect a long training time.

Return to text

Casti Reference

Casti, J.L., Alternate Realities: Mathematical models of nature and man, Wiley, 1989.

Processing Element

The fundamental computational block in the system. In neural networks PEs are also called neurons or units.

Epoch

One complete presentation of the input data to the network being trained

linear regression

is the process of fitting (minimization of the sum of the square of the deviations) a cloud of samples by a linear model

mean square error

is the average of the square difference between the desired response and the actual system output (the error)

least squares

is an analytic procedure that minimizes the MSE in linear optimization problems (i.e. problems that are linear in the unknowns)

correlation coefficient

correlation coefficient is the ratio of the variance of the linear regressor over the variance of the desired response

Adaptive systems

Systems that change their parameters (through algorithms) in order to meet a pre-specified function, which is either an input-output map or an internal constraint.

Performance surface

is the total error surface plotted in the space of the system coefficients (weights)

Supervised learning

learning or adaptation is supervised when there is a desired response that can be used by the system to guide the learning

Unsupervised learning

learning is unsupervised when the system parameters are adapted using only the information of the input and are constrained by pre-specified internal rules

gradient

is a vector that always points to the direction of maximum change, with a magnitude equal to the slope of the tangent to the curve at the point.

steepest descent

is a search procedure that seeks the next operating point in the direction opposite to the gradient

Least Mean Square

or LMS is a steepest descent search algorithm that uses a very efficient estimate of the gradient (the product of the error times the input)

step size

or learning rate is the constant that scales the gradient to correct the old weights

on-line training

is a learning procedure that modifies the weights after the presentation of every sample

epoch

one complete presentation of the training data.

batch training

is the adaptation of the weight based on an epoch update

training set

is the ensemble of input/desired response pairs used to train the system

test set

is the ensemble of input/desired response data used to verify the performance of the trained system. This data is NOT used for training

learning curve

is a plot of the MSE across iterations

weight track

is a weight space plot of the weight locations during adaptation

geometric ratio

is the ratio of two consecutive terms in a geometric progression

time constant of adaptation

is the exponent of the exponentially fitted envelop of the weight’s geometric progression

rattling

is the perturbation around the optimal weight value produced by a nonzero learning rate

misadjustment

is the normalized excess MSE produced by the rattling

learning rate scheduling

is the choice of a variable stepsize, which starts large in the begining of training and decreases progressively towards the end of adaptation

eigenvalue spread

is the ratio of the largest over the smallest eigenvalue

normalized LMS

is the LMS algorithm with a stepsize normalized by an estimate of the input data variance

big O notation

is an approximate way to express the complexity of a computer algorithm, where only the largest factor is shown. Normally we are interested in multiplications since they are the most time consuming to execute in general purpose computers

adaline

Bernard Widrow called the linear processing element ADALINE for adaptive linear element

Eq. 4

$$J = \frac{1}{2N}\sum_{i=1}^{N}\varepsilon_i^2$$

Eq. 14

$$r^2 = \frac{\sum_i \left(y_i - \bar{d}\right)^2 - \frac{\sum_i (\varepsilon_i (y_i - \bar{d}))^2}{\sum_i (y_i - \bar{d})^2}}{\sum_i \left(d_i - \bar{d}\right)^2}$$

Eq.6

$$w = \frac{\sum_i (x_i - \bar{x})(d_i - \bar{d})}{\sum_i (x_i - \bar{x})^2} \qquad b = \frac{\sum_i x_i^2 \sum_i d_i - \sum_i x_i \sum_i x_i d_i}{N[\sum_i (x_i - \bar{x})^2]}$$

Eq.9

$$J = \frac{1}{2N}\sum_i \left(d_i - wx_i\right)^2 = \frac{1}{2N}\sum_i \left(x_i^2 w^2 - 2d_i x_i w + d_i^2\right)$$

Eq.3

$$d_i - (b + wx_i) = d_i - \tilde{d}_i = \varepsilon_i$$

Eq.10

$$\nabla J = \frac{\partial J}{\partial w}$$

Eq.12

$$\nabla J(k) = \frac{\partial}{\partial w} J(k) = \frac{\partial}{\partial w}\frac{1}{2N}\sum \varepsilon^2 \approx \frac{1}{2}\frac{\partial}{\partial w}\left(\varepsilon^2(k)\right) = -\varepsilon(k)x(k)$$

Eq.11

$$w(k+1) = w(k) - \eta\nabla J(k)$$

Eq.13

$$w(k+1) = w(k) + \eta\varepsilon(k)x(k)$$

Eq.54

$$J = J_{\min} + \frac{1}{2N}(w - w^*)\sum_i x_i^2 (w - w^*)$$

Eq.16

$$\lambda = \frac{1}{N}\sum_i x_i^2$$

Eq.17

$$|\rho| = |1 - \eta\lambda| < 1 \Rightarrow \eta < \frac{2}{\lambda}$$

Eq.19

$$\tau = \frac{1}{\eta\lambda}$$

Eq.25

$$\sum_i x_{ij} d_i = \sum_{k=0}^{p} w_k \sum_i x_{ik} x_{ij} \qquad \mathrm{j = 0,1,\ldots p}$$

Eq.28

$$P = RW^* \quad \text{or} \quad W^* = R^{-1}P$$

Eq.31

$$\nabla J = 0 = RW - P \quad \text{or} \quad W^* = R^{-1}P$$

Eq.68

$$J_{\min} = \frac{1}{2}\left[\sum_i \frac{d_i^2}{N} - P^T W^*\right]$$

EQ.69

$$J = J_{\min} + \frac{1}{2}(W - W^*)^T R(W - W^*)$$

Eq.65

$$P = \begin{bmatrix} P_0 \\ P_1 \\ \cdots \\ P_p \end{bmatrix}$$

Eq.27

$$P_j = \frac{1}{N}\sum_i x_{ij} d_i$$

Eq.36

$$\eta < \frac{2}{\lambda_{\max}}$$

Eq.21

$$\eta(n+1) = \eta(n) - \beta$$

Eq.33

$$W(k+1) = W(k) - \eta \nabla J(k)$$

Eq.32

$$\nabla J = \left[\frac{\partial J}{\partial w_0}, \ldots, \frac{\partial J}{\partial w_p} \right]^T$$

Eq.41

$$W(k+1) = W(k) + \eta R^{-1} \varepsilon(k) X(k)$$

Eq.5

$$\begin{cases} \dfrac{\partial J}{\partial b} = 0 \\ \dfrac{\partial J}{\partial w} = 0 \end{cases}$$

Eq.45

$$\sum_{i=1}^{N} d_i = Nb + w \sum_{i=1}^{N} x_i$$
$$\sum_{i=1}^{N} x_i d_i = b \sum_{i=1}^{N} x_i + w \sum_{i=1}^{N} x_i^2$$

Eq.53

$$J_{\min} = \frac{1}{2N} \left[\sum_i d_i^2 - \frac{\left(\sum_i d_i x_i \right)^2}{\sum_i x_i^2} \right]$$

Eq.52

$$w^* = \frac{\sum_i x_i d_i}{\sum_i x_i^2}$$

Eq.55

$$J = J_{\min} + \frac{\lambda}{2}(w - w^*)^2$$

Eq.30

$$J = \left[W^T R W - 2P^T W + \sum_i \frac{d_i^2}{N} \right]$$

Eq.26

$$R_{kj} = \frac{1}{N} \sum_i x_{ik} x_{ij}$$

Eq.63

$$R = \begin{bmatrix} R_{00} & R_{01} & \dots & R_{0p} \\ R_{10} & R_{11} & \dots & R_{1p} \\ \dots & \dots & \dots & \dots \\ R_{P0} & R_{p1} & \dots & R_{pp} \end{bmatrix}$$

Eq.40

$$W(k+1) = W(k) - R^{-1} \nabla J(k)$$

Widrow

Bernard Widrow was one of the first researchers that explored engineering applications of adaptive systems. We are going to hear a lot about him in this book.

Eq.7

$$r^2 = \frac{\sum_i \left(y_i - \bar{d}\right)^2}{\sum_i \left(d_i - \bar{d}\right)^2}$$

Eq.15

$$w(k+1) = w^* + (1 - \eta\lambda)^k (w(0) - w^*)$$

Widrow and Stearns

Adaptive Signal Processing, Prentice Hall, 1985 (Chapter 4).

Linear Models

Consult for instance the textbook Intro. to Linear Models by Dunteman, Sage Publications, 1984.

Eq.49

$$\sum_i \left(d_i - \bar{d}\right)^2 = \sum_i \left(y_i - \bar{d}\right)^2 + \sum_i \left(d_i - y_i\right)^2$$

outlier

is a noisy point that does not follow the characteristics of the input (or desired response) data.

RLS

is an on-line algorithm to compute the optimal weights (as opposed to the batch process to solve the least squares). Unfortunately it is also much more computational intensive than the LMS. To know more please consult

Adaptive Filter Theory by Haykin, Prentice Hall, 1996

Gauss

Karl Friedrich Gauss (1777-1855) was a mathematical genius who proposed the use of least squares to solve sets of linear equations. He realized that in optimization problems involving Gaussian distribution models, a quadratic equation was obtained (after taking the logarithm), which leads to an easy solution for the optimum.

covariance

is the sum of the crossproducts of the two variables with the means removed.

standard deviation

is the square root of the variance. The variance is the second moment of the data with respect to the mean.

Estimation theory

see for instance

autocorrelation

is a measure of similarity of the samples' distribution, which is computed by the sum of the crossproducts between the data set and its shifted versions. The autocorrelation is a function of the shift.

crosscorrelation

measures the similarity between two different data sets, and it is computed by the sum of the crossproducts between the two data sets at different lags (it is a function of the lag).

Derivation of Solution

Let us just take the derivative of J with respect to the weights, using the matrix operations.

$$\frac{\partial J}{\partial w} = Rw + w^T R - 2p \quad = Rw + R^T w - 2p = 2Rw - 2p$$

since the transpose of R is equal to itself (Toeplitz). If we equate this to zero we obtain the optimal weights, i.e.

$$w^* = R^{-1} p$$

which is the equation in the text

Return to Text

Contour

is a curve linking all the points with the same value of J (J=constant). The contour plot for J is formed by concentric ellipsoids (ellipses for the 2D case).

eigenvalues

are the scaling constants in the eigenvalue equation of a matrix. Here the matrix is the input autocorrelation matrix. Eigenvalues can be considered as the projections of the data along the eigenvectors.

Z scores

is a statistical terminology that means that all the variables are zero mean variables. See ???????

Newton’s Derivation

The equation can be easily proved if we recall the gradient of the performance surface

$$\nabla J = Rw - p$$

left multiply by R^{-1} to obtain

$$R^{-1}p = w - R^{-1}\nabla J$$

and then substitute in the optimal solution Eq. 28 to obtain

$$w^* = w - R^{-1}\nabla J$$

From this equation we can derive the incremental equation presented in the text

Return to Text

ill-conditioned

A matrix is ill-conditioned when the determinant is almost zero. See the appendix.

gradient definition and construction

The gradient is formally defined in terms of partial derivatives of a function f(x,y). Let us consider a function f(x,y) that has partial derivatives at x0,y0. The gradient of *f* at x0,y0 is defined by

$$gradf(x_0,y_0) = \nabla f(x_0,y_0) = f_x(x_0,y_0)u_x + f_y(x_0,y_0)u_y$$

where u_x, u_y are the unit vectors along x and y and f_x, f_y are the partial derivatives of *f* along the x and y directions respectively, which are given by

$$f_x = \frac{\partial f(x,y)}{\partial x}, \qquad f_y = \frac{\partial f(x,y)}{\partial y}$$

The gradient is associated with the concept of a directional derivative of a function. Let us assume we have a direction $u = au_x + bu_y$. The directional derivative of *f* at x0,y0 along ***u*** is

$$D_u f(x_0, y_0) = \lim_{h \to 0} \frac{f(x_0 + ha, y_0 + hb) - f(x_0, y_0)}{h}$$

So the gradient can be defined as a function of the ordered derivatives as

$$D_u f(x_0, y_0) = (gradf\ \ (x_0, y_0)) \cdot u$$

where the operation is the dot product of two vectors (for $v = cu_x + du_y$ $v \cdot u = ac + bd$).

This expression means that the maximum value of the directional derivative as a function of the direction ***u*** is given by the size of the gradient and it occurs exactly when the direction ***u*** coincides with the gradient direction.

Moreover, we can also find this direction pretty easily. Let us consider the curve C(x,y) defined as the line in the x,y plane where the function *f* has a constant value (this line is called the level curve or the contour of *f*). At a point x0,y0 in C the rate of change of *f* in the direction of the unit vector ***u*** tangent to C must be zero (see the definition above), i.e.

$$D_u f(x_0, y_0) = (gradf\ \ (x_0, y_0)) \cdot u = 0$$

But this implies that the gradient vector is perpendicular to the tangent vector ***u*** of the level curve at x0,y0. This explains the graphical construction outlined in the text.

Return to Text

development of the phone system

A good example is the telephone system. Long and meticulous research was conducted at Bell Laboratories on human perception of speech. This created the specification for the required bandwidth and noise level for speech intelligibility. Then engineers perfected the microphone that would translate the pressure waves into electrical waves to meet the specification. Then these electrical waves where transmitted through copper wires over long distances to a similar device, still preserving the required specification. For increased functionality the freedom of reaching any other telephone was added to the system. So switching of calls had to be implemented. This created the phone system. Initially, the switching among lines was done by operators. Then we invented a machine that would automatically switch the calls. Operators were still used for special services such as directory assistance. But now that the fundamental engineering aspects are stable, we are asking machines to automatically recognize speech and directly assist callers.

The development of the phone system is an excellent example of engineering design. Once we have a vision we try to understand the principles at work, create specifications and a system architecture. The fundamental principles at work are found by applying the scientific method. The phenomenon under analysis is first studied with physics or mathematics. The importance of models is that they translate general principles and through *deduction* we can apply them to particular cases like the ones we are interested in. These disciplines create approximate models of the external world using the principle of divide-and-conquer. First the problem is divided in manageable pieces, each is studied independently of the others and protocols among the pieces are drawn such that the system can work as a whole, meeting the specifications drawn *a priori*. This is what engineering design is today.

Return to the Text

Mars’ pathfinder mission

When the machines have to autonomously interact with the environment, or have to operate near the optimum set point, we can not specify all the functions a priori and in a deterministic way. Take for instance the Mars Pathfinder mission. It was totally impossible to specify all the possible conditions that the rover Sojourner would face, even if remotely controlled from Earth. So the problem could not be solved by a sequence of instructions determined a priori in JPL’s laboratory on Earth. The vehicle was given high level instructions (way points) and was equipped with cameras and laser sensors that would see the terrain. The information from the sensors was analyzed and catalogued in general classes. For each class a procedure was designed to accomplish the goal of moving from point A to point B. This is the type of engineering systems that we will be building more and more in the future.

The big difference from the initial machines and Sojourner is that the environment is intrinsically in the loop of the machine function. This brings a very different set of problems, because as we said earlier, the environment is complex and unpredictable. If our physical model does not capture the essentials of the environment, then errors accumulate over time and the solution becomes impractical. So we do not have anymore the luxury of dictating the rules of the game, as we did for the early machine building era.

It turns out that animals and humans do Sojourner type of tasks effortlessly.

Return to the Text

Index

1

1. Experimental Model Building 5

2

2. Data Collection 9

3

3. Least Squares 12

4

4. Least squares as a search for the parameters of a linear system 15

5

5. Estimation of the gradient - the LMS algorithm 21

6

6. Getting a grip on adaptation 26

7

7. Regression for multiple variables 34

8

8. Analytic versus Iterative solutions 47
8. Newton's method 45

B

batch versus online learning 58

C

Chapter I- Data Fitting with Linear Models 4
computation of correlation coefficient 57
Conclusions 51
convergence for multiple weights case 64

D

Derivation of correlation coefficient 57
derivation of normal equations 61
derivation of the time constant of adaptation 60
development of the phone system 79

E

End of Chapter 1 55
estimation of eigenvalue spread 65

L

least sqaure derivation 55

M

Mars' pathfinder mission 79
more derivation of performance surface 58

more on derivation of largest stepsize... 59
more on scheduling stepsizes... 60
multiple variable correlation coefficient.. 63

P

performance surface properties.. 62

T

The linear Regression Model.. 48

V

variance 56, 57