NN V: The generalized delta learning rule

We now focus on generalizing the delta learning rule for feedforward layered neural networks.

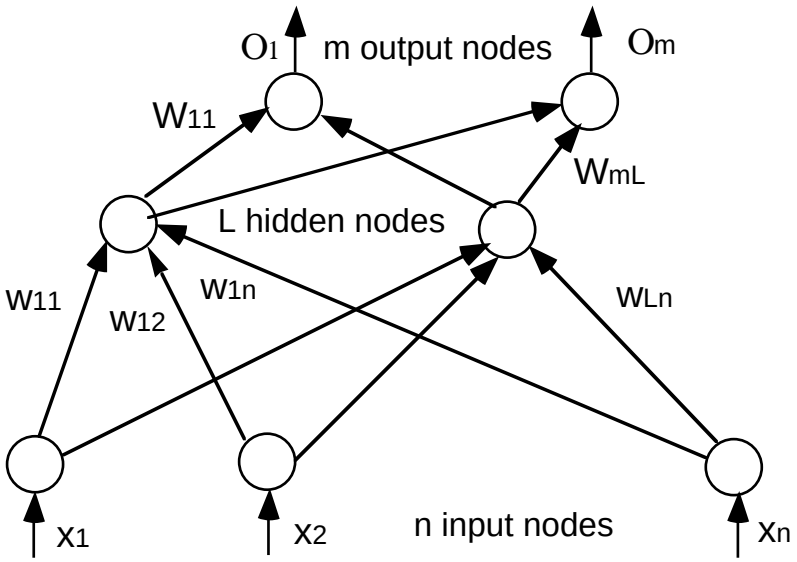
The architecture of the two-layer network considered below is shown in the figure. It has strictly speaking, two layers of processing neurons.

If, however, the layers of nodes are counted, then the network can also be labeled as a three-layer network. There is no agreement in the literature as to which approach is to be used to describe network architectures.

In this text we will use the term *layer* in reference to the actual number of existing and processing neuron layers. Layers with neurons whose outputs are not directly accesible are called internal or hidden

layers.

Thus the network of the figure is a two-layer network, which can be called a single hidden-layer network.

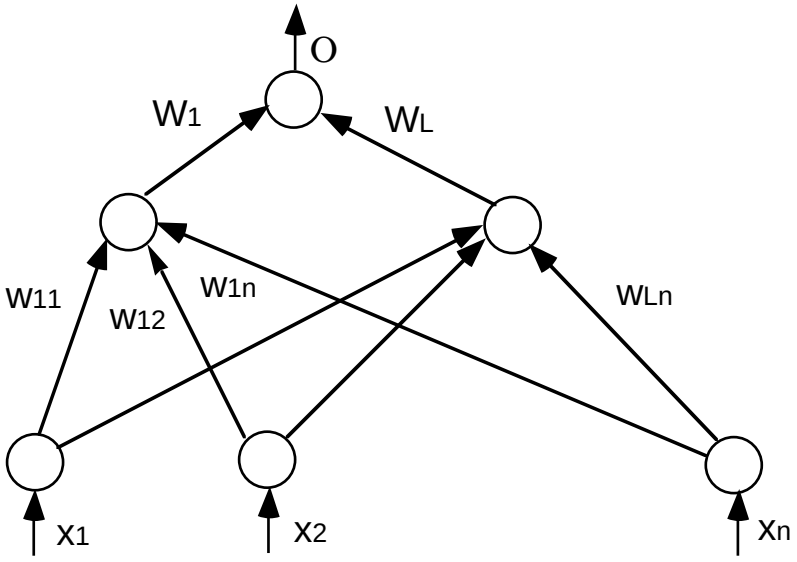


Layered neural network with two continuous perceptron layers.

The generalized delta rule is the most often used supervised learning algorithm of feedforward multi-

layer neural networks.

For simplicity we consider only a neural network with one hidden layer and one output node.



Two-layer neural network with one output node.

The measure of the error on an input/output training pattern

$$(x^k, y^k)$$

is defined by

$$E_k(W, w) = \frac{1}{2}(y^k - O^k)^2$$

where O^k is the computed output and the overall measure of the error is

$$E(W, w) = \sum_{k=1}^{K} E_k(W, w).$$

If an input vector x^k is presented to the network then it generates the following output

$$O^k = \frac{1}{1 + \exp(-W^T o^k)}$$

where o^k is the output vector of the hidden layer

$$o_l^k = \frac{1}{1 + \exp(-w_l^T x^k)}$$

and w_l denotes the weight vector of the l-th hidden neuron, $l = 1, \ldots, L$.

The rule for changing weights following presentation of input/output pair k is given by the gradient descent method, i.e. we minimize the quadratic error function by using the following iteration process

$$W := W - \eta \frac{\partial E_k(W, w)}{\partial W},$$
$$w_l := w_l - \eta \frac{\partial E_k(W, w)}{\partial w_l},$$

for $l = 1, \ldots, L$, and $\eta > 0$ is the learning rate.

By using the chain rule for derivatives of composed functions we get

$$\frac{\partial E_k(W,w)}{\partial W} = \frac{1}{2}\frac{\partial}{\partial W}\left[y^k - \frac{1}{1+\exp(-W^T o^k)}\right]^2 =$$

$$-(y^k - O^k)O^k(1-O^k)o^k$$

i.e. the rule for changing weights of the output unit is

$$W := W + \eta(y^k - O^k)O^k(1-O^k)o^k = W + \eta\delta_k o^k$$

that is

$$W_l := W_l + \eta\delta_k o_l^k,$$

for $l = 1, \ldots, L$, and we have used the notation

$$\delta_k = (y^k - O^k)O^k(1 - O^k).$$

Let us now compute the partial derivative of E_k with respect to w_l

$$\frac{\partial E_k(W, w)}{\partial w_l} = -O^k(1 - O^k)W_l o_l^k(1 - o_l^k)x^k$$

i.e. the rule for changing weights of the hidden units is

$$w_l := w_l + \eta\delta_k W_l o_l^k(1 - o_l^k)x^k,$$

for $l = 1, \ldots, L$. That is

$$w_{lj} := w_{lj} + \eta\delta_k W_l o_l^k(1 - o_l^k)x_j^k,$$

for $j = 1, \ldots, n$.

Summary 1. *The generalized delta learning rule (error backpropagation learning)*

We are given the training set

$$\{(x^1, y^1), \ldots, (x^K, y^K)\}$$

where $x^k = (x_1^k, \ldots, x_n^k)$ *and* $y^k \in \mathbf{R}$, $k = 1, \ldots, K$.

- **Step 1** $\eta > 0$, $E_{\max} > 0$ are chosen
- **Step 2** Weigts w are initialized at small random values, $k := 1$, and the running error E is set to 0
- **Step 3** Training starts here. Input x^k is presented, $x := x^k$, $y := y^k$, and output O is computed

  $$O = \frac{1}{1 + \exp(-W^T o)}$$

  where o_l is the output vector of the hidden layer

  $$o_l = \frac{1}{1 + \exp(-w_l^T x)}$$

- **Step 4** Weights of the output unit are updated

$$W := W + \eta\delta o$$

where $\delta = (y - O)O(1 - O)$.

- **Step 5** Weights of the hidden units are updated

$$w_l = w_l + \eta\delta W_l o_l(1 - o_l)x,\ l = 1, \ldots, L$$

- **Step 6** Cumulative cycle error is computed by adding the present error to E

$$E := E + \frac{1}{2}(y - O)^2$$

- **Step 7** If $k < K$ then $k := k+1$ and we continue the training by going back to **Step 2**, otherwise we go to **Step 8**

- **Step 8** The training cycle is completed. For $E < E_{\max}$ terminate the training session. If $E > E_{\max}$ then $E := 0$, $k := 1$ and we initiate a new training cycle by going back to **Step 3**

Exercise 1. *Derive the backpropagation learning rule*

with bipolar sigmoidal activation function

$$f(t) = \frac{2}{(1 + \exp -t) - 1}.$$

Effectivity of neural networks

Funahashi (1989) showed that infinitely large neural networks with a single hidden layer are capable of approximating all continuous functions. Namely, he proved the following theorem

Theorem 1. *Let $\phi(x)$ be a nonconstant, bounded and monotone increasing continuous function. Let $K \subset \mathbf{R}^n$ be a compact set and*

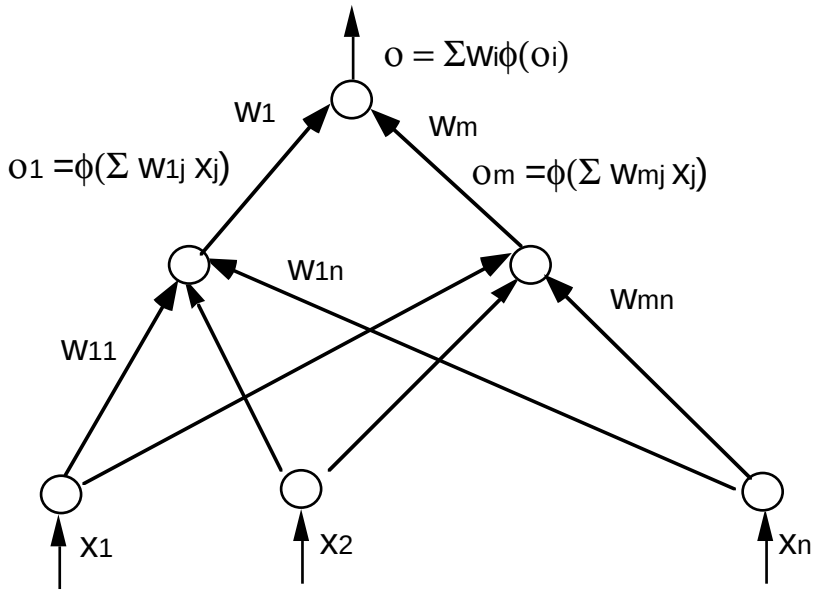
$$f : K \to \mathbf{R}$$

be a real-valued continuous function on K. Then for arbitrary $\epsilon > 0$, there exists an integer N and real constants w_i, w_{ij} such that

$$\tilde{f}(x_1, \ldots, x_n) = \sum_{i=1}^{N} w_i \phi(\sum_{j=1}^{n} w_{ij} x_j)$$

satisfies

$$\|f - \tilde{f}\|_\infty = \sup_{x \in K} |f(x) - \tilde{f}(x)| \leq \epsilon.$$



Funahashi's network.

In other words, any continuous mapping can be approximated in the sense of uniform topology on K by input-output mappings of two-layers networks whose output functions for the hidden layer are $\phi(x)$ and are linear for the output layer.

The Stone-Weierstrass theorem from classical real analysis can be used to show certain network architectures possess the universal approximation capability.

By employing the Stone-Weierstrass theorem in the designing of our networks, we also guarantee that the networks can compute certain polynomial expressions: if we are given networks exactly computing two functions, f_1 and f_2, then a larger network can exactly compute a polynomial expression of f_1 and f_2.

Theorem 2. *(Stone-Weierstrass) Let domain K be a compact space of n dimensions, and let $\mathcal{G}$ be a set of continuous real-valued functions on K, satisfying the following criteria:*

1. *The constant function $f(x) = \mathbf{1}$ is in $\mathcal{G}$.*

2. *For any two points $x_1 \neq x_2$ in K, there is an f in $\mathcal{G}$ such that $f(x_1) \neq f(x_2)$.*

3. *If f_1 and f_2 are two functions in $\mathcal{G}$, then fg and $\alpha_1 f_1 + \alpha_2 f_2$ are in $\mathcal{G}$ for any two real numbers α_1 and α_2.*

Then $\mathcal{G}$ is dense in $\mathcal{C}(K)$, the set of continuous real-valued functions on K. In other words, for any $\epsilon > 0$ and any function g in $\mathcal{C}(K)$, there exists g in $\mathcal{G}$ such that

$$\|f - g\|_\infty = \sup_{x \in K} |f(x) - g(x)| \leq \epsilon.$$

The key to satisfying he Stone-Weierstrass theorem is to find functions that *transform multiplication into addition* so that products can be written as summations.

There are at least three generic functions that accomplish this transfomation: exponential functions, partial fractions, and step functions.

The following networks satisfy the Stone-Weierstrass theorem.

- **Decaying-exponential networks**

  Exponential functions are basic to the process of transforming multiplication into addition in several kinds of networks:

$$\exp(x_1)\exp(x_2) = \exp(x_1 + x_2).$$

  Let $\mathcal{G}$ be the set of all continuous functions that can be computed by arbitrarily large decaying-exponential networks on domain

$$K = [0, 1]^n :$$

$$\mathcal{G} = \Bigg\{ f(x_1, \ldots, x_n)$$

$$= \sum_{i=1}^{N} w_i \exp(-\sum_{j=1}^{n} w_{ij} x_j), w_i, w_{ij} \in \mathbf{R} \Bigg\}.$$

Then $\mathcal{G}$ is dense in $\mathcal{C}(K)$

- **Fourier networks**
- **Exponentiated-function networks**
- **Modified logistic networks**
- **Modified sigma-pi and polynomial networks**

  Let $\mathcal{G}$ be the set of all continuous functions that can be computed by arbitrarily large modified sigma-pi or polynomial networks on domain

$$K = [0, 1]^n :$$

$$\mathcal{G} = \left\{ f(x_1, \ldots, x_n) = \sum_{i=1}^{N} w_i \prod_{j=1}^{n} x_j^{w_{ij}}, w_i, w_{ij} \in \mathbf{R} \right\}.$$

  Then $\mathcal{G}$ is dense in $\mathcal{C}(K)$.

- **Step functions and perceptron networks**
- **Partial fraction networks**