Learning Processes :

Learning is a process by which the free parameters of a NN are adapted through a process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place.

Five basic learning rules :

① Error-Correction learning ② Memory-based learning ③ Hebbian Learning.

④ Competitive learning &

⑤ Boltzmann Learning.

Error-Correction learning is rooted in optimum Filtering.

* Memory-based learning operates by memorizing the training data explicitly.

* Hebbian & Competitive learning are bolh inspired by neurobiological considerations.

* Boltzmann is based on statistical mechanics.

Error-Correction Learning:

Consider the simple case of a neuron K constituting the only computational node in the output layer of a feedforward neural network as shown in the fig.

Neuron K is driven by a signal vector $x(n)$ produced by one or more layers of hidden neurons, which are themselves driven by an input vector (stimulus) applied to the source nodes (ie input layer) of the neural network.

n denotes discrete time

The output signal of neuron K is denoted by $y_K(n)$.

The output signal is compared to a desired response or target output, denoted by $d_K(n)$.

An error signal denoted by $e_K(n)$ is produced & given as

$$e_K(n) = d_K(n) - y_K(n) \quad \text{——} (1)$$

The corrective adjustments are designed to make the output signal $y_K(n)$ come closer to the desired response $d_K(n)$ in a step-by-step manner. This is done by minimizing the cost function or index of performance. $\mathcal{E}(n)$

$$\mathcal{E}(n) = \frac{1}{2} e_k^2(n) \quad \text{— (2)}$$

$\mathcal{E}(n)$ is the instantaneous value of the error energy. The step-by-step adjustments to the synaptic weights of neuron k are continued until the system reaches a steady state. At that point the learning process is terminated.

The learning process is referred to as error-correction learning.

$\mathcal{E}(n)$ the cost function leads to a learning rule commonly referred to as the delta rule or Widrow-Hoff rule.

Let $w_{kj}(n)$ denote the value of synaptic weight w_{kj} of neuron k excited by element $x_j(n)$ of the signal vector $x(n)$ at time step n.

According to the delta rule, the adjustment $\Delta w_{kj}(n)$ applied to the synaptic weight w_{kj} at time step n is defined by

$$\Delta w_{kj}(n) = \eta e_k(n) x_j(n) \quad \text{— (3)}$$

η is a positive constant that determines the rate of learning

η → learning-rate parameter

The updated value of synaptic weight w_{kj} is determined by

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n) \quad \text{---(4)}$$

In effect, $w_{kj}(n)$ and $w_{kj}(n+1)$ may be viewed as the old & new values of synaptic weight w_{kj}

$$w_{kj}(n) = z^{-1}[w_{kj}(n+1)] \quad \text{---(5)}$$

where z^{-1} is the unit-delay operator

z^{-1} represents a storage element

Method of Steepest Descent

In the method of steepest descent, the successive adjustments applied to the weight vector w are in the direction of steepest descent, that is in a direction opposite to the gradient vector $\nabla \mathcal{E}(w)$.

For convenience of presentation we write

$$g = \nabla \mathcal{E}(w) \quad \text{---} (1)$$

Accordingly, the steepest descent algorithm is formally described by

$$w(n+1) = w(n) - \eta g(n) \quad (2)$$

$$\Delta w_{kj}(n) = \eta \delta_k(n) x_j(n)$$

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n)$$

$$w(n+1) - w(n) = -\eta g(n)$$

where η is a positive constant called the stepsize or learning-rate parameter and $g(n)$ is gradient vector evaluated at the point $w(n)$.

In going from iteration n to $n+1$ the algorithm applies the correction

$$\Delta w(n) = w(n+1) - w(n) \quad \text{---} (3)$$

$$\Delta w(n) = -\eta g(n)$$

First-order Taylor series expansion around $w(n)$ to approximate $\mathcal{E}(w(n+1))$ as

$$\mathcal{E}(w(n+1)) = \mathcal{E}(w(n)) + g^T(n) \Delta w(n)$$

the use of which is justified for small η. Substituting eqn (3) in this approximate relation yields

$$\mathcal{E}(w(n+1)) = \mathcal{E}(w(n)) - \eta g^T(n) g(n)$$
$$= \mathcal{E}(w(n)) - \eta \|g(n)\|^2$$

Which shows that, for a positive learning-rate parameter η, the cost function is decreased as the algorithm progresses from one iteration to the next.

* when η is small, the transient response of the algorithm is overdamped.

* when η is large, the transient response of the algorithm is underdamped.

Linear Least-Squares Filter:

Linear least-squares filter has two distinctive characteristics First, the single neuron around which it is built is linear, as shown in the model

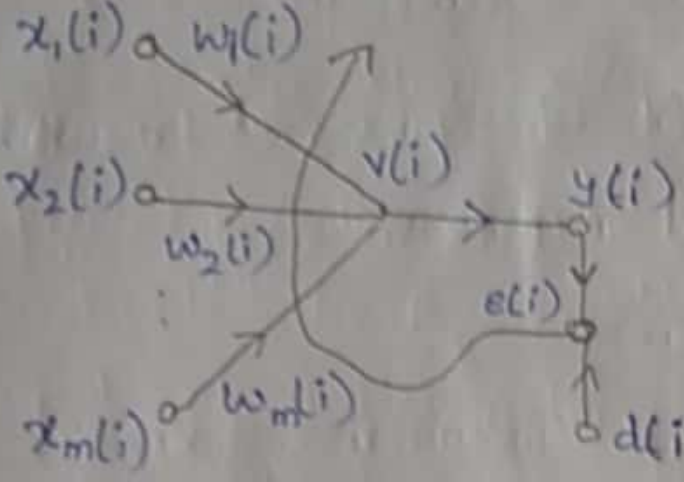

fig: model of neuron

Second the cost function $\mathcal{E}(w)$ used to design the filter consists of the sum of error squares as defined in $\mathcal{E}(w) = \frac{1}{2}\sum_{i=1}^{n} e^2(i)$

& $y(i) = v(i) = \sum_{k=1}^{m} w_k(i)\, x_k(i)$

$$y(i) = x^T(i)\, w(i) \quad \text{where } w(i) = [w_1$$

where $w(i) = [w_1(i), w_2(i) \ldots w_m(i)]^T$

$$e(i) = d(i) - y(i)$$

Express the error vector $e(n)$ as follows

$$e(n) = d(n) - [x(1), x(2) \ldots x(n)]^T w(n)$$
$$= d(n) - x(n)\, w(n) \quad \text{——— (1)}$$

where $d(n)$ is the n-by-1 desired response vector.

$$d(n) = [d(1), d(2) \ldots d(n)]^T$$

and $x(n)$ is the n-by-m data matrix

$$x(n) = [x(1), x(2) \ldots x(n)]^T$$

Differentiating eqn (1) with respect to $w(n)$ yields the gradient matrix.

$$\nabla e(n) = -x^T(n)$$

Correspondlingly, the Jacobian of $e(n)$ is

$$J(n) = -x(n) \quad \text{——} (2)$$

Substitute eqn (1) & (2) in the below eqn

$$w(n+1) = w(n) - (J^T(n) J(n))^{-1} J^T(n) e(n)$$

$$w(n+1) = w(n) + (x^T(n) x(n))^{-1} x^T(n) (d(n) - x(n) w(n))$$

$$w(n+1) = w(n) + (x^T(n) x(n))^{-1} x^T(n).d(n) - (x^T(n) x(n))^{-1} x(n) w(n)$$

$$= w(n) - \frac{x(n) w(n) x^T(n)}{x^T(n) x(n)} + (x^T(n) x(n))^{-1} x^T(n).d(n)$$

$$w(n+1) = (x^T(n) x(n))^{-1} x^T(n).d(n) \quad \text{——} (3).$$

The term $(x^T(n) x(n))^{-1} x^T(n)$ is pseudoinverse of the ~~matrix~~ data matrix $x(n)$.

$$x^+(n) = (x^T(n) x(n))^{-1} x^T(n) \quad \text{——} (4).$$

Rewrite eqn $w(n+1) = x^+(n).d(n)$

The weight vector $w(n+1)$ solves the linear least-squares

Wiener Filter: Limiting form of the linear least-Squares filter for an Ergodic Environment

The input vector $x(i)$ and desired response $d(i)$ are drawn from an ergodic environment that is also stationary.

The second-order statistics is described by

* Correlation matrix of the input vector $x(i)$; it is denoted by R_x
* Cross-correlation vector between the input vector $x(i)$ and desired response $d(i)$; it is denoted by γ_{xd}

These two quantities are defined as follows, respectively.

$$R_x = E[x(i)x^T(i)]$$

$$= \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{\infty} x(n)x^T(n)$$

$$= \lim_{n\to\infty} \frac{1}{n} X^T(n)X(n)$$

&

$$\gamma_{xd} = E[x(i)d(i)]$$

$$= \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} x(i)d(i)$$

$$= \lim_{n\to\infty} \frac{1}{n} x^T(n).d(n)$$

Where E denotes the statistical expectation operator.

Accordingly we may reformulate the linear least-Squares soln of eqn (11) as follows

$$w(n+1) = (X^T(n)\,x(n))^{-1} X^T(n).d(n) \quad \text{---(1)}$$

$$w_o = \lim_{n\to\infty} w(n+1)$$

$$= \lim_{n\to\infty} (X^T(n)\,x(n))^{-1} . X^T(n).d(n)$$

$$= \lim_{n\to\infty} \frac{1}{n} \underbrace{(X^T(n)\,x(n))^{-1}}_{R_x} . \lim_{n\to\infty} \frac{1}{n} \underbrace{X^T(n).d(n)}_{r_{xd}}$$

$$w_o = R_x^{-1} r_{xd}$$

where R_x^{-1} is the inverse of the correlation matrix R_x

The weight vector w_o is called the Wiener Solution to the linear optimum filtering problem.