

Lecture 10.
MLP (II):
Single Neuron Weight Learning
and Nonlinear Optimization

Outline

- Single neuron case: Nonlinear error correcting learning
- Nonlinear optimization

Solving XOR Problem

A training sample consists of a feature vector (x_1, x_2) and a label z . In XOR problem, there are 4 training samples $(0, 0; 0)$, $(0, 1; 1)$, $(1, 0; 1)$, and $(1, 1; 0)$.

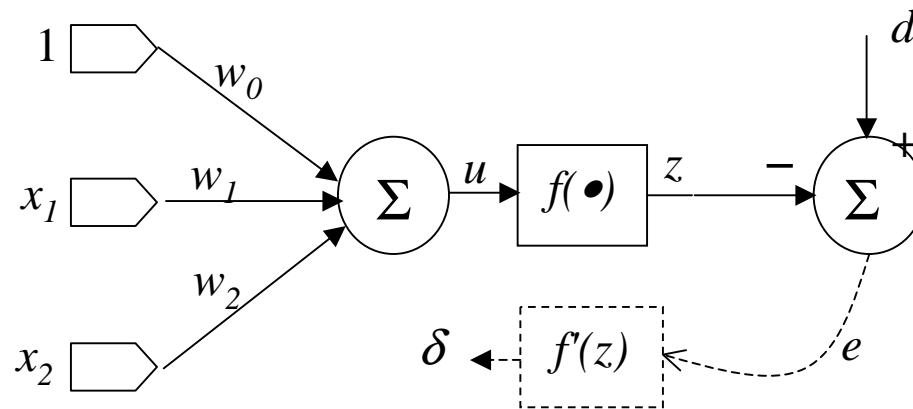
These four training samples give four equations for 9 variables

(x_1, x_2)	z
$(0, 0)$	$0 = f(w_{10}^{(2)} + w_{11}^{(2)} f(w_{10}^{(1)}) + w_{12}^{(2)} f(w_{20}^{(1)}))$
$(0, 1)$	$1 = f(w_{10}^{(2)} + w_{11}^{(2)} f(w_{10}^{(1)} + w_{12}^{(1)}) + w_{12}^{(2)} f(w_{20}^{(1)} + w_{22}^{(1)}))$
$(1, 0)$	$1 = f(w_{10}^{(2)} + w_{11}^{(2)} f(w_{10}^{(1)} + w_{11}^{(1)}) + w_{12}^{(2)} f(w_{20}^{(1)} + w_{21}^{(1)}))$
$(1, 1)$	$0 = f(w_{10}^{(2)} + w_{11}^{(2)} f(w_{10}^{(1)} + w_{11}^{(1)} + w_{12}^{(1)}) + w_{12}^{(2)} f(w_{20}^{(1)} + w_{21}^{(1)} + w_{22}^{(1)}))$

Finding Weights of MLP

- In MLP applications, often there are large number of parameters (weights).
- Sometimes, the number of unknowns is more than the number of equations (as in XOR case). In such a case, the solution may not be unique.
- The objective is to solve for a set of weights that minimize the actual output of the MLP and the corresponding desired output. Often, the cost function is a sum of square of such difference.
- This leads to a nonlinear least square optimization problem.

Single Neuron Learning



Nonlinear least square cost function:

$$E = \sum_{k=1}^K [e(k)]^2 = \sum_{k=1}^K [d(k) - z(k)]^2 = \sum_{k=1}^K [d(k) - f(\mathbf{W}\mathbf{x}(k))]^2$$

Goal: Find $\mathbf{W} = [w_0, w_1, w_2]$ by minimizing E over given training samples.

Gradient based Steepest Descent

The weight update formula has the format of

$$w_i(t+1) = w_i(t) + \eta \nabla_{w_i} E = w_i(t) + \eta \frac{\partial E}{\partial w_i(t)}$$

t is the *epoch* index. During each epoch, K training samples will be applied and the weight will be updated once.

Since
$$\frac{\partial E}{\partial w_i} = \sum_{k=1}^K \frac{\partial [e(k)]^2}{\partial w_i} = \sum_{k=1}^K 2[d(k) - z(k)] \left(-\frac{\partial z(k)}{\partial w_i} \right)$$

And
$$\frac{\partial z(k)}{\partial w_i} = \frac{\partial f(u)}{\partial u} \frac{\partial u}{\partial w_i} = f'(u) \frac{\partial}{\partial w_i} \left(\sum_{j=0}^2 w_j x_j \right) = f'(u) x_i$$

Hence
$$\frac{\partial E}{\partial w_i} = -2 \sum_{k=1}^K [d(k) - z(k)] f'(u(k)) x_i(k)$$

Delta Error

Denote $\delta(k) = [d(k) - z(k)]f'(u(k))$, then

$$\frac{\partial E}{\partial w_i} = -2 \sum_{k=1}^K \delta(k) x_i(k)$$

$\delta(k)$ is the error signal $e(k) = d(k) - z(k)$ modulated by the derivative of the activation function $f'(u(k))$ and hence represents the amount of correction needs to be applied to the weight w_i for the given input $x_i(k)$. Therefore, the weight update formula for a single-layer, single neuron perceptron has the following format

$$w_i(t+1) = w_i(t) + \eta \sum_{k=1}^K \delta(k) x_i(k)$$

This is the error-correcting learning formula discussed earlier.

Nonlinear Optimization

Problem: Given a nonlinear cost function $E(W) \geq 0$. Find W such that $E(W)$ is minimized.

Taylor's Series Expansion

$$E(W) \approx E(W(t)) + (\nabla_W E)^T (W - W(t)) + (1/2)(W - W(t))^T H_W(E) (W - W(t))$$

where

$$\nabla_W (E) = \left[\frac{\partial E}{\partial W_1}, \frac{\partial E}{\partial W_2}, \dots, \frac{\partial E}{\partial W_N} \right]^T$$

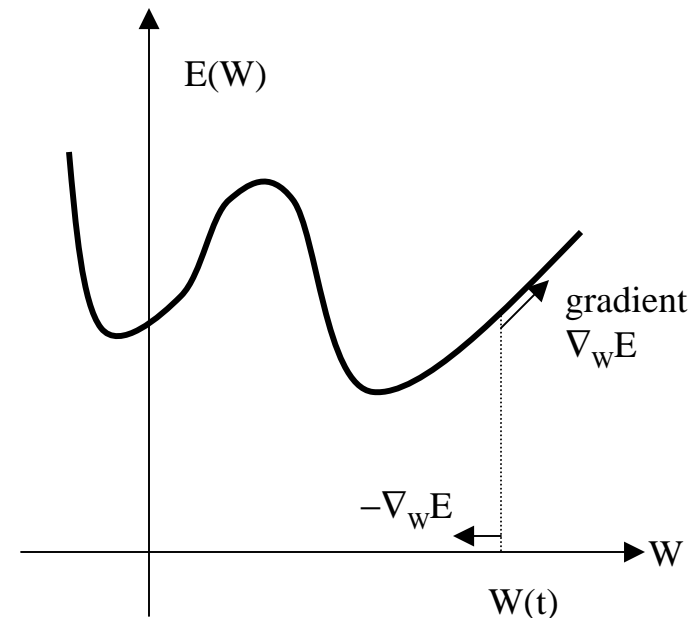
$$H_W (E) = \begin{bmatrix} \frac{\partial^2 E}{\partial W_1^2} & \frac{\partial^2 E}{\partial W_1 \partial W_2} & \dots & \frac{\partial^2 E}{\partial W_1 \partial W_N} \\ \frac{\partial^2 E}{\partial W_2 \partial W_1} & \frac{\partial^2 E}{\partial W_2^2} & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 E}{\partial W_N \partial W_1} & \frac{\partial^2 E}{\partial W_N \partial W_2} & \dots & \frac{\partial^2 E}{\partial W_N^2} \end{bmatrix}$$

Steepest Descent

- Use a first order approximation:

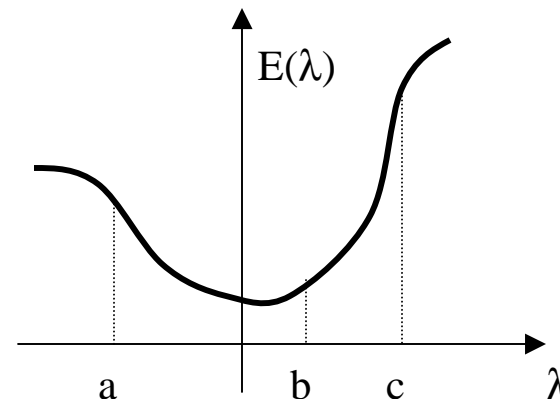
$$E(W) \approx E(W(t)) + (\nabla_{W(t)} E)^T (W - W(t))$$
- The amount of reduction of $E(W)$ from $E(W(t))$ is proportional to the inner product of $\nabla_{W(t)} E$ and $\Delta W = W - W(t)$.
- If $|\Delta W|$ is fixed, then the inner product is maximized when the two vectors are aligned. Since $E(W)$ is to be reduced, we should have

$$\Delta W \propto -\nabla_{W(t)} E \equiv -g(t)$$
- This is called the steepest descent method.



Line Search

- Steepest descent only gives the direction of ΔW but not its magnitude.
- Often we use a present step size η to control the magnitude of ΔW
- Line search allows us to find the locally optimal step size along $-g(t)$.
- Set $E(W(t+1), \lambda) = E(W(t) + \lambda(-g))$. Our goal is to find λ so that $E(W(t+1))$ is minimized.
- Along $-g$, select points a, b, c such that $E(a) > E(b), E(c) > E(b)$.
- Then λ is selected by assuming $E(\lambda)$ is a quadratic function of λ .



Conjugate Gradient Method

Let $d(n)$ be the search direction. The conjugate gradient method decides the new search direction $d(n+1)$ as:

$$\underline{d}(t+1) = -\underline{g}(t+1) + \beta(t+1)\underline{d}(t)$$

where

$$\beta(t+1) = \frac{[\underline{g}(t+1)]^T [\underline{g}(t+1) - \underline{g}(t)]}{[\underline{g}(t)]^T \underline{g}(t)}$$

$\beta(t+1)$ is computed here using a *Polak-Ribiere formula*.

Conjugate Gradient Algorithm

1. Choose $W(0)$.
2. Evaluate $g(0) = \nabla_{W(0)} E$ and set $d(0) = -g(0)$
3. $W(t+1) = W(t) + \alpha d(t)$. Use line search to find α that minimizes $E(W(t) + \alpha d(t))$.
4. If not converged, compute $g(t+1)$, and $\beta(t+1)$. Then $d(t+1) = -g(t+1) + \beta(t+1)d(t)$ go to step 3.

Newton's Method

2nd order approximation:

$$E(W) \approx E(W(t)) + (g(t))^T(W-W(t)) + (1/2) (W-W(t))^T H_{W(t)}(E)(W-W(t))$$

Setting $\nabla_W E = 0$ w.r.p. W (not $W(t)$), and solve $\Delta W = W - W(t)$, we have

$$H_{W(t)}(E) \Delta W + g(W(t)) = 0$$

Thus, $\Delta W = -H^{-1} g(t)$

This is the Newton's method.

Difficulties

H^{-1} : too complicate to compute!

Solution

Approximate H^{-1} by different matrices:

1. Quasi-Newton Method
2. Levenberg-marquardt method