A Linear Regression

It is useful to review linear regression because the mathematics of single-layer perceptrons are very similar, because more general networks are often cascades of single-layer networks, and because linear analyses are often useful first-order approximations of nonlinear systems. Consider a linear system

$$y = \mathbf{x}^T \mathbf{w} \tag{A.1}$$

where $\mathbf{x}$ is an input vector and $\mathbf{w}$ is a weight vector to be determined. Let d be the desired output for the given input $\mathbf{x} \in \mathbf{R}^N$ and assume $\mathbf{x}$ and d have stationary statistics. The output y is one-dimensional here but the derivation is easily generalized to higher output dimensions. The single-sample error is $e = d - y$ and the squared error is

$$\begin{aligned} e^2 &= e^T e \\ &= (d - \mathbf{x}^T \mathbf{w})^T (d - \mathbf{x}^T \mathbf{w}) \\ &= d^2 - 2d\mathbf{x}^T \mathbf{w} + \mathbf{w}^T \mathbf{x}\mathbf{x}^T \mathbf{w}. \end{aligned} \tag{A.2}$$

Let the error function be $1/2$ the mean squared error (to suppress a factor of 2 later on)

$$\begin{aligned} 2E &= E\left[e^2\right] \\ &= E\left[d^2\right] - 2E\left[d\mathbf{x}^T\right]\mathbf{w} + \mathbf{w}^T E\left[\mathbf{x}\mathbf{x}^T\right]\mathbf{w}. \end{aligned} \tag{A.3}$$

Note, E is the error function but $E[\cdot]$ denotes an expected value. Let $\mathbf{P} = E[d\mathbf{x}]$ and $\mathbf{R} = E\left[\mathbf{x}\mathbf{x}^T\right]$. $\mathbf{P}$ is the input-target correlation vector with elements $p_j = E\left[dx_j\right]$ and $\mathbf{R}$ is the input autocorrelation matrix with elements $r_{ij} = E\left[x_i x_j\right]$. Then (A.3) can be written

$$2E = E\left[d^2\right] - 2\mathbf{P}^T \mathbf{w} + \mathbf{w}^T \mathbf{R}\mathbf{w}, \tag{A.4}$$

which is a quadratic function of $\mathbf{w}$. $\mathbf{R}$ is a real symmetric matrix and thus positive-semi-definite, $\mathbf{w}^T \mathbf{R}\mathbf{w} \geq 0$, so E has a single global minimum. The derivative of E with respect to $\mathbf{w}$ is

$$\frac{\partial E}{\partial \mathbf{w}} = \mathbf{R}\mathbf{w} - \mathbf{P}. \tag{A.5}$$

Setting this to zero produces

$$\mathbf{R}\mathbf{w} = \mathbf{P}, \tag{A.6}$$

which can be solved to obtain the optimum weight vector $\mathbf{w}^*$

$$\mathbf{w}^* = \mathbf{R}^{-1}\mathbf{P}. \tag{A.7}$$

Numerical analysis texts suggest several ways to solve systems of linear equations that may be preferable to inversion when $\mathbf{R}$ is poorly conditioned.

Substitution into (A.4) and simplifying produces an expression for the minimum error obtained

$$\begin{aligned} 2E_{min} &= E\left[d_k^2\right] + \mathbf{w}^{*T}\mathbf{R}\mathbf{w}^* - 2\mathbf{P}^T\mathbf{w}^* \\ &= E\left[d_k^2\right] - \mathbf{P}^T\mathbf{w}^* \\ E_{min} &= \frac{1}{2}\left(\sigma_d^2 + \mu_d^2 - \mathbf{P}^T\mathbf{w}^*\right) \end{aligned} \tag{A.8}$$

where μ_d and σ_d are the mean and standard deviation of the target. This may be smaller when $\mu_d = 0$, which is reasonable because (A.1) does not include an offset term.

It can be shown that the error is a quadratic function of the difference $\mathbf{w} - \mathbf{w}^*$

$$2E = 2E_{min} + (\mathbf{w} - \mathbf{w}^*)^T\mathbf{R}(\mathbf{w} - \mathbf{w}^*). \tag{A.9}$$

It is minimum at $\mathbf{w} = \mathbf{w}^*$ and increases quadratically with the difference $\mathbf{w} - \mathbf{w}^*$. At the optimum, the error is uncorrelated with the input

$$\begin{aligned} e &= d - \mathbf{x}^T\mathbf{w} \\ e\mathbf{x} &= d\mathbf{x} - \mathbf{x}\mathbf{x}^T\mathbf{w} \\ E\left[e\mathbf{x}\right] &= \mathbf{P} - \mathbf{R}\mathbf{w} \\ &= \mathbf{P} - \mathbf{R}\mathbf{R}^{-1}\mathbf{P} \\ &= \mathbf{P} - \mathbf{P} \\ &= 0. \end{aligned}$$

This makes sense because correlation would indicate that the error contains remaining linearly predictable elements that could be reduced further by modifying $\mathbf{w}$.

A.1 Newton's Method

Let $\mathbf{g} = \frac{\partial E}{\partial \mathbf{w}} = \mathbf{R}\mathbf{w} - \mathbf{P}$. Then

$$\mathbf{R}^{-1}\mathbf{g} = \mathbf{w} - \mathbf{R}^{-1}\mathbf{P}$$

$$\mathbf{R}^{-1}\mathbf{g} = \mathbf{w} - \mathbf{w}^*$$

This gives the update rule

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta \mathbf{R}^{-1} \mathbf{g}_k \tag{A.10}$$

$$\Delta \mathbf{w}_{k+1} = \eta (\mathbf{w}^* - \mathbf{w}_k) \tag{A.11}$$

(where the subscripts index time rather than vector elements). At each step, $\mathbf{w}$ is changed by a fraction η of the difference $(\mathbf{w}^* - \mathbf{w}_k)$. Because the error surface is quadratic, the solution could be obtained in a single step when $\eta = 1$. For nonlinear optimization tasks such as most neural network problems, however, the linear approximation is only locally valid and smaller step sizes are used to avoid straying too far from the region of validity; one-step convergence is not possible and iteration with a smaller step size is necessary. In the linear case, the eventual solution is the same however, $\mathbf{w}_\infty = \mathbf{w}^*$.

A.2 Gradient Descent

For large input dimensions (or small computers), storage and accurate inversion of $\mathbf{R}$ can be a problem so iterative procedures are often used. In simple gradient descent, $\mathbf{R}$ is replaced by $\mathbf{I}$ (actually, $\mathbf{R} = \mathbf{I}$ only for zero-mean, unit-variance, uncorrelated inputs) and the weight vector moves directly down the local gradient

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta \mathbf{g}. \tag{A.12}$$

The step size η controls how much $\mathbf{w}$ changes in each iteration. Although this will eventually converge to the optimal solution, the time required may be very long as the gradient, and thus the step size, approaches zero at the minimum.

Stability requires $0 < \eta < 2/\lambda_{max}$ where λ_{max} is the largest eigenvalue of $\mathbf{R}$ (see below). When $\eta > 2/\lambda_{max}$, the system will diverge. Because λ_{max} is usually unknown, a smaller than optimal step size must be used which may further slow convergence.

Convergence Rate of Gradient Descent Assuming $\mathbf{R}$ is full rank with distinct eigenvalues, it can be decomposed into $\mathbf{R} = \mathbf{V}\mathbf{D}\mathbf{V}^T$ where $\mathbf{V}$ is the matrix whose columns are eigenvectors of $\mathbf{R}$ and $\mathbf{D}$ is the diagonal matrix whose entries are the corresponding eigenvalues. Because the eigenvectors are orthonormal, $\mathbf{V}$ is unitary, $\mathbf{V}^T = \mathbf{V}^{-1}$. After changing coordinates $\mathbf{z} = \mathbf{V}^T(\mathbf{w} - \mathbf{w}^*)$, equation A.9 can be written

$$\begin{aligned} 2E &= 2E_{min} + (\mathbf{w} - \mathbf{w}^*)^T \mathbf{R} (\mathbf{w} - \mathbf{w}^*) \\ &= 2E_{min} + \mathbf{z}^T \mathbf{V}^T \mathbf{V} \mathbf{D} \mathbf{V}^T \mathbf{V} \mathbf{z} \qquad \text{(A.13)} \\ &= 2E_{min} + \mathbf{z}^T \mathbf{D} \mathbf{z}. \qquad \text{(A.14)} \end{aligned}$$

Because **D** is a diagonal matrix, E is now the sum of N uncoupled components

$$E = E_{min} + \frac{1}{2}\sum_{k=1}^{N} \lambda_k z_k^2 \tag{A.15}$$

where z_k is the projection of $\mathbf{w} - \mathbf{w}^*$ on the k^{th} eigenvector and λ_k is the corresponding eigenvalue. The gradient of E with respect to $\mathbf{z}$ becomes

$$\frac{\partial E}{\partial \mathbf{z}} = \mathbf{D}\mathbf{z}, \tag{A.16}$$

which gives the gradient descent update rule

$$\mathbf{z}(t+1) = \mathbf{z}(t) - \eta \mathbf{D}\mathbf{z}(t). \tag{A.17}$$

The components are decoupled, so we have N independent processes

$$\begin{aligned} z_k(t+1) &= z_k(t) - \eta\lambda_k z_k(t) \\ &= (1 - \eta\lambda_k) z_k(t) \end{aligned} \tag{A.18}$$

and after m time steps

$$z_k(t+m) = (1 - \eta\lambda_k)^m z_k(t).$$

For $z_k(t+m)$ to approach 0 as m becomes large, it is necessary that $|1 - \eta\lambda_k| < 1$, which requires $\lambda_k > 0$ and $0 < \eta < 2/\lambda_k$ for all k, that is,

$$0 < \eta < \frac{2}{\lambda_{max}}. \tag{A.19}$$

with fastest convergence occurring at $\eta = 1/\lambda_{max}$. Le Cun, Simard, and Pearlmutter [94] describe an iterative method for estimating λ_{max} in a neural network; see section 6.1.7.

From equation A.18, we have

$$\Delta z_k = -\eta\lambda_k z_k.$$

The continuous-time approximation

$$\frac{dz_k}{dt} = -\eta\lambda_k z(t)$$

has solution

$$z_k(t) = e^{-\eta\lambda_k t} z_k(0), \tag{A.20}$$

which shows the exponential nature of the convergence. The overall convergence is limited by the rate of convergence of the slowest component. Using the optimum $\eta = 1/\lambda_{max}$ gives

$$z_{slowest}(t) = \exp\left(-\frac{\lambda_{min}}{\lambda_{max}}t\right) z_k(0). \tag{A.21}$$

That is, the overall convergence is governed by the slowest time constant $\frac{\lambda_{max}}{\lambda_{min}}$, where λ_{min} is the smallest nonzero eigenvalue.

There may be a loophole here, however. Using (A.15), the overall error can then be expressed as

$$E = E_{min} + \frac{1}{2}\sum_{k=1}^{N} \lambda_k \exp\left(-2\frac{\lambda_k}{\lambda_{max}}t\right) z_k(0). \tag{A.22}$$

If we are satisfied when the error is small but nonzero, at 0.001 MSE say, then if $z_k(0)$ is small enough, the contribution of the kth component to the total error will be small and it will not be necessary to wait for these components to fully converge. Although small λ_k values cause slow convergence of the kth component, they also weight the contribution of the component to the total error.

An approximate expression for the distribution of eigenvalues in the case of random, uncorrelated inputs is given by Le Cun, Kanter, and Solla [92, 93] and leads to estimates of the learning time. The following points are made: Nonzero-mean inputs, correlations between inputs, and nonuniform input variances can all lead to increased spread between the minimum and maximum eigenvalues. For nonzero-mean inputs, there is an eigenvalue proportional to N so λ_{max} is much larger than in the zero-mean case and convergence time is slower. Subtracting the mean from the inputs suppresses the large eigenvalue and leads to faster training times. Nonuniform input variances can also lead to an increased spread in the eigenvalues which can be suppressed by rescaling the inputs. This result provides justification for centering and normalizing input variables.

For a multilayer network, the hidden layer outputs can be considered as inputs to the following layer. Because sigmoid outputs are always positive and therefore have a nonzero mean, while symmetric (odd) functions such as tanh are at least capable of a zero mean, this also provides justification for the empirical observation that use of tanh nonlinearities often produces faster training than sigmoid nodes. The result also provides justification for the suggestion of scaling the learning rate for each node by $1/M$, where M is the number of inputs into the node (see section 6.1.9).

A.3 The LMS Algorithm

The Widrow-Hoff learning rule [402], also called the LMS (least mean squares) algorithm or the delta rule is basically an iterative on-line implementation of linear regression. Like gradient descent, it avoids storing $\mathbf{R}$, but the LMS method reduces storage requirements even further. Gradient descent still requires $O(N)$ storage to accumulate error terms for the entire training set in order to approximate the true gradient before making a weight change. The LMS method avoids this may making weight changes as soon as errors occur. In the limit of very small learning rates, the result is the same. Like linear regression, it also minimizes the mean squared error of a linear fit so it shares many properties with linear regression and succeeds or fails in the same situations. An extensive summary of the LMS algorithm is provided by Widrow and Stearns [405].

As before, the half mean squared error is $E = \frac{1}{2}E\left[e^2\right]$ and the gradient is $\mathbf{g} = \frac{\partial E}{\partial \mathbf{w}}$. The LMS algorithm does steepest descent using an estimate of the gradient *based only on the error on the current pattern*

$$\hat{\mathbf{g}} = \frac{1}{2}\frac{\partial e^2}{\partial \mathbf{w}} \tag{A.23}$$

$$= -e\mathbf{x}. \tag{A.24}$$

That is, it does on-line rather than batch learning. The update rule is then

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta\hat{\mathbf{g}}_k \tag{A.25}$$

$$= \mathbf{w}_k + \eta e_k \mathbf{x}_k \tag{A.26}$$

where $0 < \eta < 2/\lambda_{max}$ for stability. Here, subscripts index pattern presentations rather than vector elements.

Because λ_{max} is unknown unless $\mathbf{R}$ is analyzed, an estimate must be used. $\mathbf{R}$ is nonnegative-definite, so all eigenvalues are nonnegative $\lambda_i \geq 0$ and λ_{max} can be bounded by

$$\lambda_{max} \leq \sum_i \lambda_i = \text{trace}(\mathbf{R}) \tag{A.27}$$

where $\text{trace}(\mathbf{R}) = \sum_i r_{ii} = \sum_i E\left[x_i^2\right]$. Because this places a upper bound on λ_{max} that may too high, the resulting η value may be smaller than necessary. Adaptive learning rate methods, perhaps initialized this way, may be able to improve training speed by adjusting the rate based on observed training performance. As an aside, this provides justification for centering input variables because $E\left[x_i^2\right] = \sigma_i^2 + \mu_i^2$ where σ_i^2 and μ_i are the variance and mean of input x_i; zero-mean inputs will produce lower estimates of λ_{max} and allow larger step sizes to be used.