Dynamics and Algorithms for Stochastic Search

Genevieve Beth Orr
B.A., Grinnell College, 1979
M.S. University of Wisconsin, 1982
M.S. Drexel University, 1986

A dissertation submitted to the faculty of the
Oregon Graduate Institute of Science & Technology
in partial fulfillment of the
requirements for the degree
Doctor of Philosophy
in
Computer Science and Engineering

October 1995

The dissertation "Dynamics and Algorithms for Stochastic Search" by Genevieve B. Orr has been examined and approved by the following Examination Committee:

Todd Leen
Associate Professor
Thesis Research Adviser

Steve Otto
Assistant Professor

Etienne Barnard
Associate Professor

Andrew Fraser
Associate Professor
Portland State University

Dedication

To my parents, Arthur and Dorothy.

Acknowledgements

I would first like to thank my research advisor Dr. Todd Leen for his dedication to teaching, for his encouragement and support in my research at OGI, and for the confidence he gave me in pursuing my own ideas in the field.

I would also like to thank the other members of my committee, Dr. Steve Otto, Dr. Etienne Barnard, and Dr. Andrew Fraser for their support and their valuable suggestions on the thesis.

I would like to thank my officemates Steve Rehfuss and Nanda Kambhatla for the good technical discussions and all of the help they have given me over the many years we shared an office together. I would also like to thank the neural nets group at OGI for all of those fun tea time discussions.

Finally, I would like to thank all my fellow students at OGI who gave me the strength to persevere through the courses, the qualifying exams, the research proficiency, and finally, the writing of this thesis. In particular, thanks to Bennet, Jon, Harini, Ravi, Lisa, Scott, Ira, Nanda, and Judy for their emotional support and wonderful friendship.

Contents

List of Figures

Abstract

Dynamics and Algorithms for Stochastic Search

Genevieve Beth Orr, Ph.D.
Oregon Graduate Institute of Science & Technology, 1995

Supervising Professor: Todd Leen

In this thesis we develop a mathematical formulation for the learning dynamics of stochastic or on-line learning algorithms in neural networks. We use this formulation to 1) model the time evolution of the weight space densities during learning, 2) predict convergence regimes with and without momentum, and 3) develop a new efficient algorithm with few adjustable parameters which we call adaptive momentum.

In stochastic learning, the weights are updated at each iteration based on a single exemplar randomly chosen from the training set. Treating the learning dynamics as a Markov process, we show that the weight space probability density $P(w,t)$ can be cast as a Kramers-Moyal series

$$\frac{\partial P(w,t)}{\partial t} = L_{KM}\ P(w,t) \tag{0.1}$$

where L_{KM} is an infinite-order linear differential operator, the terms of which involve powers of the learning rate μ. We present several approaches for truncating this series so that approximate solutions can be obtained. One approach is the small noise expansion where the weights are modeled as a sum of a deterministic and noise component.

However, in order to provide more accurate solutions, we also develop a perturbation expansion in μ. We demonstrate the technique on equilibrium weight-space densities.

Unlike batch learning, stochastic updates are noisy but fast to compute. The speed-up can be dramatic if training sets are highly redundant, and the noise can decrease the likelihood of becoming trapped in poor local minima. However, acceleration techniques based on estimating the local curvature of the cost surface can not be implemented stochastically because the estimates of second order effects are much too noisy. Disregarding such effects can greatly hinder learning in problems where the condition number of the hessian is large. A matrix of learning rates (the inverse hessian) that scales the stepsize according to the curvature along the different eigendirections of the hessian is needed. We propose adaptive momentum as a solution. It results in an *effective learning rate matrix* that approximates the inverse hessian. No explicit calculation of the hessian or its inverse is required. This algorithm is only $\mathcal{O}(n)$ in both space and time, where n is the dimension of the weight vector.

Chapter 1

Introduction

The goal of this thesis is to improve understanding of the behavior of stochastic learning algorithms by mathematically modeling the weight dynamics. We then use this understanding to develop and implement a computationally efficient method for speeding learning. A stochastic learning algorithm is a method of estimating an optimal value of a parameter (e.g. weight in a network) by iteratively updating that parameter based on both the current parameter value and an individual datum (input) presented from the environment. The fluctuations of the incoming data appear as random noise, making it natural to treat these algorithms probabilistically. Stochastic learning can be contrasted with batch learning which updates the parameter value at each iteration based on an *average over all* inputs. Batch learning is deterministic and does not exhibit the fluctuations present during stochastic learning.

Stochastic learning algorithms for neural networks have important advantages over batch training methods. They require less storage and less computation at each iteration which is particularly important when training sets are large and redundant. The inherent noise in the learning process also lessens the chance of becoming stuck in local optima. At the same time, the convergence (e.g. in mean square) is guaranteed if the learning rate is annealed (decreased slowly towards zero) at late times to remove the noise.

The general form of the stochastic algorithm we consider is

$$\omega(t+1) = \omega(t) + \mu(t)\, H[\omega(t), x(t)] \tag{1.1}$$

where t is the time[1], $\omega(t) \in \mathbb{R}^m$ is the weight vector, $x(t) \in \mathbb{R}^n$ is the data exemplar presented at time t, $\mu(t)$ is the learning rate, and $H[\ldots] \in \mathbb{R}^m$ is the update function. The exemplars $x(t)$ can be either inputs or, in the case of supervised learning, input/target pairs. For batch learning, the update function becomes $H[\omega_t] \equiv \langle H[\omega_t, x_t] \rangle_x$ where $\langle \cdot \rangle_x$ denotes average over all exemplars in the training set. Typically, the learning rate is either held constant $\mu(t) = \mu_0$ or is annealed according, say, to the schedule $\mu(t) = \mu_0/t$. For gradient algorithms, the update function is the negative of the gradient of some cost function, $\mathcal{E}$.

1.1 Modeling Stochastic Search

In the first half of this thesis, we examine the late time convergence behavior for stochastic algorithms from a mathematical perspective. We treat weight updates probabilistically as a Markov process whose weight-space density can be described using the infinite order differential equation referred to as the Kramers-Moyal Expansion (KME). Much of our analysis focuses on meaningful ways of truncating the Kramers-Moyal Expansion, e.g. to the second order Fokker-Planck equation (FPE), so that approximate solutions can be computed. We examine the case where learning rate is held constant as well as the case where the learning rate is annealed. Convergence behavior in these two regimes is quite different and thus needs to be treated separately. Most previous theoretical development has examined what happens for a constant learning rate. We review these results and extend them to the annealed regime.

1.2 Speeding Learning using Curvature

For many optimization algorithms, whether stochastic or batch, convergence rates can be very sensitive to the choice of model parameters (e.g μ). Techniques for speeding batch algorithms often involve estimating either directly or indirectly the optimal values

[1]To simplify notation we will often express the time as a subscript, e.g. $\omega(t) = \omega_t$.

of these parameters based on local properties (e.g. curvature) of the cost surface.

Basic gradient descent algorithms do not take curvature into account. The learning rate parameter is picked in some ad hoc manner and the weights are moved in a direction opposite the gradient of the cost function. There are two problems with this. First, the gradient direction is often not the most direct path to the minimum. Second, a poor choice of learning rate parameter can have a large effect on convergence rate. If slightly too large, divergence can result. If too small, learning can be very slow.

Second order extensions of gradient descent attempt to correct these problems by modeling the cost surface, $\mathcal{E}$, as locally quadratic. If this assumption is valid then the optimal weight vector ω^* can be predicted in from the current weight ω_c in one (batch) step by

$$\omega^* = \omega_c - \left(\nabla_\omega^2 \mathcal{E}\right)^{-1} \nabla_\omega \mathcal{E} \Big|_{\omega=\omega_c}. \tag{1.2}$$

This is the idea behind Newton's Algorithm. Note that the learning rate "parameter" in this case is a matrix (the inverse Hessian) that depends on the current position of the weight. The Hessian is the second derivative of the cost function with respect to the weights and will be denoted by $R \equiv \nabla_\omega^2 \mathcal{E}$.

If the surface is not precisely quadratic, the predicted optimal weight is still likely to move the current weight much closer to the true optimal weight. Repeated updates can then be made. The downside of this method is that the Hessian is a matrix of size m^2 where m is the number of weights. For large networks, computing, storing, and then inverting R will generally offset any gains made in speed-up. Convergence takes fewer iterations but each iteration is very costly.

To reduce computation and storage space, approximations of second order methods are often made. For example, instead of using the full Hessian, the linearized Hessian (see Chapter 8) is used. The linearized Hessian has the advantage that it can be written as an outer product thus simplifying some computation. In addition, the linearized Hessian is positive definite thus providing added stability.

Another approximation is to assume that the Hessian is block diagonal, i.e. that the

correlation between weights for example *between* layers is zero. The most extreme case is to assume that the Hessian is diagonal thus making storage requirements only $\mathcal{O}(m)$ and making inversion trivial. How much is lost by neglecting off-diagonal elements depends on the problem. For poorly conditioned problems whose weight axes are not aligned with the eigendirections of the Hessian, this assumption can be quite poor.

Other simplifications include the quasi-Newton methods (e.g. BFGS) which keeps a positive definite estimate of the inverse Hessian directly. Storage is still $\mathcal{O}(m^2)$ but no matrix inversion is needed so that computation time is reduced from $\mathcal{O}(m^3)$ to $\mathcal{O}(m^2)$. These algorithms are known to work well for small networks. However, as the number of weights enter into the thousands (not uncommon for real world problems) the computation time can still become excessive. Storage requirements may also exceed maximum capacity although as memory becomes cheaper and more available this may be less of a problem.

These techniques all use batch learning because accurate estimates of local curvature are required; Stochastic estimates of local curvature are just too noisy. One approach to remedy this problem is to use averages over time (iterations) instead of batch averages over the training set to smooth out the noise. Time averaging decreases the numerical sensitivity to noise while still retaining the efficiency of stochastic search.

1.3 Time Averages

One example of time averaging was suggested by Venter [Ven67]. His algorithm estimates the diagonal components of the Hessian during the *annealing* phase of stochastic search. At each timestep the algorithm computes a finite difference approximation of the Hessian from two stochastic gradient estimates computed at weights a small distance c_t to either side of the current weight. The difference approximations are averaged over time. An important and necessary component of this algorithm is that the finite difference interval c_t is annealed along with the learning rate. The weight update for each *component* of

the weight has the form

$$\omega_{t+1} = \omega_t - \frac{A_t^{-1}}{t}\,\frac{1}{2}\left(\frac{\partial \mathcal{E}(\omega_t + c_t, x_{2t})}{\partial \omega} + \frac{\partial \mathcal{E}(\omega_t - c_t, x_{2t+1})}{\partial \omega}\right) \tag{1.3}$$

where

$$\begin{aligned} A_t &= \min(\,\max(a, B_n),\, b) \\ & \quad 0 < a < \lambda < b < \infty, \text{ where } \lambda \equiv \left\langle \partial^2 \mathcal{E}/\partial \omega^2 \right\rangle_x \\ B_t &= \frac{1}{t}\sum_{j=1}^{t} \frac{1}{2c_j}\left(\frac{\partial \mathcal{E}(\omega_j + c_j, x_{2j})}{\partial \omega} - \frac{\partial \mathcal{E}(\omega_j - c_j, x_{2j+1})}{\partial \omega}\right) \\ c_t &= \frac{c}{t^{\gamma}} \qquad \text{where } 0 < \gamma < \frac{1}{2}. \end{aligned}$$

Note that two inputs, x_{2t} and x_{2t+1}, are used at each iteration t. This algorithm has several disadvantages. First, it assumes a diagonal Hessian. Second, it is not clear how to best choose the parameters γ, a, b, or c.

One very simple example of time averaging that has been observed to improve convergence rates for gradient descent learning is to add a *momentum* term. Momentum, as we will show, amounts to having weight updates based on an exponential average of previous gradients rather than just the current gradient. In this thesis we extend the theoretical analysis of equation (1.1) to include convergence results for stochastic learning with momentum. We show that if the momentum parameter is chosen properly, convergence rates can often be improved. Of course, the problem of choosing the optimal learning rate parameter is now replaced with the equally difficult task of choosing the optimal momentum parameter.

1.4 Adaptive Momentum

The second half of this thesis examines using momentum as a potential method for speeding stochastic search. We study constant momentum for both constant and annealed learning rates. In addition we present a novel technique for speeding learning during the annealing phase of stochastic search. We refer to this as adaptive momentum. Adaptive

momentum was inspired by our theoretical results on constant momentum obtained in the first half of the thesis. It is a stochastic form of the optimal momentum parameter *matrix* that adjusts itself based on the local curvature. No momentum parameter needs to be set by the user. It achieves fast asymptotic convergence rates independent of the learning rate. It is also efficient: Given m weights, each iteration is $\mathcal{O}(m)$ in both space and time. We apply the algorithm to both linear and nonlinear problems of varying size and complexity.

Chapter 2

Weight-Space Probability Densities

In this chapter we consider algorithms of the form

$$\omega(t+1) = \omega(t) + \mu\, H^i[\omega(t), x(t)] \tag{2.1}$$

where the $x(t)$ are now treated as i.i.d. random variables with some known density $\rho(x)$ and where μ is a *constant*. Equation (2.1) describes a random walk on ω with fixed timesteps ($\Delta t = 1$) and variable spatial steps $\Delta\omega$. The weights are thus random variables whose probability density at time t we denote by $P(\omega, t)$.

In the first part of this chapter we develop a Kramers-Moyal Expansion (KME) for the time evolution of $P(\omega, t)$. Unfortunately the KME, being of infinite order, is not in general[1] amenable to analytic solution, thus, a large part of our analysis is devoted to developing principled methods of approximation. We also develop a *backwards* KME for describing the distribution of first passage times. At the end of the chapter we compare the theoretical predictions with simulations for several small problems.

2.1 Kramers-Moyal Expansion and Weight-Space Densities

In this section we present two different derivations of the Kramers-Moyal expansion (KME) for $P(\omega, t)$. The first is via the Kolmogorov equation which results in a differential

[1] See [LM93] for a case where it solvable.

difference equation. Transition to continuous time gives the KME. The second derivation is via the (continuous time) Master Equation. Both derivations require conditions for which the transition to continuous time is valid.

2.1.1 via the Kolmogorov Equation

Equation (2.1) describes a Markov process whose density evolves according to the Kolmogorov equation

$$P(\omega', (n+1)\tau) \;=\; \int d\omega\, P(\omega, n\tau)\, T(\omega'|\omega) \tag{2.2}$$

where τ has been introduced as a timescale ($\tau = 1$ in (2.1)) and where $T(\omega'|\omega)$ is the *single* step transition probability from state ω to ω',

$$T(\omega'|\omega) = \int dx\; \rho(x)\; \delta(\omega' - \omega - \mu\, H[\omega, x]\,) \tag{2.3}$$

where δ is the Dirac Delta function. Equation (2.2) is an integral equation that is generally difficult to evaluate, however, it can be recast as a differential-difference equation [LO94] by expanding the transition probability (2.3) as a power series in μ (see Appendix A) to give

$$\tau\; \frac{P(\omega, (n+1)\tau) \;-\; P(\omega, n\tau)}{\tau} \;= \\ \sum_{i=1}^{\infty} \frac{(-\mu)^i}{i!} \sum_{j_1,\ldots j_i=1}^{m} \frac{\partial^i}{\partial\omega_{j_1}\,\partial\omega_{j_2}\ldots\partial\omega_{j_i}}\; (\,\langle H_{j_1} H_{j_2} \ldots H_{j_i}\rangle_x\; P(\omega, n\tau)\,) \tag{2.4}$$

where ω_{j_α} and H_{j_α} are the j_α^{th} component of weight, and weight update, respectively. Letting $t = n\tau$ and assuming $\tau \ll t$, the left side approximates[2] a derivative to give

$$\tau\;\; \partial_t P(\omega, t) = \sum_{i=1}^{\infty} \frac{(-\mu)^i}{i!} \sum_{j_1,\ldots j_i=1}^{m} \frac{\partial^i}{\partial\omega_{j_1}\,\partial\omega_{j_2}\ldots\partial\omega_{j_i}}\; (\,\langle H_{j_1} H_{j_2} \ldots H_{j_i}\rangle_x\; P(\omega, t)\,)\,. \tag{2.5}$$

This equation is called the Kramers-Moyal expansion. Approximate solutions can be found by truncating to finite order. In particular, we can truncate to second order in μ to obtain a Fokker-Planck equation. This is the subject of Section §2.2.

[2] See appendix D for a more detailed discussion of the transition to continuous time.

2.1.2 via the Master Equation

Heskes [HSK92] uses an alternative derivation of (2.5) via a master equation. A master equation can be obtained from (2.1) by treating time between transitions as a random variable. In (2.1) we assumed that transitions occur precisely at $t = 1, 2, \ldots$ so that at $t = n\tau$ there have been exactly n transitions. Drawing on the work of Bedeaux [BLLS71], Heskes instead sets the number of transitions in time t be Poisson distributed. That is, the probability of there being *exactly* i transitions in time t is

$$\phi(i,t) = \left(\frac{t}{\tau}\right)^i \frac{e^{-\frac{t}{\tau}}}{i!} \tag{2.6}$$

where τ is the average time between transitions. Note that both the mean and the variance of the number of transitions in time t is equal to t/τ. We now have that that the *average* rather than the *exact* number of transitions at $t = n\tau$ is n. Defining $p_i(\omega)$ as the probability of being at ω after exactly i transitions, we then can write the probability of being at weight ω at time t as

$$P_\phi(\omega,t) = \sum_{i=0}^{\infty} \phi(i,t)\, p_i(\omega). \tag{2.7}$$

Note that P_ϕ is not quite the same as the P in (2.2). At late times, however, the spread of ϕ about the mean t/τ becomes small relative to t so that P_ϕ is a good approximation of P[3].

To obtain the master equation, we first differentiate (2.7) with respect to time

$$\begin{aligned}
\partial_t P_\phi(\omega,t) &= \sum_{i=1}^{\infty} \left\{ \frac{i\, t^{i-1}}{\tau^i} \frac{e^{-\frac{t}{\tau}}}{i!} - \left(\frac{t}{\tau}\right)^i \frac{e^{-\frac{t}{\tau}}}{\tau\, i!} \right\} p_i(\omega) && (2.8)\\
&= \sum_{i=1}^{\infty} \frac{1}{\tau} \left\{ \left(\frac{t}{\tau}\right)^{i-1} \frac{e^{-\frac{t}{\tau}}}{(i-1)!} - \left(\frac{t}{\tau}\right)^i \frac{e^{-\frac{t}{\tau}}}{i!} \right\} p_i(\omega) && (2.9)\\
&= \frac{1}{\tau} \left(\sum_{i=1}^{\infty} \phi(i-1,t)\, p_i(\omega) - \sum_{i=0}^{\infty} \phi(i,t)\, p_i(\omega) \right)\\
&= \frac{1}{\tau} \left(\sum_{i=1}^{\infty} \phi(i-1,t)\, p_i(\omega) \;-\; P_\phi(\omega,t) \right). && (2.10)
\end{aligned}$$

[3]Bedeaux also discusses bounds on the difference between the solution of the master equation and P_ϕ when other densities for $\phi(i,t)$ are used. Note that if we let $\phi(i,t) = \delta(t-i)$ then we return to our original formulation where transitions occur precisely at integer values of t.

The first term on the right side can be written as

$$\begin{aligned}\frac{1}{\tau}\sum_{i=1}^{\infty}\phi(i-1,t)\,p_i(\omega) &= \frac{1}{\tau}\sum_{i=1}^{\infty}\phi(i-1,t)\int d\omega'\,T(\omega|\omega')p_{i-1}(\omega') \\ &= \int d\omega'\,\frac{T(\omega|\omega')}{\tau}\sum_{i=0}^{\infty}\phi(i,t)\,p_i(\omega') \\ &= \int d\omega'\,W(\omega|\omega')\,P_\phi(\omega',t) \qquad (2.11)\end{aligned}$$

where

$$W(\omega|\omega') \equiv \frac{T(\omega|\omega')}{\tau} \qquad (2.12)$$

is the transition probability from state ω' to ω *per unit time.* Inserting (2.11) into (2.10) gives the master equation

$$\begin{aligned}\partial_t P_\phi(\omega,t) &= \int d\omega'\,W(\omega|\omega')\,P_\phi(\omega',t) - \frac{1}{\tau}\,P_\phi(\omega,t) \\ &= \int d\omega'\,\left[\,W(\omega|\omega')\,P_\phi(\omega',t) - W(\omega'|\omega)\,P_\phi(\omega,t)\,\right]. \qquad (2.13)\end{aligned}$$

Just as we expanded the Kolmogorov equation in Appendix A, we can expand the right side of (2.13) in a power series [Gar90] resulting in the Kramers-Moyal expansion in (2.5) but with P replaced by P_ϕ. Recall, for $\tau \ll t$, P_ϕ and P can be treated as approximations of each other.

In summary, the weight-space probability density associated with the discrete time stochastic algorithm given in (2.1) can be modeled at late times ($\tau \ll t$) by the Kramers-Moyal expansion, an infinite order partial differential equation. This expansion can be derived either via a Kolmogorov equation as in subsection (2.1.1) or via a master equation as just shown here.

2.2 Fokker-Planck Equation

The Kramers-Moyal expansion is in general intractable so that approximations must be made. Knowing how to do this in a meaningful way is difficult. In this chapter and in chapter 3 we present several approaches.

One approach is to reduce the KME to a second order equation known as the Fokker-Planck equation. The general form of a nonlinear Fokker-Planck (1-dimension) is

$$\frac{\partial P}{\partial t} = -\frac{\partial}{\partial y}A(y)P + \frac{1}{2}\frac{\partial^2}{\partial y^2}B(y)P \tag{2.14}$$

where $A(y)$ and $B(y) > 0$ are any real differentiable functions. A is usually referred to as the *drift* term and B as the *diffusion* or *fluctuation* term. The *linear* Fokker-Planck equation refers to the case where A is linear in y and B is a constant.

Using the Fokker-Planck equation is appealing because it has been well studied in the context of diffusion processes in physics and chemistry. Thus there is a well understood physical interpretation that fits well with our intuition about the way weights evolve in time.

There is another somewhat more obscure reason for using a Fokker-Planck equation that is a result of the Pawula Theorem (see [Ris89] for details and proof). Stated in terms of the Kramers-Moyal expansion for the transition probability $P(\omega, t|\omega', t')$, the Pawula Theorem in Risken's words concludes that "the expansion may stop either after the first term or after the second term, if it does not stop after the second term it must contain an infinite number of terms." That is, if only a finite number of terms are kept past the second term, the transition probability must have negative values. However, as we shall see in the next chapter, this does not necessarily mean that these extra terms are useless.

Using the Fokker-Planck equation to model stochastic learning algorithms for neural networks is not new (e.g. see [HSK92, RS88, DV93, RSW90, HPS93]), yet there still remains a limited understanding of the conditions under which such an approximation is valid. In this section, we assume that the Kramers-Moyal equation (or equivalently, the master equation) accurately models the probabilistic late time development of (2.1). Given this, we are interested in understanding in what way a Fokker-Planck equation is an appropriate limit or approximation of the Kramers-Moyal expansion.

2.2.1 Fokker-Planck Limit

The infinite order Kramers-Moyal equation (2.5) is unsolvable analytically as it stands, however, one can ask what it reduces to in the limit of infinitely small stepsize, i.e. $\mu \to 0$. However, Gardiner [Gar90] shows that the master equation (2.13) (or equivalently the Kramers-Moyal equation) reduces to a Fokker-Planck equation in the limit of infinitely small jump size *only* if the "scaling assumption" holds. The scaling assumption states that the average stepsize and the variance of the stepsize must both be proportional to the same parameter which, in this case, is μ. If the scaling assumption does not hold, we can not assume that the higher order terms in (2.5) vanish. Unfortunately, as Heskes points out, the scaling assumption does not hold for (2.1) as can be seen from the fact that $\langle \Delta\omega \rangle_x \sim \mathcal{O}(\mu)$ and $\langle (\Delta\omega - \langle \Delta\omega \rangle_x)^2 \rangle_x \sim \mathcal{O}(\mu^2)$. Thus, there is no *limiting* case of the KME that results in a Fokker-Planck equation.

2.2.2 The Fokker-Planck Approximation: Truncation of the Kramers-Moyal Expansion

Although the Fokker-Planck equation can not be considered as a *limit* of the master equation, it can be considered as an *approximation* to it. By viewing the Kramers-Moyal expansion (2.5) as an expansion of the master equation in the "small" parameter μ, we can truncate (2.5) to obtain the *nonlinear* Fokker-Planck equation (1-dimension)

$$\tau \ \frac{\partial P(\omega,t)}{\partial t} = -\mu \frac{\partial}{\partial \omega} \left(\alpha_1(\omega)\ P(\omega,t)\right) + \frac{\mu^2}{2} \frac{\partial^2}{\partial \omega^2} \left(\alpha_2(\omega)\ P(\omega,t)\right) \tag{2.15}$$

where $\alpha_i \equiv \langle H^i(\omega,x) \rangle_x$. We present the XOR problem [LO92, OL93] later in the chapter to illustrate how well (2.15) approximates the density $P(\omega,t)$.

2.2.3 The Small Noise Expansion for Behavior within a Basin

The argument against truncating the Kramers-Moyal expansion as above to obtain a nonlinear Fokker-Planck Equation is that $P(\omega,t)$ implicitly depends on μ so we can not be certain of the order of the truncated terms. Heskes [Hes94] justifies this statement by substituting the stationary solution of the one-dimensional Fokker-Planck equation back

into the Kramers-Moyal expansion. He finds in this case that "all higher order terms are of the same order of magnitude in η [the learning rate] as the first two terms."

To be confident that the truncated terms can be neglected, we *explicitly* model the μ dependence in (2.5) using a technique referred to as the small noise expansion [Hes94, Gar90, vK92]. The small noise expansion begins with the *ansatz* that ω can be written as

$$\omega = \phi(t) + \sqrt{\mu}\xi \tag{2.16}$$

where ξ is a random variable that represents the nondeterministic deviations about a deterministic path $\phi(t)$. Note that both ϕ and ξ (and thus also $P(\xi, t)$) are assumed to be independent of μ (at least to $\mathcal{O}(\mu)$). This formulation is consistent with our intuition about the behavior of (2.1) after the weights have settled into a single basin of attraction: ϕ represents the "true gradient" path and the size of the fluctuations about this path depends on the learning rate μ.

Making a change of variable to ξ [HSK92] transforms the Kramers-Moyal expansion into the set of equations (see appendix (B))

$$\frac{d\phi(t)}{dt} = c\,\alpha_1^{(0)}(\phi(t)) \tag{2.17}$$

$$\frac{\partial P(\xi,t)}{\partial t} = c\sum_{m=2}^{\infty}\sum_{i=1}^{m}\frac{(-1)^i\mu^{\frac{m-2}{2}}}{i!(m-i)!}\alpha_i^{(m-i)}(\phi(t))\frac{\partial^i}{\partial\xi^i}\left\{\xi^{m-i}P(\xi,t)\right\} \tag{2.18}$$

where[4]

$$\alpha_i^{(k)}(\phi(t)) \equiv \frac{\partial^k}{\partial\omega^k}\left\langle H^i(\omega,x)\right\rangle_x\Big|_{\omega=\phi(t)} \tag{2.19}$$

and where we have assumed that the timescale τ scales as $\mu = c\tau$ for some constant c.

Equation (2.17) for ϕ describes the deterministic motion along the true gradient, while (2.18) describes the fluctuations about the true gradient. To lowest order (m=2), (2.18) becomes the *linear* Fokker-Planck equation (with c=1)

$$\frac{\partial P(\xi,t)}{\partial t} = -\alpha_1^{(1)}(\phi(t))\,\frac{\partial}{\partial\xi}\left(\xi\,P(\xi,t)\right) + \frac{1}{2}\,\alpha_2^{(0)}(\phi(t))\,\frac{\partial^2}{\partial\xi^2}P(\xi,t) \tag{2.20}$$

[4]Note that the previously defined α_i is related to $\alpha_i^{(k)}$ by $\alpha_i = \alpha_i^{(0)}$.

Note that μ does not appear. Thus, to lowest order, $P(\xi, t)$ is independent of μ and so fulfills the conditions of the original *ansatz*.

Equation (2.18) can be used to solve for $\langle\xi\rangle$, and $\langle\xi^2\rangle$ since

$$\frac{d\langle\xi\rangle}{dt} = \int d\xi\, \xi \frac{\partial P(\xi,t)}{\partial t} \tag{2.21}$$

$$\frac{d\langle\xi^2\rangle}{dt} = \int d\xi\, \xi^2 \frac{\partial P(\xi,t)}{\partial t}. \tag{2.22}$$

Using the right hand side of (2.18) and integrating by parts gives

$$\frac{d\langle\xi\rangle}{dt} = c\sum_{m=2}^{\infty} \frac{\mu^{\frac{m-2}{2}}}{(m-1)!}\, \alpha_1^{(m-1)} \langle\xi^{m-1}\rangle \tag{2.23}$$

$$\frac{d\langle\xi^2\rangle}{dt} = c\sum_{m=2}^{\infty} \mu^{\frac{m-2}{2}} \left\{ \frac{2\,\alpha_1^{(m-1)}}{(m-1)!} \langle\xi^m\rangle + \frac{\alpha_2^{(m-2)}}{(m-2)!} \langle\xi^{m-2}\rangle \right\} \tag{2.24}$$

Keeping only the $m = 2$ terms gives

$$\frac{d\langle\xi\rangle}{dt} = c\,\alpha_1^{(1)} \langle\xi\rangle \quad \Rightarrow \quad \langle\xi\rangle_t = \langle\xi\rangle_{t_0} e^{\gamma(t_0,t)/2} \tag{2.25}$$

$$\frac{d\langle\xi^2\rangle}{dt} = 2c\,\alpha_1^{(1)} \langle\xi^2\rangle + c\alpha_2^{(0)}$$

$$\Rightarrow \quad \langle\xi^2\rangle_t = \langle\xi^2\rangle_{t_0} e^{\gamma(t_0,t)} + c\int_{t_0}^{t} ds\, e^{\gamma(s,t)} \alpha_2^{(0)}(\phi(s)) \tag{2.26}$$

where

$$\gamma(t_1, t_2) \equiv 2c \int_{t_1}^{t_2} \alpha_1^{(1)}(\phi(\tau))\, d\tau.$$

The ansatz, however, does *not* seem appropriate for describing the transitions *between* basins. For example, consider gradient descent. Equation (2.17) shows that the deterministic component decays down the gradient ($\alpha_1^{(0)}$ is negative the gradient). To move between basins would require moving up the gradient for at least a short period of time. In addition, equation (2.25) diverges if $\alpha_1^{(1)}$ is positive. This corresponds to regions where the Hessian of cost function is negative, i.e. the regions between basins.

Note also that Fokker-Planck equation in (2.20) in linear in ξ. Thus Heskes argues that if we restrict ourselves to a Fokker-Planck equation, only the linearized equation for $P(\xi, t)$ is meaningful. In other words, using a nonlinear Fokker-Planck as in equation (2.15) can only produce spurious results. In addition, this linearized Fokker-Planck is

only valid at late times when the weight-density is completely contained within a basin, i.e. it is not able to describe basin hopping. However, as the example with the XOR problem will demonstrate, the nonlinear Fokker-Planck equation in (2.15) when applied directly to ω (as opposed to ξ) is able to accurately model the complex nonlinear behavior that is involved in basin hopping. The results hardly seem spurious. Why it does such a good job is not clear. When the learning rate is large, (2.15) does fail, but that is not surprising because the higher order terms are expected to become more important as μ increases. In addition, a continuous time equation ceases to be a good approximation of the behavior of the weights when the (finite) stepsize is large.

2.2.4 Higher Order Terms

Van Kampen [vK92, pp.267-272] explores the effect of keeping the next higher order term in (2.18). Not only do higher powers of ξ appear but so do derivatives up to fourth order; a far different result than would have been obtained had the next term in the Kramers-Moyal expansion been included. However, the resulting equation for $P(\xi, t)$ is no longer independent of μ violating the original *ansatz*. The next chapter presents an alternative approach for obtaining higher order terms for the special case of equilibrium weight densities.

2.3 The Backward Kramers-Moyal Expansion and First Passage Times

One way of characterizing convergence of the weights is by the *first* passage time; the time required for a network initialized at ω_0 to first pass into an ϵ neighborhood D of a global or local optimum ω^* (see Figure 2.1). In this section we derive analytic expression for the first passage time.

We begin by writing the weight space density *conditioned* on the initial weight ω_0 at $t = 0$ as

$$P(\omega, n|\omega_0, 0) = \int d\omega'\, P(\omega, n|\omega', 1)\, T(\omega'|\omega_0). \tag{2.27}$$

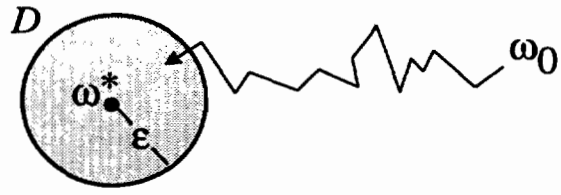


Figure 2.1: First Passage Time. Sample search path entering ϵ neighborhood of optimum ω^*

That is, the probability of transitioning from ω_0 to ω in n iterations equals the probability of first transitioning to ω' in *one* timestep, times the probability of then transitioning from ω' to ω in $n-1$ timesteps, summed over all ω'. Substituting in the expression for $T(\omega'|\omega_0)$ gives

$$P(\omega, n|\omega_0, 0) = \int d\omega' \, P(\omega, n|\omega_0, 1) \int dx \; \rho(x) \; \delta(\omega' - \omega_0 - \mu \, H[\omega_0, x] \,) \quad (2.28)$$

$$= \langle P(\omega, n-1|\omega_0 + \mu \, H[\omega_0, x] \,), 0) \rangle_x . \quad (2.29)$$

In the last step we have used the property of time-shift invariance, i.e. $P(\omega, n|\omega', m) = P(\omega, n-1|\omega', m-1)$. This property holds because we are using a constant learning rate and we have assumed that the input density $\rho(x)$ is stationary.

Next we let $G_\epsilon(n; \omega_0)$ denote the probability that a network initialized at ω_0 has *not* passed into the region D by the n^{th} iteration. We obtain $G_\epsilon(n; \omega_0)$ by integrating $P(\omega, n \mid \omega_0, 0)$ over weights ω *not in* D;

$$G_\epsilon(n; \omega_0) = \int_{D^c} d\omega \; P(\omega, n \mid \omega_0, 0) \quad (2.30)$$

where D^c is the complement of D. Substituting equation (2.29) into (2.30) and integrating over ω we obtain the recursion

$$G_\epsilon(n; \omega_0) = \langle \, G_\epsilon(n-1; \omega_0 + \mu H[\omega_0, x \,]) \, \rangle_x \quad . \quad (2.31)$$

Before any learning takes place, we assume that none of the networks in the ensemble have entered D. Thus the initial condition for G_ϵ is

$$G_\epsilon(0; \omega_0) = 1 \, , \quad \omega_0 \in D^c \quad . \quad (2.32)$$

Networks that have entered D are removed from the ensemble (i.e. the surface of D is an absorbing boundary). Thus G_ϵ satisfies the boundary condition

$$G_\epsilon(n;\omega_0) \;=\; 0\,, \quad \omega_0 \in D \;. \tag{2.33}$$

Finally, the probability that the network has *not* passed into the region D on or before iteration $n-1$ minus the probability the network has *not* passed into D on or before iteration n is simply the probability that the network *has* passed into D *exactly at iteration* n. Thus the first passage time is

$$P_\epsilon^{fpt}(n;\omega_0) = G_\epsilon(n-1;\omega_0) \;-\; G_\epsilon(n;\omega_0) \tag{2.34}$$

where $P_\epsilon^{fpt}(n;\omega_0)$ is the probability that a network initialized at ω_0 *first* enters the ϵ-neighborhood D at the n^{th} iteration.

Finally the recursion (2.31) for G_ϵ can be expanded in a power series in μ to obtain the *backward* Kramers-Moyal equation

$$G_\epsilon(n;\omega) \;-\; G_\epsilon(n-1;\omega) \;=$$
$$\sum_{i=1}^{\infty} \frac{\mu^i}{i!} \sum_{j_1,\cdots j_i=1}^{m} \langle H_{j_1} H_{j_2} \ldots H_{j_i} \rangle_x \; \frac{\partial^i}{\partial\omega_{j_1}\,\partial\omega_{j_2}\ldots\partial\omega_{j_i}}\, G_\epsilon(n-1;\omega) \;. \tag{2.35}$$

Note that we can also truncate (2.35) to second order in μ to obtain the *backward* Fokker-Planck equation for G

$$\tau \;\frac{\partial G_\epsilon(t;\omega)}{\partial t} = -\mu\alpha_1(\omega)\frac{\partial}{\partial\omega}G_\epsilon(t;\omega) + \frac{\mu^2}{2}\alpha_2(\omega)\;\frac{\partial^2}{\partial\omega^2}G_\epsilon(t;\omega). \tag{2.36}$$

2.4 Simulations

In this section we examine the XOR problem and competitive learning. We compare simulations of the weight-space densities and first passage times with the theoretically predicted values. The theoretical values are computed using the nonlinear Fokker-Planck equation obtained by truncating the Kramers-Moyal Equation. The recursive solution (2.31) for the first passage time is also examined. We postpone until the next chapter the comparison of the nonlinear Fokker-Planck equation with the small noise expansion.

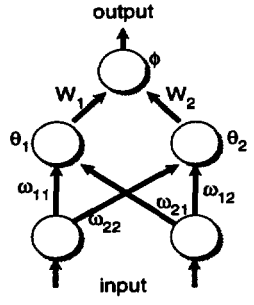


Figure 2.2: XOR architecture

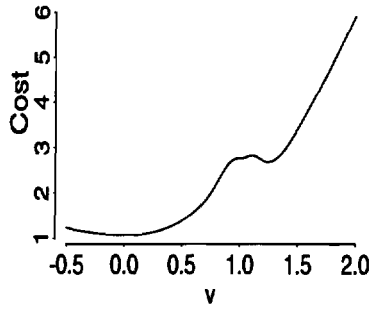


Figure 2.3: XOR. Cost function for 1-D slice.

2.4.1 The XOR Problem: Weight-Space Densities

The XOR problem is a well studied, simple but nontrivial nonlinear problem. We use the standard 2-input/2-hidden/1-output architecture (9 weights = 6 connecting weights + 3 biases) shown in Figure 2.2. The network is trained by stochastic gradient descent on the cross-entropy error function [LP91] given by

$$\mathcal{E}(t) = \ln\left(y_t^{d_t}\ (1-y_t)^{1-d_t}\right) \tag{2.37}$$

where y_t is the activation of the output node and d_t is the desired target value at timestep t. The hidden and output nodes use the sigmoid response function $f(x) = 1/(1+e^{-x})$. To provide global optima at *finite* weight values, the output targets are set to δ and $1-\delta$, with $\delta << 1$.

For computational tractability, we reduce the state space dimension by constraining the search to one-dimensional subspaces of the weight space. Figure 2.3 displays the cost function along one such subspace. The parameterization v is chosen to pass through a global optimum at $v = 0$, and a known local optimum [LP91] at $v = 1.0$. In this one-dimensional slice, another local optimum[5] occurs at $v = 1.24$.

Figure 2.4a shows the evolution of $P(v,t)$ estimated by simulation of 10,000 networks, each receiving a different random sequence of the four input/target patterns. Initially the density is peaked up about the local optimum at $v = 1.24$. At intermediate times, there is a spike of density at the local optimum at $v = 1.0$. This spike is narrow since the diffusion coefficient is small there. Figure 2.4b shows the evolution of $P(v,t)$ obtained

[5]The local optimum at $v = 1$ is a local minimum even when viewed in the entire 9 dimensional space whereas local minimum at $v = 1.24$ is a local minimum only in the 1-dimensional space.

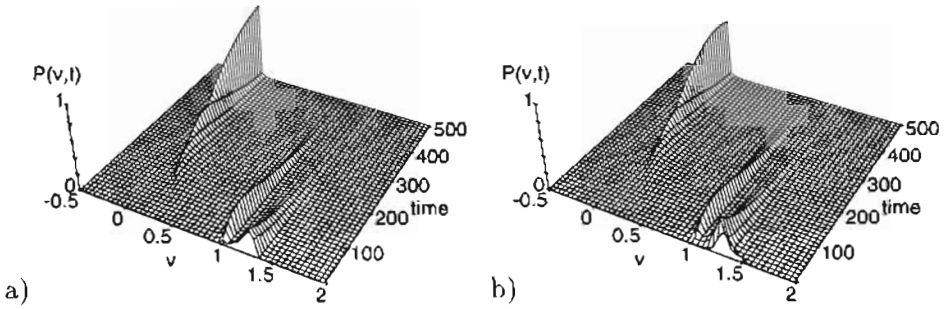


Figure 2.4: 1-D XOR. Time evolution of weight-space density ($\mu = .025$): a) simulation, b) Fokker-Planck solution.

by numerical integration[6] of the Fokker-Planck equation.

Agreement between the simulated and the Fokker-Planck density is very good. It is important to note that the nonlinear Fokker-Planck equation is able to accurately capture the highly nonlinear behavior of the density as it transitions *between* basins of attraction as well as the behavior within a basin. This result contradicts Heskes [HK93] claims that only a *linear* Fokker-Planck equation has any meaning, that is, any features resulting from the nonlinearity can not be trusted. However, our empirical evidence here suggests that for "small" learning rates the nonlinear Fokker-planck provides a good estimate of $P(\omega, t)$ even when transitions between basins occurs.

For large learning rates, the agreement between the Fokker-Planck predictions and simulations is quite poor. An example of this is shown below for XOR first passage times. In such cases, the contribution of the higher order truncated terms can not be neglected. In addition, the continuous time assumption ceases to be a good approximation especially at early times.

We next look at the weight-space density for a 2-D slice through the XOR weight space. Figure (2.5) shows the time evolution of the weight-space probability density computed by numerical integration of the forward Kolmogorov equation. The learning rate is $\mu = 0.25$. Each graph in this figure displays the 2-dimensional cost surface with the weight-space density superimposed on top of it. The weights are initialized at the back right (see spike at $t = 0$). The density moves down the cost surface and ends up in a global minimum at $t = 100$. Notice that at $t = 34$ some of the density temporarily spends

[6] We use a Crank-Nicholson differencing scheme for the numerical integration [PFTV87, for example].

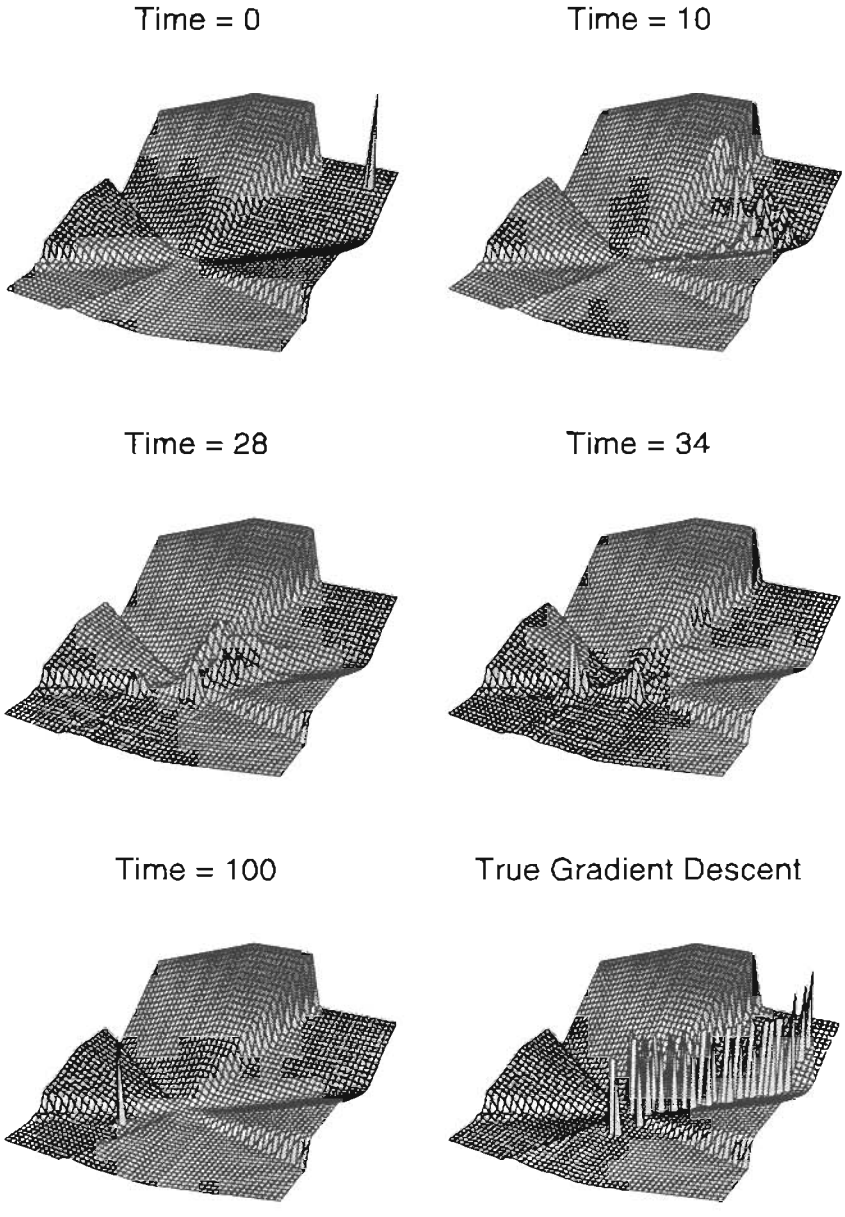


Figure 2.5: 2-D XOR: Weight-space density for stochastic gradient descent computed by numerical integration of the Kolmogorov equation. Bottom right graph shows the path taken using true (batch) gradient descent where each spike represents the deterministic position of the weight at one instant in time.

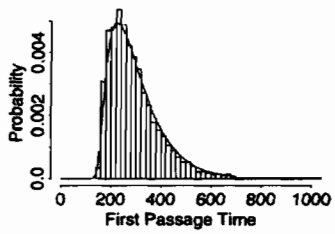


Figure 2.6: 1-D XOR. Simulated (histogram) and theoretical (solid line) distributions of first passage times for the cost function in (2.3) with $v_0 = 1.2$, $\epsilon = .1$, and $\mu = .025$.

time in a local minimum (the right spike) before moving on to the global minimum. For comparison, the bottom right graph shows the path (all 100 timesteps) taken by the weights during true gradient descent (i.e. batch) training. True gradient descent is deterministic so the "density" at each time-step is a delta function. After 100 timesteps, the weights trained by true gradient descent are trapped in a local minimum.

2.4.2 The XOR Problem: First Passage Times

For the cost function shown in Figure 2.3, we calculate the distribution of first passage times for networks, initialized at $v = 1.2$, entering within $\epsilon = 0.1$ of the global optimum at $v = 0$. For this example we numerically integrate the backward Fokker-Planck equation given in (2.36). In Figure 2.6 we compare the theoretical predictions (solid line) with simulations from an ensemble of 10,000 networks (histogram) initialized at $v = 1.2$. For this example the agreement is good at the small learning rate ($\mu = 0.025$) used, but degrades for larger μ as higher order terms in the expansion (2.35) become significant.

Figure 2.7a displays another 1-dimensional subspace for the XOR problem. Figure 2.7b compares simulations (histogram) with the Fokker-Planck solution (dashed). Here, the Fokker-Planck solution is quite poor because the steepness of the cost function results in large contributions from higher order terms in (2.35). We also display in Figure 2.7b the exact density (solid) computed from the recursion in (2.31). As expected, the exact and simulated densities agree very well.

2.4.3 Competitive Learning: First Passage Times and Basin Hopping

As a final example of first passage times, we consider competitive learning with two 2-D weight vectors symmetrically placed about the center of a rectangle. Inputs are uniformly distributed in a rectangle of width 1.1 and height 1. Figure 2.8 displays the

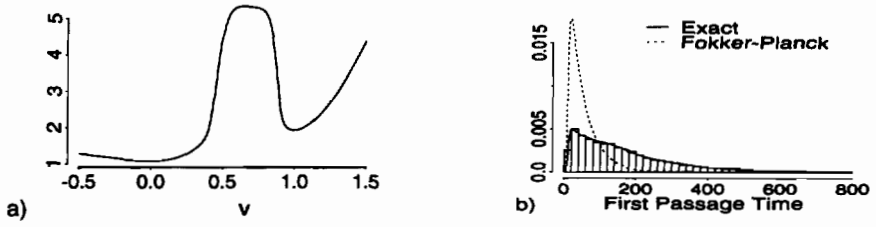


Figure 2.7: 1-D XOR - second example. a) Cost function. b) Simulated (histogram) and theoretical (lines) distributions of first passage times with $v_0 = 1.0$, $\epsilon = .1$, and $\mu = .05$.

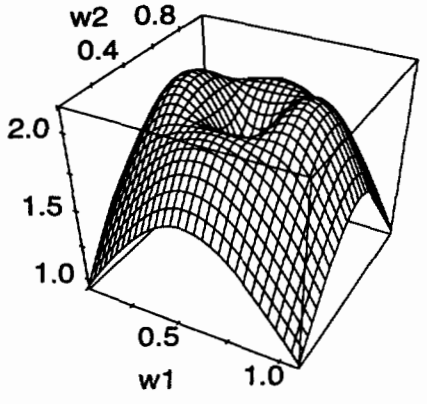


Figure 2.8: Competitive Learning. Inverted log of cost function.

log of the cost function inverted so as to better see that there are two global and two local optima.

Figure 2.9a shows a sample path with weights started near the local optimum (circles) and switching to hover around the global optimum. The measured and predicted (from numerical integration of (2.31)) distribution of times required to first pass within a distance $\epsilon = 0.1$ of the global optimum is shown in Figure 2.9b.

2.5 Summary

In this chapter we have studied the time-evolution of the weight density during learning using the Kramers-Moyal expansion derived either from the Kolmogorov equation or a master equation. We discuss how to truncate the Kramers-Moyal equation to obtain approximate solutions to the full expansion. We present two approaches for truncating. The naive approach is to view the Kramers-Moyal expansion as an expansion in the

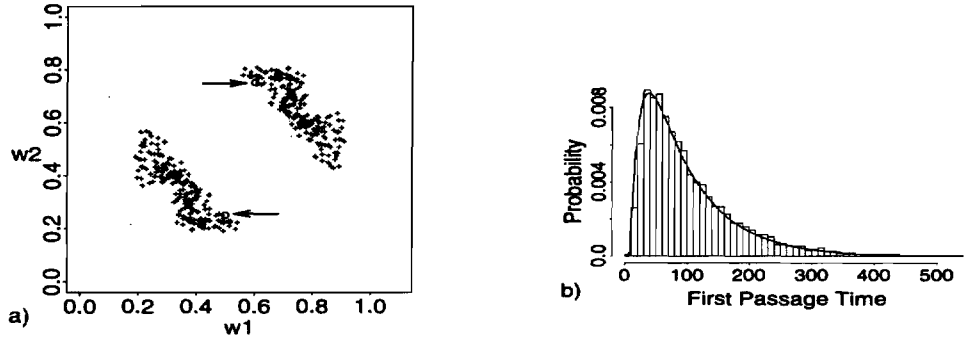


Figure 2.9: Competitive Learning. a) Data (small dots) and sample weight path (crosses). Arrows point to initial weight configuration (circles). b) First passage times. $\epsilon = .1$ and $\mu = .05$.

small parameter μ and to truncate to some order in μ. For example, truncating to second order results in the nonlinear Fokker-Planck equation (2.15) which surprisingly does a good job at predicting the complex time evolution *across basins* of the density for the 1-dimensional XOR problem when the learning rate is small.

The argument against truncating in this manner is that the density implicitly depends on μ so that the order of the truncated terms can not be determined strictly by the explicit power of μ. The small noise *ansatz* in (2.16) was then introduced as a way to explicitly model the μ dependence. This results in a deterministic equation for the average trajectory and a *linear* Fokker-Planck equation for the behavior of the noise about this trajectory.

Heskes used the results from the small noise expansion to argue that any nonlinear behavior present in the nonlinear Fokker-Planck equation can only be spurious. However, our examples with the XOR problem clearly show that much of the nonlinear behavior is being captured by the nonlinear Fokker-Planck equation. Despite this, we do believe that to properly truncate the Kramers-Moyal equation the μ dependence must be made explicit. The next chapter presents another approach for doing this that makes it possible to compute solutions beyond the lowest order.

Chapter 3

Perturbation Analysis for LMS

In this chapter we use perturbation theory to explicitly model the μ dependence in the Kramers-Moyal expansion for the *equilibrium* density $P_e(\omega) \equiv P(\omega, \infty)$. This technique enables us to recursively compute any finite order approximation of $P(\omega)$ in terms of the lower order approximations. Thus, unlike the small noise expansion where equations above the second order appear difficult to solve, this technique makes it possible to obtain equilibrium solutions to any finite order. In the first section we describe the general technique[1]. We next illustrate the technique by computing the Least Mean Square (LMS) equilibrium densities. We find that the lowest order solutions correspond exactly to the solution obtained using the small noise expansion in Appendix D.

3.1 Perturbation Theory

The Kramers-Moyal expansion (2.5) in one-dimension can be written as

$$\begin{aligned} \tau \ \partial_t P(\omega,t) &= \sum_{i=1}^{\infty} \frac{(-\mu)^i}{i!} \frac{\partial^i}{\partial \omega^i} \ (\ \langle H^i(\omega,x)\rangle_x \, P(\omega,t) \ ) \\ &= \mathcal{L}_{KM} \, P(\omega,t). \end{aligned} \tag{3.1}$$

where $\mathcal{L}_{KM}$ is the differential operator implicitly defined by (3.1). At equilibrium the left hand side is zero so we have

$$0 = \mathcal{L}_{KM} \, P_e(\omega). \tag{3.2}$$

Equation (3.2) is a specific example of a problem of the general form $\mathcal{L}P = 0$ where $\mathcal{L}$ is some known operator. In this section we discuss a perturbative approach to solving such problems.

[1] Unpublished notes of T. K. Leen.

We first make the assumption that the operator $\mathcal{L}$ and the solution P can be written as a power series in some small parameter α

$$\mathcal{L} = \mathcal{L}_0 + \alpha\mathcal{L}_1 + \alpha^2\mathcal{L}_2 + \ldots \tag{3.3}$$

$$P = P^{(0)} + \alpha P^{(1)} + \alpha^2 P^{(2)} + \ldots \tag{3.4}$$

where the $\mathcal{L}_i$ and $P^{(i)}$ are independent of α. Substituting (3.3) and (3.4) into $\mathcal{L}P = 0$ we obtain

$$0 = \mathcal{L}_0 P^{(0)} + \alpha(\mathcal{L}_0 P^{(1)} + \mathcal{L}_1 P^{(0)}) + \alpha^2(\mathcal{L}_0 P^{(2)} + \mathcal{L}_1 P^{(1)} + \mathcal{L}_2 P^{(0)}) + \ldots \tag{3.5}$$

Since α is arbitrary, the above sum is zero only if each coefficient of α^i vanishes independently, that is,

$$0 = \mathcal{L}_0 P^{(0)} \tag{3.6}$$

$$0 = \mathcal{L}_0 P^{(1)} + \mathcal{L}_1 P^{(0)} \tag{3.7}$$

$$\vdots$$

$$0 = \sum_{k=0}^{i} \mathcal{L}_k P^{(i-k)}. \tag{3.8}$$

This set of equations is solved recursively as follows. First the eigenvalues λ and the eigenvectors P_λ and Q_λ of the eigensystems

$$\mathcal{L}_0 P_\lambda = -\lambda P_\lambda \tag{3.9}$$

$$\mathcal{L}_0^\dagger Q_\lambda = -\lambda Q_\lambda \tag{3.10}$$

are solved where $\mathcal{L}_0^\dagger$ is the adjoint[2] of $\mathcal{L}_0$ and where we require that $\{P_\lambda\}$ and $\{Q_\lambda\}$ each form a complete set. It can be shown that $(P_\lambda, Q_{\lambda'}) = \delta_{\lambda,\lambda'}$ where (a, b) denotes the inner product of a and b.

The solution of (3.6) is then $P^{(0)} = P_{\lambda=0}$. Subsequent $P^{(i)}$ are found by assuming that they can be written as a linear combination of P_λ,

$$P^{(i)} = \sum_\lambda a_\lambda^{(i)} P_\lambda. \tag{3.11}$$

[2]If L is an operator defined on a space S of measurable functions, then the adjoint $L^\dagger$ of L is defined by $(f, Lg) = (L^\dagger f, g)$, $\forall f, g \in S$. Here, (a, b) denotes the inner product of a and b.

Then, starting with (3.8):

$$\begin{aligned}\sum_{k=1}^{i} \mathcal{L}_k P^{(i-k)} &= -\mathcal{L}_0 P^{(i)} \\ &= -\sum_{\lambda} a_\lambda^{(i)} \mathcal{L}_0 P_\lambda \\ &= \sum_{\lambda} a_\lambda^{(i)} . \lambda P_\lambda\end{aligned}$$

Taking the inner product of both sides with $Q_{\lambda'}$ gives

$$\begin{aligned}\sum_{k=1}^{i} (\mathcal{L}_k P^{(i-k)}, Q_{\lambda'}) &= \sum_{\lambda} a_\lambda^{(i)} \lambda \, (P_\lambda, Q_{\lambda'}) \\ &= \sum_{\lambda} a_\lambda^{(i)} \lambda \, \delta_{\lambda\lambda'} \\ &= a_{\lambda'}^{(i)} \lambda' .\end{aligned}$$

Substituting λ for λ', solving for $a_\lambda^{(i)}$, and plugging back into (3.11) gives $P^{(i)}$ in terms of $P^{(k<i)}$,

$$P^{(i)} = \sum_{\lambda} a_\lambda^{(i)} P_\lambda = \sum_{\lambda} \frac{P_\lambda}{\lambda} \sum_{k=1}^{i} (\mathcal{L}_k P^{(i-k)}, Q_\lambda) . \tag{3.12}$$

3.2 Computing the LMS Equilibrium Density

We now use the above method to compute the LMS equilibrium density. The first section below describes the LMS algorithm.

3.2.1 The LMS Algorithm

The stochastic LMS learning algorithm is given by

$$\omega(t+1) \;=\; \omega(t) \;+\; \mu \nabla_\omega \mathcal{E}(\omega(t), z(t)) \tag{3.13}$$

where $\omega(t) \in \mathbb{R}^m$ is the weight and $z(t) = \{x(t), d(t)\}$ is the input/target pair at time t. The cost function is given by the squared error

$$\mathcal{E}(\omega(t), z(t)) = \frac{1}{2}(d(t) - \omega(t) \cdot x(t))^2. \tag{3.14}$$

We assume that the training data is generated according to a "signal plus noise" model, that is,

$$d(t) = \omega^* \cdot x(t) + \epsilon(t) \tag{3.15}$$

where ω^* is the optimal weight vector and $\epsilon(t)$ is i.i.d. noise with zero mean and variance σ_ϵ^2. Defining the weight error vector as $v(t) \equiv \omega(t) - \omega^*$, we obtain the weight error update equation

$$v(t+1) = v(t) - \mu\, H[v(t), z(t)] \tag{3.16}$$

with

$$H[v(t), z(t)] = \nabla_v \mathcal{E}(v(t), z(t)) = (\epsilon(t) - x(t) \cdot v(t))\, x(t). \tag{3.17}$$

3.2.2 Perturbative Approximation of the Equilibrium Density

Following the procedure in Section §3.1, we want to solve for successive approximations of $P_e(v)$ in

$$\begin{aligned} 0 &= \mathcal{L}_{KM}\, P_e(v) = \sum_{i=1}^{\infty} \frac{(-\mu)^i}{i!} \frac{\partial^i}{\partial v^i} \left(\langle H^i(v,z) \rangle_z P_e(v) \right) \\ &= \mu \left(-\frac{\partial}{\partial v} \langle H(v,z) \rangle_z + \mu \frac{1}{2!} \frac{\partial^2}{\partial v^2} \langle H^2(v,z) \rangle_z + \mu^2 \frac{(-1)}{3!} \frac{\partial^3}{\partial v^3} \langle H^3(v,z) \rangle_z + \cdots \right) P_e(v) \end{aligned} \tag{3.18}$$

where H is given by (3.17) and $\langle \cdot \rangle_z$ denotes expectation over x and ϵ. The first step is to write (3.18) as $\mathcal{L}P = 0$ where $\mathcal{L}$ and P have the form given by equations (3.3) and (3.4). A natural choice would be to let $\alpha = \mu$, $\mathcal{L} = \mathcal{L}_{KM}/\mu$, and

$$\mathcal{L}_{i-1} = \frac{(-1)^i}{i!} \frac{\partial^i}{\partial v^i} \langle H^i(v,z) \rangle_z. \tag{3.19}$$

However, there is a better choice for LMS that results in an $\mathcal{L}_0$ whose eigensystem is completely known. We first expand H^i in equation (3.17) to give

$$\begin{aligned} \frac{1}{\mu} \mathcal{L}_{KM}\, P_e(v) &= \frac{1}{\mu} \sum_{i=1}^{\infty} \frac{(-\mu)^i}{i!} \frac{\partial^i}{\partial v^i} \left(\langle (\epsilon - x\, v)^i\, x^i \rangle\, P_e(v) \right) \\ &= \frac{1}{\mu} \sum_{i=1}^{\infty} \frac{(-\mu)^i}{i!} \frac{\partial^i}{\partial v^i} \left(\sum_{k=0}^{i} \binom{i}{k} (-1)^{i-k} \left\langle \epsilon^k \right\rangle \left\langle x^{2i-k} \right\rangle v^{i-k} \right). \end{aligned} \tag{3.20}$$

We next make the change of variable[3]

$$y = \frac{v}{\sqrt{\mu \sigma_\epsilon^2}} \tag{3.21}$$

[3]It is probably not accidental that this corresponds up to a constant to the same change of variable used in the small noise expansion. Note, at equilibrium, $\phi = 0$.

so that (3.20) becomes after simplification

$$0 = \frac{1}{\mu}\mathcal{L}_{KM}\, P_e(v) = \sum_{m=2}^{\infty} (-\sqrt{\mu})^{m-2} \langle x^m \rangle \sum_{i=\lceil \frac{m}{2} \rceil}^{m} \binom{i}{2i-m} \frac{\langle \epsilon^{2i-m} \rangle}{i!\, \sigma_\epsilon^{2i-m}} \frac{\partial^i}{\partial y^i} \left(y^{m-i} P_e(y) \right) \tag{3.22}$$

Note that since $\langle \epsilon \rangle = 0$, the $m = 1$ term is 0. Thus, the lowest order term ($m = 2$) is

$$\mathcal{L}_0\, P_e(v) = R \left(\frac{\partial}{\partial y} y + \frac{1}{2} \frac{\partial^2}{\partial y^2} \right) P_e(y) \tag{3.23}$$

where $R \equiv \langle x^2 \rangle$ is the Hessian of the cost function for LMS averaged over the inputs. The operator $\mathcal{L}_0$ in (3.23) is the Ornstein-Uhlenbeck operator [Gar90, Ris89] whose eigensystem is known.

Now (3.22) is an expansion in $\sqrt{\mu}$ rather than μ therefore it seems more appropriate to set $\alpha = \sqrt{\mu}$. However, in the case of gaussian inputs and noise that we examine next, the odd powered terms vanish so that equating α with μ is still appropriate.

3.2.3 Gaussian Inputs and Noise

If we assume that x and ϵ are gaussian distributed with zero mean then all odd moments are 0 and even moments are given by

$$\langle x^{2n} \rangle = \frac{(2n)!\, R^n}{n!\, 2^n} \text{ and } \langle \epsilon^{2n} \rangle = \frac{(2n)!\, \sigma_\epsilon^{2n}}{n!\, 2^n}. \tag{3.24}$$

Equation (3.22) then simplifies to

$$0 = \sum_{l=1}^{\infty} (\mu R)^l \sum_{j=l}^{2l} 2^{-j} \binom{j}{l} \binom{2l}{j} \frac{\partial^j}{\partial y^j} \left(y^{2l-j} P_e(y) \right). \tag{3.25}$$

Thus, we can write

$$0 = \frac{1}{\mu}\mathcal{L}_{KM}\, P_e(y) = (\mathcal{L}_0 + \mu \mathcal{L}_1 + \ldots)\; P_e(y)$$

where

$$\mathcal{L}_{l-1} = R^l \sum_{j=l}^{2l} 2^{-j} \binom{j}{l} \binom{2l}{j} \frac{\partial^j}{\partial y^j} y^{2l-j}. \tag{3.26}$$

The first three $\mathcal{L}_l$ are

$$\begin{aligned}
\mathcal{L}_0 &= \frac{1}{2} R \left(2\, \partial_y y + \partial_y^2 \right) \\
\mathcal{L}_1 &= \frac{3}{8} R^2 \left(4\, \partial_y^2 y^2 + 4\, \partial_y^3 y + \partial_y^4 \right) \\
\mathcal{L}_2 &= \frac{5}{16} R^3 \left(8\, \partial_y^3 y^3 + 12\, \partial_y^4 y^2 + 6\, \partial_y^5 y + \partial_y^6 \right)
\end{aligned}$$

where $\partial_y^i \equiv \frac{\partial^i}{\partial y^i}$.

The eigenfunctions and eigenvalues of $\mathcal{L}^{(0)}$ are [Gar90]

$$P_{\lambda_k}(y) = \frac{e^{-y^2}}{\sqrt{\pi\, 2^k\, k!}}\, \mathcal{H}_k(y) \tag{3.27}$$

$$\lambda_k = kR \tag{3.28}$$

where $\mathcal{H}_k(y)$ is the k^{th} Hermite polynomial. The adjoint of $\mathcal{L}_0$ is the associated backward equation

$$\mathcal{L}_0^\dagger = R\left(y\,\partial_y + \frac{1}{2}\partial_y^2\right) \tag{3.29}$$

with eigenfunctions

$$Q_{\lambda_k}(y) = \frac{1}{\sqrt{2^k\, k!}}\, \mathcal{H}_k(y)\,. \tag{3.30}$$

Thus $P_e^{(0)}(y) = P_{\lambda=0}(y)$ is a gaussian with variance $\frac{1}{2}$, independent of μ. Higher order corrections, $P_e^{(i)}(y)$, can be obtained from (3.12). The first several are

$$\begin{aligned}
P_e^{(0)}(y) &= \frac{1}{\sqrt{\pi}}\, e^{-y^2} \qquad \Rightarrow \qquad P_e^{(0)}(v) = \frac{1}{\sqrt{\pi\mu\sigma_\epsilon^2}}\, e^{-v^2/\mu\sigma_\epsilon^2} \\
P_e^{(1)}(y) &= \frac{3}{4}\, R\,(-1+2y^2)\, P_e^{(0)}(y) \qquad \Rightarrow \qquad P_e^{(1)}(v) = \frac{3}{4}\, R\,(-1+\frac{2v^2}{\mu\sigma_\epsilon^2})\, P_e^{(0)}(v) \\
P_e^{(2)}(y) &= \frac{9}{32}\, R^2\,(-1-4y^2+4y^4)\, P_e^{(0)}(y) \qquad \Rightarrow \\
&\qquad\qquad P_e^{(2)}(y) = \frac{9}{32}\, R^2\,(-1-\frac{4v^2}{\mu\sigma_\epsilon^2}+\frac{4v^4}{\mu^2\sigma_\epsilon^4})\, P_e^{(0)}(v).
\end{aligned} \tag{3.31}$$

3.3 Comparison of Approaches: An Example

So far we have discussed three approaches for approximating weight-space probability densities; the nonlinear Fokker-Planck equation, the small noise expansion, and the perturbative expansion. In this section we apply each of these techniques to computing the LMS equilibrium density and compare the results with the exact density obtained from simulation. We assume gaussian zero mean inputs and noise, and a constant learning rate.

The i^{th} order perturbative density we define to be

$$P_{e,i}^{(pert)}(v) \equiv P_e^{(0)}(v) + \mu P_e^{(1)}(v) + \cdots + \mu^i P_e^{(i)}(v) \tag{3.32}$$

where the first several $P_e^{(k)}(v)$ are given in (3.31).

We define $P_e^{(sn)}(v)$ to be the LMS equilibrium density obtained from the lowest order truncation of small noise expansion in (2.20). At equilibrium ($t = \infty$), $\phi = 0$ and the lefthand side of (2.20) is zero,

$$0 = \mathcal{L}_{sn} P_e^{(sn)}(\xi) \equiv R\partial_\xi(\xi\, P_e^{(sn)}(\xi)) + \frac{R\sigma_\epsilon^2}{2}\partial_\xi^2 P_e^{(sn)}(\xi). \tag{3.33}$$

Expressed in v coordinates, we find that $\mathcal{L}_{sn}$ is identical to $\mu\mathcal{L}_0$ implying that $P_e^{(sn)}(v) = P_e^{(0)}(v)$. This is not surprising since at equilibrium ($\phi = 0$) both the perturbative and small noise expansions are derived from essentially the identical transformation: $y = \sqrt{\mu\sigma_\epsilon^2}v$ versus $\xi = \sqrt{\mu}v$.

The nonlinear Fokker-Planck equation refers to (2.15), i.e., the truncated Kramers-Moyal expansion which, for LMS at equilibrium with gaussian inputs, is

$$0 = \mathcal{L}_{FP} P_e^{(FP)}(v) = \mu R\partial_v\{v\, P_e^{(FP)}(v)\} + \frac{\mu^2}{2}\partial_v^2\left\{(R\sigma_\epsilon^2 + 3R^2v^2)\, P_e^{(FP)}(v)\right\}. \tag{3.34}$$

It has the closed form solution [LM93]

$$P_e^{(FP)}(v) = \frac{1}{K}\left(1 + \frac{3R}{\sigma_\epsilon^2}v^2\right)^{-(1+\frac{1}{3\mu R})} \qquad \text{with } K = B\left(\frac{1}{2}, \frac{1}{2} + \frac{1}{3\mu R}\right) \tag{3.35}$$

where $B(\cdot,\cdot)$ is the Beta function. $\mathcal{L}_{FP}$ is somewhere between the 0^{th} and 1^{st} order perturbative operators. Writing out the terms separately we have

$$\mathcal{L}_{FP}(v) = \mu R\partial_v v + \frac{\mu^2 R\sigma_\epsilon^2}{2}\partial_v^2 + \frac{\mu^2 3R^2}{2}\partial_v^2 v^2. \tag{3.36}$$

The first two terms correspond precisely $\mu\mathcal{L}_0(v)$, the 0^{th} order perturbative operator. The third term corresponds to *part* of $\mu^2\mathcal{L}_1(v)$. Note that $\mathcal{L}_1$ also contains terms with 3^{rd} and 4^{th} order derivatives. Thus the nonlinear Fokker-Planck equation is capturing some but not all of the $\mathcal{O}(\mu^2)$ effects.

Figure 3.1a compares $P_e^{(FP)}$ (dotted) and $P_{e,0}^{(pert)}(v)$ (solid) with the simulated density (dashed) for gaussian inputs and noise with $\mu = .05$, $\sigma_\epsilon^2 = 1$, and $R = 4$. Figure 3.1b does the same except that $P_{e,0}^{(pert)}(v)$ has been replaced with $P_{e,1}^{(pert)}(v)$. Recall that $P_e^{(sn)}(v)$, the lowest order solution from the small noise expansion is identical with $P_{e,0}^{(pert)}(v)$. For other choices of inputs, the behavior was qualitatively the same.

Figure 3.1a shows that the 0^{th} order perturbative solution does a better job at fitting the true density than the nonlinear Fokker-Planck. Thus, we do more harm than good

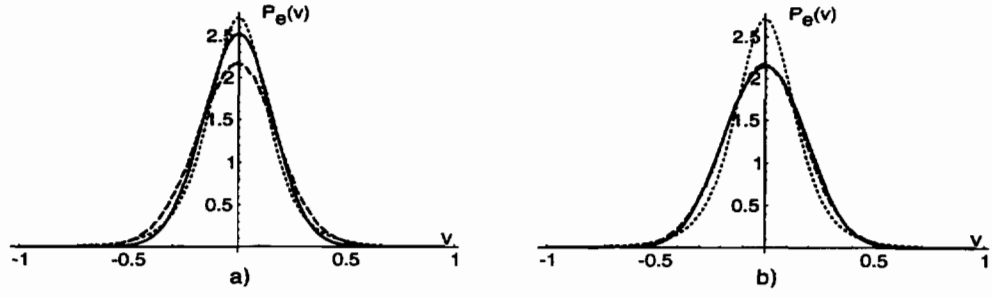


Figure 3.1: a) Comparison of densities from Fokker-Planck (dotted), simulation (dashed), and 0^{th} order perturbative density (solid) for gaussian inputs and gaussian noise with $\mu = .05$, $\sigma_\epsilon^2 = 1$, and $R = 4$. b) Same as a) but with 1^{st} order perturbative density. Note that for all k, the k^{th} order perturbative density is identical to the k^{th} order density from the small noise expansion.

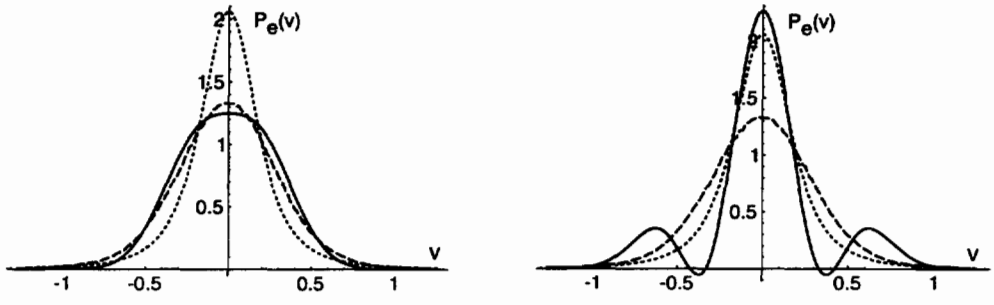


Figure 3.2: a) Comparison of densities from Fokker-Planck (dotted), simulation (dashed), and 1^{st} order perturbative density (solid) for gaussian inputs and gaussian noise with $\mu = .10$, $\sigma_\epsilon^2 = 1$, and $R = 4$. b) Same as a) but with 4^{th} order perturbative density.

by including only part of the $\mathcal{O}(\mu^2)$ effects. This is similar to the conclusion drawn from the pre-equilibrium behavior of LMS derived in Appendix D.

From Figure 3.1b we see that $P_{e,1}^{(pert)}(v)$ is almost a perfect fit with the simulated density. As higher order corrections are added the solution improves and then begins to degrade. Oscillations appear and there are regions in the tails where the density is negative (consistent with the Pawula theorem). This is particularly noticeable for larger learning rates. For example, Figures 3.2a and 3.2b display the 1^{st} and 4^{th} order perturbative densities, respectively, compared against the simulated and Fokker-Planck densities for $\mu = .10$. While $P_{e,1}^{(pert)}$ is a much better fit than the $P_e^{(FP)}$, $P_{e,4}^{(pert)}$ is much worse and exhibits oscillations. Unfortunately, it is not clear how to tell *a priori* how many corrections produce the best fit.

Moments of the Perturbative Densities

In the previous section we saw that there was a point at which adding higher order perturbative corrections only degraded the solution. However, we show in this section that the estimates of the moments continue to improve as higher order corrections are included even though the density seems to degrade.

The n^{th} moment of v can be computed from the perturbative densities as follows

$$\langle v^n \rangle = \int dv\, v^n P_e(v) = \langle v^n \rangle_0 + \mu \langle v^n \rangle_1 + \mu^2 \langle v^n \rangle_2 \cdots \tag{3.37}$$

where

$$\langle v^n \rangle_k \equiv \int dv\, v^n P_e^{(k)}(v).$$

Equation (3.37) represents a power series expansion in μ for the n^{th} moment of v. Using the densities given in (3.31) we find that

$$\langle v^2 \rangle_0 = \frac{\mu\sigma_\epsilon^2}{2} \quad \langle v^2 \rangle_1 = \frac{3\mu\sigma_\epsilon^2 R}{4} \quad \langle v^2 \rangle_2 = \frac{9\mu\sigma_\epsilon^2 R^2}{8} \tag{3.38}$$

$$\langle v^4 \rangle_0 = \frac{3\mu^2\sigma_\epsilon^4}{4} \quad \langle v^4 \rangle_1 = \frac{9\mu^2\sigma_\epsilon^4 R}{4} \quad \langle v^4 \rangle_2 = \frac{81\mu^2\sigma_\epsilon^4 R^2}{16}. \tag{3.39}$$

These terms can be compared to the series expansion obtained directly from the expansion of the exact moments. The exact moments for the LMS weight error equilibrium density are easily calculated directly from the update equation

$$v(t+1) = v(t) + \mu\,(\epsilon - v\,x)\,x.$$

Raising each side to the n^{th} power, taking expectations, setting $\langle v^n \rangle = \lim_{t\to\infty} \langle v^n(t) \rangle = \lim_{t\to\infty} \langle v^n(t+1) \rangle$, and solving for $\langle v^n \rangle$ gives

$$\langle v^n \rangle = -\sum_{m=0}^{n-1} \langle v^m \rangle \langle \epsilon^{n-m} \rangle \sum_{k=n-m}^{n} \binom{n}{k} \binom{k}{m+k-n} \mu^k (-1)^{m+k-n} \left\langle x^{2k+m-n} \right\rangle \div \sum_{i=1}^{n} \binom{n}{i} (-\mu)^i \langle x^{2i} \rangle. \tag{3.40}$$

Note that, since we are using gaussian inputs and noise, all odd moments of v will be zero. The first two nonzero moments computed from (3.40) are

$$\langle v^2 \rangle = \frac{\mu\sigma_\epsilon^2}{2 - 3\mu R} = \mu\sigma_\epsilon^2 \left(\frac{1}{2} + \frac{3\mu R}{4} + \frac{9(\mu R)^2}{8} + \ldots \right) \tag{3.41}$$

$$\begin{aligned}\langle v^4 \rangle &= \frac{3\mu^2\sigma_\epsilon^4(2-6\mu R+21\mu^2R^2)}{8-48\mu R+174\mu^2R^2-390\mu^3R^3+315\mu^4R^4} \\ &= 3\mu^2\sigma_\epsilon^4\left(\frac{1}{4}+\frac{3\mu R}{4}+\frac{27(\mu R)^2}{16}+6(\mu R)^3+\ldots\right). \qquad (3.42)\end{aligned}$$

The first three terms in these expansions exactly correspond to first three terms obtained by inserting (3.39) into (3.37) with $n = 2$ and 4. It is difficult to show that the moments agree in general because the $P_e^{(k)}(v)$ are computed recursively, however, we believe this to be the case.

Thus, even though the densities obtained from the perturbative expansion seem to degrade as higher order terms are added, the perturbative moments continue to improve and, in fact, correspond to the expansion of the exact moments.

3.4 Summary

In this chapter we presented a method for recursively computing equilibrium weight-space densities to any finite order. We applied it to LMS and compared with previous approaches. For the particular transformation used, the perturbative densities turned out to exactly correspond to the densities computed using the small noise expansion. The density obtained by just truncating the KME to second order contained some but not all the $\mathcal{O}(\mu^2)$ dependence.

As the first several perturbative corrections were included, the fit to the simulated density clearly improved. However, as even higher order terms were included the density estimates started to degrade even the estimate of the moments continued to improve. Oscillations appeared and the density even became negative in regions. We currently have no method of *a priori* knowing which order solution provides the best fit.

Chapter 4

Convergence Regimes for Annealed Learning

In this chapter we study the convergence behavior of stochastic learning *with an annealed learning rate*. We characterize the convergence rate by the time rate of change of the squared weight error. We first derive a small noise expansion for the weight density and use it to compute the evolution of the squared weight error for LMS. Analysis of nonlinear problems is more difficult. However, we note that learning rates are typically annealed only at late times *after* the weights have settled into a particular basin of attraction. By assuming that the weights are already fluctuating about some local optimum, we are able to derive asymptotic convergence results for nonlinear as well as linear networks.

Our analysis shows for annealed learning that the convergence behavior can be classified into two regimes based on whether the learning rate is above or below a specified critical value. When below, the convergence rate is suboptimal. When above, the convergence rate is optimal, i.e. proportional to $\frac{1}{t}$.

4.1 Small Noise Expansion

The derivation for the small noise expansion for annealed learning is similar to that for constant learning so that most of the details have been relegated to Appendix C. We use the annealing schedule $\mu(t) = \frac{\mu_0}{t}$ so that the *ansatz* becomes

$$v = \phi(t) + \sqrt{\frac{\mu_0}{t}}\xi. \tag{4.1}$$

With this change of variable, the KME reduces to the set of equations (see Appendix C)

$$\frac{d\phi}{dt} = \frac{\mu_0}{\tau}\frac{\alpha_1^{(0)}(\phi)}{t} \tag{4.2}$$

$$\frac{\partial \mathcal{P}(\xi,t)}{\partial t} = -\frac{1}{2\,t}\,\frac{\partial \xi\,\mathcal{P}(\xi,t)}{\partial \xi} + \frac{1}{\tau}\sum_{m=2}^{\infty}\left(\sqrt{\frac{\mu_0}{t}}\right)^m \sum_{i=1}^{m}\frac{(-1)^i}{i!\,(m-i)!}\;\alpha_i^{(m-i)}(\phi(t))\,\frac{\partial^i}{\partial \xi^i}\left(\xi^{m-i}\,\mathcal{P}(\xi,t)\right). \tag{4.3}$$

Keeping the lowest order noise terms on the right side of (4.3) leaves

$$\frac{\partial \mathcal{P}(\xi,t)}{\partial t} = -\frac{1}{2\,t}\,\frac{\partial \xi\,\mathcal{P}(\xi,t)}{\partial \xi} \tag{4.4}$$

which is independent of μ and so fullfills the original *ansatz.* If we also include the next lowest order term[1] then we have

$$\frac{\partial \mathcal{P}(\xi,t)}{\partial t} = \frac{1}{2\,t}\left\{-\left(1+\frac{2\mu_0\,\alpha_1^{(1)}}{\tau}\right)\frac{\partial \xi\,\mathcal{P}(\xi,t)}{\partial \xi} + \frac{\mu_0\,\alpha_2^{(0)}}{\tau}\,\frac{\partial^2 \mathcal{P}}{\partial \xi^2}\right\}. \tag{4.5}$$

From (4.5) we obtain

$$\langle \xi \rangle = \langle \xi_0 \rangle e^{\frac{\gamma(t_0,t)}{2}} \tag{4.6}$$

$$\langle \xi^2 \rangle = \langle \xi_0^2 \rangle\, e^{\gamma(t_0,t)} + \mu_0 \int_{t_0}^{t} \frac{dq}{q}\, e^{\gamma(q,t)}\,\alpha_2^{(0)}(\phi(q)) \tag{4.7}$$

where

$$\gamma(t_1,t_2) \equiv \int_{t_1}^{t_2} \frac{dq}{q}\,\left[1+2\,\mu_0\,\alpha_1^{(1)}(\phi(q))\,\right],$$

and where $\langle \xi_0 \rangle$ and $\langle \xi_0^2 \rangle$ are $\langle \xi \rangle$ and $\langle \xi^2 \rangle$ at $t = 0$, respectively. We next apply these results to LMS.

4.1.1 LMS

For LMS with update equation

$$v(t+1) = v(t) + \frac{\mu_0}{t}H(v,t) \;= v(t) + \frac{\mu_0}{t}\;(\epsilon - v(t)\,x(t))\,x(t) \tag{4.8}$$

the $\alpha_i^{(j)}$ coefficients are

$$\begin{aligned} \alpha_1^{(0)} &= \langle H(\phi,t)\rangle_x = -R\phi \\ \alpha_1^{(1)} &= \left\langle \tfrac{\partial H(\phi,t)}{\partial v}\right\rangle_x = -R \\ \alpha_2^{(0)} &= \langle H^2(\phi,t)\rangle_x = R\sigma_\epsilon^2 + S\phi^2. \end{aligned} \tag{4.9}$$

[1] If we assume that τ scales as μ then this term would be included in the lowest (i.e. $\mathcal{O}(1)$) term.

Thus, the equations of motion become

$$\begin{aligned}
\phi(t) &= \phi_0 \left(\frac{t_0}{t}\right)^{\frac{\mu_0 R}{\tau}} \\
e^{\gamma(t_1,t_2)} &= \exp \int_{t_1}^{t_2} \frac{dq}{q} \left[1 - \frac{2\,\mu_0 R}{\tau}\right] = \left(\frac{t_1}{t_2}\right)^{\frac{2\mu_0 R}{\tau}-1} \\
\langle \xi \rangle &= \langle \xi_0 \rangle \left(\frac{t_0}{t}\right)^{\frac{\mu_0 R}{\tau}-\frac{1}{2}} \\
\langle \xi^2 \rangle &= \langle \xi_0^2 \rangle \left(\frac{t_0}{t}\right)^{\frac{2\mu_0 R}{\tau}-1} + \frac{\mu_0}{\tau} \int_{t_0}^{t} \frac{dq}{q} \left(\frac{q}{t}\right)^{\frac{2\mu_0 R}{\tau}-1} \left(R\sigma_\epsilon^2 + S\phi^2\right) \\
&= \langle \xi_0^2 \rangle \left(\frac{t_0}{t}\right)^{\frac{2\mu_0 R}{\tau}-1} + \frac{\mu_0 R \sigma_\epsilon^2}{2\mu_0 R - \tau} \left(1 - \left(\frac{t_0}{t}\right)^{\frac{2\mu_0 R}{\tau}-1}\right) - \\
&\qquad \frac{\mu_0 S\, \phi_0^2}{2\,\tau} \left(\frac{t_0}{t}\right)^{\frac{2\mu_0 R}{\tau}-1} \left(1 - \frac{t_0}{t}\right).
\end{aligned}$$

If we set $\langle \xi_0 \rangle = \langle \xi_0^2 \rangle = 0$ and transform back to v we obtain

$$\langle v \rangle = \phi + \sqrt{\frac{\mu_0}{t}}\,\langle \xi \rangle = \langle v_0 \rangle \left(\frac{t_0}{t}\right)^{\frac{\mu_0 R}{\tau}} \tag{4.10}$$

$$\begin{aligned}
\langle v^2 \rangle &= \phi^2 + \frac{\mu_0}{t} \langle \xi^2 \rangle \\
&= \langle v_0^2 \rangle \left(\frac{t_0}{t}\right)^{\frac{2\mu_0 R}{\tau}} + \frac{\mu_0^2 R \sigma_\epsilon^2}{2\mu_0 R - \tau} \left[\frac{1}{t} - \frac{1}{t_0}\left(\frac{t_0}{t}\right)^{\frac{2\mu_0 R}{\tau}}\right] + \\
&\qquad \frac{\mu_0^2 S}{2\,\tau} \langle v_0^2 \rangle \left(\frac{t_0}{t}\right)^{\frac{2\mu_0 R}{\tau}} \left[\frac{1}{t_0} - \frac{1}{t}\right]
\end{aligned} \tag{4.11}$$

where $\langle v_0 \rangle$ and $\langle v_0^2 \rangle$ are the expected mean and variance at the initial time t_0.

Figure 4.1 compares simulations (dashed) with the theoretical formula (solid) given in (4.11). In the simulations, we first had the networks settle to equilibrium before annealing was started. The graphs in Figure 4.1 display only the results once annealing was started. For the theoretical curves, $t_0 = 1$ and $\langle v_0^2 \rangle$ was set equal to the weight variance of the simulations when annealing was just turned on. That is, we forced the curves to equal each other at $t = 1$. The agreement is good particularly for small μ_0. This is not surprising since we have kept only the terms of lowest order in μ_0 in the Kramers-Moyal expansion. The theoretical curves tend to roll over earlier than the simulations.

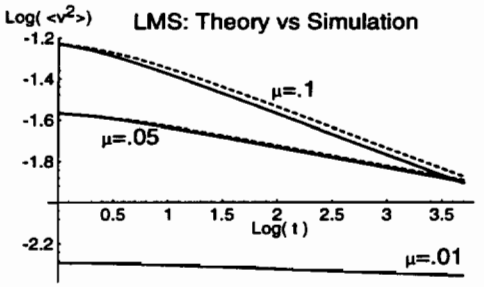


Figure 4.1: Simulations (dashed) vs theoretical (solid) predictions of the squared weight error of 1-D LMS for $R = 1$, $\sigma_\epsilon = 1$, $\tau = 1$, and $\mu_0 = .01, .05$, and 0.1. Simulations used 10000 networks.

4.2 Nonlinear Problems

Evaluating the even the lowest order equations of the small noise expansion is much more difficult for nonlinear problems; further simplification is required. We therefore make the reasonable assumption that annealing is turned on late in the training when the deterministic component is very close to a local optimum. The α coefficients can then be expanded about the minimum (ϕ=0)

$$\alpha_j^{(i)}(\phi) = \sum_{k=0}^{\infty} \frac{\phi^k}{k!} \left. \frac{d^k \alpha_j^{(i)}}{d\phi^k} \right|_{\phi=0} . \tag{4.12}$$

Keeping only the lowest order nonzero term gives

$$\alpha_1^{(0)} = \langle H(\phi, x) \rangle_x |_{\phi=0} \; \phi \equiv -R\,\phi \tag{4.13}$$

$$\alpha_1^{(1)} = \left. \frac{d \, \langle H(\phi, x) \rangle_x}{d\phi} \right|_{\phi=0} \equiv R \tag{4.14}$$

$$\alpha_2^{(0)} = \langle H^2(\phi, x) \rangle_x |_{\phi=0} \equiv D \tag{4.15}$$

where R and D are constants.

We now put equations (4.14) and (4.15) into equations (4.2), (4.6), and (4.7). For simplicity, we have set $\tau = 1$. Solving yields the equations of motion for the weight error at late times for the general nonlinear case

$$\langle v \rangle = \langle v_0 \rangle \left(\frac{t_0}{t} \right)^{\mu_0 R} \tag{4.16}$$

$$\langle v^2 \rangle = \langle v^2 \rangle_0 \left(\frac{t_0}{t} \right)^{2\mu_0 R} + \frac{\mu_0^2 D}{2\mu_0 R - 1} \left[\frac{1}{t} - \frac{1}{t_0} \left(\frac{t_0}{t} \right)^{2\mu_0 R} \right]$$

$$= \left(\langle v^2\rangle_0 + \frac{\mu_0^2 D}{2\mu_0 R - 1}\,\frac{1}{t_0}\right)\left(\frac{t_0}{t}\right)^{2\mu_0 R} + \frac{\mu_0^2 D}{2\mu_0 R - 1}\,\frac{1}{t}. \tag{4.17}$$

In [LO94, OL94], we also obtain (4.17) by using the Kramers-Moyal expansion more directly. Defining

$$C \equiv \int d^N v\ (v\,v^T)\ P(v,t)$$

where $v \in \mathbb{R}^N$ is the N-dimensional weight error, we then have

$$\frac{dC}{dt} = \int d^N v\ (v\,v^T)\ \frac{\partial P(v,t)}{\partial t} \tag{4.18}$$

$$= \int d^N v\ (v\,v^T)\ \left(\sum_{i=1}^{\infty} \left(\frac{(-\mu)^i}{t^i\ i!}\ \frac{\partial^i}{\partial v^i}\ \left(\ \langle H^i(v,x)\rangle_x\ P(v,t)\right)\right)\right) \tag{4.19}$$

$$= \frac{\mu_0}{t}\int d^N v\ P(v,t)\ \left\{v\ \left\langle H(v,x)^T\right\rangle_x + \left\langle H(v,x)\,v^T\right\rangle_x\right\} + \frac{\mu_0^2}{t^2}\int d^N v\ P(v,t)\ \left\langle H(v,x)\,H(v,x)^T\right\rangle_x. \tag{4.20}$$

Solving equation (4.20) by keeping only the lowest order nonzero terms in the expansion of $H(v,x)$ about the local minimum also yields (4.17).

Note that if we apply (4.17) to LMS we obtain

$$\langle v^2\rangle = \langle v_0^2\rangle\ \left(\frac{t_0}{t}\right)^{2\mu_0 R} + \frac{\mu_0^2 R\sigma_\epsilon^2}{2\mu_0 R - 1}\left[\frac{1}{t} - \frac{1}{t_0}\left(\frac{t_0}{t}\right)^{2\mu_0 R}\right] \tag{4.21}$$

instead of equation (4.11). This is because we have set $\alpha_2^{(0)} = D = R\sigma_\epsilon^2$ (only lowest order terms are kept) instead of $\alpha_2^{(0)} = R\sigma_\epsilon^2 + S\phi^2$. When ϕ is small this is a good approximation. Figure 4.2 compares the expected squared weight error from simulations for 1-D LMS versus theoretical predictions from equation (4.21). As can be seen, the agreement is good.

4.3 Asymptotic Convergence Regimes

Examination of (4.17) reveals that the late time convergence behavior falls into two regimes. If $2\mu_0 R < 1$, the first term on the right dominates so that the late time convergence rate is proportional to $\left(\frac{1}{t}\right)^{2\mu_0 R}$. Note that this is slower than $\frac{1}{t}$. If $2\mu_0 R > 1$, the second term dominates so that at late times the convergence rate is proportional to $\frac{1}{t}$. Thus, the fastest or "optimal" rate of convergence is proportional to $\frac{1}{t}$ and is achieved

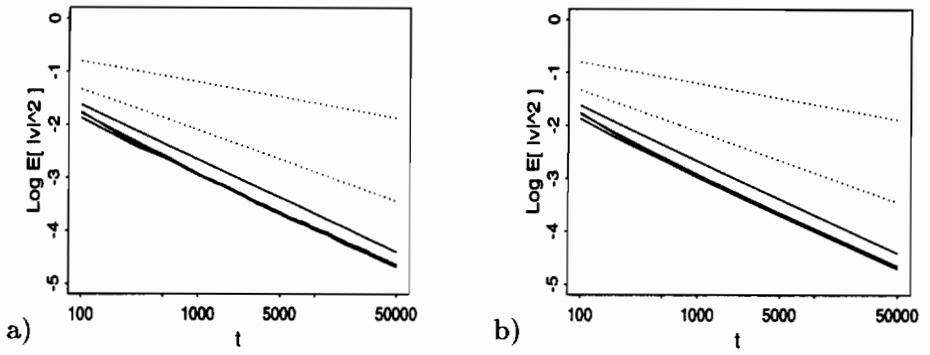


Figure 4.2: 1-D LMS: a) Simulation results from an ensemble of 2000 networks with $R = 1.0$ and $\mu = \mu_0/t$. b) Theoretical predictions. Curves correspond to (top to bottom) $\mu_0 = 0.2, 0.4, 0.6, 0.8, 1.0, 1.5$.

if $\mu_0 > \mu_{crit} \equiv \frac{1}{2R}$. Summarizing, the two regimes are

$$\text{Regime 1 } (\mu_0 < \mu_{crit}): \qquad \langle v^2 \rangle \sim \left(\frac{1}{t}\right)^{\mu_0/\mu_{crit}}$$

$$\text{Regime 2 } (\mu_0 > \mu_{crit}): \qquad \langle v^2 \rangle \sim \frac{1}{t}$$

where the convergence rate is suboptimal in regime 1 and optimal in regime 2. In regime 2, we define μ_{opt} to be the μ_0 that minimizes coefficient of $\frac{1}{t}$ in the expression for $\langle v^2 \rangle$. From (4.17) we obtain $\mu_{opt} = \frac{1}{R}$. These results are the same as those derived by Goldstein [Gol87].

The two regimes can be seen clearly for LMS in Figure 4.2. The dotted curves correspond to regime 1 where the learning rate is suboptimal. The solid curves correspond to regime 2.

4.4 Summary

We have shown that the small noise expansion can be used to obtain expressions for the time evolution of the weight error for stochastic learning with constant learning rate and annealed learning. The expressions for annealed learning reveal that late time

convergence rates can be classified into two regimes where convergence rate refers to the time rate of change of the weight error. For learning rates above a critical value, convergence is optimal, that is, proportional to $\frac{1}{t}$. The optimal learning rate is found to be R^{-1}.

Chapter 5

Stochastic Learning with Constant Learning Rate and Constant Momentum

The main focus of the remainder of this thesis is to examine methods for speeding stochastic learning. In this chapter we examine adding momentum to standard stochastic learning with a constant learning rate. We summarize previous results and then we attempt a small noise expansion for the weights. Most of the results are for linear networks due to the difficulty of evaluating nonlinear problems. Chapter 6 discusses momentum with *annealed learning* where we obtain asymptotic results for both linear and nonlinear networks. Chapters 7 and 8 then present an algorithm with annealed learning where the momentum is automatically adapted so as to obtain fast convergence.

5.1 Momentum

Learning is slow if the learning rate is not chosen well. Ideally, we want a learning rate matrix proportional to the inverse Hessian so that the stepsize is scaled according to the local curvature along the different eigendirections of the Hessian. However, many algorithms and speed-up techniques based on estimating curvature (e.g. conjugate gradient, quasi-Newton, etc) cannot be used stochastically because estimates of second order effects are too noisy. One simple alternative that has been observed to speed learning is called momentum.

In stochastic learning with momentum each weight update includes not only the standard update term but also an additional term (called the momentum term) proportional

to the *previous* weight update. The general form is

$$\omega(t+1) = \omega(t) + \mu\, H[\omega(t), x(t)] + \beta(\omega(t) - \omega(t-1)) \tag{5.1}$$

where H is the weight update function as before, μ is the learning rate (assumed constant in this chapter), and β is the momentum parameter which is a constant in the range $0 \leq \beta < 1$. We refer to this equation as learning with *constant* momentum because the parameter β is held fixed, in contrast to an adaptive β discussed later in the Chapters (7) and (8).

By expanding out the right side of (5.1) we find that adding momentum is equivalent to updating the weights based on an exponential average of H evaluated at past timesteps,

$$\omega(t+1) = \omega(t) + \mu \sum_{i=0}^{t} \beta^i\, H[\omega(t-i), x(t-i)] \tag{5.2}$$

where the weighting factor in this average is just β. Thus, momentum may not only speed learning but may also be an efficient method of smoothing out the noise in the stochastic updates as compared with the costly batch mode average over training set.

To develop some intuition about momentum we can look at momentum with gradient descent under simplified conditions. If $\beta^t \ll 1$ and if the gradient is not changing rapidly over the significant terms in the exponential average then the finite sum can be replaced by an infinite sum and the gradient can be pulled out to give

$$\begin{aligned} \mu_0 \sum_{i=0}^{t} \beta^i\, \nabla\mathcal{E}_{t-i} &\approx \mu\, \nabla\mathcal{E}_t \sum_{i=0}^{\infty} \beta^i \\ &= \frac{\mu}{1-\beta}\, \nabla\mathcal{E}_t \end{aligned} \tag{5.3}$$

where for simplicity we use the notation $\nabla\mathcal{E}_t \equiv \nabla\mathcal{E}(\omega(t), x(t))$. Equation (5.3) says that momentum effectively increases the learning rate by a factor $\frac{1}{1-\beta}$.

We define the *effective* learning rate, μ_{eff}, for stochastic learning with momentum parameter β as the learning rate that would be needed (using $\beta = 0$) to obtain an equivalent rate of convergence. Thus, under these conditions $\mu_{eff} = \frac{\mu}{1-\beta}$. Note that if a true[1] exponential average were used, the effective learning rate would be μ and not $\mu/(1-\beta)$.

[1]The more commonly used method of exponentially averaging a variable y based on a sequence of estimates $\widehat{y}$, has the form $y_t = (1-\beta)\widehat{y}_t + \beta y_{t-1} = (1-\beta)\sum_{i=0}^{t} \beta^i \widehat{y}_{t-i}$. The exponential average used with momentum does not include the initial factor of $(1-\beta)$.

To analyze the time evolution of the weights more precisely we again treat the update equation as a Markov process. Note, that equation (5.1) is not first order. However it can be recast as one by introducing a new variable $\Omega(t)$ so that the system of equations $z(t) \equiv \{\omega(t), \Omega(t)\}$ is first order. One choice is to set $\Omega(t) = \omega(t-1)$. Expressed in terms of the weight error $v \equiv \omega - \omega^*$, equation equation (5.1) then becomes the first order system

$$\begin{aligned} v(t+1) &= v(t) + \mu\, H[v(t), x(t)] + \beta(v(t) - \Omega(t)) \\ \Omega(t+1) &= v(t). \end{aligned} \tag{5.4}$$

5.2 Previous Results for LMS

In this section we review the results of Tuğay and Tanik [TT89], and Shynk and Roy [SR90] for LMS with momentum. Both sets of authors study stability regions and time constants for convergence in mean and mean square. Their analysis is based on examining the roots of the transition matrix A defined by

$$E[z(t+1)] = A\, E[z(t)] \tag{5.5}$$

where $z(t+1) \equiv \{v(t+1), v(t)\}^T$ as in (5.4) and v is the weight error *expressed in the eigen-coordinates of the Hessian.* Tuğay and Tanik use the diagram repeated here in Figure 5.1 to display the roots as a function of $\epsilon_i \equiv \mu\lambda_i$ for fixed β. The roots occur in pairs. They start out real (β and 1) for $\epsilon_i = 0$. Increasing ϵ_i, we see that when ϵ_i reaches $(1 - \sqrt{\beta})^2$, the roots become complex occurring in complex conjugate pairs. When ϵ_i reaches $(1 + \sqrt{\beta})^2$, the roots go back to being real. Finally, when $\epsilon_i \geq 2(1+\beta)$, one of the real valued roots becomes larger than one and the algorithm diverges. In general, i^{th} root of A is complex if

$$\frac{(1 - \sqrt{\beta})^2}{\lambda_i} < \mu < \frac{(1 + \sqrt{\beta})^2}{\lambda_i}. \tag{5.6}$$

We note that the maximum of the two roots in the pair is smallest when the roots are in the complex region. In this case, the magnitude of both roots in the pair is $\sqrt{\beta}$ for all ϵ_i falling in this complex region.

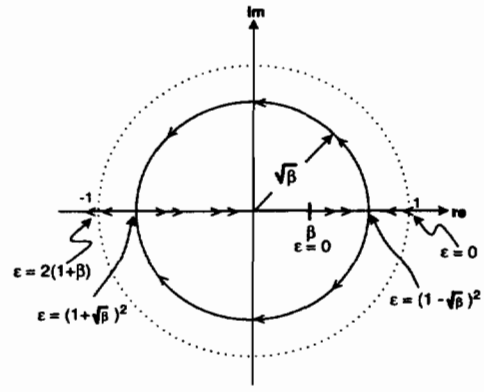


Figure 5.1: Roots of the transition matrix A plotted in the complex plane as a function of $\epsilon \equiv \mu\lambda$. We assume β is fixed.

5.2.1 Stability Regions

For convergence in mean, Shynk and Roy find the stability conditions to be

$$|\beta| < 1 \quad \text{and} \quad 0 < \mu < \frac{2(1+\beta)}{\lambda_{max}} \qquad \text{(convergence in mean)} \tag{5.7}$$

where λ_i is the i^{th} eigenvalue of the Hessian of the cost function and λ_{max} is the largest such eigenvalue. Although these conditions allow for $-1 < \beta < 0$, Roy and Shynk note that in their simulations, "a negative β can lead to degraded performance, even though the algorithm is stable". This is consistent with our own simulations where a negative β was never observed to improve learning over $\beta = 0$. Note also that a negative β would result in an alternating sum in equation (5.2) which could result in oscillatory behavior.

Two necessary conditions for convergence in mean square [SR90] are

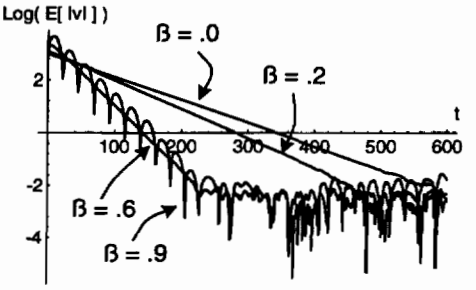
$$\begin{aligned} 0 < \mu &< \frac{2(1-\beta^2)}{(3+\beta)\lambda_{max}} \\ \sum_{i=1}^{N} \frac{\mu\lambda_i}{2(1-\beta^2-\mu\lambda_i)} &< 1 \qquad \text{(convergence in mean square).} \end{aligned} \tag{5.8}$$

Equations in (5.7) and (5.8) predict the *region* of stability but say nothing about the rate of convergence as a function of μ and β. To do this, we look at the time constants.

5.2.2 Rate of Convergence: Time Constants

The time constant τ_i is a measure of the convergence rate of the mean of the weight error $\langle v_i \rangle$ along the i^{th} eigendirection of the Hessian R. It is implicitly defined by the equation

a)



b)

β	τ	
	Sim.	Theory
0	49.7	50.0
.2	39.4	40.0
.6	17.6	20.0
.9	17.6(env)	5.0

Figure 5.2: 1-D LMS with $R = 1$, $\mu = .02$, and $\sigma_\epsilon = 1$ (a) Plots of $\log(|\langle v \rangle|)$ for simulations using an ensemble of 1000 networks with $v_0 = 1000$. (b) Comparison of simulation and theory for the time constant *before* equilibrium where $\tau_{theory} = \frac{1-\beta}{\mu R}$ and $\tau_{sim} = -1/s$ where s is the slope of $\log(\langle v \rangle)$ vs t. For $\beta = .9$, $\langle v \rangle$ oscillates about zero, so we take s to be the slope of the *envelope* of $\log(\langle v \rangle)$.

$\langle v_i \rangle \propto e^{-t/\tau_i}$ and is related to the root r_i of the transition matrix A by $\tau_i = -1/\ln(r_i)$. For standard LMS without momentum, the time constant is $\tau_i^{LMS} = 1/(\mu\lambda_i)$.

Roy and Shynk [SR90] show for slow convergence (i.e. $\mu\lambda_i << 1 - \beta$), that the time constant for LMS with momentum (referred to as MLMS) is

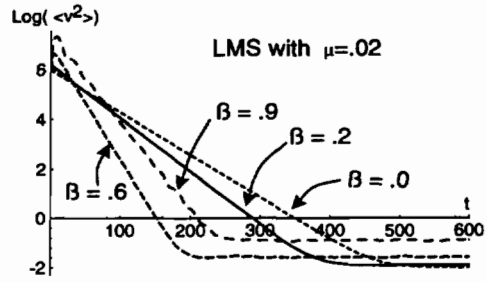
$$\tau_i^{MLMS} \approx \frac{1+\beta}{1+2\beta}\,\tau_i^{LMS} \quad \overset{\beta \ll 1}{\longrightarrow} \quad (1-\beta)\,\tau_i^{LMS} = \frac{1-\beta}{\mu\lambda_i}. \tag{5.9}$$

This predicts that as β is increased, the convergence rate of $\langle v \rangle$ improves (τ gets smaller) and that the effective learning rate is $\mu/(1-\beta)$. However, this result is only valid for small μ and β. The table in (5.2b) compares the predicted time constants with simulations for $\mu = .02$, $R = 1$, $\sigma_\epsilon^2 = 1$, and various β. The agreement is fairly good up to $\beta = .6$ where the conditions for slow convergence begin to break down. The corresponding graphs of $\log(|\langle v \rangle|)$ are shown in (5.2a). We plot $|\langle v \rangle|$ rather than $\langle v \rangle$ so that the log can be taken. Oscillations *about* zero thus look like "scallops". Note for $\beta = .9$, the eigenvalues of the transition matrix are complex and oscillatory behavior is observed in $\langle v \rangle$.

Figure 5.3a plots simulations of $\log[v^2]$ for LMS for $\mu = .02$ and varying β (note that all curves satisfy the stability condition in equations (5.8)). Consistent with the time constant measurements for $\langle v \rangle$, we see that the convergence behavior for $\langle v^2 \rangle$ is not monotonic in β. That is, as β is increased the convergence rate increases up to at least $\beta = .6$, however, by the time $\beta = .9$ the convergence rate has degraded.

We see from Figure 5.3a that the variance flattens out at some nonzero equilibrium value. This equilibrium variance of the weight error increases as β increases. This should

a)



b)

β	$\langle v^2 \rangle_{t=\infty}$ Sim.	Theory
0	.011	.010
.2	.013	.013
.6	.027	.026
.9	.125	.126

Figure 5.3: 1-D LMS with $R = 1$, $\mu = .02$, and $\sigma_\epsilon = 1$ (a) Plots of $\log(\langle v^2 \rangle)$ for simulations using an ensemble of 1000 networks with $v_0 = 1000$. (b) Comparison of simulation and theory for the *equilibrium* weight variance $\langle v^2 \rangle_{t=\infty}$.

be expected for the following reason. When momentum is not present, increasing the learning rate will increase the equilibrium variance (see discussion in the next section). Since momentum increases the effective learning rate then increasing momentum, while keeping the learning rate fixed, should also increase the variance. A related effect is shown by Roy and Shynk: Increasing the momentum increases the misadjustment by a factor of $1 + \beta$. The misadjustment is the excess mean squared error $M = MSE - MSE_{min}$ where MSE_{min} is the mean squared error at the optimal weight ω^*.

Given that the effective learning rate is $\mu_{eff} = \mu/(1-\beta)$ we can examine the effect of varying β and μ while keeping μ_{eff} fixed. Figure 5.4 displays simulations for 1-D LMS all with the same μ_{eff}. We see that all the curves are similar, except that as β is increased there is a small rise in $\log(E[v^2])$ at very early times. Although this rise is slightly corrected by a steeper slope, the best performance is still with $\beta = 0$. It appears then that updating weights using momentum cannot outperform using an optimally chosen learning rate without momentum.

One situation where we had hoped momentum would help is on poorly conditioned problems, i.e. problems whose condition number $\rho \equiv \lambda_{max}/\lambda_{min}$ is much larger than 1. To guarantee convergence in mean we require that the learning rate be constrained by $\mu < 1/\lambda_{max}$. However, $\mu \sim 1/\lambda_{max} \Rightarrow \mu \ll 1/\lambda_{min}$ so that convergence along the λ_{min} direction will be very slow. Thus, the overall convergence rate, which is determined by the slowest time constant, will be very poor. The question is whether momentum can speed learning along the λ_{min} direction without seriously degrading performance along the λ_{max} direction.

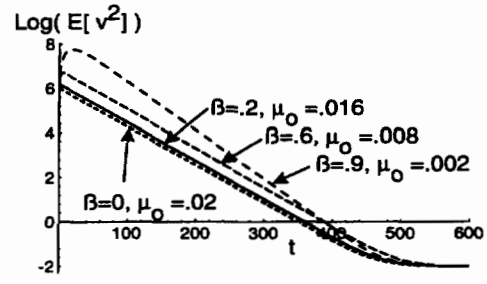


Figure 5.4: Simulations of 1-D LMS with $v_0 = 1000$, $R = 1$, $\sigma_\epsilon = 1$ for an ensemble of 1000 networks. μ and β are varied so as to keep the effective learning rate fixed at $\mu_{eff} = \frac{\mu}{1-\beta} = .02$.

5.2.3 Minimizing the Maximum τ_i

Unfortunately, Tuğay and Tanik find that for stochastic learning with constant learning rate, momentum cannot improve learning for poorly conditioned problems. They came to this conclusion by finding the μ and β that minimize the largest time constant

$$\tau^* \equiv \min_{\mu,\beta} \max_i \tau_i. \tag{5.10}$$

Recall that for a given β, the maximum τ_i is minimized if all the roots of A are complex, i.e. μ satisfies equation (5.6) for all λ_i. This is equivalent to placing the following bounds on the ρ and μ:

$$\rho < \left(\frac{1+\sqrt{\beta}}{1-\sqrt{\beta}}\right)^2 \quad \text{and} \quad \frac{(1-\sqrt{\beta})^2}{\lambda_{min}} < \mu < \frac{(1+\sqrt{\beta})^2}{\lambda_{max}}. \tag{5.11}$$

When (5.11) holds, the time constants are bounded by $\tau_i \leq -\frac{2}{\ln\beta}$ which we note is independent of μ. Choosing the smallest possible β that satisfies the bound on ρ then results in the lowest bound on τ_i.

When ρ is large, Tuğay and Tanik show that the above analysis leads to

$$\tau^*(MLMS) \approx \sqrt{\frac{\tau^*(LMS)}{2}}$$

which is a significant improvement over LMS.

However, it all falls apart when one remembers that the condition for convergence in mean square given in (5.8) must also be met. This condition combined with (5.11) results in the more restrictive bound

$$\rho < \frac{2(1-\beta^2)}{(3+\beta)(1-\sqrt{\beta})^2}. \tag{5.12}$$

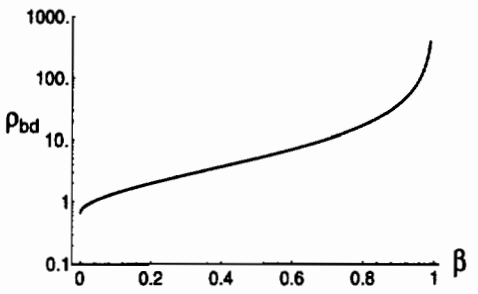


Figure 5.5: Upper bound on the condition number as function of momentum β.

Figure 5.5 displays this bound as a function of β. When ρ is large, we see that this condition forces β to be near 1 making the bound $-\frac{2}{\ln\beta}$ large. As a result, $\tau^*(MLMS) \approx \tau^*(LMS)$. Thus, Tuğay and Tanik conclude that using momentum to minimize the maximum time constant is not very useful. In fact, they conclude that the best convergence rate with MLMS will always be about the same as the best convergence rate achieved with LMS. The only possible advantage of MLMS is that it is less sensitive to noise and results in smoother learning curves.

5.3 LMS: Equilibrium Weight Variance

Once equilibrium has been reached, we must anneal the learning rate to reduce the noise or, alternatively, we must switch to batch mode. Since either option is costly, we would prefer to find a learning method that reduces the weight variance as much as possible before annealing. Does the averaging in momentum provide an efficient mechanism for reducing the noise? We explore this question in this section.

As can be seen from where the curves flatten out in Figure 5.3a, that the squared weight error at equilibrium increases as β is increased. This is not surprising since the effective learning rate is increased. We compute the exact equilibrium variance for 1-D LMS with momentum by starting with the equation (A6a) in Appendix A of Roy and Shynk [SR90]

$$\begin{aligned} C(t+1) &= \left[(1+\beta)^2 + 2\mu^2R^2 - 2\mu(1+\beta)R + \mu^2R^2\right] C(t) + \beta^2 C(t-1) \\ &\quad +[2\mu\beta R - 2\beta(1+\beta)]D(t) + \mu^2\sigma_\epsilon^2 R \qquad (5.13) \\ D(t+1) &= (1+\beta-\mu R)C(t) - \beta D(t) \qquad (5.14) \end{aligned}$$

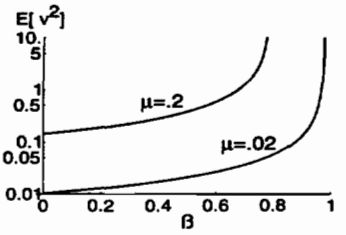


Figure 5.6: Fixed μ: Equilibrium weight variance for 1-D LMS as a function of β for $\mu = .02$, $R = 1$, $\sigma_\epsilon = 1$.

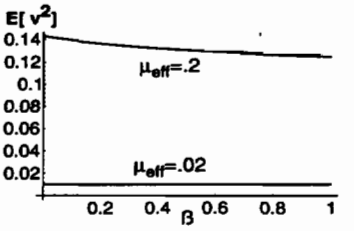


Figure 5.7: Fixed μ_{eff}: Equilibrium weight variance for 1-D LMS as a function of β for $R = 1$, $\sigma_\epsilon = 1$, and $\mu_{eff} = .02$, and .2.

where $C(t) \equiv \langle v^2(t) \rangle$ and $D(t) \equiv \langle v(t)v(t-1) \rangle$. Gaussian inputs have been assumed so that $S \equiv \langle x^4 \rangle = 3R^2$.

Setting $C(t+1) = C(t)$ and $D(t+1) = D(t)$, we solve for the equilibrium variance to get

$$\langle v^2 \rangle = \frac{(1+\beta)\sigma_\epsilon^2 \mu_0}{2(1+\beta)(1-\beta) - \mu_0 R(3+\beta)} \tag{5.15}$$

which reduces to the correct LMS variance $\left\langle v^2_{\beta=0} \right\rangle = \frac{\mu\sigma_\epsilon^2}{2-3\mu R}$ when $\beta = 0$. Figure 5.3b compares the equilibrium variance predicted from (5.15) with that computed from the simulations. Note that for β small, (5.15) reduces to $\langle v^2 \rangle \approx \left\langle v^2_{\beta=0} \right\rangle / (1-\beta)$.

Figure 5.6 shows how $\langle v^2 \rangle$ computed using (5.15) varies as a function of β for 1-D LMS with $R = 1$, $\sigma_\epsilon^2 = 1$, and either $\mu_0 = .2$ or $\mu_0 = .02$. The increase in $\langle v^2 \rangle$ is gradual for small β but is rapid as β gets closer to some critical value. The critical value corresponds to where the denominator of (5.15) becomes zero. Note, that the weights converge as long as the denominator is positive. Thus, a condition for convergence in mean square is

$$\mu R < \frac{2(1-\beta)(1+\beta)}{3+\beta}$$

which is the same result as in equation (5.8).

To answer our original question of whether momentum is an effective way of reducing the noise at equilibrium, we examine the asymptotic variance as a function of β while keeping the *effective* learning rate fixed. To do this, we rewrite (5.15) in terms of μ_{eff} to

give

$$\begin{aligned}\langle v^2 \rangle &= \frac{(1+\beta)\sigma_\epsilon^2 \mu_{eff}}{2(1+\beta) - \mu_{eff} R(3+\beta)} \\ &= \frac{\mu_{eff}\sigma_\epsilon^2}{2 - 2R\mu_{eff}} \quad \text{for } \beta = 1 \\ &= \frac{\mu_{eff}\sigma_\epsilon^2}{2 - 3R\mu_{eff}} \quad \text{for } \beta = 0.\end{aligned} \tag{5.16}$$

As long as $R\mu_{eff} \ll 1$, the difference between $\beta = 0$ and $\beta = 1$ is slight. Figure 5.7 plots equation (5.16) as a function of β for two different values of μ_{eff}. As can be seen, increasing β does decrease the equilibrium variance but only very slightly. This can also be seen in the simulation in Figure 5.4 where all the curves flatten out at about the same value. Thus the averaging in momentum does not provide an efficient mechanism for reducing late time noise.

5.4 Small Noise Expansion

In Appendix E we perform, rather unsuccessfully, a small noise expansion on stochastic learning with momentum and constant learning rate. We used the same coordinates Roy and Shynk with $U(t) \equiv \{v(t), \Omega(t)\}^T \equiv \{v(t), v(t-1)\}^T$ where v is assumed to be the weight error. U is then written as the sum of a deterministic component and noise component

$$\begin{pmatrix} v \\ \Omega \end{pmatrix} = \begin{pmatrix} \phi_v \\ \phi_\Omega \end{pmatrix} + \sqrt{\mu} \begin{pmatrix} \xi_\omega \\ \xi_\Omega \end{pmatrix}. \tag{5.17}$$

The resulting transformed KME equation is

$$\begin{aligned}&\tau \left(-\frac{\dot{\phi_\omega}}{\sqrt{\mu}} \frac{\partial}{\partial \xi_\omega} - \frac{\dot{\phi_\Omega}}{\sqrt{\mu}} \frac{\partial}{\partial \xi_\Omega} + \frac{\partial}{\partial s} \right) \mathcal{P}(\xi_\omega, \xi_\Omega, s) = \\ &\sum_{n=0}^{\infty} \mu^{\frac{n}{2}} \sum_{i=1}^{\infty} \sum_{j=0}^{i} \sum_{l=0}^{\min(j,n)} \sum_{p=\max(0,i-n)}^{i-l} \frac{(-1)^i}{i!\,(n+p-i)!} \begin{pmatrix} i \\ j \end{pmatrix} \begin{pmatrix} j \\ l \end{pmatrix} \begin{pmatrix} i-l \\ p \end{pmatrix} \beta^{j-l} \\ &\alpha_l^{(n+p-i)}(\phi_\omega)\, (\widetilde{\Delta\phi})^{i-l-p} \frac{\partial^i}{\partial {\xi_\omega}^j \partial {\xi_\Omega}^{i-j}} \left\{ {\xi_\omega}^{n+p-i} (\Delta\xi)^p\, \mathcal{P}(\xi_\omega, \xi_\Omega, s) \right\}.\end{aligned} \tag{5.18}$$

From the part of the $n = 1$ term we are able to extract the deterministic equations

$$\tau \frac{d\phi_v}{dt} = \beta\,(\phi_v - \phi_\Omega) + \mu\,\langle H(\phi_v, \phi_\Omega, x) \rangle_x \tag{5.19}$$

$$\tau\,\frac{d\phi_\Omega}{dt} = \phi_v - \phi_\Omega. \tag{5.20}$$

However the resulting noise equations were not very decipherable (see Appendix E). It may be that a different set of transformations would produce a set of more intuitively sensible equations. However, we did try many variations without much success.

Solving the deterministic equations for LMS with initial conditions $\phi(0) = c\{1,1\}^T$ gives

$$\phi_v(t) = \frac{c}{2h}\left((-a+h+\mu R)\, e^{-(a+h)t/\tau} + (a+h-\mu R)\, e^{-(a-h)t/\tau}\right) \tag{5.21}$$

where $a = (1-\beta-\mu R)/2$ and $h = \sqrt{a^2-\mu R}$.

5.5 Summary

When the learning rate is small ($\mu R \ll 1$) momentum effectively increases the learning rate and the convergence rate by a factor of $1/(1-\beta)$. However, adding momentum does not appear to improve learning beyond what can be achieved using the optimal learning rate without momentum even when the problem is poorly conditioned. One slight advantage of using momentum is that the exponential averaging can smooth learning and slightly decrease the equilibrium weight variance.

Chapter 6

Stochastic Learning with Annealed learning and Constant Momentum

Weights in the noise regime fluctuate about some local optimum w^*. To reduce this noise, the learning rate must be annealed. In this chapter, we extend the convergence rate results for annealed learning discussed in Chapter (4) to include momentum.

In the noise regime, we assume that any deterministic component of the weight error $v \equiv \omega - \omega^*$ is essentially zero. This enables us to use an approach similar to that in Section (4.2) to directly compute the time evolution of the weight error correlation [LO94].

6.1 Time Evolution of the Weight Error Correlation

The discrete time algorithm with momentum can be written as a system of equations:

$$\begin{aligned} v(t+1) &= v(t) + \mu(t)\, H[v(t), x(t)] + \beta\, \Omega(t) && (6.1) \\ \Omega(t+1) &\equiv v(t+1) - v(t) \\ &= \Omega(t) + \mu(t)\, H[v(t), x(t)] + (\beta - 1)\, \Omega(t), && (6.2) \end{aligned}$$

or in continuous time,

$$\begin{aligned} \frac{dv(t)}{dt} &= \mu(t)\, H[v(t), x(t)] + \beta\, \Omega(t) && (6.3) \\ \frac{d\Omega(t)}{dt} &= \mu(t)\, H[v(t), x(t)] + (\beta - 1)\, \Omega(t) && (6.4) \end{aligned}$$

where $v(t) \in \mathbb{R}^N$ is the weight error, $\mu(t)$ is the learning rate, H is the weight update function and $x(t)$ is the data fed to the algorithm at time t. Defining $Z \equiv (v, \Omega)^T \in R^{2N}$

and

$$\tilde{H}[Z(t),x(t)] \equiv \begin{pmatrix} \mu(t)\, H[v(t),x(t)] + \beta\, \Omega(t) \\ \mu(t)\, H[v(t),x(t)] + (\beta - 1)\, \Omega(t) \end{pmatrix}$$

we get

$$Z[t+1] = Z[t] + \tilde{H}[Z(t),x(t)]$$

or

$$\frac{dZ(t)}{dt} = \tilde{H}[Z(t),x(t)].$$

We define the generalized weight error correlation matrix as

$$\tilde{C} \equiv E[\, Z\, Z^T\,] = \int d^N Z\; Z\, Z^T\, P(Z,t)\;, \tag{6.5}$$

where $P(Z,t)$ is the probability density of Z at time t whose time evolution can be written according to the Kramers-Moyal expansion

$$\frac{\partial P(Z,t)}{\partial t} = \sum_{i=1}^{\infty} \frac{(-1)^i}{i!} \sum_{j_1,\ldots j_i=1}^{2N} \frac{\partial^i}{\partial Z_{j_1}\, \partial Z_{j_2} \ldots \partial Z_{j_i}} \left\{ \left\langle \tilde{H}_{j_1}\, \tilde{H}_{j_2} \ldots \tilde{H}_{j_i} \right\rangle_x P(Z,t) \right\}\;, \tag{6.6}$$

where $\tilde{H}_{j_k}$ denotes the j_k^{th} component of the $2N$-component vector $\tilde{H}$, and $\langle \ldots \rangle_x$ denotes averaging over the density of inputs.

Note that the convergence (in mean square) to ω^* is characterized by the average squared norm of the weight error $E[\,|v|^2\,] = \text{Trace } C$ where $C \equiv E[\,v\,v^T\,]$ is the upper left quadrant of $\tilde{C}$.

Differentiating $\tilde{C}$ with respect to time and using the Kramers-Moyal expansion (6.6) we obtain an equation of motion for the generalized weight error correlation

$$\frac{d\tilde{C}}{dt} = \int d^N Z\;\; Z\, Z^T \frac{\partial P(Z,t)}{\partial t}. \tag{6.7}$$

Integrating by parts leaves

$$\frac{d\tilde{C}}{dt} = \int d^N Z\; P(Z,t) \left[\, Z \left\langle \tilde{H}(Z,x)^T \right\rangle_x + \left\langle \tilde{H}(Z,x) \right\rangle_x Z^T \,\right] + \int d^N Z\; P(Z,t) \left\langle \tilde{H}(Z,x)\, \tilde{H}(Z,x)^T \right\rangle_x \tag{6.8}$$

where we have assumed that $P(Z,t)$ and its derivatives are all zero at infinity.

6.1.1 Asymptotics of the Weight Error Correlation

We now consider the late time behavior of (6.8) where we assume that the density $P(Z,t)$ becomes sharply peaked about $Z = 0$. Thus we can expand $\tilde{H}(Z,x)$ in a power series about $Z = 0$ and retain the lowest order non-trivial terms in (6.8). Thus we retain the *linear* part of the drift vector

$$\text{drift} \equiv \left\langle \tilde{H}(Z,x) \right\rangle_x \approx KZ$$

where

$$\begin{aligned} K &\equiv \begin{pmatrix} -\mu(t)\, R & \beta I \\ -\mu(t) R & (\beta - 1) I \end{pmatrix} \\ R_{ij} &\equiv \left\langle \left. \frac{\partial^2 \mathcal{E}}{\partial v_i \partial v_j} \right|_{v=0} \right\rangle_x \end{aligned}$$

and where I is the $N \times N$ identity. We retain the *constant* part of the diffusion matrix

$$\begin{aligned} \text{diffusion} &\equiv \left\langle \tilde{H}(Z,x)\, \tilde{H}(Z,x)^T \right\rangle_x \approx \mu(t)^2\ \tilde{D} \\ \tilde{D} &\equiv \begin{pmatrix} D & D \\ D & D \end{pmatrix} \\ D &\equiv \left\langle H(0,x)\, H(0,x)^T \right\rangle_x \end{aligned}$$

With these approximations, the time evolution of the weight correlation matrix $\tilde{C} \equiv E[ZZ^T]$ evolves according to

$$\frac{d\tilde{C}}{dt} = K\tilde{C} + \tilde{C}\, K^T + \mu(t)^2\, \tilde{D} \tag{6.9}$$

To evaluate this we define the evolution operator to be

$$U(t_2, t_1) \equiv \exp\left[\int_{t_1}^{t_2} d\tau\ K(\tau)\right] \tag{6.10}$$

and the solution to equation (6.9) becomes

$$\tilde{C} = U(t,t_0)\ \tilde{C}\ (t_0)\ U^T(t,t_0) + \int_{t_0}^{t} d\tau\ \mu(\tau)^2\, U(t,\tau)\ \tilde{D}\ U^T(t,\tau) \tag{6.11}$$

Again we emphasize that to study convergence of ω^* we need only evaluate Trace C which is the sum of the first N diagonal elements of $\tilde{C}$.

6.1.2 μ_0/t Learning Schedules

In Appendix F we evaluate (6.11) for learning rate schedules of the form $\mu(t) = \mu_0/t$. The analysis assumes, without loss of generality, that the coordinates are chosen so that R is diagonal with eigenvalues λ_i $i = 1 \ldots N$. In this section we summarize and examine the results. This analysis shows that the expected value of $|v|^2$ *at late times* can be written approximately as

$$E[|v|^2] \approx \sum_{i=1}^{N} \left\{ E[v_i^2(t_0)] \left(\frac{t_0}{t}\right)^{\frac{2\mu_0\lambda_i}{1-\beta}} + \frac{\mu_0^2 D_{ii}}{(1-\beta)(2\mu_0\lambda_i - 1 + \beta)} \left(\frac{1}{t} - \frac{1}{t_0}\left(\frac{t_0}{t}\right)^{\frac{2\mu_0\lambda_i}{1-\beta}} \right) \right\} \tag{6.12}$$

where

$$D_{ij} \equiv \left\langle \frac{\partial \mathcal{E}}{\partial v_i} \frac{\partial \mathcal{E}}{\partial v_j}\bigg|_{v=0} \right\rangle_x$$

and where v_i are the components of v along the eigendirections of the Hessian. Note that we are assuming that we are in a region of the weightspace for which $\lambda_i > 0$. Therefore, we must have $\beta < 1$ to prevent the exponent of $\frac{t_0}{t}$ being negative. Defining

$$\mu_{eff} \equiv \frac{\mu_0}{1-\beta} \quad \text{and} \quad \mu_{crit} \equiv \frac{1}{2\lambda_{min}}$$

we again find that there are two asymptotic regimes:

Regime 1 $\mu_{eff} > \mu_{crit}$: $E[|v|^2] \sim \frac{1}{t}$

Regime 2 $\mu_{eff} < \mu_{crit}$: $E[|v|^2] \sim \left(\frac{1}{t}\right)^{\frac{\mu_{eff}}{\mu_{crit}}}$ $\left(\text{Note, } \frac{\mu_{\text{eff}}}{\mu_{\text{crit}}} = \frac{2\mu_0\lambda_{min}}{1-\beta} < 1\right)$.

Comparison with Simulation

Figure 6.1 compares the simulated and theoretical results. The learning rate μ_0 is held fixed while β is varied. Regime 1 (solid) corresponds to $\beta > .6$. As can be seen, all of the curves in regime 1 all have about same slope while the slopes of the curves in regime 2 (dotted) vary and are less steep than those in regime 1.

Just as in the constant learning rate case, the results also imply that the asymptotic behavior is the same for combinations of μ_0 and β for which $\mu_{eff} = \mu_0/(1-\beta)$ is held fixed. Figure 6.2 displays results of simulations for which $\mu_{eff} = .5$. The algorithm is

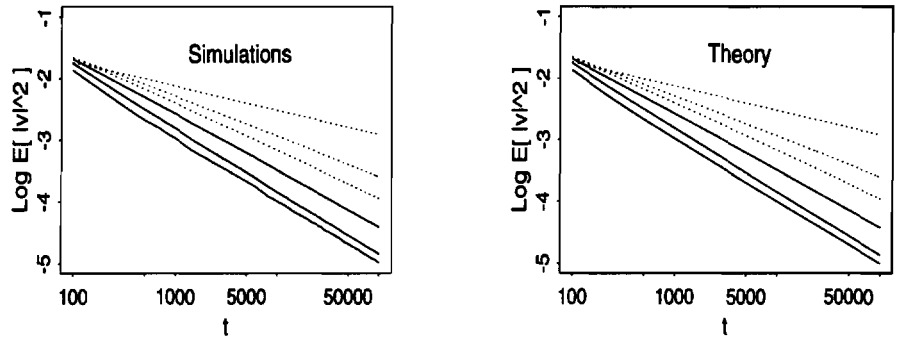


Figure 6.1: LEFT – Simulation results from an ensemble of 2000 one-dimensional LMS algorithms with momentum with $R = 1.0$, $\mu_0 = 0.2$. RIGHT – Theoretical predictions from equation (6.12). Curves correspond to (top to bottom) $\beta = 0.0$, 0.4, 0.5, 0.6, 0.7, 0.8 .

run at constant learning rate until the weights settle to their equilibrium state, then annealing is turned on at $t_0 = 1$. At early times (e.g. $t < 100$), the curves are quite different[1]. At late times, the curves essentially overlap as long as β is small. However for large β, $E[|v|^2]$ shifts upward slightly.

6.2 Optimal Momentum, β_{opt}

For annealed learning without momentum the optimal scalar learning rate is $\mu_{0,opt} = 1/\lambda_{min}$. Since adding momentum increases the effective learning rate by a factor of $1/(1-\beta)$, we define the "optimal" β as the value of momentum that satisfies $\mu_{eff} \equiv \mu_0/(1-\beta) = \mu_{0,opt}$. Solving for β gives $\beta_{opt} = 1 - \mu\lambda_{min}$. For the 1-D simulations in Figure 6.1, $\beta_{opt} = .8$ which corresponds to the bottom most curve.

Ideally, however, the optimal learning rate is a *matrix* equal to the inverse Hessian, $\mu_{0,opt} = R^{-1}$. When such a learning rate is used, the stepsize of the weight update is

[1]The curves vary quite a bit at early times even if $E[|v|^2]$ at $t = t_0$ is set to be the same for all μ_0.

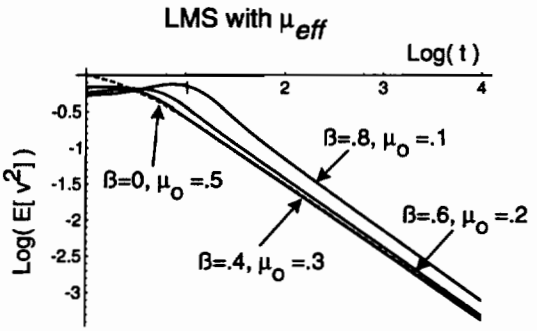


Figure 6.2: 1-D LMS: Trade-off between μ_0 and β, for $\mu_{eff} = \mu_0/(1-\beta) = .5$. $R = 1$, $\sigma_\epsilon = 1$, and 10000 networks. The dashed curve corresponds to $\beta = 0$.

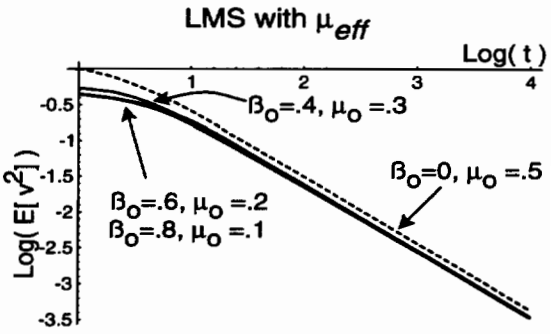


Figure 6.3: 1-D LMS: Trade-off between μ_0 and β_0, for $\mu_{eff} = \mu_0/(1-\beta_0) = .5$ when $\beta_t = \frac{t-1}{t}\beta_0$. $R = 1$, $\sigma_\epsilon = 1$, and 10000 networks. The dashed curve corresponds to $\beta_0 = 0$.

scaled independently along *each direction of the Hessian* so that optimally fast convergence is obtained *along each direction.* We can now ask, given a *scalar* μ_0, what *matrix* momentum parameter results in an effective learning rate matrix equal to $\mu_{0,opt}$? Setting

$$\mu_{eff} \equiv \mu_0(I - \beta_{opt})^{-1} = R^{-1}$$

where I is the identity matrix and solving for β_{opt}, we obtain the "optimal" momentum *matrix*

$$\beta_{opt} = I - \mu_0 R \tag{6.13}$$

In correspondence with the restriction $0 \leq \beta < 1$ when β is a scalar, we now require that β_{opt} have eigenvalues $0 \leq \lambda_i^{(\beta)} < 1$. From (6.13), we see that this leads to the restriction on μ_0 that $\mu_0 < 1/\lambda_{max}$ where λ_{max} is the largest eigenvalue of R.

Thus, using a scalar μ_0 with the momentum matrix β_{opt} should give approximately the same convergence behavior as using $\mu_{0,opt} = R^{-1}$ without momentum. The advantage of using β_{opt} instead of $\mu_{0,opt}$ is that the Hessian R is much easier to compute than the inverse Hessian R^{-1}. In fact, in the next chapter, we discuss how to *stochastically* estimate β_{opt}.

6.2.1 Ramped Momentum

Recall that adding momentum for constant $\mu = \mu_0$ is equivalent to computing an exponential average of past gradients. For annealed learning the expanded form of momentum

is

$$\omega(t+1) = \omega(t) - \mu_0 \sum_{i=1}^{t} \frac{\beta^i}{t-i} \widehat{\nabla \mathcal{E}}_{t-i}. \tag{6.14}$$

Thus, when annealed learning is used, the sum no longer has the form of a standard exponential weighted average because of the factors $\frac{1}{t-i}$. A slight modification to the previous form of stochastic learning with momentum and annealed learning is

$$\omega(t+1) = \omega(t) - \frac{\mu_0}{t} \widehat{\nabla \mathcal{E}}_t + \beta_t(\omega(t) - \omega(t-1)). \tag{6.15}$$

where

$$\beta_t \equiv \frac{t-1}{t} \beta_0 \tag{6.16}$$

for some constant $0 < \beta_0 < 1$. We refer to this as *ramped* beta because at $t = 1$ we have $\beta_t = 0$, and at late times $\beta_t \approx \beta_0$. Expanding out (6.15) gives

$$\omega(t+1) = \omega(t) - \frac{\mu_0}{t} \sum_{i=1}^{t} \beta_0^i \widehat{\nabla \mathcal{E}}_{t-i}. \tag{6.17}$$

This has the more standard form of an exponential average of past gradients since the $\frac{1}{t}$ factor is now pulled out of the sum. It also has the effect of weighting the older gradients less. Figure 6.3 displays the results of using ramped momentum for the same parameters that were used in Figure 6.2. Note that the dashed curves are the same in both figures since $\beta = 0$. As can be seen, the behavior particularly at early times is much better for ramped momentum.

6.2.2 Simulations with Large Condition Number

In this section we present simulations for a 4-D LMS network. The training data consists of 1000 examples for which the eigenvalues of Hessian R range from about 10^{-5} to 1. Thus the condition number is $\rho \approx 10^5$. We first train the network at constant learning rate ($\mu_0 = .5$) until no improvement is observed either in the mean squared error $\mathcal{E}$ or in the squared weight error $E[|v|^2]$. We then anneal the learning rate to remove the noise. We anneal according to μ_0/t_a where t_a is the time initialized to 1 when annealing is turned on.

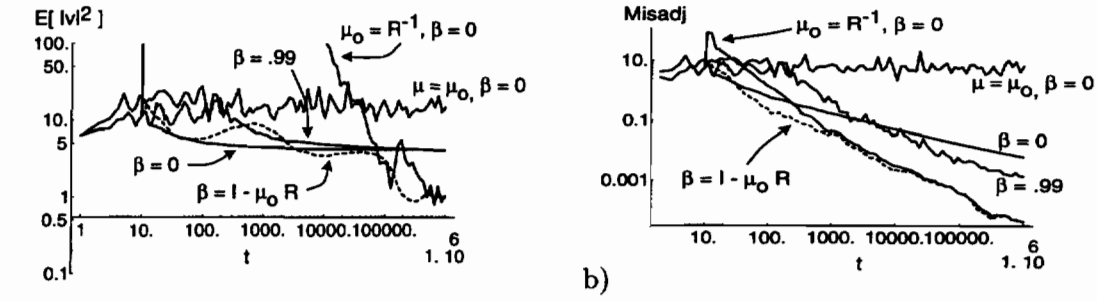
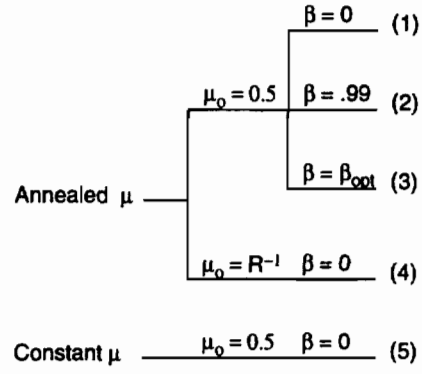


Figure 6.4: 4-D LMS with $\rho = 10^5$: a) $E[|v|^2]$, b) Misadjustment. Learning rate is annealed starting at $t = 10$ according to $\mu = \mu_0/t_a$. Curves correspond to 1) $\mu_0 = .5$ with $\beta = 0, .99, I - \mu_0 R$ (dashed), and 2) $\mu_0 = R^{-1}$ without momentum. The learning curve for constant learning rate $\mu = \mu_0$ is also shown for comparison. Each curve is an average of 10 runs.

Five approaches are compared and displayed in Figure 6.4. They are



The first three use an annealed learning rate with scalar learning rate parameter $\mu_0 = .5$. They differ according to the choice of momentum parameter: $\beta = 0$, $\beta = .99$, and the optimal momentum parameter $\beta = I - \mu_0 R$ (dashed). The other two approaches are shown for comparison. They are 1) the optimal learning rate $\mu_0 = R^{-1}$ with annealing and 2) a constant learning rate $\mu = \mu_0 = .5$ (no annealing). When annealing is used, it is turned on at $t = 10$ in the figure (where t_a is set to 1).

Figure 6.4a displays the squared weight error while Figure 6.4b displays the misadjustment. At late times the best performance is observed with μ_{opt} (approach 4 above) and with $\mu_0 = .5$ combined with β_{opt} (approach 3 above). In (6.4b) it is easy to see that using $\mu_0 = .5$ with β_{opt} results in convergence behavior very similar to $\mu_0 = R^{-1}$ while using $\mu_0 = .5$ with the scalar β results in slower convergence.

When the optimal learning rate $\mu_0 = R^{-1}$ is used, the squared weight error first

increases dramatically to as high as 10^6 (see Figure 6.4a) before converging very quickly. To prevent the large initial rise, one can use the annealing schedule $\mu_0/(t_a + \tau)$. When $\tau = 10$ or 1000 are used, the initial rise is much smaller but the value of $E[|v|^2]$ and the MSE are essentially unchanged at late times.

6.3 Summary

In this chapter we examined the convergence of stochastic learning with an annealed learning rate combined with momentum. As in the previous chapter where a constant learning rate was studied, we find that adding momentum 1) increases the effective learning rate by a factor of $1/(1-\beta)$ and 2) does not does not improve convergence over using the optimal learning rate without momentum.

We also defined an optimal momentum matrix and showed that the convergence behavior was comparable to using μ_{opt} without momentum. The advantage is that β_{opt} is defined in terms of R while μ_{opt} is defined in terms of the more difficult to compute R^{-1}. In the next two chapters we show how β_{opt} can be stochastically evaluated very efficiently so that no direct computation or storage of R is even needed.

Chapter 7

Adaptive Momentum for Linear Networks

In this chapter we show how to estimate the optimal momentum matrix *stochastically* for linear networks. We refer to this algorithm as adaptive momentum. When used during the annealing phase of learning, adaptive momentum results in an effective learning rate that is close to the optimal learning rate matrix, $\mu_{opt} = R^{-1}$. Since μ_{opt} scales the learning rate appropriately along the different eigendirections of the Hessian, it is particularly useful for problems that are poorly conditioned.

Note that adaptive momentum requires setting very few parameters. Not only does the algorithm automatically compute the appropriate momentum parameter but we also find that the algorithm is quite insensitive to the choice of μ_0, only requiring that $\mu_0 < 1/\lambda_{max}$.

Simulations on linear problems of varying sizes and condition numbers are presented and compared with the theoretical predictions. In the next chapter, we extend these ideas to nonlinear networks.

7.1 Adaptive Momentum for LMS with annealed learning

The LMS algorithm with N inputs and M outputs with annealed learning and momentum is

$$\omega_{t+1} = \omega_t - \frac{\mu_0}{t} \widehat{\nabla \mathcal{E}}_t + \beta_t(\omega_t - \omega_{t-1}) \tag{7.1}$$

$$\mathcal{E}_t \equiv \frac{1}{2M} \sum_{i=1}^{M} \left(d_t - w_t^T x_t\right)^2 \tag{7.2}$$

where $\omega \in \mathbb{R}^{N\times M}$ is the weight matrix, $\mathcal{E} \in \mathbb{R}$ is the cost function, $d \in \mathbb{R}^M$ is the target vector, and $x \in \mathbb{R}^N$ is the input vector. We shall only consider the case for one output ($M = 1$) since a network with M outputs can be treated as M separate single output networks.

The previous chapter showed that an effective learning rate of $\mu_{0,opt} = R^{-1}$ is achieved when the momentum parameter is $\beta_{opt} = I - \mu_0 R$, where $\mu_0 < 1/\lambda_{max}$. For linear networks is $R = \left\langle x\, x^T \right\rangle$. In general, R is unknown or expensive to compute, however, we can stochastically estimate it using $\widehat{R}(t) = x_t\, x_t^T$ where x_t is the input at time t. Thus we define adaptive momentum for linear networks as

$$\beta(t) \equiv I - \mu_0 x_t\, x_t^T, \quad (1 \text{ output}, M = 1). \tag{7.3}$$

The momentum term in (7.1) then becomes

$$\beta_t \Delta\omega_t = (I - \mu_0 x_t\, x_t^T)\Delta\omega_t = \Delta\omega_t - \mu_0 x_t\, (x_t^T \Delta\omega_t) \tag{7.4}$$

where $\Delta\omega_t \equiv \omega_t - \omega_{t-1}$. The dot product $x_t^T \Delta\omega_t$ is a scalar that is the same for each weight component so that it needs to be computed only once at each weight vector update rather than N times. Thus, computation of the momentum term is $\mathcal{O}(N)$.

An alternative to setting $\mu_0 < 1/\lambda_{max}$, which requires knowing λ_{max}, is to bound the value of $\beta(t)$ to be above zero on a sample by sample basis. This method, which we refer to as "capping", is discussed in detail in Appendix G. Capping is less desirable than setting $\mu_0 = 1/\lambda_{max}$. However, it does prevent divergence when μ_0 is chosen too large.

7.2 Simulations for LMS

In this section we present simple 1, 2, and 4 dimensional simulations for LMS with momentum. We also present a higher dimensional example from image compression. We find that adaptive momentum consistently performs as well or better than constant momentum. In addition, we find that not only are convergence rates asymptotically optimal (i.e. $\propto 1/t$) for adaptive momentum but that the magnitude of $E[|v|^2]$ at late times appears to be *independent* of the learning rate parameter.

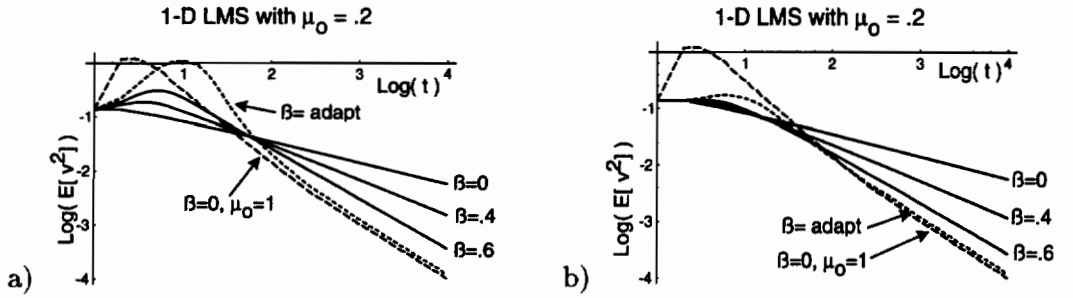


Figure 7.1: 1-D LMS: a) Simulation results from an ensemble of 2000 one-dimensional LMS networks with momentum using $R = 1.0$, $\mu_0 = 0.2$, and $\sigma^2 = 1$. For comparison, a simulation using $\mu_{opt} = 1$ and no momentum is also included. b) Same as a) but with ramped β.

7.2.1 1-D LMS

Figure 7.1a displays, for various values of β, graphs of $\text{Log}(E[|v|^2])$ versus $\text{Log}(t)$ for simulations using 2000 1-D networks with $R = 1$, $\mu_0 = 0.2$ and $\sigma^2 = 1$. Inputs are gaussian distributed with zero mean. Also shown for comparison is a simulation using the optimal learning rate parameter, $\mu_{opt} = \frac{1}{R} = 1$ *without* momentum. The networks were first run at constant learning rate $\mu_0 = .2$ without momentum until equilibrium was reached. Momentum was *not* used in the constant learning rate phase due to stability[1] problems for large β. One explanation for the initial increase in $E[v^2]$ for the $\beta > 0$ curves may be because momentum increases variance[2]. Note that $\beta_{opt} = 1 - \mu_0\lambda = .6$.

Figure 7.1b displays simulations for the same parameters but with ramped momentum. As can be seen, the large rise in $E[v^2]$ at early times is reduced, however, late time behavior is not significantly changed.

[1]Convergence regions are different for constant and annealed learning. Some of the combinations of μ and β that result in convergence for annealed learning instead result in divergence if used with a constant learning rate. In transitioning from a constant to an annealed learning phase, it would seem advantageous to pick parameters so that the variance is matched at the boundary. However, we do not address this issue here.

[2]If we had used momentum during the constant learning rate phase, the equilibrium variance would have been larger than what is seen here for the $\beta > 0$ curves. Thus $E[v^2]$ at the point annealing is just turned on ($t = 0$) would also have been larger. Of course, in some cases, the equilibrium variance would have been infinite (see previous footnote).

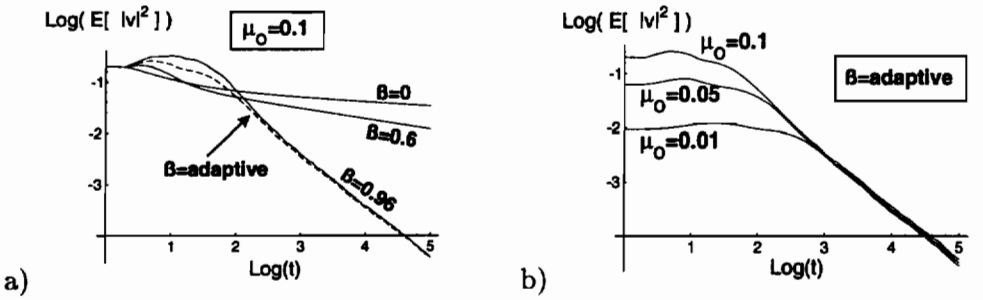


Figure 7.2: 2-D LMS Simulations: Behavior of $\log(E[|v|^2])$ over an ensemble of 1000 networks with $\lambda_1 = .4$ and $\lambda_2 = 4$, $\sigma_\epsilon^2 = 1$. a) $\mu_0 = 0.1$ with various β. Dashed curve corresponds to adaptive momentum. b) β adaptive for various μ_0.

2-D LMS

Figure 7.2 displays $\log(E[|v|^2])$ averaged over an ensemble of 1000 networks trained with either constant or adaptive momentum. The eigenvalues of the Hessian R are $\lambda_1 = .4$ and $\lambda_2 = 4$ and the variance of ϵ is $\sigma_\epsilon^2 = 1$. In Figure 7.2a the learning rate parameter is the same for all curves ($\mu_0 = 0.1$) while β is varied. As the momentum is increased, the convergence rate improves. The best performance occurs for both the optimal *scalar*[3] momentum ($\beta_{opt}^{(scal)} = 1 - \mu_0\lambda_{min} = .96$) and adaptive momentum. Note, however, that adaptive momentum is preferable in practice because the value of λ_{min} (and thus the value of $\beta_{opt}^{(scal)}$) is, in general, unknown.

Figure 7.2b shows the behavior of $\log(E[|v|^2])$ for adaptive momentum while μ_0 is varied. After a few hundred iterations the value of $\log(E[|v|^2])$ is independent of μ_0 (in all cases $\mu_0 < 1/\lambda_{max}$).

7.2.2 4-D LMS with $\rho \approx 10^5$

We now return to the 4-D LMS problem first presented in Section §6.2.2. In Figure 7.3 we compare

1. $\mu_0 = \mu_{0,opt} \equiv R^{-1}$ with $\beta = 0$,
2. $\mu_0 = .5$ with $\beta = \beta_{opt} \equiv I - \mu_0 R$,
3. $\mu_0 = .5$ with $\beta = \beta_{adapt}$.

[3]For this particular example, we did not try the optimal learning rate matrix $\mu_{opt,0} = R^{-1}$.

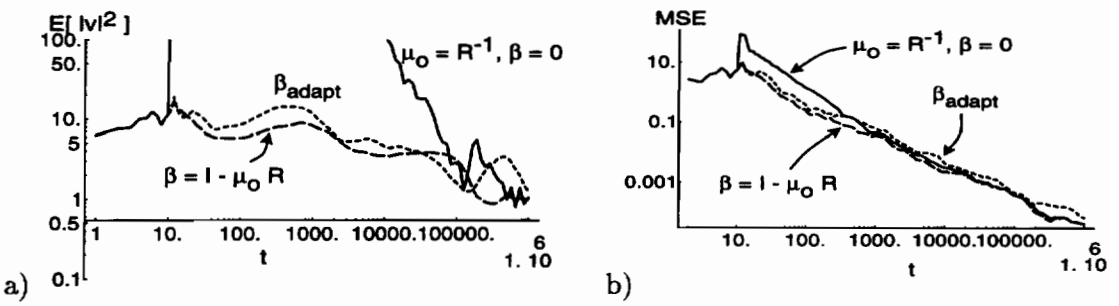


Figure 7.3: 4-D LMS with $\rho = 10^5$: a) $E[|v|^2]$, b) Misadjustment. Learning rate is annealed starting at $t = 10$ according to $\mu = \mu_0/t_a$. Curves correspond to 1) $\mu_0 = .5$ with $\beta = I - \mu_0 R$ (long dash), and 3) $\mu_0 = .5$ with β_{adapt} (short dash) Each curve is an average of 10 runs.

Results for the first two cases are repeated from figure 6.4. As can be seen, the convergence behavior of the misadjustment is similar for all three. $E[|v|^2]$ is similar for β_{opt} and β_{adapt} while $E[|v|^2]$ initially diverges for μ_{opt}. At late times, all three are similar.

7.2.3 Image Compression

For a larger problem we consider predicting the center block of an image based on the 8 surrounding blocks. The image used is commonly known as "Lenna" or "the girl with the hat" [GG93] and is shown in Figure 7.4. The block size is 4×4 making the target dimension 16 (1 block) and the input dimension 128 (8 blocks of size 4×4). There are a total of 3844 examples that have been divided into 2914 training examples and 930 test examples. The eigenvalues range from 1.06×10^{-5} to 19.98. so that the condition number for this problem is about $\rho = 1.9 \times 10^6$.

Figure 7.5 displays the learning curves for

1. annealed learning with $\beta = \beta_{adapt}$,
2. annealed learning with $\beta = 0$, and
3. constant learning rate (for comparison purposes).

As before, we have first trained (not shown completely) at constant learning rate $\mu_0 = .026$ until the MSE and the weight error have leveled out. As can be seen β_{adapt} does much better than annealing without momentum.

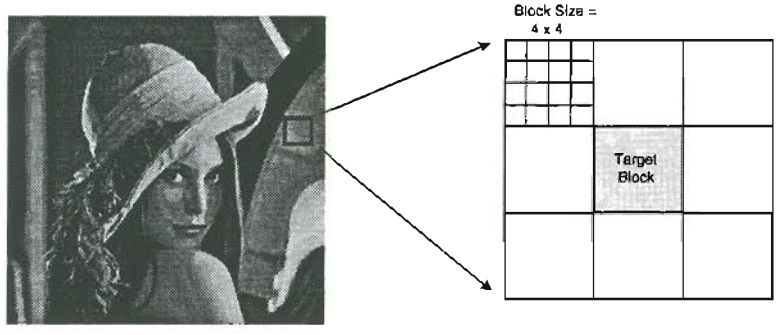


Figure 7.4: Lenna Image: Target block is predicted from the 8 adjacent surrounding blocks. Each block is 4×4.

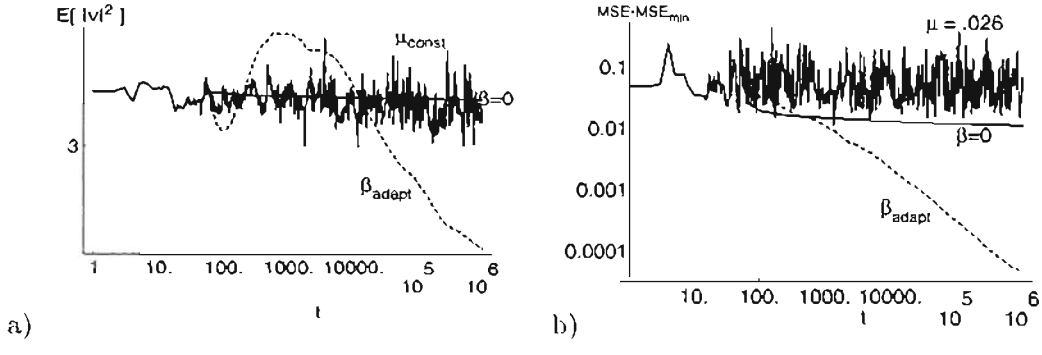


Figure 7.5: Lenna: a) $E[|v|^2]$, b) Misadjustment. Learning rate is annealed starting at $t = 50$ according to $\mu = \mu_0/t_a$, $\mu_0 = .026$. Curves correspond to $\mu_0 = .026$ with $\beta = 0$ and $\mu_0 = .026$ with β_{adapt} (dashed). The learning curve for constant learning rate $\mu = \mu_0$ is shown for comparison.

7.3 Choosing μ_0

In general, we want the learning rate to be as large as possible in the constant learning rate phase. For stochastic LMS with constant learning rate, convergence in mean square requires $\mu_0 < \frac{2R}{S}$ (in 1-D) where, as before, $R = \langle x^2 \rangle$ and $S = \langle x^4 \rangle$. For gaussian inputs ($S = 3R^2$) this bound is $\mu_0 < \frac{2}{3\lambda}$ (1-D) or $\mu < \frac{2}{3\lambda_{max}}$ (multiple-D). Most problems are not gaussian. However, rather than estimate S, a more expedient approach is to just set $\mu_0 = \frac{1}{2\lambda_{max}}$, i.e. a little below the gaussian bound.

To compute λ_{max}, we use the "on-line" algorithm developed by LeCun, Simard, and Pearlmutter [LSP93]. It is based on the idea that repeatedly multiplying a vector by the Hessian will ultimately align that vector along the eigenvector, $e_{\lambda_{max}}$, associated with the maximum eigenvalue λ_{max}. If the vector is also normalized before each multiplication then the norm of the final vector will be equal to λ_{max}. That is,

1. Start with an arbitrary vector p_0.
2. Iterate: $p_{t+1} = R\mathcal{N}(p_t)$, where $\mathcal{N}(p)$ is the operator that normalizes p.
3. Then

$$\begin{aligned} \lim_{t\to\infty} ||p_t|| &= \lambda_{max} \\ \lim_{t\to\infty} p_t &= e_{\lambda_{max}}. \end{aligned}$$

To implement this, we compute the product $R\mathcal{N}(p)$ using the approximation

$$R\mathcal{N}(p) \approx \frac{\widehat{\nabla\mathcal{E}}(\omega + \alpha\mathcal{N}(p)) - \widehat{\nabla\mathcal{E}}(\omega)}{\alpha} \tag{7.5}$$

where α is some small constant. The "on-line" version is then implemented using a running average of p so that

$$p_{t+1} = (1-\gamma)p_t + \gamma\frac{\widehat{\nabla\mathcal{E}}(\omega + \alpha\mathcal{N}(p_t)) - \widehat{\nabla\mathcal{E}}(\omega)}{\alpha} \tag{7.6}$$

for some small constant γ. To reduce the noise at late times, one can also anneal γ.

We set $\alpha = \gamma = .01$ and let the algorithm run for 1000 iterations. For Lenna, the predicted maximum eigenvalue was $\lambda_{max} = 19.35$ as compared to the exact value $\lambda_{max} = 19.98$. We have found that this algorithm works well at predicting λ_{max} to

within a few percent for linear problems and is fairly insensitive to the values of α and γ.

In the next chapter we discuss in detail an online forward-backward algorithm for computing the product of the Hessian times a vector. This algorithm can replace the finite difference approximation for $R\mathcal{N}(p)$ given in equation (7.5) with the exact (online) product. Note that the number of parameters is then reduced since α is no longer needed.

7.4 Summary

In this chapter we have introduced adaptive momentum, β_{adapt}, for linear networks. Adaptive momentum is a stochastic implementation of the optimal momentum matrix, β_{opt}, derived in the previous chapter. We describe an efficient implementation of β_{adapt} and test it on linear networks of varying sizes and condition number. We find that β_{adapt} is quite insensitive to the choice of μ_0 and that its performance is comparable to β_{opt}.

In the constant learning rate phase, we want learning rate to be $\mu_0 \approx \frac{2R}{S}$. Since S is not known, we instead set $\mu_0 = \frac{1}{2\lambda_{\max}}$. To compute λ_{max}, we use an "on-line" algorithm by LeCun, *et. al.*, [LSP93]. Knowing λ_{max} is also useful for adaptive momentum in the annealing phase so as to insure that $\mu_0 < \frac{1}{\lambda_{\max}}$.

Chapter 8

Adaptive Momentum for Nonlinear Networks

In this chapter we present an algorithm for nonlinear adaptive momentum for use in the annealing phase of stochastic search. As in the linear case, adaptive momentum is a stochastic implementation of the optimal momentum parameter. Despite the complexity of nonlinear networks we have discovered an efficient implementation of nonlinear adaptive momentum. We present our implementation and demonstrate its effectiveness on several small and large networks.

8.1 Form of β_{adapt}

As discussed in Chapter (6), the optimal momentum parameter is

$$\beta_{opt}(\omega_t) = I - \mu_0 R(\omega_t) \qquad (\text{for } \mu_0 < \frac{1}{\lambda_{max}}) \tag{8.1}$$

where $R(\omega_t)$ is the Hessian of the cost function evaluated at the weight ω_t. Note that, unlike linear networks, the Hessian is not constant throughout the weight-space. A stochastic estimate of β_{opt} is then

$$\beta_{adapt}(\omega_t, x_t) = I - \mu_0 R_t(\omega_t, x_t) \qquad (\text{for } \mu_0 < \frac{1}{\lambda_{max}}) \tag{8.2}$$

where $R_t(\omega_t, x_t)$ is the instantaneous estimate of $R(\omega_t)$ based on the single exemplar x_t presented at the t^{th} timestep. The momentum *term* then becomes

$$\beta_{adapt}\ \Delta\omega_t = (I - \mu_0 R_t)\ \Delta\omega_t = \Delta\omega_t - \mu_0 R_t \Delta\omega_t \tag{8.3}$$

where $\Delta\omega_t \equiv (\omega_t - \omega_{t-1})$. Written in this way we see that we do not need to explicitly calculate R_t, rather we just need to compute the *product* $R_t\Delta\omega$. This can done efficiently

using the algorithm developed by Pearlmutter [Pea94] for computing the product of the Hessian and an arbitrary vector. The algorithm requires only one forward-backward propagation to compute this product.

8.1.1 Fast Multiplication of Hessian times a Vector

Pearlmutter observes that given an arbitrary vector z (of the appropriate dimension), the product Rz can be written as

$$R(\omega)\, z = \frac{\partial}{\partial r} \frac{\partial \mathcal{E}(\omega + rz, x)}{\partial \omega}\bigg|_{r=0} \tag{8.4}$$

where $\mathcal{E}(\omega)$ is the cost function. He then defines the differential operator $\mathcal{R}_z\{\cdot\}$

$$\mathcal{R}_z\{f(\omega)\} \equiv \frac{\partial}{\partial r} f(\omega + rz)\bigg|_{r=0} \tag{8.5}$$

where $f(\omega)$ is some arbitrary function. The product Rz can then be written in terms of the operator $\mathcal{R}_z\{\cdot\}$

$$R_t\, z = \mathcal{R}_z \left\{ \frac{\partial \mathcal{E}(\omega, x)}{\partial \omega} \right\}. \tag{8.6}$$

Now $\mathcal{R}_z\{\cdot\}$ follows all of the usual rules for differential operators (e.g. chain rule, etc). Therefore, if the standard forward-backward equations for computing $\frac{\partial \mathcal{E}(\omega,x)}{\partial \omega}$ are substituted into (8.6) and simplified using these rules, we obtain a set of forward-backward equations for computing $R_t z$. We present the details below.

Let x_i denote the net input to a node i, y_i the output of a node i, and ω_{ij} the weight connecting node j to node i. The standard forward computation is then

$$\begin{aligned} y_i &= x_i && \text{(input nodes)} \\ x_i &= \sum_j \omega_{ij} y_j && \text{(non-input nodes)} \\ y_i &= \sigma_i(x_i) \end{aligned} \tag{8.7}$$

where σ_i is the nonlinearity of node i. Applying the operator $\mathcal{R}_z\{\cdot\}$ to the above forward pass equations gives

$$\begin{aligned} \mathcal{R}_z\{y_i\} &= \mathcal{R}_z\{x_i\} = 0 && \text{(input nodes)} \\ \mathcal{R}_z\{x_i\} &= \sum_j (\omega_{ij} \mathcal{R}_z\{y_j\} + z_{ij} y_j) && \text{(non-input nodes)} \\ \mathcal{R}_z\{y_i\} &= \mathcal{R}_z\{x_i\} \sigma_i'(x_i) && \text{(non-input nodes)} \end{aligned} \tag{8.8}$$

where we have made use of the fact that $\mathcal{R}_z\{\omega\} = z$.

The standard backward pass equations to compute the gradient are

$$\begin{aligned}
\frac{\partial \mathcal{E}}{\partial y_i} &= e_i(y_i) && \text{(output nodes)} \\
\frac{\partial \mathcal{E}}{\partial y_i} &= \sum_j \omega_{ji} \frac{\partial \mathcal{E}}{\partial x_j} && \text{(non-output nodes)} \\
\frac{\partial \mathcal{E}}{\partial x_i} &= \sigma_i'(x_i) \frac{\partial \mathcal{E}}{\partial y_i} \\
\frac{\partial \mathcal{E}}{\partial \omega_{ji}} &= y_i \frac{\partial \mathcal{E}}{\partial x_j}.
\end{aligned} \tag{8.9}$$

where $e_i \equiv \frac{d\mathcal{E}}{dy_i}$. For the squared error cost function, $e_i = y_i - d_i$ where d_i is the desired target. Applying the operator $\mathcal{R}_z\{\cdot\}$ to the above backward pass equations gives

$$\begin{aligned}
\mathcal{R}_z\left\{\frac{\partial \mathcal{E}}{\partial y_i}\right\} &= \mathcal{R}_z\{e_i(y_i)\} = e_i'(y_i)\mathcal{R}_z\{y_i\} && \text{(output nodes)} \\
\mathcal{R}_z\left\{\frac{\partial \mathcal{E}}{\partial y_i}\right\} &= \sum_j \left(\omega_{ji}\mathcal{R}_z\left\{\frac{\partial \mathcal{E}}{\partial x_j}\right\} + z_{ji}\frac{\partial \mathcal{E}}{\partial x_j}\right) && \text{(non-output nodes)} \\
\mathcal{R}_z\left\{\frac{\partial \mathcal{E}}{\partial x_i}\right\} &= \sigma_i'(x_i)\,\mathcal{R}_z\left\{\frac{\partial \mathcal{E}}{\partial y_i}\right\} + \mathcal{R}_z\{x_i\}\sigma_i''(x_i)\frac{\partial \mathcal{E}}{\partial y_i} \\
\mathcal{R}_z\left\{\frac{\partial \mathcal{E}}{\partial \omega_{ji}}\right\} &= y_i\,\mathcal{R}_z\left\{\frac{\partial \mathcal{E}}{\partial x_j}\right\} + \mathcal{R}_z\{y_i\}\frac{\partial \mathcal{E}}{\partial x_j}.
\end{aligned} \tag{8.10}$$

The product $Rz = \mathcal{R}_z\{\partial \mathcal{E}/\partial \omega_{ji}\}$ is then obtained from the last equation. Note that only a single forward-back pass is needed beyond the normal forward-backward propagation pass needed to compute the activations and gradients.

8.2 Linearized Hessian

To improve stability we can force R to be positive definite by using the linearized Hessian. Suppose we have the cost function

$$\mathcal{E} = \frac{1}{2}\sum_{k=1}^{m}(d_k - y_k(x,\omega))^2 \tag{8.11}$$

where y_k is the output of the k^{th} output node when input x is presented and d_k is the associated target. Then the instantaneous estimate of the Hessian is

$$\frac{\partial^2 \mathcal{E}}{\partial \omega_{ij}\partial \omega_{lm}} = \sum_{k=1}^{m}\left((d_k - y_k)\frac{\partial^2 y_k}{\partial \omega_{ij}\partial \omega_{lm}} + \frac{\partial y_k}{\partial \omega_{ij}}\frac{\partial y_k}{\partial \omega_{lm}}\right). \tag{8.12}$$

The linearized Hessian is obtained by assuming the outputs are on average close to the targets (this is valid at late times in the training) so that the first term can be neglected to give

$$\frac{\partial^2 \mathcal{E}}{\partial \omega_{ij} \partial \omega_{lm}} \approx \sum_{k=1}^{m} \frac{\partial y_k}{\partial \omega_{ij}} \frac{\partial y_k}{\partial \omega_{lm}}.$$

This approximation can be further justified if we assume that we have an unbiased model, i.e. $y_k(x) = E[d_k|x]$ when the network is fully trained. In this case, the expected value of the first term can be written as

$$\begin{aligned}
\left\langle (y_k(x) - d_k) \frac{\partial^2 y_k}{\partial \omega_{ij} \partial \omega_{lm}} \right\rangle_{x,q} &= \int \left(\int d(d_k)' d_k' P(d_k'|x) - d_k \right) \frac{\partial^2 y_k}{\partial \omega_{ij} \partial \omega_{lm}} P(d_k|x) P(x) d(d_k) dx \\
&= \int d_k' P(d_k'|x) \frac{\partial^2 y_k}{\partial \omega_{ij} \partial \omega_{lm}} P(x) d^{(k)\prime} dx - \int d_k P(d_k|x) \frac{\partial^2 y_k}{\partial \omega_{ij} \partial \omega_{lm}} P(x) d(d_k) dx \\
&= 0
\end{aligned}$$

Thus, the expected value of the Hessian and the linearized Hessian are the same for a fully trained network.

8.2.1 Computing the Linearized Hessian

The linearized Hessian can be obtained from the full Hessian by setting the targets to $d_k = y_k$ (see equation (8.12)). The forward equations in (8.8) remain the same and the backward equations in (8.10) reduce to

$$\begin{aligned}
\mathcal{R}_z \left\{ \frac{\partial \mathcal{E}}{\partial y_i} \right\} &= \mathcal{R}_z \{y_i\} && \text{(output nodes)} \\
\mathcal{R}_z \left\{ \frac{\partial \mathcal{E}}{\partial y_i} \right\} &= \sum_j \omega_{ji} \mathcal{R}_z \left\{ \frac{\partial \mathcal{E}}{\partial x_j} \right\} && \text{(non-output nodes)} \\
\mathcal{R}_z \left\{ \frac{\partial \mathcal{E}}{\partial x_i} \right\} &= \sigma_i'(x_i) \mathcal{R}_z \left\{ \frac{\partial \mathcal{E}}{\partial y_i} \right\} \\
\mathcal{R}_z \left\{ \frac{\partial \mathcal{E}}{\partial \omega_{ji}} \right\} &= y_i \mathcal{R}_z \left\{ \frac{\partial \mathcal{E}}{\partial x_j} \right\}. && (8.13)
\end{aligned}$$

We tested the algorithm using both the linearized and the full Hessian and found no significant differences in the result. Since the linearized equations are simpler to compute and in some circumstances may be more stable, we chose to use the linearized equations.

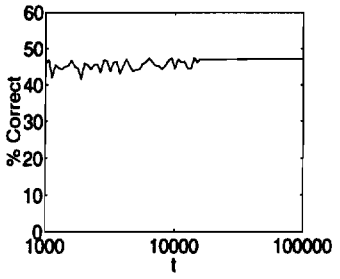


Figure 8.1: Phoneme Classification on training set: Learning rate is held fixed until the noise regime is reached, then either 1) annealing ($\frac{\mu_0}{t}$) is turned on (solid) or annealing is turned on with adaptive momentum (dotted). $\mu_0 = 1$.

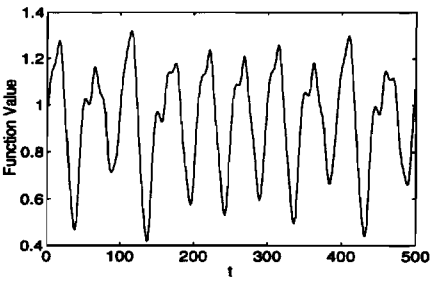


Figure 8.2: Mackey-Glass Time Series for $\tau = 17$, $a = 0.2$, and $b = 0.1$.

8.3 Simulations

8.3.1 Phoneme Classification

We next use phoneme classification as an example of a large nonlinear problem. The database consists of 9000 phoneme vectors taken from 48 50-second speech monologues. Each input vector consists of 70 perceptual linear predictive (PLP[1]) coefficients [Her90]. There are 39 target classes. The architecture was a standard fully connected feedforward network with 71 (includes bias) input nodes, 70 hidden nodes, and 39 output nodes for a total of 7700 weights.

Figure 8.1 plots the percent of phonemes correctly classified as a function of number of input presentations (timesteps). We trained the network with a fixed learning rate of $\mu_0 = 1$ for a long time until the noise regime was reached. The early training is not shown in the figure, however, it will be discussed a little later. At around $t = 15000$ in the figure the annealing schedule of μ_0/t was started. The solid curve corresponds to $\beta = 0$ and the dotted curve corresponds to adaptive momentum. For $\beta = 0$, the noise is reduced but the percent classification does not improve. For adaptive momentum, there is a large improvement in the percent classification.

[1]PLP is an all pole model of the auditory spectrum of speech that incorporates three basic concepts from the psychophysics of hearing: critical band resolution curves, the equal loudness curve, and the intensity-loudness power-law relation.

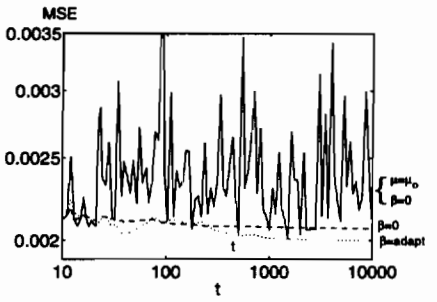


Figure 8.3: Mackey-Glass: Learning rate is held fixed until the noise regime is reached, then ($t = 10$ in figure) annealing ($\frac{\mu_0}{t}$) is turned on with either $\beta = 0$ or adaptive momentum (dotted). Learning curve for constant learning rate without momentum is also shown for comparison. $\mu_0 = .127$.

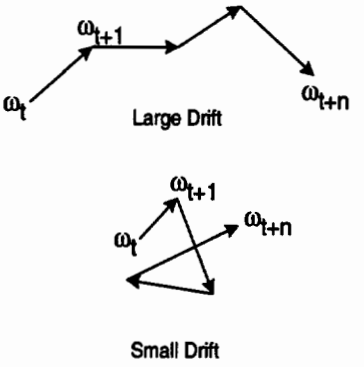


Figure 8.4: Large drift indicates large correlation between weight updates. Small drift indicates that the weights are in the noise regime.

8.3.2 Mackey-Glass

In this section we look at the chaotic time series generated by integrating[2] the Mackey-Glass differential delay equation

$$\dot{x} = -bx(t) + \frac{ax(t-\tau)}{1 + (x(t-\tau))^{10}}.$$

The series for $\tau = 17$, $a = 0.2$, and $b = 0.1$ is shown in figure (8.2).

We trained a network with 4 inputs, 38 hidden units, and one output using gradient descent to predict $x(t + 85)$ from the inputs {x(t), x(t-6), x(t-12), x(t-18)}. As in the phoneme problem, we first trained the network at constant learning rate. The LeCun, et.al. algorithm estimated the maximum eigenvalue of the Hessian at our starting weight value to be about $\lambda_{max} = 3.93$ so that $\mu_0 = .5/\lambda_{max} = .127$. Once the MSE leveled out we turned on annealing with either $\beta = 0$ or adaptive momentum. Figure 8.3 displays the results. The learning curve without annealing is also shown for comparison. Since we do not know the precise minimum MSE we can not plot the misadjustment.

[2]Code for generating the series was provided by John Moody.

8.4 When to Anneal

For adaptive momentum to be useful, we need a method that automatically detects when the noise regime has been reached. To do this we turn to Moody and Darken's Adaptive-Search-Then-Converge (ASTC) algorithm which uses a statistic called the drift to monitor the noise present in the weight updates. Small drift indicates low correlation between subsequent weight changes, i.e. weights in the noise regime. Large drift indicates that weight changes are highly correlated and that learning is still taking place (see Figure 8.4).

Darken [Dar93] discusses several ways of defining the drift. We use the version that is most practical from an implementation perspective

$$d(t) \equiv \sqrt{T}\frac{\langle \nabla \mathcal{E}_s \rangle_T}{\sqrt{\langle (\nabla \mathcal{E}_s - \langle \nabla \mathcal{E}_s \rangle_T)^2 \rangle_T}} \tag{8.14}$$

where $\langle \cdot \rangle_T$ indicates an exponential average over $s \leq t$ with weighting factor $1/T$. T starts out small and is slowly increased over time. The numerator is proportional to the average stepsize while the denominator is proportional to the standard deviation of the stepsize. Darken shows that for $\mu_0 < \mu_{opt}$, the drift should increase at a rate of $A + B\sqrt{t}$ for some constants A and B. However, for large t this increases very slowly so that from a practical standpoint he argues that it can be taken as a constant that he sets at 2. Thus, $d(t) > 2$ indicates that $\mu_0 < \mu_{opt}$.

The drift is computed at each iteration. The algorithm starts out with a constant learning rate (called "search" mode). When all components of the drift have changed sign at least once, the learning rate is annealed (called "converge" mode). At this point the algorithm continues to switch between search and converge modes for the remainder of the training as follows:

1. When in converge mode: If one or more components of the drift exceed threshold=2, then switch to search mode.

2. When in search mode: If all the components of the drift, that had exceeded threshold=2 in the previous converge mode, change sign then switch to converge mode.

We tried several other values of the threshold but without improvement. One problem with this algorithm is that the learning rate is either constant or decreasing, i.e. there

is no principled mechanism for increasing the learning rate if it is initially set too small. As a result, it is best to start with a learning rate that is as large as possible.

Several variants of the learning rate schedule can be used during the converge phase. For example, we can let $\mu(t) = \mu_0/(\tau + t)$ where μ and τ are parameters that must be chosen. At late times this behaves as μ_0/t, however, it drops off more slowly at early times. In order to include as few adjustable parameters as possible, we implement the simple μ_0/t schedule.

Convergence behavior is very sensitive to the initial value of μ_0. If μ_0 is too large the weights diverge and the algorithm (eventually) switches to converge mode where the learning rate is then decreased. However, recuperation from the initial divergence can take many iterations. On the other hand, if μ_0 is too small, convergence is very slow because the algorithm has no mechanism for increasing the learning rate. Ideally, we want the initial learning rate (during search mode) to be as large as possible without having the weights diverge. As previously discussed in Section §7.3, the bound for stochastic LMS with constant learning rate and gaussian inputs is $\mu_0 < \frac{2}{3\lambda_{max}}$ for convergence in mean square. Clearly, for nonlinear problems where the cost surface is not quadratic and inputs are often far from gaussian this bound will not be correct. However, for lack of a better initial estimate, we again set $\mu_0 = \frac{1}{2\lambda_{max}}$, i.e., slightly below this bound.

We approximate λ_{max} for nonlinear networks just as we did for linear networks, using the algorithm by LeCun, *et. al.* [LSP93] described in Section §7.3. However, the Hessian is not constant throughout the weight space for nonlinear problems so that even if our initial estimate of the λ_{max} is accurate, the eigenvalue spectrum may change after a number of weight updates. One possibility is to periodically recompute $\mu_0 = \frac{1}{2\lambda_{max}}$, however, in our current implementation μ_0 is computed only once at the start of training.

8.4.1 ASTC with Adaptive Momentum (MASTC)

To combine ASTC with adaptive momentum (referred to as MASTC) we use the drift to *first* detect the noise regime. Once detected, annealing is turned on with adaptive momentum and the drift is no longer computed. There is no switching back and forth between the two phases because a) we assume that once we are in some noise regime there is no switching between basins of local minima, and b) since adaptive momentum results in an effective learning rate that is optimal for that basin there should be no need to return to a constant learning rate. When ASTC is used *without* adaptive momentum,

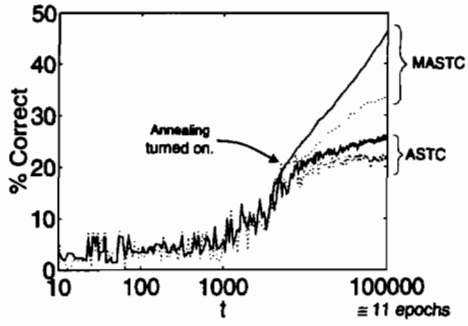


Figure 8.5: Phoneme Classification: Percent Correct on training set (solid) and test set (dotted) as function of the number of input presentations. Top two curves correspond to MASTC. The lower two curves correspond to ASTC. $\mu_0 = 1$.

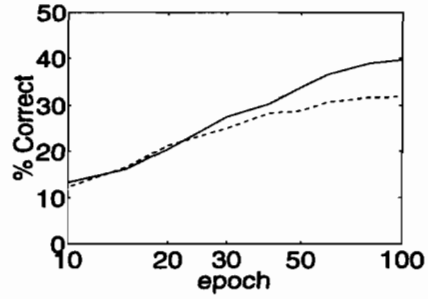


Figure 8.6: Phoneme Classification using Conjugate Gradient Descent: Percent Correct on training set (solid) and test set (dotted) as function of the number of epochs. One epoch equals one pass through the data, i.e. 9000 input presentations.

the scalar learning rate μ_0 is far from optimal particularly if the Hessian is poorly conditioned. Thus the learning rate schedule μ_0/t generally decreases the learning rate too quickly[3]. As a result, the algorithm thrashes between search and converge modes in an attempt to correct for this.

8.5 ASTC and MASTC Applied to Phoneme Classification

We now return to the phoneme classification problem in section §8.3.1 . We compare 1) ASTC, 2) MASTC, and 3) conjugate gradient descent which is a batch algorithm. The maximum eigenvalue was computed to be about $\lambda_{max} = 1.33$, however, a learning rate of $\mu_0 = 1$ did not result in divergence. Figure 8.5 displays the results for both training set (solid) and test set (dotted). MASTC (top two curves) performs much better than straight ASTC. Though not shown in this figure, we actually saw improvement in the classification by keeping the learning rate constant far past the point where ASTC first switches to converge. This means that ASTC signals the arrival in the noise regime too

[3]The optimal learning rate along the i^{th} eigendirection of the Hessian is $\mu_{0,opt} = \frac{1}{\lambda_i}$. However, we know that $\mu_0 < 1/\lambda_{max}$ otherwise the weights would diverge in the search phase. Thus, μ_0 is too small along the eigendirections for which $\lambda_i < \lambda_{max}$.

early. Further analysis is required to determine how to correct this problem.

Both ASTC and MASTC train faster than conjugate gradient descent[4] (CGD). Figure 8.6 displays the classification performance as a function of epoch. One epoch corresponds to one pass through the data (9K inputs).

Timing estimates showed that one "epoch"[5] of MASTC is roughly equivalent to about 9 epochs of CGD. However, this number possibly can be decreased because the CGD code has been carefully optimized whereas MASTC was not. After 100 epochs the CGD performance was only about 39% on the training set. The performance for ASTC with adaptive momentum after the only 100,000 input presentations was about 47%. Performance on the test set was only slightly better for MASTC.

A rough comparison of the relative complexity of the two calculations shows that one epoch of adaptive momentum should take about 3 times as long as one epoch of CGD. To show this, we let N be the number of weights in the network and we assume that the number of training examples, T, is large compared to N. This means that the calculation of the gradient is the dominant factor in CGD since the gradient calculation is $\mathcal{O}(NT)$ and the remaining calculations are $\mathcal{O}(N)$. We shall also assume that the total number of weights is large compared to the number of nodes so that terms on the order of the number of nodes can be neglected.

For *each* training example, the gradient calculation requires one forward-backward pass as given in equations (8.7) and (8.9). The forward calculation requires approximately $2N$ add/multiplies. For example, consider a network with n_0 input nodes, n_1 hidden nodes, and n_2 output nodes. The total number of weights is $N = n_1(n_0+n_1)$. To compute the hidden layer activations, we must perform $n_0 n_1$ multiples plus $n_1(n_0 - 1)$ adds. The output activations then require $n_1 n_2$ multiplies plus $n_2(n_1 - 1)$ adds. The total number of add/multiples is then

$$n_1 n_0 + n_1(n_0 - 1) + n_1 n_2 + n_2(n_1 - 1) = 2n_1(n_2 + n_0) - (n_1 + n_2) \approx 2n_1(n_0 + n_2) = 2N. \tag{8.15}$$

Note that the sigmoid calculation has not been included because it is on the order of the

[4]Conjugate gradient descent was performed using *nopt* written by Etienne Barnard and made available through the Center for Spoken Language Understanding at the Oregon Graduate Institute.

[5]For stochastic gradient descent, we define an epoch to be n input presentations, where n is the number of training examples. Because the n inputs are chosen at random with replacement from the entire training set, some inputs may occur several times and some may not occur at all over the course of one epoch.

number of nodes rather than the number of weights.

The gradient is then obtained from the backward calculation given in equations (8.9). The gradients for the top layer weights are given by

$$\frac{\partial \mathcal{E}}{\partial \omega_{ji}} = y_i \, \sigma_j' e_j = y_i \, \delta_j$$

where $\delta_i \equiv \frac{\partial \mathcal{E}}{\partial x_i}$, $i = 1, \ldots, n_1$, and $j = 1, \ldots, n_2$. Computing δ_j is $\mathcal{O}(n_2)$ and so can be neglected. Thus, the number of multiplies is $n_1 n_2$. The gradients for the next layer are given computed as

$$\frac{\partial \mathcal{E}}{\partial \omega_{ik}} = y_k \, \sigma_i' \sum_j \omega_{ji} \, \delta_j = y_k \, \delta_i$$

where i and j range as before, and $k = 1, \ldots, n_0$. Computing the δ_i require approximately $2n_1 n_2$ add/multiplies. Computing the gradients from the δ_i's then requires $n_0 n_1$ multiples. Thus the backward calculation takes approximately

$$n_1 n_2 + 2n_1 n_2 + n_0 n_1 = 3n_1 n_2 + n_0 n_1 = 2n_1 n_2 + N$$

If the number of inputs is on the order of the number of outputs (i.e. $n_0 \sim n_2$), then $N \approx 2n_0 n_1 \approx n_1 n_2$ so that the above expression is approximately $2N$. Thus the forward-backward calculation for *one* training example is $\mathcal{O}(2N + 2N = 4N)$.

For batch learning, the gradient for each example must be added into the overall average gradient. This requires N adds for each example. Therefore, to compute the average gradient for batch mode is approximately $\mathcal{O}(5N)$ per example.

For stochastic gradient descent, each training example requires the $4N$ add/multiplies as discussed above for the forward-backward pass. In addition, each weight update

$$\omega_{t+1} = \omega_t - \mu \nabla \mathcal{E}_t$$

requires N multiples (by the learning rate) plus N subtractions to obtain ω_{t+1}. This gives approximately $6N$ add/multiplies.

We now include the adaptive momentum term

$$\Delta \omega_t + \mu R_t \Delta \omega_t \tag{8.16}$$

where $R\Delta\omega_t$ is computed using the linearized hessian given in equations (8.8) and (8.13). The forward calculation in (8.8) is similar in complexity to the forward equations in (8.7)

except that there is an additional matrix multiplication (from the z term) requiring approximately $2N$ add/multiples. The backward equations in (8.13) have the same complexity as the standard backward equations in (8.9) so that the total forward-backward calculation for computing $R\Delta\omega_t$ takes about

$$4N \text{ (forward)} + 2N \text{ (backward)} = 6N$$

add/multiplies.

Given $R\Delta\omega_t$, the momentum term in (8.16) then requires N multiplies (by μ), plus N adds (to add in $\Delta\omega_t$). Finally adding the whole result to update ω_{t+1} takes another N adds. Thus, the momentum terms requires $6N+N+N+N = 9N$ add/multiples on top of the 6N add/multiplies for the standard stochastic update calculations. Thus, stochastic gradient descent with adaptive momentum takes about $6N+9N = 15N$ add/multiplies, or about 3 times as many as CGD.

We also note that adaptive momentum is used only during the annealing phase and not throughout the entire training process. Before annealing, the drift term in ASTC is computed to determine when annealing should be turned on. Once annealing is turned on, the drift is no longer computed but adaptive momentum is turned on. The complexity of the drift is similar to that of adaptive momentum. The numerator of the drift is an exponential average of the weight change. This requires 3N add/multiples per training example. The denominator is an exponential average of the standard deviation of the weight change and requires about 5N add/multiplies. Finally, the numerator must get divided by the denominator. Thus, the drift computation is about the same complexity as the computation of the adaptive momentum term, 9N. There is also the additional cost ($\sim N$) associated with checking to see if the drift is above or below threshold.

8.6 Summary

In this chapter we have presented an efficient algorithm for computing nonlinear adaptive momentum that makes use of Pearlmutter's algorithm [Pea94] for computing the product of the Hessian and an arbitrary vector. We test the algorithm on a phoneme classification problem and on the Mackey-Glass problem during the (late-time) annealing phase of learning.

For adaptive momentum to be effective, we need to know when the noise regime has been reached. To this end, we use the drift statistic from Moody and Darken's ASTC

algorithm to signal this transition. Unfortunately, the drift statistic tends to signal the noise regime too early. Further work is needed to correct this.

Chapter 9

Future Directions

To develop a complete algorithm for stochastic search there are still major issues that must be solved. The first and most critical one is how to speed learning *before* the noise regime has been reached. This is important because most of the training time is spent in the constant learning rate phase. In fact, for some problems, over training can occur even before the noise regime is reached, thus making annealing unnecessary. One possible technique that we discuss below is adaptive momentum modified to work in the constant learning rate phase.

Another issue is how to improve ASTC so that 1) it better detects the noise regime and 2) has some principled method for increasing the learning rate when needed. This latter point, however, is tied to the issue of optimizing performance in the constant learning rate phase, i.e. solving one probably solves the other.

In the long run, however, a better approach to training neural networks may be a hybrid algorithm, i.e. one that uses a combination of batch and stochastic techniques.

9.1 Stochastic Algorithms for the Search Phase

As was mentioned in the introduction, speed-up techniques often work by estimating algorithm parameters (e.g. learning rates) by modeling the local curvature of the cost surface. For stochastic algorithms this is difficult because stochastic estimates of curvature are too noisy to be of much use. An alternative is to use time averaging to remove the noise. However, a problem with time averaging is that it is difficult to know how to weight past values. For example, suppose we have the quantity (call it X) whose exponential average is given by

$$\langle X \rangle_t = (1-\alpha)X_t + \alpha \langle X \rangle_{t-1}$$

$$= (1-\alpha)\sum_{i=0}^{t}\alpha^i X_{t-i}.$$

If X_t is changing rapidly, then the exponential weighting factor α should be small so that more recent values of X_t are weighted more heavily than earlier values. In such a case, the estimate of $\langle X\rangle_t$ can still be quite noisy and will lag the true value. If X_t is not changing rapidly then it is better to have the weighting factor be large so that more terms are contributing in a significant way to the average. The more terms included, the more the noise will be reduced. In adaptive momentum, we average the gradients. In the noise regime the average gradient is close to zero and not changing rapidly so that the update term is very approximately

$$\begin{aligned}\omega(t+1) &= \mu_0\sum_{i=0}^{t}(I-\mu_0 R)^i\,\nabla\mathcal{E}(\omega_{t-i},x_{t-i})\\ &\approx \left(\mu_0\sum_{i=0}^{\infty}(I-\mu_0 R)^i\right)\langle\nabla\mathcal{E}(\omega_t,x_t)\rangle\\ &= R^{-1}\,\langle\nabla\mathcal{E}(\omega_t,x_t)\rangle_x. \qquad (9.1)\end{aligned}$$

This is precisely the Newton step update. However, if the gradient is changing rapidly as it is in the constant learning rate phase, it can not be pulled out of the sum. This means we are no longer approximating the Newton update. One approach to correct this may be to rescale the gradients in the exponential average in accordance to how much they are expected to change.

Another difficulty in applying adaptive momentum to the constant learning rate phase is that the optimal momentum parameter does not have a simple form. Even for the very simple case of 1-D LMS, the optimal β must satisfy [TT89]

$$(1-\sqrt{\beta})^2 < \mu_0\lambda < (1+\sqrt{\beta})^2. \qquad (9.2)$$

In multiple dimensions λ could be replaced with R. Solving for β would require negotiating square roots of matrices. Would stochastic estimates be possible? Hopefully, approximations could be made that would be efficient to calculate and accurate enough to make a difference in the learning speed.

9.2 Detecting the Noise Regime

One of the problems with ASTC was that, for both linear and nonlinear problems, it consistently switched from search phase to converge phase to soon. However, the *first* converge phase was usually very short (sometimes only a few iterations) indicating that the algorithm quickly realizes that it switched too quickly. Unfortunately, by this point the learning rate had already decreased quite significantly[1].

ASTC detects end of the first search phase when the all components of the drift have changed sign at least once. One possible reason that this might occur too early may be due to the noise present in the drift estimate. Recall that the drift is computed as an exponential time average whose weighting factors are hardwired. The noise, which tends to be larger early in the training, may result in spurious sign changes. One simple solution would be not register the occurrence of any sign changes until a few epochs have passed. This of course is not particularly satisfactory because there may be instances where annealing needs to be done early, for example, if the initial learning rate chosen is too large. Alternative solutions need to be found.

9.3 Hybrid Algorithms

It may be that some mix between batch and stochastic is needed. By "mix" we mean either 1) batch and stochastic are used at different stages of training, or 2) the batch size itself is varied. For example, in the first approach, it may be that stochastic is better to use early in the training where the noise is useful in exploring the cost surface and where careful estimates of the cost surface are not yet necessary. Once a good basin is found, some fast batch algorithm such as conjugate gradient descent might be more effective at finding the minimum.

The second approach could be used as an alternative to annealing the learning rate. The amount of noise present in the weight updates is controlled either by the learning rate or by the batch size. Batch learning (zero noise) generally refers to averaging over the entire training set at each iteration. One could also use "small batches" where

[1]This would be less of a problem if we used an annealing schedule that rolled over more slowly than $\frac{\mu_0}{t}$, e.g $\frac{\mu_0}{t+\tau}$. However, such schedules invariably introduce additional parameters that must be adjusted. One of our goals was to minimize the number of adjustable parameters so we did not use these alternative schedules.

the weight updates at each iteration are averaged over some subset (perhaps randomly chosen) of the training set. A batch size of 1 would correspond to pure stochastic while a batch size equal to the size of the training set would correspond to pure batch. Thus, an alternative to annealing the learning rate would be to slowly increase the batch size as needed so as to decrease the noise *without* annealing. We would need to 1) determine the batch size schedule that optimizes learning and 2) determine how this method compares in performance with standard annealing. At each iteration, the cost of the batch average must be weighed against the amount of noise that can be tolerated. As an aside, we also note that small batches could be practical when algorithms are run on multiprocessor systems where batch algorithms are easier to implement in parallel.

Bibliography

[BLLS71] Dick Bedeaux, Katja Lakatos-Lindenberg, and Kurt E. Shuler. On the relation between master equations and random walks and their solutions. *Journal of Mathematical Physics*, 12(10):2116–2123, 1971.

[Dar93] Christian Darken. *Learning Rate Schedules for Stochastic Gradient Algorithms.* PhD thesis, Yale University, 1993.

[DV93] R. Der and Th. Villmannn. Dynamics of self organized feature mapping. unpublished, 1993.

[Gar90] C.W. Gardiner. *Handbook of Stochastic Methods, 2nd Ed.* Springer-Verlag, Berlin, 1990.

[GG93] Allen Gersho and Robert M. Gray. *Vector Quantization and Signal Compression.* Kluwer Academic Publishers, Norwell, MA, 1993.

[Gol87] Larry Goldstein. Mean square optimality in the continuous time Robbins Monro procedure. Technical Report DRB-306, Dept. of Mathematics, University of Southern California, LA, 1987.

[Her90] Hynek Hermansky. Perceptual linear predictive (PLP) analysis of speech. *Journal of the Acoustical Society of America*, 87(4), April 1990.

[Hes94] Tom M. Heskes. On Fokker-Planck approximations of on-line learning processes. *Journal of Physics A*, 27:5145–5160, 1994.

[HK93] Tom M. Heskes and Bert Kappen. On-line learning processes in artificial neural networks. In J. Taylor, editor, *Mathematical Foundations of Neural Networks*, pages 199–233. Elsevier, Amsterdam, 1993.

[HPS93] L. Hansen, R. Pathria, and P. Salamon. Stochastic dynamics of supervised learning. *Journal of Physics A*, pages 26:63–71, 1993.

[HSK92] Tom M. Heskes, Eddy T.P. Slijpen, and Bert Kappen. Learning in neural networks with local minima. *Physical Review A*, 46(8):5221–5231, 1992.

[LM93] Todd K. Leen and John E. Moody. Weight space probability densities in stochastic learning: I. Dynamics and equilibria. In Giles, Hanson, and Cowan, editors, *Advances in Neural Information Processing Systems, vol. 5*, pages 451–458. Morgan Kaufmann, San Mateo, CA, 1993.

[LO92] Todd K. Leen and Genevieve B. Orr. Weight-space probability densities and convergence times for stochastic learning. In *International Joint Conference on Neural Networks*, volume IV, pages 158–164. IEEE, June 1992.

[LO94] Todd K. Leen and Genevieve B. Orr. Optimal stochastic search and adaptive momentum. In Cowan, Tesauro, and Alspector, editors, *Advances in Neural Information Processing Systems, vol. 6*, pages 477–484. Morgan Kaufman, San Mateo, CA, 1994.

[LP91] P. Lisboa and S. Perantonis. Complete solution of the local minima in the XOR problem. *Network: Computation in Neural Systems*, 2:119, 1991.

[LSP93] Yann LeCun, Patrice Y. Simard, and Barak Pearlmutter. Automatic learning rate maximization by on-line estimation of the Hessian's eigenvectors. In Giles, Hanson, and Cowan, editors, *Advances in Neural Information Processing Systems, vol. 5*, pages 156–163. Morgan Kaufmann, San Mateo, CA, 1993.

[OL93] Genevieve B. Orr and Todd K. Leen. Weight space probability densities in stochastic learning: II. Transients and basin hopping times. In Giles, Hanson, and Cowan, editors, *Advances in Neural Information Processing Systems, vol. 5*, pages 507–514. Morgan Kaufmann, San Mateo, CA, 1993.

[OL94] Genevieve B. Orr and Todd K. Leen. Momentum and optimal stochastic search. In M. C. Mozer, P. Smolensky, D. S. Touretzky, J. L. Elman, and A. S. Weigend, editors, *Proceedings of the 1993 Connectionist Models Summer School*, pages 351–357. Lawrence Erlbaum Associates, Hillsdale, NJ, 1994.

[Pea94] Barak A. Pearlmutter. Fast exact multiplication by the Hessian. *Neural Computation*, 6:147–160, 1994.

[PFTV87] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes – the Art of Scientific Computing.* Cambridge University Press, New York, 1987.

[Ris89] H. Risken. *The Fokker-Planck Equation.* Springer-Verlag, Berlin, 1989.

[RS88] H. Ritter and K. Schulten. Convergence properties of Kohonen's topology conserving maps: Fluctuations, stability and dimension selection. *Biological Cybernetics*, 60:59–71, 1988.

[RSW90] G. Radons, H.G. Schuster, and D. Werner. Fokker-Planck description of learning in backpropagation networks. In *International Neural Network Conference - INNC 90, Paris*, volume II, pages 993–996. Kluwer Academic Publishers, Norwell, MA, July 1990.

[SR90] John J. Shynk and Sumit Roy. Analysis of the momentum LMS algorithm. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 38(12):2088–2098, 1990.

[TT89] Mehmet Ali Tugay and Yalcin Tanik. Properties of the momentum LMS algorithm. *Signal Processing*, 18:117–127, 1989.

[Ven67] J. H. Venter. An extension of the Robbins-Monro procedure. *Annals of Mathematical Statistics*, 38:181–190, 1967.

[vK92] N. van Kampen. *Stochastic Precesses in Physics and Chemistry.* North-Holland, Amsterdam, 1992.

Appendix A

Expansion of the Kolmogorov Equation

In this appendix we derive the differential difference equation in (2.4) by starting with the Kolmogorov equation given in (2.2). With single step transition probability defined in (2.3), the Kolmogorov equation is

$$\begin{aligned} P(\omega', (n+1)\tau) &= \int d\omega\, P(\omega, n\tau) \int dx\, \rho(x)\, \delta(\omega' - \omega - \mu\, H[\omega, x]\,) \\ &= \int dx\, \rho(x) \int d\omega\, P(\omega, n\tau)\, \delta(\omega' - \omega - \mu\, H[\omega, x]\,) \end{aligned} \tag{A.1}$$

where $\omega, \omega' \in I\!R^m$. We next make a Taylor series expansion of the δ function where ω_{j_α} and H_{j_α} are the j_α^{th} component of weight, and weight update, respectively,

$$\begin{aligned} P(\omega', (n+1)\tau) &= \int dx\, \rho(x) \int d\omega\, P(\omega, n\tau) \sum_{i=1}^{\infty} \frac{(-\mu)^i}{i!} \\ &\qquad \sum_{j_1,\ldots j_i=1}^{m} H_{j_1} H_{j_2} \ldots H_{j_i}\, \frac{\partial^i}{\partial\omega'_{j_1}\, \partial\omega'_{j_2} \ldots \partial\omega'_{j_i}}\, \delta(\omega' - \omega) \\ &= \int dx\, \rho(x) \sum_{i=1}^{\infty} \frac{(-\mu)^i}{i!} \sum_{j_1,\ldots j_i=1}^{m} \frac{\partial^i}{\partial\omega'_{j_1}\, \partial\omega'_{j_2} \ldots \partial\omega'_{j_i}} \\ &\qquad \int d\omega\, \delta(\omega' - \omega)\, P(\omega, n\tau)\, H_{j_1} H_{j_2} \ldots H_{j_i} \\ &= \int dx\, \rho(x) \sum_{i=1}^{\infty} \frac{(-\mu)^i}{i!} \sum_{j_1,\ldots j_i=1}^{m} \\ &\qquad \frac{\partial^i}{\partial\omega'_{j_1}\, \partial\omega'_{j_2} \ldots \partial\omega'_{j_i}} \left\{ H_{j_1} H_{j_2} \ldots H_{j_i}\, P(\omega', n\tau) \right\}. \end{aligned} \tag{A.2}$$

Integrating over x and moving the $i = 0$ term to the left side leaves the differential-difference

$$\tau\, \frac{P(\omega', (n+1)\tau) \;-\; P(\omega', n\tau)}{\tau} \;=$$

$$\sum_{i=1}^{\infty} \frac{(-\mu)^i}{i!} \sum_{j_1,\ldots j_i=1}^{m} \frac{\partial^i}{\partial\omega'_{j_1}\,\partial\omega'_{j_2}\ldots\partial\omega'_{j_i}} \left(\langle H_{j_1} H_{j_2} \ldots H_{j_i} \rangle_x \; P(\omega', n\tau) \right). \quad \text{(A.3)}$$

Appendix B

Small Noise Expansion for Stochastic learning with Constant Learning Rate

The Kramers-Moyal equation for stochastic learning with a constant learning rate in 1-dimension is

$$\tau \ \partial_t P(\omega,t) = \sum_{i=1}^{\infty} \frac{(-\mu)^i}{i!} \frac{\partial^i}{\partial \omega^i} \ \{ \ \langle H^i(\omega,x)\rangle_x P(\omega,t)\} \, . \tag{B.1}$$

The small noise expansion requires making a change of variable from $\{\omega,t\}$ to $\{\xi,s\}$ where

$$\begin{aligned} \xi &= \frac{1}{\sqrt{\mu}}(\omega - \phi(t)). \\ s &= t \end{aligned}$$

The partial derivatives transform as

$$\begin{aligned} \frac{\partial}{\partial t} &= \frac{\partial \xi}{\partial t}\frac{\partial}{\partial \xi} + \frac{\partial s}{\partial t}\frac{\partial}{\partial s} = \frac{-\dot{\phi}}{\sqrt{\mu}}\frac{\partial}{\partial \xi} + \frac{\partial}{\partial s} \\ \frac{\partial}{\partial \omega} &= \frac{\partial \xi}{\partial \omega}\frac{\partial}{\partial \xi} + \frac{\partial s}{\partial \omega}\frac{\partial}{\partial s} = \frac{1}{\sqrt{\mu}}\frac{\partial}{\partial \xi} \end{aligned}$$

and the density transforms as

$$P(\omega,t) = \mathcal{P}(\xi,s)\left|\frac{\partial(\xi,s)}{\partial(\omega,t)}\right| = \frac{1}{\sqrt{\mu}}\mathcal{P}(\xi,s) \tag{B.2}$$

where $\mathcal{P}(\xi,s)$ denotes the density of ξ at time s and $\frac{\partial(\xi,s)}{\partial(\omega,t)}$ is the Jacobian.

With these coordinate changes, (B.1) becomes (renaming $s \to t$)

$$\tau \frac{\partial P(\xi,t)}{\partial t} - \frac{\tau \dot{\phi}}{\sqrt{\mu}}\frac{\partial P(\xi,t)}{\partial \xi} = \sum_{i=1}^{\infty} \frac{(-\sqrt{\mu})^i}{i!} \frac{\partial^i}{\partial \xi^i}\Big(\ \langle H^i(\phi(t)+\sqrt{\mu}\,\xi)\rangle_x P(\xi,t)\Big) \, . \tag{B.3}$$

Expanding $H^i(\phi(t)+\sqrt{\mu}\,\xi,x)$ in a power series about ϕ gives

$$\left\langle H^i(\phi(t)+\sqrt{\mu}\,\xi,x)\right\rangle_x = \sum_{k=0}^{\infty} \frac{(\sqrt{\mu}\,\xi)^k}{k!}\,\alpha_i^{(k)}(\phi) \tag{B.4}$$

where

$$\alpha_i^{(k)}(\phi) \equiv \frac{\partial^k}{\partial\omega^k}\left\langle H^i(\omega,x)\right\rangle_x\Big|_{\omega=\phi(t)}. \tag{B.5}$$

Now inserting this into (B.3) results in

$$\tau\,\frac{\partial P(\xi,t)}{\partial t} - \frac{\tau\,\dot\phi}{\sqrt{\mu}}\frac{\partial P(\xi,t)}{\partial\xi} = \sum_{i=1}^{\infty}\sum_{k=0}^{\infty}\frac{(-1)^i(\sqrt{\mu})^{i+k}}{i!\;k!}\,\alpha_i^{(k)}(\phi)\,\frac{\partial^i}{\partial\xi^i}\left(\xi^k\,P(\xi,t)\right). \tag{B.6}$$

Next make a change of index $m=i+k$ to give

$$\tau\,\frac{\partial P(\xi,t)}{\partial t} - \frac{\tau\,\dot\phi}{\sqrt{\mu}}\frac{\partial P(\xi,t)}{\partial\xi} = \sum_{m=1}^{\infty}\sum_{i=1}^{m}\frac{(-1)^i(\sqrt{\mu})^m}{i!\;(m-i)!}\,\alpha_i^{(m-i)}(\phi)\,\frac{\partial^i}{\partial\xi^i}\left(\xi^{m-i}\,P(\xi,t)\right). \tag{B.7}$$

Since ϕ is arbitrary, we can define it so that the $m=1$ term on the RHS cancels the second term on the LHS

$$-\frac{\tau\,\dot\phi}{\sqrt{\mu}}\frac{\partial P(\xi,t)}{\partial\xi} = -\sqrt{\mu}\,\alpha_1^{(0)}(\phi)\,\frac{\partial P(\xi,t)}{\partial\xi}. \tag{B.8}$$

Simplifying, obtain the system of equations

$$\frac{d\phi(t)}{dt} = \frac{\mu}{\tau}\,\alpha_1^{(0)}(\phi(t)) \tag{B.9}$$

$$\frac{\partial P(\xi,t)}{\partial t} = \frac{\mu}{\tau}\sum_{m=2}^{\infty}\sum_{i=1}^{m}\frac{(-1)^i\mu^{(m-2)/2}}{i!(m-i)!}\alpha_i^{(m-i)}(\phi(t))\frac{\partial^i}{\partial\xi^i}\left\{\xi^{m-i}P(\xi,t)\right\}. \tag{B.10}$$

Since the choice of timescale is arbitrary, we can assume that τ scales as $\mu = c\tau$ where c is some constant. Then the transformed equations become

$$\frac{d\phi(t)}{dt} = c\,\alpha_1(\phi(t)) \tag{B.11}$$

$$\frac{\partial P(\xi,t)}{\partial t} = c\sum_{m=2}^{\infty}\sum_{i=1}^{m}\frac{(-1)^i\mu^{(m-2)/2}}{i!(m-i)!}\alpha_i^{(m-i)}(\phi(t))\frac{\partial^i}{\partial\xi^i}\left\{\xi^{m-i}P(\xi,t)\right\} \tag{B.12}$$

which to lowest order ($m=2$), are independent of μ.

Appendix C

Small Noise Expansion for Annealed Learning

The Kramers-Moyal equation for stochastic learning with an annealed learning rate in 1-dimension is

$$\tau \ \partial_t P(\omega,t) = \sum_{i=1}^{\infty} \frac{(-\mu_0)^i}{t^i \ i!} \frac{\partial^i}{\partial \omega^i} \ \{ \ \langle H^i(\omega,x)\rangle_x P(\omega,t) \ \} . \tag{C.1}$$

The small noise expansion assumes one can make a change of variable from $\{\omega,t\}$ to $\{\xi,s\}$ where

$$\begin{aligned} \xi &= (\omega-\phi(t))\sqrt{\frac{t}{\mu_0}}. \\ s &= t \end{aligned}$$

The partial derivatives transform as

$$\begin{aligned} \frac{\partial}{\partial t} &= \frac{\partial \xi}{\partial t}\frac{\partial}{\partial \xi} + \frac{\partial s}{\partial t}\frac{\partial}{\partial s} \\ &= \left(\frac{\xi}{2s} - \sqrt{\frac{s}{\mu_0}}\,\dot{\phi}(s)\right)\frac{\partial}{\partial \xi} + \frac{\partial}{\partial s} \\ \frac{\partial}{\partial \omega} &= \frac{\partial \xi}{\partial \omega}\frac{\partial}{\partial \xi} + \frac{\partial s}{\partial \omega}\frac{\partial}{\partial s} \\ &= \sqrt{\frac{s}{\mu_0}}\frac{\partial}{\partial \xi} \end{aligned}$$

and the density transforms as

$$P(\omega,t) = \mathcal{P}(\xi,s)\left|\frac{\partial(\xi,s)}{\partial(\omega,t)}\right| = \sqrt{\frac{s}{\mu_0}}\mathcal{P}(\xi,s) \tag{C.2}$$

where $\mathcal{P}(\xi,s)$ denotes the density of ξ at time s and $\frac{\partial(\xi,s)}{\partial(\omega,t)}$ is the Jacobian.

With these coordinate changes, (C.1) becomes

$$\tau\left(\frac{\xi}{2s}-\sqrt{\frac{s}{\mu_0}}\,\dot{\phi}(s)\right)\frac{\partial}{\partial\xi}\left\{\sqrt{\frac{s}{\mu_0}}\mathcal{P}(\xi,s)\right\}+\tau\frac{\partial}{\partial s}\left\{\sqrt{\frac{s}{\mu_0}}\mathcal{P}(\xi,s)\right\}=$$
$$\sum_{i=1}^{\infty}\frac{(-\mu_0)^i}{s^i\,i!}\left(\sqrt{\frac{s}{\mu_0}}\right)^i\frac{\partial^i}{\partial\xi^i}\left\{\left\langle H^i\left(\phi+\sqrt{\frac{\mu_0}{s}}\xi,x\right)\right\rangle_x\sqrt{\frac{s}{\mu_0}}\mathcal{P}(\xi,s)\right\}. \quad \text{(C.3)}$$

Expanding $\langle H^i\rangle_x$ about ϕ

$$\left\langle H^i(\phi+\sqrt{\tfrac{\mu_0}{s}}\xi,x)\right\rangle_x=\sum_{k=0}^{\infty}\left(\sqrt{\frac{\mu_0}{s}}\xi\right)^k\frac{1}{k!}\left\langle\tfrac{\partial^k}{\partial\omega^k}H^i(\phi,x)\right\rangle_x=\sum_{k=0}^{\infty}\left(\sqrt{\frac{\mu_0}{s}}\xi\right)^k\frac{\alpha_i^{(k)}(\phi)}{k!} \quad \text{(C.4)}$$

where

$$\alpha_i^{(k)}(\phi)\equiv\left\langle\tfrac{\partial^k}{\partial\omega^k}H^i(\phi,x)\right\rangle_x \quad \text{(C.5)}$$

and putting into (C.3) gives after simplification

$$\frac{\partial\mathcal{P}(\xi,s)}{\partial s}+\frac{\mathcal{P}(\xi,s)}{2\,s}+\left(\frac{\xi}{2s}-\sqrt{\frac{s}{\mu_0}}\,\dot{\phi}(s)\right)\frac{\partial\mathcal{P}(\xi,s)}{\partial\xi}=$$
$$\frac{1}{\tau}\sum_{i=1}^{\infty}\sum_{k=0}^{\infty}\frac{(-1)^i}{k!\,i!}\left(\sqrt{\frac{\mu_0}{s}}\right)^{i+k}\alpha_i^{(k)}(\phi)\frac{\partial^i}{\partial\xi^i}\left(\xi^k\mathcal{P}(\xi,s)\right). \quad \text{(C.6)}$$

Making the change of index $m=i+k$ we finally arrive at

$$\frac{\partial\mathcal{P}(\xi,s)}{\partial s}+\frac{\mathcal{P}(\xi,s)}{2\,s}+\left(\frac{\xi}{2s}-\sqrt{\frac{s}{\mu_0}}\,\dot{\phi}(s)\right)\frac{\partial\mathcal{P}(\xi,s)}{\partial\xi}=$$
$$\frac{1}{\tau}\sum_{m=1}^{\infty}\sum_{i=1}^{m}\frac{(-1)^i}{(m-i)!\,i!}\left(\sqrt{\frac{\mu_0}{s}}\right)^m\alpha_i^{(m-i)}(\phi)\frac{\partial^i}{\partial\xi^i}\left(\xi^{m-i}\mathcal{P}(\xi,s)\right). \quad \text{(C.7)}$$

We now implicitly define ϕ so that the term with $\dot{\phi}$ on the left cancels the $m=1$ term on the right,

$$-\sqrt{\frac{s}{\mu_0}}\,\dot{\phi}(s)\frac{\partial\mathcal{P}(\xi,s)}{\partial\xi}=-\frac{1}{\tau}\sqrt{\frac{\mu_0}{s}}\alpha_1^{(0)}(\phi)\frac{\partial\mathcal{P}(\xi,s)}{\partial\xi}. \quad \text{(C.8)}$$

Equations (C.7) and (C.8) become (renaming $s\to t$)

$$\frac{d\phi}{dt}=\frac{\mu_0}{\tau}\frac{\alpha_1^{(0)}(\phi)}{t} \quad \text{(C.9)}$$

$$\frac{\partial\mathcal{P}(\xi,t)}{\partial t}=-\frac{\xi}{2\,t}\frac{\partial\mathcal{P}(\xi,t)}{\partial\xi}-\frac{\mathcal{P}(\xi,t)}{2\,t}+ \quad \text{(C.10)}$$
$$\frac{1}{\tau}\sum_{m=2}^{\infty}\left(\sqrt{\frac{\mu_0}{t}}\right)^m\sum_{i=1}^{m}\frac{(-1)^i}{i!\,(m-i)!}\,\alpha_i^{(m-i)}(\phi(t))\frac{\partial^i}{\partial\xi^i}\left(\xi^{m-i}\,\mathcal{P}(\xi,t)\right).$$

Keeping only the $m = 2$ term and setting $c = \frac{\mu_0}{\tau}$ leaves the system of equations

$$\begin{aligned}
\frac{d\phi}{dt} &= c\,\frac{\alpha_1^{(0)}(\phi)}{t} \\
\frac{\partial \mathcal{P}(\xi,t)}{\partial t} &= -\frac{\xi}{2\,t}\,\frac{\partial \mathcal{P}(\xi,t)}{\partial \xi} - \frac{\mathcal{P}(\xi,t)}{2\,t} + \\
&\qquad \frac{c}{t}\left(-\alpha_1^{(1)}(\phi)\frac{\partial}{\partial \xi}\xi\,\mathcal{P}(\xi,t) + \frac{1}{2}\alpha_2^{(0)}(\phi)\frac{\partial^2}{\partial \xi^2}\mathcal{P}(\xi,t)\right).
\end{aligned}$$

Note that the lowest order terms are independent of μ thus fulfilling the original *ansatz.*

From this, we can solve for the time evolution of $\langle\xi\rangle$ and $\langle\xi^2\rangle$

$$\frac{\partial\langle\xi\rangle}{\partial t} = \int \frac{\partial \mathcal{P}(\xi,t)}{\partial t}\,\xi\,d\xi = \frac{\langle\xi\rangle}{t}\left(\frac{1}{2} + c\,\alpha_1^{(1)}(\phi)\right) \tag{C.11}$$

$$\begin{aligned}
\frac{\partial\langle\xi^2\rangle}{\partial t} &= \int \frac{\partial \mathcal{P}(\xi,t)}{\partial t}\,\xi^2\,d\xi \\
&= \frac{1}{t}\left\{\left(1 + 2\,c\,\alpha_1^{(1)}(\phi)\right)\,\langle\xi^2\rangle + c\,\alpha_2^{(0)}\right\}.
\end{aligned} \tag{C.12}$$

Solving yields

$$\langle\xi\rangle = \langle\xi\rangle_0 e^{\frac{\gamma(t_0,t)}{2}} \tag{C.13}$$

$$\langle\xi^2\rangle = \langle\xi^2\rangle_0\, e^{\gamma(t_0,t)} + c\int_{t_0}^{t}\frac{dq}{q}\,e^{\gamma(q,t)}\,\alpha_2^{(0)}(\phi(q)) \tag{C.14}$$

where $\langle\xi\rangle_0$ and $\langle\xi^2\rangle_0$ are $\langle\xi\rangle$ and $\langle\xi^2\rangle$ evaluated at $t = t_0$, respectively, and

$$\gamma(t_1,t_2) \equiv \int_{t_1}^{t_2}\frac{dq}{q}\left[1 + 2\,c\,\alpha_1^{(1)}(\phi(q))\right].$$

Appendix D

Time Evolution of 1-D LMS: Comparison with Discrete and Continuous Time Solutions

Neural network learning takes place in discrete time. However, continuous time dynamics are more amenable to analysis and approximation. Thus, we have analyzed the time evolution of the weights by simplifying the exact discrete time equations using two stages of approximation. First, we assumed that for small learning rates and for late times we can transition to continuous time equations. We then approximate the continuous time equations by carefully expanding out in μ and truncating to some desired order. Both stages of approximations are necessary because the dynamical equations for nonlinear problems are otherwise intractable.

In this appendix we explore the effect of these approximations by examining a very simple system that is solvable exactly: 1-D linear networks. We solve both the discrete and continuous time equations for the first two moments of the weight error, $\langle v \rangle$ and $\langle v^2 \rangle$. We compare these results to each other and also to the moments obtained from the small noise expansion. We assume a constant learning rate throughout.

D.1 Discrete-Time Evolution of Weights

In this section we derive the exact discrete time expression for the time evolution of the expected weight error and squared weight error for LMS with a constant learning rate. As before, the update function for LMS is $H[v, x] = (\epsilon_t - v_t x_t)x_t$ so that the update

equations for the weight error and squared weight error become

$$v_{t+1} = v_t + \mu\,(\epsilon_t - v_t\,x_t)\,x_t \tag{D.1}$$

$$v_{t+1}^2 = v_t^2 + 2\mu\,(\epsilon - v_t\,x_t)\,x_t v_t + \mu^2\,(\epsilon - v_t\,x_t)^2\,x_t^2 \tag{D.2}$$

where ϵ is zero mean noise. Taking expectation over the inputs gives

$$\begin{aligned}\langle v_{t+1}\rangle &= (1-\mu\,R)\langle v_t\rangle \\ \langle v_{t+1}^2\rangle &= (1-2\mu\,R+\mu^2 S)\langle v_t^2\rangle + \mu^2\,\sigma_\epsilon^2\,R\end{aligned} \tag{D.3}$$

where $R = \langle x^2\rangle$, $S = \langle x^4\rangle$, and $\sigma_\epsilon^2 = \langle \epsilon^2\rangle$. We solve these difference equations to obtain

$$\begin{aligned}\langle v_t\rangle &= (1-\mu\,R)^t\langle v_0\rangle \\ \langle v_t^2\rangle &= (1-2\mu\,R+\mu^2 S)^t\langle v_0^2\rangle + \frac{1-(1-2\mu\,R+\mu^2 S)^t}{2R-\mu S}\,\mu\sigma_\epsilon^2 R\end{aligned} \tag{D.4}$$

where $\langle v_0\rangle$ and $\langle v_0^2\rangle$ are the first and second moments of v at $t=0$, respectively. Note that the equilibrium variance (at $t=\infty$) is the same as previously predicted, $\langle v_\infty^2\rangle = \frac{\mu R\sigma_\epsilon^2}{2R-\mu S}$.

Assuming that μ is small, we can use the identity

$$(1+a)^b = e^{b\ln(1+a)} = e^{b(a-\frac{a^2}{2}+\ldots)} \approx e^{ba} \quad (|a|<1) \tag{D.5}$$

to rewrite the equations in (D.4) to lowest order in μ

$$\begin{aligned}\langle v_t\rangle &\approx \langle v_0\rangle\,e^{-\mu R t} \\ \langle v_t^2\rangle &\approx \langle v_0^2\rangle\,e^{-2\mu R t} + \frac{\mu\sigma_\epsilon^2}{2}\left(1-e^{-2\mu R t}\right).\end{aligned} \tag{D.6}$$

We can also introduce a timescale τ where $t = n\tau$ and n is the iteration number. The above equations then become

$$\begin{aligned}\left\langle v_{n=t/\tau}\right\rangle &\approx \langle v_0\rangle\,e^{-\frac{\mu R t}{\tau}} \\ \left\langle v_{n=t/\tau}^2\right\rangle &\approx \langle v_0^2\rangle\,e^{-\frac{2\mu R t}{\tau}} + \frac{\mu\sigma_\epsilon^2}{2}\left(1-e^{-\frac{2\mu R t}{\tau}}\right).\end{aligned} \tag{D.7}$$

D.2 Continuous-Time Evolution of Weights

To transition to continuous time we assume the timescale τ defined as above is small relative to t (i.e. n is large) so we can expand $\langle v(n\tau+\tau)\rangle$ about $t = n\tau$

$$\langle v(t+\tau)\rangle = \langle v(t)\rangle + \tau\frac{d\langle v(t)\rangle}{dt} + \tau^2\frac{d^2\langle v(t)\rangle}{dt^2}\cdots. \tag{D.8}$$

To lowest order in τ we have

$$\tau \frac{d\langle v(t)\rangle}{dt} \approx \langle v(t+\tau)\rangle - \langle v(t)\rangle = \mu\langle H[v,x]\rangle \tag{D.9}$$

or

$$\frac{d\langle v\rangle}{dt} \approx -\frac{\mu R}{\tau}\langle v\rangle. \tag{D.10}$$

Similarly, for $\langle v^2\rangle$ we find

$$\frac{d\langle v^2(t)\rangle}{dt} \approx \frac{\mu}{\tau}(-2R+\mu S)\langle v^2\rangle + \frac{\mu^2 R\sigma_\epsilon^2}{\tau}. \tag{D.11}$$

Solving yields

$$\begin{aligned}
\langle v(t)\rangle &= \langle v_0\rangle\, e^{-\frac{\mu R t}{\tau}} \\
\langle v^2(t)\rangle &= \langle v_0^2\rangle\, e^{(-2R+\mu S)\frac{\mu t}{\tau}} + \frac{\mu R\sigma_\epsilon^2}{2R-\mu S}\left(1 - e^{(-2R+\mu S)\frac{\mu t}{\tau}}\right) \\
&= \frac{\mu R\sigma_\epsilon^2}{2R-\mu S} + \left(\langle v_0^2\rangle - \frac{\mu R\sigma_\epsilon^2}{2R-\mu S}\right) e^{(-2R+\mu S)\frac{\mu t}{\tau}}.
\end{aligned} \tag{D.12}$$

The equation for $\langle v^2\rangle$ correctly predicts the equilibrium variance ($t = \infty$). Otherwise, these equations do not look quite like either (D.4) or (D.7). However, we recall that the above continuous time equations are only valid for small τ. If we let $\tau \to 0$ then $\langle v(t)\rangle$ goes to zero for all t (assuming that $-2R + \mu S < 0$) and $\langle v^2(t)\rangle$ becomes a constant. However, if we assume that the timescale τ scales with μ so that $\frac{\mu}{\tau}$ is constant[1], we obtain in the limit as $\tau, \mu \to 0$

$$\begin{aligned}
\langle v(t)\rangle &= \langle v_0\rangle\, e^{-\frac{\mu R t}{\tau}} \\
\langle v^2(t)\rangle &= \langle v_0^2\rangle e^{-\frac{2\mu R t}{\tau}} + \frac{\mu\sigma_\epsilon^2}{2}\left(1 - e^{-\frac{2\mu R t}{\tau}}\right).
\end{aligned} \tag{D.13}$$

These are identical to the lowest order discrete equations in (D.7). Since we have 1) truncated at $\mathcal{O}(\tau)$ in (D.8) and 2) set $\tau \propto \mu$, then we must assume that the continuous time equations are only valid to $\mathcal{O}(\mu)$.

[1]Letting $\tau \to 0$ is equivalent to letting $n \to \infty$ thus we expect $\langle v\rangle$ and $\langle v^2\rangle$ to approach their equilibrium values in this limit. However, if we simultaneously let $\mu \to 0$ so that $\mu\tau$ is constant then we are in a sense slowing down the learning process at the same rate that we are speeding up time.

D.3 Small Noise Expansion applied to LMS with a Constant Learning Rate

We now use the small noise expansion discussed in Section §2.2.3 to approximate the continuous time equations of the previous section. The weight error at time t is described by $v(t) = \phi(t) + \sqrt{\mu}\xi$ where ϕ represents the deterministic motion and ξ represents the fluctuations about the deterministic path. The lowest order[2] ($m = 2$) time evolution of ϕ and ξ were found from equations (2.17), (2.25), and (2.26) to be (with $t_0 = 0$)

$$\frac{d\phi(t)}{dt} = \frac{\mu}{\tau}\alpha_1^{(0)}(\phi(t)) \tag{D.14}$$

$$\frac{d\langle\xi\rangle}{dt} = \frac{\mu}{\tau}\alpha_1^{(1)}\langle\xi\rangle \quad \Rightarrow \quad \langle\xi(t)\rangle = \langle\xi(0)\rangle e^{\gamma(0,t)/2} \tag{D.15}$$

$$\frac{d\langle\xi^2\rangle}{dt} = 2\frac{\mu}{\tau}\alpha_1^{(1)}\langle\xi^2\rangle + \frac{\mu}{\tau}\alpha_2^{(0)}$$

$$\Rightarrow \quad \langle\xi^2(t)\rangle = \langle\xi^2(0)\rangle e^{\gamma(0,t)} + \frac{\mu}{\tau}\int_0^t ds\, e^{\gamma(s,t)}\alpha_2^{(0)}(\phi(s)) \tag{D.16}$$

where

$$\gamma(t_1, t_2) \equiv \frac{2\mu}{\tau}\int_{t_1}^{t_2}\alpha_1^{(1)}(\phi(s))\, ds.$$

Inserting the coefficients for 1-D LMS

$$\alpha_1^{(0)} = -R\phi, \quad \alpha_1^{(1)} = -R. \quad \alpha_2^{(0)} = R\sigma_\epsilon^2 + S\phi^2 \tag{D.17}$$

we can compare the equations for the time evolution of $\langle v\rangle$ and $\langle v^2\rangle$ in (D.10) and (D.11) with those derived here using the small noise expansion

$$\frac{d\langle v\rangle}{dt} = \frac{d\phi}{dt} + \sqrt{\mu}\frac{d\langle\xi\rangle}{dt} = \frac{\mu}{\tau}\left(\alpha_1^{(0)} + \sqrt{\mu}\alpha_1^{(1)}\langle\xi\rangle\right) = -\frac{\mu}{\tau}R\left(\phi + \sqrt{\mu}\langle\xi\rangle\right)$$

$$= -\frac{\mu}{\tau}R\langle v\rangle \tag{D.18}$$

$$\frac{d\langle v^2\rangle}{dt} = \frac{d\phi^2}{dt} + \mu\frac{d\langle\xi^2\rangle}{dt} = 2\phi\frac{\mu}{\tau}\alpha_1^{(0)} + \frac{\mu^2}{\tau}\left(2\alpha_1^{(1)}\langle\xi^2\rangle + \alpha_2^{(0)}\right)$$

$$= \frac{\mu}{\tau}\left(-2R\langle v^2\rangle + \mu S\phi^2\right) + \frac{\mu^2 R\sigma_\epsilon^2}{\tau}. \tag{D.19}$$

Comparing these with (D.10) and (D.11) we see that the equation for $\langle v\rangle$ is identical. If the equation for $\langle v^2\rangle$ in (D.19) had an additional $\frac{\mu^3}{\tau}S\langle\xi^2\rangle$ then the ϕ^2 would be replaced

[2] If all orders are retained, the solution should be identical to solving the entire Kramers-Moyal Expansion which, in turn, should be identical to the solution obtained of the full continuous time equation given in the previous section.

with a $\langle v^2 \rangle$ and the equation would match (D.11) precisely. However, only the lowest order ($m = 2$) equations for the noise ξ have been kept so we would not expect this term to be present. The term $\frac{\mu^3}{\tau} S \langle \xi^2 \rangle$ does not appear until $m = 4$.

Solving equations (D.14)-(D.16) we obtain

$$\phi(t) = \phi(0)\, e^{-\frac{\mu R t}{\tau}} \tag{D.20}$$

$$\langle \xi(t) \rangle = \langle \xi(0) \rangle\, e^{-\frac{\mu R t}{\tau}} \tag{D.21}$$

$$\begin{aligned} \langle \xi^2(t) \rangle &= \langle \xi^2(0) \rangle\, e^{-\frac{2\mu R t}{\tau}} + \frac{\mu}{\tau} \int_0^t ds\, e^{-\frac{2\mu R(t-s)}{\tau}} \left(R\sigma_\epsilon^2 + S\,\phi^2(0)\, e^{-\frac{2\mu R s}{\tau}} \right) \\ &= \langle \xi^2(0) \rangle\, e^{-\frac{2\mu R t}{\tau}} + \frac{\sigma_\epsilon^2}{2}(1 - e^{-\frac{2\mu R t}{\tau}}) + \frac{\mu}{\tau} S\,\phi^2(0)\, t\, e^{-\frac{2\mu R t}{\tau}}. \end{aligned} \tag{D.22}$$

With $\langle \xi(0) \rangle = 0$, $\langle v_0 \rangle = \phi(0)$ and $\langle v_0^2 \rangle = \phi^2(0) + \mu \langle \xi^2(0) \rangle$, the equations for the weight error become

$$\begin{aligned} \langle v(t) \rangle &= \phi(t) + \sqrt{\mu}\, \langle \xi(t) \rangle = \phi(0)\, e^{-\frac{\mu R t}{\tau}} + \sqrt{\mu}\, \langle \xi(0) \rangle\, e^{-\frac{\mu R t}{\tau}} \\ &= \langle v_0 \rangle\, e^{-\frac{\mu R t}{\tau}} \end{aligned} \tag{D.23}$$

$$\begin{aligned} \langle v^2(t) \rangle &= \phi^2(t) + \mu\, \langle \xi^2(t) \rangle \\ &= \phi^2(0)\, e^{-\frac{2\mu R t}{\tau}} + \mu \langle \xi^2(0) \rangle\, e^{-\frac{2\mu R t}{\tau}} + \frac{\mu \sigma_\epsilon^2}{2}(1 - e^{-\frac{2\mu R t}{\tau}}) + \frac{\mu^2 S\, \phi^2(0)\, t}{\tau}\, e^{-\frac{2\mu R t}{\tau}} \\ &= \left(\langle v_0^2 \rangle + \phi^2(0)\, \frac{\mu^2 S\, t}{\tau} \right) e^{-\frac{2\mu R t}{\tau}} + \frac{\mu \sigma_\epsilon^2}{2} \left(1 - e^{-\frac{2\mu R t}{\tau}} \right). \end{aligned} \tag{D.24}$$

As expected, the mean is identical to the exact continuous time solution. Equation (D.24) can be obtained from the full continuous time solution in (D.12) by expanding and truncating to $\mathcal{O}(\mu)$ as follows. Let $\frac{\mu R \sigma_\epsilon^2}{2R - \mu S} \approx \frac{\mu \sigma_\epsilon^2}{2}$ and $e^{\frac{\mu^2 S t}{\tau}} \approx 1 + \frac{\mu^2 S t}{\tau}$. Note that the last approximation is quite poor as t becomes large, however, the effect is muted by the $e^{-\frac{2\mu R t}{\tau}}$ multiplicative factor. Inserting these approximations into (D.12) and truncating[3] to $\mathcal{O}(\mu)$ (remembering that $\frac{\mu}{\tau}$ is $\mathcal{O}(1)$) then gives (D.24).

D.4 Comparison to Simulations

Figure D.1 compares the prediction of $\langle v^2 \rangle$ using the discrete time equation (D.4), the full continuous time equation (D.12), and the small noise expansion (D.24). A large learning rate ($\mu = .2$) has been chosen so as to highlight the differences. If μ is very

[3] To obtain (D.13) we let μ and τ actually go to zero.

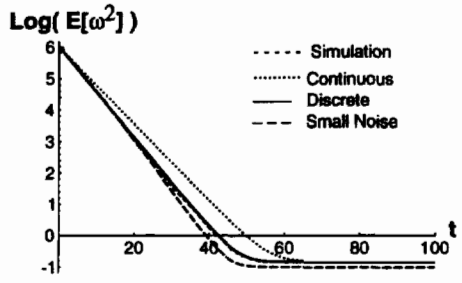


Figure D.1: Predictions of the squared weight error for 1-D LMS with gaussian inputs with $R = 1$, $\sigma_\epsilon^2 = 1$, $\langle v_0 \rangle = 1000$, and $\mu = .2$. Simulations used 10000 networks. Theoretical curves were computed using $\langle v_0^2 \rangle = 10^6$, $\phi^2(0) = 10^6$, $\langle \xi^2(0) \rangle = 0$, and $S = 3R^2$.

small (e.g. .002), then all of the curves would be indistinguishable to the eye. For $\mu = .2$, the discrete time curve lies right on top of the simulations, as it should. All of the curves except the small noise expansion accurately predict the equilibrium variance. What is somewhat surprising is that the small noise expansion predicts the pre-equilibrium slope more accurately than the full continuous time equation. Perhaps this should be expected because the continuous time solution can only be guaranteed to be accurate to $\mathcal{O}(\mu)$ even though it contains terms that are $\mathcal{O}(\mu^2)$. For 1-D LMS, these extra terms help when it comes to predicting the equilibrium variance, however, they hurt when it comes to predicting the pre-equilibrium slope. In contrast, the lowest order small noise expansion keeps *only* those terms that are $\mathcal{O}(\mu)$ while throwing away everything else.

These results would seem to imply that the higher order terms for the equilibrium density calculated in the previous chapter using a perturbative expansion are not valid. However, we note that the full continuous time equations predict the correct equilibrium variance. We believe that this is not accidental. To study equilibrium behavior, we can let $\tau \to 0$ while keeping μ fixed. The scaling was necessary so that the pre-equilibrium behavior would still be visible.

D.5 Summary

In this chapter, we have examined the very simple network of 1-D LMS to better understand the approximations being made in first transitioning to continuous time and then approximating the continuous time solutions using the small noise expansion. We chose

a 1-D LMS network because its discrete and continuous time equations are easily solved exactly thus making precise comparisons possible. We discover that the continuous time equations can only be trusted up to $\mathcal{O}(\mu)$ if we assume that the timescale τ scales with μ. In such a case, it seems preferable to use the lowest order small noise expansion rather than the full continuous time solution because the small noise expansion only keeps $\mathcal{O}(\mu)$ terms.

Appendix E

Small Noise Expansion for Constant Learning with Momentum

The algorithm for stochastic learning with a constant learning rate and constant momentum in 1-dimension can be written as

$$v(t+1) = v(t) + \mu H[v(t), x(t)] + \beta(v(t) - \Omega(t)) \quad \text{(E.1)}$$

$$\Omega(t+1) = \omega(t) \quad \text{(E.2)}$$

$$= \Omega(t) + (\omega(t) - \Omega(t)). \quad \text{(E.3)}$$

In vector form it can be written as

$$\begin{pmatrix} v(t+1) \\ \Omega(t+1) \end{pmatrix} = \begin{pmatrix} v(t) \\ \Omega(t) \end{pmatrix} + \begin{pmatrix} \mu H[v(t), x(t)] + \beta(v(t) - \Omega(t)) \\ v(t) - \Omega(t) \end{pmatrix} \equiv \begin{pmatrix} v(t) \\ \Omega(t) \end{pmatrix} + \begin{pmatrix} G_1 \\ G_2 \end{pmatrix} \quad \text{(E.4)}$$

The Kramers-Moyal equation for this is then

$$\tau \ \partial_t P(v, \Omega, t) = \sum_{i=1}^{\infty} \frac{(-1)^i}{i!} \sum_{j=0}^{i} \binom{i}{j} \frac{\partial^i}{\partial v^j \partial \Omega^{i-j}} \left(\left\langle G_1^j G_2^{i-j} \right\rangle_x P(v, \Omega, t) \right). \quad \text{(E.5)}$$

The small noise expansion requires making a change of variable from $\{v, \Omega, t\}$ to $\{\xi_\omega, \xi_\Omega, s\}$ where

$$\xi_\omega = (v - \phi_\omega(t)) \frac{1}{\sqrt{\mu}}$$

$$\xi_\Omega = (\Omega - \phi_\Omega(t)) \frac{1}{\sqrt{\mu}}$$

$$s = t.$$

The partial derivatives transform as

$$\begin{aligned}
\frac{\partial}{\partial t} &= -\frac{\dot{\phi_\omega}}{\sqrt{\mu}}\frac{\partial}{\partial \xi_\omega} - \frac{\dot{\phi_\Omega}}{\sqrt{\mu}}\frac{\partial}{\partial \xi_\Omega} + \frac{\partial}{\partial s} \\
\frac{\partial}{\partial v} &= \frac{1}{\sqrt{\mu}}\frac{\partial}{\partial \xi_\omega} \\
\frac{\partial}{\partial \Omega} &= \frac{1}{\sqrt{\mu}}\frac{\partial}{\partial \xi_\Omega}
\end{aligned} \tag{E.6}$$

and the density transforms as

$$P(v,\Omega,t) = \mathcal{P}(\xi_\omega,\xi_\Omega,s)\left|\frac{\partial(\xi_\omega,\xi_\Omega,s)}{\partial(v,\Omega,t)}\right| = \frac{1}{\mu}\mathcal{P}(\xi_\omega,\xi_\Omega,s) \tag{E.7}$$

where $\frac{\partial(\xi_\omega,\xi_\Omega,s)}{\partial(v,\Omega,t)}$ is the Jacobian.

We can also rewrite $\left\langle G_1^j G_2^{i-j}\right\rangle_x$ as

$$\begin{aligned}
\left\langle G_1^j G_2^{i-j}\right\rangle_x &= \left\langle (\mu H + \beta(v-\Omega))^j\ (v-\Omega)^{i-j}\right\rangle_x \\
&= \sum_{l=0}^{j}\binom{j}{l}\mu^l\ \left\langle H^l\right\rangle_x \beta^{j-l}\ (v-\Omega)^{i-l}.
\end{aligned} \tag{E.8}$$

Expanding $H^l[v=\phi_\omega+\sqrt{\mu}\xi_\omega,x]$ about $v=\phi_\omega$ gives

$$\begin{aligned}
\left\langle H^l[v,x]\right\rangle_x &= \sum_{k=0}^{\infty}\frac{\mu^{k/2}}{k!}\,{\xi_\omega}^k\,\frac{\partial^k}{\partial v^k}\left\langle H^l[v,x]\right\rangle_x\Big|_{v=\phi_\omega} \\
&= \sum_{k=0}^{\infty}\frac{\mu^{k/2}}{k!}\,{\xi_\omega}^k\alpha_l^{(k)}(\phi_\omega)
\end{aligned} \tag{E.9}$$

where

$$\alpha_l^{(k)}(\phi_\omega) \equiv \frac{\partial^k}{\partial v^k}\left\langle H^l[v,x]\right\rangle_x\Big|_{v=\phi_\omega}. \tag{E.10}$$

We can also rewrite

$$(v-\Omega)^{i-l} = (\Delta\phi+\sqrt{\mu}\Delta\xi)^{i-l} = \sum_{p=0}^{i-l}\binom{i-l}{p}\mu^{\frac{p}{2}}(\Delta\xi)^p(\Delta\phi)^{i-l-p} \tag{E.11}$$

where $\Delta\xi \equiv \xi_\omega - \xi_\Omega$ and $\Delta\phi \equiv \phi_\omega - \phi_\Omega$. We also note that $\Delta\phi \propto \mu$ so we define

$$\widetilde{\Delta\phi} \equiv \frac{1}{\mu}\Delta\phi \tag{E.12}$$

Now putting (E.9), (E.11), and (E.12) into (E.8) gives

$$\left\langle G_1^j G_2^{i-j} \right\rangle_x = \sum_{l=0}^{j} \sum_{k=0}^{\infty} \sum_{p=0}^{i-l} \left(\begin{array}{c} j \\ l \end{array} \right) \left(\begin{array}{c} i-l \\ p \end{array} \right) \frac{\mu^{i+\frac{k-p}{2}}}{k!} \beta^{j-l} \alpha_l^{(k)}(\phi_\omega) \, (\widetilde{\Delta\phi})^{i-l-p} \, {\xi_\omega}^k (\Delta\xi)^p. \tag{E.13}$$

Using (E.13) and the transformations in (E.6) and (E.7), (E.5) becomes

$$\begin{aligned} &\tau \left(-\frac{\dot{\phi_\omega}}{\sqrt{\mu}} \frac{\partial}{\partial \xi_\omega} - \frac{\dot{\phi_\Omega}}{\sqrt{\mu}} \frac{\partial}{\partial \xi_\Omega} + \frac{\partial}{\partial s} \right) \mathcal{P}(\xi_\omega, \xi_\Omega, s) = \\ &\sum_{i=1}^{\infty} \sum_{j=0}^{i} \sum_{l=0}^{j} \sum_{k=0}^{\infty} \sum_{p=0}^{i-l} \mu^{\frac{i+k-p}{2}} \frac{(-1)^i}{i!\,k!} \left(\begin{array}{c} i \\ j \end{array} \right) \left(\begin{array}{c} j \\ l \end{array} \right) \left(\begin{array}{c} i-l \\ p \end{array} \right) \beta^{j-l} \, \alpha_l^{(k)}(\phi_\omega) \, (\widetilde{\Delta\phi})^{i-l-p} \\ &\frac{\partial^i}{\partial {\xi_\omega}^j \partial {\xi_\Omega}^{i-j}} \left\{ {\xi_\omega}^k (\Delta\xi)^p \, \mathcal{P}(\xi_\omega, \xi_\Omega, s) \right\} \end{aligned} \tag{E.14}$$

Making the change of index $n = i + k - p$ and rearranging the order of summation we finally arrive at

$$\begin{aligned} &\tau \left(-\frac{\dot{\phi_\omega}}{\sqrt{\mu}} \frac{\partial}{\partial \xi_\omega} - \frac{\dot{\phi_\Omega}}{\sqrt{\mu}} \frac{\partial}{\partial \xi_\Omega} + \frac{\partial}{\partial s} \right) \mathcal{P}(\xi_\omega, \xi_\Omega, s) = \\ &\sum_{n=0}^{\infty} \mu^{\frac{n}{2}} \sum_{i=1}^{\infty} \sum_{j=0}^{i} \sum_{l=0}^{\min(j,n)} \sum_{p=\max(0,i-n)}^{i-l} \frac{(-1)^i}{i!\,(n+p-i)!} \left(\begin{array}{c} i \\ j \end{array} \right) \left(\begin{array}{c} j \\ l \end{array} \right) \left(\begin{array}{c} i-l \\ p \end{array} \right) \beta^{j-l} \\ &\alpha_l^{(n+p-i)}(\phi_\omega) \, (\widetilde{\Delta\phi})^{i-l-p} \frac{\partial^i}{\partial {\xi_\omega}^j \partial {\xi_\Omega}^{i-j}} \left\{ {\xi_\omega}^{n+p-i} (\Delta\xi)^p \, \mathcal{P}(\xi_\omega, \xi_\Omega, s) \right\} \end{aligned} \tag{E.15}$$

The $n = 0$ term on the right is

$$\mathrm{RHS}_{n=0} = \sum_{i=1}^{\infty} \sum_{j=0}^{i} \frac{(-1)^i}{i!} \left(\begin{array}{c} i \\ j \end{array} \right) \beta^j \frac{\partial^i}{\partial {\xi_\omega}^j \partial {\xi_\Omega}^{i-j}} \left\{ (\Delta\xi)^i \, \mathcal{P} \right\} \tag{E.16}$$

and the $n = 1$ term is

$$\begin{aligned} \mathrm{RHS}_{n=1} = \sqrt{\mu} &\sum_{i=1}^{\infty} \sum_{j=0}^{i} \sum_{l=0}^{\min(j,1)} \sum_{p=\max(0,i-1)}^{i-l} \frac{(-1)^i}{i!\,(1+p-i)!} \left(\begin{array}{c} i \\ j \end{array} \right) \left(\begin{array}{c} j \\ l \end{array} \right) \left(\begin{array}{c} i-l \\ p \end{array} \right) \beta^{j-l} \\ &\alpha_l^{(1+p-i)} \, (\widetilde{\Delta\phi})^{i-l-p} \frac{\partial^i}{\partial {\xi_\omega}^j \partial {\xi_\Omega}^{i-j}} \left\{ {\xi_\omega}^{1+p-i} (\Delta\xi)^p \, \mathcal{P} \right\} \end{aligned} \tag{E.17}$$

It makes most intuitive sense to cancel the $\dot{\phi_\omega}$ and $\dot{\phi_\Omega}$ terms on the left side of (E.15) with the $n = 1$, $i = 1$, $l = 0$, and $p = 0$ terms on the right because the resulting deterministic equations resemble the continuous time versions of (E.1) and (E.3)

$$\tau \dot{\phi_\omega} = \mu \langle H[\phi_\omega, x] \rangle + \beta \, (\phi_\omega - \phi_\Omega) \tag{E.18}$$

$$\tau \dot{\phi_\Omega} = \phi_\omega - \phi_\Omega \tag{E.19}$$

Taking the lowest order term on the right ($n = 0$) leaves the equations for the noise

$$\tau \frac{\partial \mathcal{P}(\xi_\omega, \xi_\Omega, s)}{\partial s} = \sum_{i=1}^{\infty} \sum_{j=0}^{i} \frac{(-1)^i}{i!} \binom{i}{j} \beta^j \frac{\partial^i}{\partial \xi_\omega{}^j \partial \xi_\Omega{}^{i-j}} \left\{ (\Delta\xi)^i \mathcal{P} \right\}. \quad \text{(E.20)}$$

E.1 Deterministic Solution for LMS

For LMS, we have

$$\alpha_1^{(0)} = \langle H \rangle_x|_{v=\phi_\omega} = -R\phi_\omega \quad \text{(E.21)}$$

so that the deterministic equations (E.18) and (E.19) can then be written as

$$\tau \frac{d\phi(t)}{dt} = A\phi(t) \quad \text{(E.22)}$$

where

$$\phi = \begin{pmatrix} \phi_\omega \\ \phi_\Omega \end{pmatrix} \quad \text{(E.23)}$$

$$A = \begin{pmatrix} \beta - \mu R & -\beta \\ 1 & -1 \end{pmatrix}. \quad \text{(E.24)}$$

The solution can formally be written as

$$\phi(t) = \phi(0) e^{At/\tau}.$$

To extract the actual components of ϕ we write

$$A = PDP^{-1}$$
$$\phi(t) = \phi(0) P e^{Dt/\tau} P^{-1}$$

where

$$D = \begin{pmatrix} -a - h & 0 \\ 0 & -a + h \end{pmatrix}$$
$$P = \begin{pmatrix} 1 - a - h & 1 - a + h \\ 1 & 1 \end{pmatrix}$$
$$P^{-1} = \frac{1}{2h} \begin{pmatrix} -1 & 1 - a + h \\ 1 & -1 + a + h \end{pmatrix}$$
$$a = \frac{1}{2}(1 - \beta + \mu R)$$
$$h = \sqrt{a^2 - \mu R}.$$

For the initial condition $\phi(0) = c \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ we have

$$\phi(t) = \begin{pmatrix} \phi_\omega(t) \\ \phi_\Omega(t) \end{pmatrix} = \frac{c}{2h} \begin{pmatrix} (-a+h+\mu R)\, e^{-(a+h)t/\tau} + (a+h-\mu R)\, e^{-(a-h)t/\tau} \\ (-a+h)\, e^{-(a+h)t/\tau} + (a+h)\, e^{-(a-h)t/\tau} \end{pmatrix}$$

Note that $\phi_\omega - \phi_\Omega$ is $\mathcal{O}(\mu)$ as originally assumed,

$$\phi_\omega - \phi_\Omega = \mu \frac{Rce^{-at/\tau}}{h} \left(e^{-ht/\tau} - e^{ht/\tau}\right) = -\frac{\mu Rce^{-at/\tau}}{2h} \sinh\left(\frac{ht}{\tau}\right).$$

Appendix F

Evaluation of the Squared Weight Error for μ_0/t Learning Rate Schedules and Constant Momentum

In this appendix we evaluate the squared weight error $E[|v|^2]$ for annealed learning with momentum. $E[|v|^2]$ is the trace of the upper left quadrant of weight error correlation matrix, $\tilde{C}$. We have from equations (6.5) and (6.11) in chapter (6) that

$$\tilde{C} \equiv E[\, Z\, Z^T\,] = \int d^N Z\; Z\, Z^T\; P(Z,t) \tag{F.1}$$

$$= U(t,t_0)\;\tilde{C}\;(t_0)\;U^T(t,t_0) + \int_{t_0}^{t} d\tau\;\mu(\tau)^2\,U(t,\tau)\;\tilde{D}\;U^T(t,\tau) \tag{F.2}$$

where

$$U(t_2,t_1) \equiv \exp\left[\int_{t_1}^{t_2} d\tau\; K(\tau)\right] \tag{F.3}$$

$$K \equiv \begin{pmatrix} -\mu(t)\,R & \beta I \\ -\mu(t)R & (\beta-1)I \end{pmatrix} \tag{F.4}$$

$$R \equiv \left\langle \nabla_v^2 \mathcal{E}\,\Big|_{v=0} \right\rangle_x \tag{F.5}$$

$$Z \equiv \{v,\Omega\}. \tag{F.6}$$

We assume a learning rate schedule of the form $\mu(t) = \mu_0/t$. We also assume, without loss of generality, that the coordinates are chosen so that R is diagonal with eigenvalues $\lambda_i \;\; i = 1 \ldots N$.

F.1 Computing the Evolution Operator U

To compute U, we first diagonalize the exponent of U

$$\int_{t_1}^{t_2} d\tau\ K(\tau) = \begin{pmatrix} -\mu_0 R \log(t_2/t_1) & \beta(t_2-t_1)I \\ -\mu_0 R \log(t_2/t_1) & (\beta-1)(t_2-t_1)I \end{pmatrix} = Q(t_2,t_1)\ \Lambda(t_2,t_1)\ Q^{-1}(t_2,t_1)$$

where Λ is a $2N \times 2N$ diagonal matrix. Letting $diag[\ a_i\ ]$ denote an $N \times N$ diagonal matrix with diagonal elements $a_1, a_2, \ldots a_N$, we can write

$$\Lambda(t_2,t_1) = \begin{pmatrix} diag[\ \eta_{i+}\ ] & 0 \\ 0 & diag[\ \eta_{i-}\ ] \end{pmatrix} \tag{F.7}$$

(F.8)

$$Q(t_2,t_1) = \begin{pmatrix} diag[\ -g_{i-}/(g_{i+}-g_{i-})\ ] & diag[\ g_{i+}/(g_{i+}-g_{i-})\ ] \\ diag[\ 1/(g_{i+}-g_{i-})\ ] & diag[\ -1/(g_{i+}-g_{i-})\ ] \end{pmatrix} \tag{F.9}$$

$$Q^{-1}(t_2,t_1) = \begin{pmatrix} I & diag[\ g_{i+}\ ] \\ I & diag[\ g_{i-}\ ] \end{pmatrix} \tag{F.10}$$

where

$$\begin{aligned} \eta_{i\pm} &= \frac{1}{2}(-(1-\beta)(t_2-t_1) - \mu_0\lambda_i \log(t_2/t_1) \pm \sqrt{h_i}) \\ g_{i\pm} &= ((1-\beta)(t_2-t_1) - \mu_0\lambda_i \log(t_2/t_1) \mp \sqrt{h_i})/(2\mu_0\lambda_i \log(t_2/t_1)) \\ h_i &= (1-\beta)^2(t_2-t_1)^2 - 2(\beta+1)(t_2-t_1)\mu_0\lambda_i \log(t_2/t_1) + \mu_0^2\lambda_i^2 \log^2(t_2/t_1)\ . \end{aligned}$$

The evolution operator U is then

$$\begin{aligned} U(t_2,t_1) &= \exp\left[\int_{t_1}^{t_2} d\tau\ K(\tau)\right] = Qe^{\Lambda}Q^{-1} \\ &= \begin{pmatrix} diag\left[\ \frac{-(g_{i-})\exp(\eta_{i+})+(g_{i+})\exp(\eta_{i-})}{(g_{i+})-(g_{i-})}\ \right] & diag\left[\ \frac{(g_{i+})(g_{i-})(\exp(\eta_{i-})-\exp(\eta_{i+}))}{(g_{i+})-(g_{i-})}\ \right] \\ diag\left[\ \frac{\exp(\eta_{i+})-\exp(\eta_{i-})}{(g_{i+})-(g_{i-})}\ \right] & diag\left[\ \frac{(g_{i+})\exp(\eta_{i+})-(g_{i-})\exp(\eta_{i-})}{(g_{i+})-(g_{i-})}\ \right] \end{pmatrix}. \end{aligned}$$

F.2 Homogeneous Solution

From (F.2), we see that $\widetilde{C}$ (and thus C) is composed of two terms. The first term we refer to as the homogeneous solution and the second term is the particular solution. In this section we compute the diagonal components of C_H where C_H is the homogeneous solution of C,

$$C_H = \text{Top left Quadrant of } \left\{U(t,t_0)\ \widetilde{C}\ (t_0)\ U^T(t,t_0)\right\}. \tag{F.11}$$

Multiplying out $U(t,t_0)\widetilde{C}\ (t_0)U^T(t,t_0)$, we obtain

$$\begin{aligned}(C_H)_{ii} &= \frac{1}{(g_{i+}-g_{i-})^2}\Big\{(-g_{i-}e^{\eta_{i+}}+g_{i+}e^{\eta_{i-}})^2\widetilde{C}\ (t_0)_{i,i}+\\ &(-g_{i-}e^{\eta_{i+}}+g_{i+}e^{\eta_{i-}})(g_{i+}g_{i-})(e^{\eta_{i-}}-e^{\eta_{i+}})\left(\widetilde{C}\ (t_0)_{i,i+N}+\widetilde{C}\ (t_0)_{i+N,i}\right)+\\ &(g_{i+}g_{i-})^2(e^{\eta_{i-}}-e^{\eta_{i+}})^2\widetilde{C}\ (t_0)_{i+N,i+N}\Big\}\end{aligned}$$

To simplify this, note that

$$g_{i+}-g_{i-}=-\frac{\sqrt{h_i}}{\mu_0\lambda_i\log(t/t_0)},\qquad g_{i+}g_{i-}=\frac{\beta(t-t_0)}{\mu_0\lambda_i\log(t/t_0)}\ ,$$

and

$$e^{2\eta_{i\pm}}=e^{-(1-\beta)(t-t_0)}\ e^{\pm\sqrt{h_i}}\ \left(\frac{t}{t_0}\right)^{-\mu_0\lambda_i}\ .$$

So that

$$\begin{aligned}(C_H)_{ii} &= \frac{(\mu_0\lambda_i\log(t/t_0))^2}{h_i}\ e^{-(1-\beta)(t-t_0)}\ \left(\frac{t}{t_0}\right)^{-\mu\lambda_i}\times\\ &\Big\{(-g_{i-}e^{\sqrt{h_i}/2}+g_{i+}e^{-\sqrt{h_i}/2})^2\ \widetilde{C}\ (t_0)_{i,i}+\frac{\beta(t-t_0)}{\mu_0\lambda_i\log(t/t_0)}\times\\ &(g_{i-}e^{\sqrt{h_i}}-(g_{i+}+g_{i-})+g_{i+}e^{-\sqrt{h_i}})(\widetilde{C}\ (t_0)_{i,i+N}+\widetilde{C}\ (t_0)_{i+N,i})+\\ &\left(\frac{\beta(t-t_0)}{\mu_0\lambda_i\log(t/t_0)}\right)^2(e^{-\sqrt{h_i}/2}-e^{\sqrt{h_i}/2})^2\widetilde{C}\ (t_0)_{i+N,i+N}\Big\}.\end{aligned}\tag{F.12}$$

Keeping terms in $e^{\sqrt{h_i}}$ we get

$$\begin{aligned}(C_H)_{ii} &\approx \frac{(\mu_0\lambda_i\log(t/t_0))^2}{h_i}\ e^{-(1-\beta)(t-t_0)+\sqrt{h_i}}\ \left(\frac{t}{t_0}\right)^{-\mu\lambda_i}\times\\ &\Big\{g_{i-}^2\ \widetilde{C}\ (t_0)_{i,i}+\frac{\beta(t-t_0)g_{i-}}{\mu_0\lambda_i\log(t/t_0)}(\widetilde{C}\ (t_0)_{i,i+N}+\widetilde{C}\ (t_0)_{i+N,i})+\\ &\left(\frac{\beta(t-t_0)}{\mu_0\lambda_i\log(t/t_0)}\right)^2\widetilde{C}\ (t_0)_{i+N,i+N}\Big\}.\end{aligned}$$

We can expand $\sqrt{h_i}$

$$\sqrt{h_i}=(1-\beta)(t-t_0)-\frac{1+\beta}{1-\beta}\mu_0\lambda_i\log(t/t_0)+\cdots$$

so that for t_0 and t large we have

$$\begin{aligned}\sqrt{h_i} &\approx (1-\beta)(t-t_0)\\ -(1-\beta)(t-t_0)+\sqrt{h_i} &\approx -\frac{1+\beta}{1-\beta}\ \mu_0\lambda_i\log(t/t_0)\\ g_{i-} &\approx \frac{(1-\beta)(t-t_0)}{\mu_0\lambda_i\log(t/t_0)}\ .\end{aligned}\tag{F.13}$$

$(C_H)_{ii}$ can now be simplified using the above approximations:

$$(C_H)_{ii} \approx \left(\frac{t}{t_0}\right)^{\frac{-2\mu\lambda_i}{1-\beta}} \left\{\tilde{C}\ (t_0)_{i,i} + \frac{\beta}{(1-\beta)}\left(\tilde{C}\ (t_0)_{i,i+N} + \tilde{C}\ (t_0)_{i+N,i}\right) + \left(\frac{\beta}{1-\beta}\right)^2 \tilde{C}\ (t_0)_{i+N,i+N}\right\}.$$

Assuming that mean square of the weight update (i.e. $E[v_i v_j]$) is larger than the $E[v_i \Omega_j]$ or $E[\Omega_i \Omega_j]$ we have

$$(C_H)_{ii} \approx \left(\frac{t}{t_0}\right)^{\frac{-2\mu\lambda_i}{1-\beta}} \tilde{C}\ (t_0)_{i,i}. \tag{F.14}$$

F.3 Particular Solution

We now compute the diagonal components of the particular solution C_P, where

$$C_P = \text{Top left Quadrant of } \left\{\int_{t_0}^{t} d\tau\ \frac{\mu_0^2}{\tau^2}\ U(t,\tau)\ \ \tilde{D}\ \ U^T(t,\tau)\right\} \tag{F.15}$$

and

$$U^T(t,\tau) = Q(t,\tau)\ \Lambda(t,\tau)\ Q^{-1}(t,\tau).$$

Multiplying out we obtain

$$(C_P)_{ii} = \int_{t_0}^{t} d\tau\ \frac{\mu_0^2}{\tau^2}\left(\frac{-g_{i-}e^{\eta_{i+}} + g_{i+}e^{\eta_{i-}} + (g_{i+}g_{i-})(e^{\eta_{i-}} - e^{\eta_{i+}})}{g_{i+} - g_{i-}}\right)^2\ D_{ii}$$

where $g_{i\pm}$ and $\eta_{i\pm}$ are functions of t and τ. Expanding out the integrand and keeping terms with $e^{\eta_{i+}}$ we get

$$(C_P)_{ii} \approx \int_{t_0}^{t} d\tau\ \frac{\mu_0^2}{\tau^2}\left(\frac{g_{i-}(1+g_{i+})\ e^{\eta_{i+}}}{g_{i+} - g_{i-}}\right)^2\ D_{ii}\ .$$

Letting

$$g_{i-}(1+g_{i+}) \approx \frac{t-\tau}{\mu_0\lambda_i \log(t/\tau)},\ \ g_{i-} - g_{i+} \approx \frac{(1-\beta)(t-\tau)}{\mu_0\lambda_i \log(t/\tau)},\ \text{ and }\ e^{\eta_{i+}} \approx \left(\frac{\tau}{t}\right)^{\frac{\mu_0\lambda_i}{1-\beta}}$$

we get

$$(C_P)_{ii} \approx \frac{\mu_0^2\ D_{ii}}{(1-\beta)^2}\ \int_{t_0}^{t} d\tau\ \frac{1}{\tau^2}\left(\frac{\tau}{t}\right)^{\frac{2\mu_0\lambda_i}{1-\beta}}.$$

Integrating we find

$$(C_P)_{ii} \approx \frac{\mu_0^2\ D_{ii}}{(1-\beta)\ (2\mu_0\lambda_i - 1 + \beta)}\left(\frac{1}{t} - \frac{1}{t_0}\left(\frac{t_0}{t}\right)^{\frac{2\mu_0\lambda_i}{1-\beta}}\right). \tag{F.16}$$

F.4 Squared Weight Error and Convergence Regimes

Combining the homogeneous and particular solutions in (F.14) and (F.16) we find

$$\begin{aligned} E[|v|^2] &= \text{Trace}[C_H + C_P] \\ &\approx \sum_{i=1}^{N} \left\{ \widetilde{C}\,(t_0)_{ii} \left(\frac{t_0}{t}\right)^{\frac{2\mu_0\lambda_i}{1-\beta}} + \right. \\ &\qquad \left. \frac{\mu_0^2\, D_{ii}}{(1-\beta)\,(2\mu_0\lambda_i - 1 + \beta)} \left(\frac{1}{t} - \frac{1}{t_0} \left(\frac{t_0}{t}\right)^{\frac{2\mu_0\lambda_i}{1-\beta}} \right) \right\} . \end{aligned} \tag{F.17}$$

Inspection of (F.17) reveals that there are two regimes of interest:

1. For $\mu_0 \geq \mu_{crit} \equiv \frac{1-\beta}{2\lambda_{min}}$, $E[|v|^2]$ drops off asymptotically as $\frac{1}{t}$.
2. For $\mu_0 < \mu_{crit}$, $E[|v|^2]$ drops off asymptotically as $\left(\frac{1}{t}\right)^{\frac{2\mu_0\lambda_{min}}{1-\beta}}$.

Appendix G

Capping Adaptive Momentum for Linear Networks

Adaptive momentum for linear networks was defined in chapter 7 as

$$\beta_{adapt} \equiv I - \mu_0 x_t x_t^T, \quad \text{(1 output, } M = 1\text{)}. \tag{G.1}$$

To avoid divergence, we require that $\mu_0 < 1/\lambda_{max}$ where λ_{max} is the largest eigenvalue of the Hessian of the cost function. In cases where we do not want to compute λ_{max}, an alternative is to cap each *sample* of β_{adapt} so that its eigenvalues are bounded between 0 and 1. Note that the eigenvalues of R are all positive so that we need only bound the eigenvalues of β_{adapt} from below. In 1-dimension, this is easily achieved by setting

$$\beta_{adapt} = \max(0, 1 - \mu_0 x_t^2), \quad \text{(1-dimension)}. \tag{G.2}$$

Note that this is not ideal in that it bounds each sample of β_{adapt} to be above zero which is very different from bounding $\langle \beta_{adapt} \rangle_x$ above zero. Even if $\mu_0 < \frac{1}{\lambda_{max}}$, the noise in x may result in $1 - \mu_0 x_t^2$ being negative for some x_t. Thus, even when μ_0 is well below $\frac{1}{\lambda_{max}}$, the expected value of β_{adapt} may no longer equal $\langle 1 - \mu_0 x_t^2 \rangle_x$.

We have found in our simulations that capping does prevent divergence in cases when μ_0 is larger than $\frac{1}{\lambda_{max}}$. However, when $\mu_0 < \frac{1}{\lambda_{max}}$ performance is slightly degraded when capping is used probably because on average β_{adapt} is no longer equal to $1 - \mu_0 R$. Thus, our preference is to appropriately choose μ_0 and not cap β_{adapt}. However, in the event that this is not possible, we present the idea of capping.

A natural extension of capping to multiple dimensions is to first define

$$\gamma = I - \mu_0 \, x \, x^T \tag{G.3}$$

where I is the $N \times N$ identity matrix. Then β_{adapt} is derived from γ by zeroing out γ's negative eigenvalues. To do this, we first diagonalize γ to get $\gamma = Q\Psi Q^{-1}$ where

$$\Psi = diag\Big[\, 1, \ldots, 1,\ 1 - \mu_0 x^2 \,\Big]_N$$

$$Q = \begin{pmatrix} -x_2/x_1 & -x_3/x_1 & \ldots & -x_N/x_1 & x_1/x_N \\ 1 & 0 & \ldots & 0 & x_2/x_N \\ 0 & 1 & \ddots & & \vdots \\ \vdots & & \ddots & & \\ 0 & \ldots & & 0 & x_{N-1}/x_N \\ 0 & \ldots & & 1 & 1 \end{pmatrix}$$

$$Q^{-1} = \frac{1}{x^2} \begin{pmatrix} -x_1x_2 & x^2 - x_2^2 & -x_3x_2 & \ldots & -x_Nx_2 \\ -x_1x_3 & -x_2x_3 & x^2 - x_3^2 & & -x_Nx_3 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ & & & & -x_Nx_{N-1} \\ -x_1x_N & -x_2x_N & \ldots & & x^2 - x_N^2 \\ x_1x_N & x_2x_N & \ldots & x_{N-1}x_N & x_N^2 \end{pmatrix}$$

and $x^2 \equiv x^T x$. Then we have

$$\beta_{adapt} = Q\ diag\Big[\, 1, \ldots, 1,\ \max(0, 1 - \mu_0 x^2) \,\Big]\ Q^{-1}. \tag{G.4}$$

Multiplying this out we find that

$$\beta_{adapt} = I + \frac{x x^T}{x^2} \Big(-1 + \max(0, 1 - \mu_0 x^2) \Big) \tag{G.5}$$

or

$$\beta_{adapt} = I - c\, x\, x^T, \qquad \text{where } c = \min(\mu_0, 1/x^2). \tag{G.6}$$

Biographical Note

Jenny was born March 11, 1958 in Philadelphia, PA. She attended the Philadelphia High School for Girls, graduating in 1975. From there, she went on to attend Grinnell College in Iowa where she majored in physics and mathematics. She was elected to Phi Beta Kappa in her junior year. She graduated summa cum laude in 1979, winning Grinnell's Linn Smith Prize for Excellence in Mathematics and also the H. George Apostle Prize in Physics. During the last two summers of college she worked as a summer intern with a nuclear physics group at Brookhaven National Laboratory. After college she went on to graduate school in physics at the University of Wisconsin, Madison, receiving an M.S. in 1982. She then taught mathematics and physics for several years at Spring Garden College in Philadelphia. During this time she also attended night school at Drexel University eventually obtaining an M.S. in mathematics in 1986. She then moved into industry, working for Quantics, inc, where she did a combination of operations research, mathematical modeling, and computer programming. After six years at Quantics, she was ready to try something new so she quit her job and went off to study neural networks at the Oregon Graduate Institute.