

Training Multilayer Networks

Machine Learning II

Lecture 4-b



THE GEORGE
WASHINGTON
UNIVERSITY

WASHINGTON, DC

Training Set

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \dots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

Performance Indices

Mean Square Error

$$F(\mathbf{x}) = \frac{1}{QS^M} \sum_{q=1}^Q \sum_{i=1}^{S^M} (t_{i,q} - a_{i,q}^M)^2$$

Cross Entropy

$$F(\mathbf{x}) = - \sum_{q=1}^Q \sum_{i=1}^{S^M} t_{i,q} \ln \frac{a_{i,q}^M}{t_{i,q}}$$

Approximate performance index

2nd order Taylor series expansion (quadratic)

$$F(\mathbf{x}) \cong F(\mathbf{x}^*) + \nabla F(\mathbf{x})^T|_{\mathbf{x}^*} (\mathbf{x} - \mathbf{x}^*) + \frac{1}{2} (\mathbf{x} - \mathbf{x}^*)^T \nabla^2 F(\mathbf{x})|_{\mathbf{x}^*} (\mathbf{x} - \mathbf{x}^*)$$

Gradient – Direction of increasing $F(\mathbf{x})$

$$\nabla F(\mathbf{x}) = \left[\frac{\partial F(\mathbf{x})}{\partial x_1} \quad \frac{\partial F(\mathbf{x})}{\partial x_2} \quad \cdots \quad \frac{\partial F(\mathbf{x})}{\partial x_n} \right]^T$$

Hessian – Curvature of $F(\mathbf{x})$

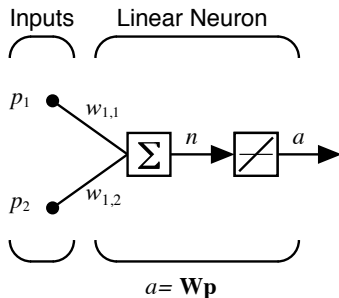
$$\nabla^2 F(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 F(\mathbf{x})}{\partial x_1^2} & \frac{\partial^2 F(\mathbf{x})}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 F(\mathbf{x})}{\partial x_1 \partial x_n} \\ \frac{\partial^2 F(\mathbf{x})}{\partial x_2 \partial x_1} & \frac{\partial^2 F(\mathbf{x})}{\partial x_2^2} & \cdots & \frac{\partial^2 F(\mathbf{x})}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 F(\mathbf{x})}{\partial x_n \partial x_1} & \frac{\partial^2 F(\mathbf{x})}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 F(\mathbf{x})}{\partial x_n^2} \end{bmatrix}$$

General optimization algorithm

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

- The search direction at iteration k is \mathbf{p}_k .
- The learning rate is α_k .
- For small learning rates, the largest reduction in $F(\mathbf{x})$ is obtained by setting $\mathbf{p}_k = -\nabla F(\mathbf{x}_k)$, the negative of the gradient direction. This is called the *steepest descent*, or gradient descent algorithm.

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \nabla F(\mathbf{x}_k)$$



$$\left\{ \mathbf{p}_1 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, \mathbf{t}_1 = [-1] \right\}, \left\{ \mathbf{p}_2 = \begin{bmatrix} 2 \\ -1 \end{bmatrix}, \mathbf{t}_2 = [-1] \right\}$$

$$\left\{ \mathbf{p}_3 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \mathbf{t}_3 = [1] \right\}, \left\{ \mathbf{p}_4 = \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \mathbf{t}_4 = [1] \right\}$$

Example performance index

$$\mathbf{U} = \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \mathbf{p}_3^T \\ \mathbf{p}_4^T \end{bmatrix}, \mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{bmatrix}, \mathbf{x} = \begin{bmatrix} w_{1,1} \\ w_{1,2} \end{bmatrix}$$

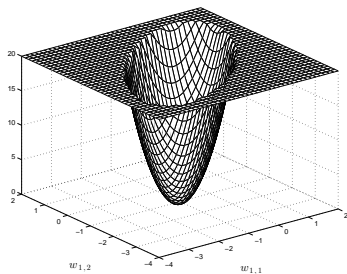
$$\begin{aligned} F(\mathbf{x}) &= \sum_{q=1}^4 (t_q - a_q)^2 = (\mathbf{t} - \mathbf{U}\mathbf{x})^T (\mathbf{t} - \mathbf{U}\mathbf{x}) \\ &= (\mathbf{t}^T \mathbf{t} - 2\mathbf{t}^T \mathbf{U}\mathbf{x} + \mathbf{x}^T \mathbf{U}^T \mathbf{U}\mathbf{x}) \\ &= c + \mathbf{d}^T \mathbf{x} + \frac{1}{2} \mathbf{x}^T \mathbf{A} \mathbf{x} \end{aligned}$$

$$\nabla F(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{d}, \nabla^2 F(\mathbf{x}) = \mathbf{A}$$

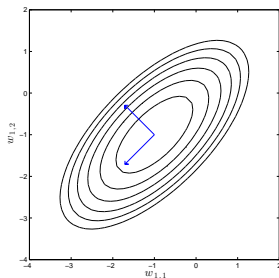
$$c = \mathbf{t}^T \mathbf{t}, \mathbf{d} = -2\mathbf{U}^T \mathbf{t}, \mathbf{A} = 2\mathbf{U}^T \mathbf{U}$$

Example performance surface

$$\mathbf{U} = \begin{bmatrix} -1 & 2 \\ 2 & -1 \\ 0 & -1 \\ -1 & 0 \end{bmatrix}, \mathbf{t} = \begin{bmatrix} -1 \\ -1 \\ 1 \\ 1 \end{bmatrix}, \mathbf{A} = \begin{bmatrix} 12 & -8 \\ -8 & 12 \end{bmatrix}, \mathbf{d} = \begin{bmatrix} 4 \\ 4 \end{bmatrix}, c = 4$$



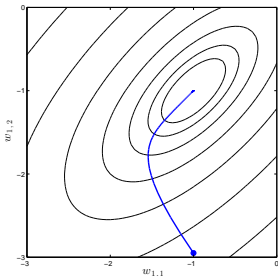
Performance surface



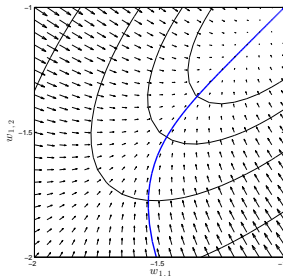
Contour plot

Example steepest descent path

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \nabla F(\mathbf{x}_k) = \mathbf{x}_k - 0.01 (\mathbf{A}\mathbf{x}_k + \mathbf{d}) = \begin{bmatrix} 0.88 & 0.08 \\ 0.08 & 0.88 \end{bmatrix} \mathbf{x}_k + \begin{bmatrix} 0.04 \\ 0.04 \end{bmatrix}$$



Steepest descent path



Zoomed path with gradients

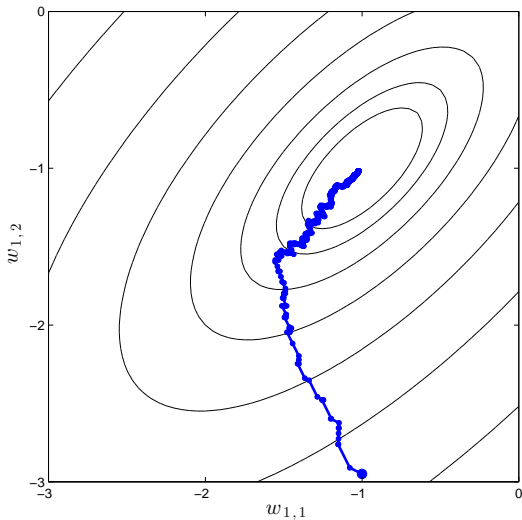
- Standard steepest descent is a batch algorithm, because the entire batch of data is used to compute the gradient.
- If we compute the gradient for one data point, it is an incremental, or stochastic algorithm.
- Mini-batches can also be used, where the algorithm operates on a subset of data – especially useful for large data sets.

$$\begin{aligned}\hat{F}(\mathbf{x}) &= (t_k - a_k)^2 \\ \nabla \hat{F}(\mathbf{x}) &= -2(t_k - a_k) \nabla a_k = -2e_k \mathbf{p}_k\end{aligned}$$

Stochastic gradient algorithm

$$\mathbf{x}_{k+1} = \mathbf{x}_k + 2\alpha e_k \mathbf{p}_k$$

Stochastic gradient trajectory



Steepest descent in multilayer networks

- For multilayer networks, the error is not a direct function of weights in the hidden layers.
- To compute the necessary gradients we need to use the chain rule.
- The chain rule is implemented one component at a time (e.g., performance function, transfer function, weight function).

$$\begin{aligned}\frac{\partial \hat{F}(\mathbf{x})}{\partial w_{i,j}^m} &= \frac{\partial \hat{F}(\mathbf{x})}{\partial n_i^m} \frac{\partial n_i^m}{\partial w_{i,j}^m} = \frac{\partial \hat{F}(\mathbf{x})}{\partial n_i^m} \frac{\partial (\sum_{l=1}^{S^{m-1}} w_{i,l}^m a_l^{m-1} + b_i^m)}{\partial w_{i,j}^m} \\ &= \frac{\partial \hat{F}(\mathbf{x})}{\partial n_i^m} a_j^{m-1}\end{aligned}$$

Derivative across transfer function

$$\frac{\partial \mathbf{a}^m}{\partial (\mathbf{n}^m)^T} = \dot{\mathbf{F}}^m(\mathbf{n}^m) = \begin{bmatrix} \frac{\partial f_1^m(\mathbf{n}^m)}{\partial n_1^m} & \frac{\partial f_1^m(\mathbf{n}^m)}{\partial n_2^m} & \dots & \frac{\partial f_1^m(\mathbf{n}^m)}{\partial n_{s^m}^m} \\ \frac{\partial f_2^m(\mathbf{n}^m)}{\partial n_1^m} & \frac{\partial f_2^m(\mathbf{n}^m)}{\partial n_2^m} & \dots & \frac{\partial f_2^m(\mathbf{n}^m)}{\partial n_{s^m}^m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_{s^m}^m(\mathbf{n}^m)}{\partial n_1^m} & \frac{\partial f_{s^m}^m(\mathbf{n}^m)}{\partial n_2^m} & \dots & \frac{\partial f_{s^m}^m(\mathbf{n}^m)}{\partial n_{s^m}^m} \end{bmatrix}$$

Derivative across weight function

$$\frac{\partial \mathbf{n}^{m+1}}{\partial (\mathbf{a}^m)^T} = \frac{\partial [\mathbf{W}^{m+1} \mathbf{a}^m + \mathbf{b}^{m+1}]}{\partial (\mathbf{a}^m)^T} = \mathbf{W}^{m+1}$$

Example transfer function derivatives

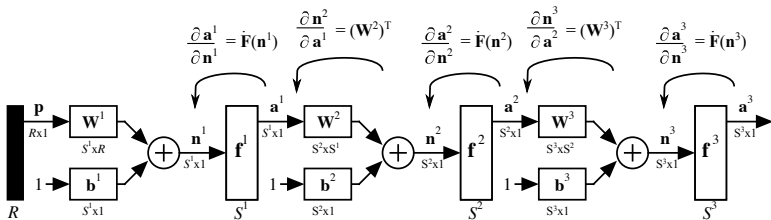
Poslin

$$\dot{\mathbf{F}}^m(\mathbf{n}^m) = \begin{bmatrix} \text{hardlim}(n_1^m) & 0 & \cdots & 0 \\ 0 & \text{hardlim}(n_2^m) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \text{hardlim}(n_{S^m}^m) \end{bmatrix}$$

Softmax

$$\dot{\mathbf{F}}^m(\mathbf{n}^m) = \begin{bmatrix} a_1^m \left(\sum_{i=1}^{S^m} a_i^m - a_1^m \right) & -a_1^m a_2^m & \cdots & -a_1^m a_{S^m}^m \\ -a_2^m a_1^m & a_2^m \left(\sum_{i=1}^{S^m} a_i^m - a_2^m \right) & \cdots & -a_2^m a_{S^m}^m \\ \vdots & \vdots & \ddots & \vdots \\ -a_{S^m}^m a_1^m & -a_{S^m}^m a_2^m & \cdots & a_{S^m}^m \left(\sum_{i=1}^{S^m} a_i^m - a_{S^m}^m \right) \end{bmatrix}$$

Multilayer chain rule (backpropagation)



$$\frac{\partial \hat{F}(\mathbf{x})}{\partial \mathbf{n}^1} = \frac{\partial \mathbf{a}^1}{\partial \mathbf{n}^1} \times \frac{\partial \mathbf{n}^2}{\partial \mathbf{a}^1} \times \frac{\partial \mathbf{a}^2}{\partial \mathbf{n}^2} \times \frac{\partial \mathbf{n}^3}{\partial \mathbf{a}^2} \times \frac{\partial \mathbf{a}^3}{\partial \mathbf{n}^3} \times \frac{\partial \hat{F}(\mathbf{x})}{\partial \mathbf{a}^3}$$

$$\frac{\partial \hat{F}(\mathbf{x})}{\partial \mathbf{n}^m} = \frac{\partial \mathbf{a}^m}{\partial \mathbf{n}^m} \times \frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{a}^m} \times \frac{\partial \hat{F}(\mathbf{x})}{\partial \mathbf{n}^{m+1}} = \dot{\mathbf{F}}(\mathbf{n}^m) (\mathbf{W}^{m+1})^T \frac{\partial \hat{F}(\mathbf{x})}{\partial \mathbf{n}^{m+1}}$$

Initializing backpropagation

For Mean Square Error

$$\frac{\partial \hat{F}(\mathbf{x})}{\partial n_i^M} = \sum_{j=1}^{S^M} \frac{\partial \hat{F}(\mathbf{x})}{\partial a_j^M} \frac{\partial a_j^M}{\partial n_i^M} = \frac{1}{S^M} \sum_{j=1}^{S^M} \frac{\partial (t_j - a_j^M)^2}{\partial a_j^M} \frac{\partial a_j^M}{\partial n_i^M}$$

$$\frac{\partial \hat{F}(\mathbf{x})}{\partial n_i^M} = \frac{-2}{S^M} \sum_{j=1}^{S^M} (t_j - a_j^M) \frac{\partial a_j^M}{\partial n_i^M}$$

$$= \frac{-2}{S^M} \sum_{j=1}^{S^M} e_j \frac{\partial a_j^M}{\partial n_i^M}$$

$$\frac{\partial \hat{F}(\mathbf{x})}{\partial \mathbf{n}^M} = \frac{\partial (\mathbf{a}^M)^T}{\partial \mathbf{n}^M} \frac{\partial \hat{F}(\mathbf{x})}{\partial \mathbf{a}^M} = \frac{-2}{S^M} \mathbf{\dot{F}}^M(\mathbf{n}^m) \mathbf{e}$$

Summary of multilayer stochastic gradient

If we define the sensitivity to be

$$\mathbf{s}^m \triangleq \frac{\partial \hat{F}(\mathbf{x})}{\partial \mathbf{n}^m}$$

Then the stochastic gradient for mean square error can be computed as

$$\mathbf{s}^M = \frac{-2}{S^M} \dot{\mathbf{F}}^M(\mathbf{n}^M) \mathbf{e}$$

$$\mathbf{s}^m = \dot{\mathbf{F}}(\mathbf{n}^m) (\mathbf{W}^{m+1})^T \mathbf{s}^{m+1}$$

$$\frac{\partial \hat{F}(\mathbf{x})}{\partial w_{ij}^m} = s_i^m a_j^{m-1}$$

$$\frac{\partial \hat{F}(\mathbf{x})}{\partial b_i^m} = s_i^m$$