One-Layer Neural Networks

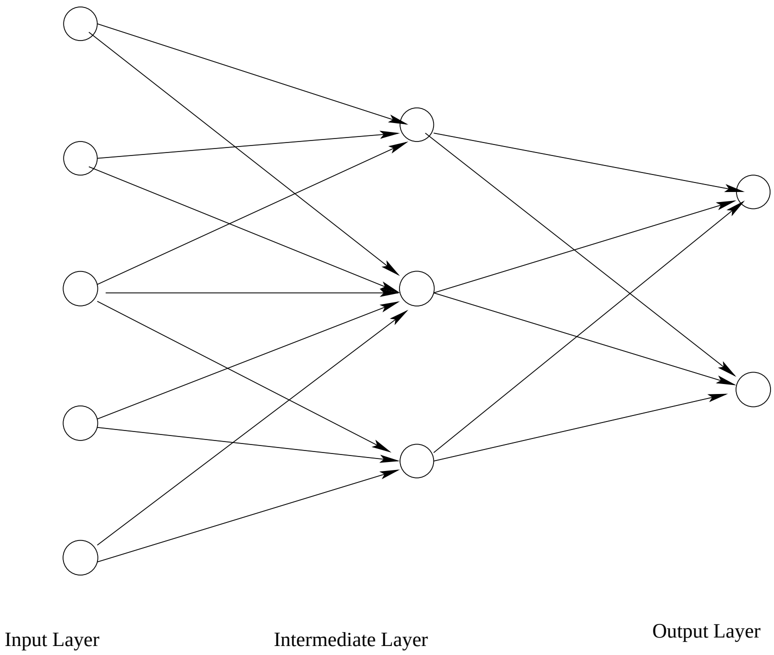

Figure 1: Single Layer Neural Network

Feed Forward Network

Definitions

Vertices

V^0 – Vertices in input layer
V^1 – Vertices in intermediate layer
V^2 – Vertices in output layer

Edges

$E^{0,1}$ – Edges between input and intermediate layer, i.e.,

$(i, j) \in E^{0,1} \Leftrightarrow$ edge between node i in V^0 and node j in V^1

$E^{1,2}$ – Edges between intermediate and output layer

Weights

$w^0_{i,j}$ – weight on edge (i, j) in $E^{0,1}$
$w^1_{i,j}$ – weight on edge (i, j) in $E^{1,2}$
b^1_j – bias on node j in V^1
b^2_j – bias on node j in V^2

Training Vectors

Input – $T^k, k = 1, \ldots, N$; T^k is a vector with components

$$x^0_i, \; i = 1, \ldots, |V^0|$$

Output – $C^k, k = 1, \ldots, N$; C^k is a vector with components

$$c_j, \; j = 1, \ldots, |V^2|$$

C^k represents the correct output for input T^k.

Computations

For a given input vector with components $x_i^0, i = 1, \ldots, |V^0|$ and specified weights and biases on the network, the following computations are made.

At intermediate node j, the input is

$$y_j^1 = \left(\sum_{(i,j) \in E^{0,1}} w_{i,j}^0 \; x_i^0 \right) + b_j^1.$$

and the output is

$$x_j^1 = f(y_j^1)$$

where f is the **intermediate layer transfer function**.

At output node j, the input is

$$y_j^2 = \left(\sum_{(i,j) \in E^{1,2}} w_{i,j}^1 \; x_i^1 \right) + b_j^2.$$

and the output is

$$x_j^2 = g(y_j^2)$$

where g is the **output layer transfer function**.

Thus the outputs

$$\mathbf{x}^2 = F(\mathbf{w}^0, \mathbf{w}^1, \mathbf{b}^1, \mathbf{b}^2)$$

are functions of $|E^{0,1}| + |E^{1,2}| + |V^0| + |V^1|$ parameters.

Transfer Functions

Examples of transfer functions:

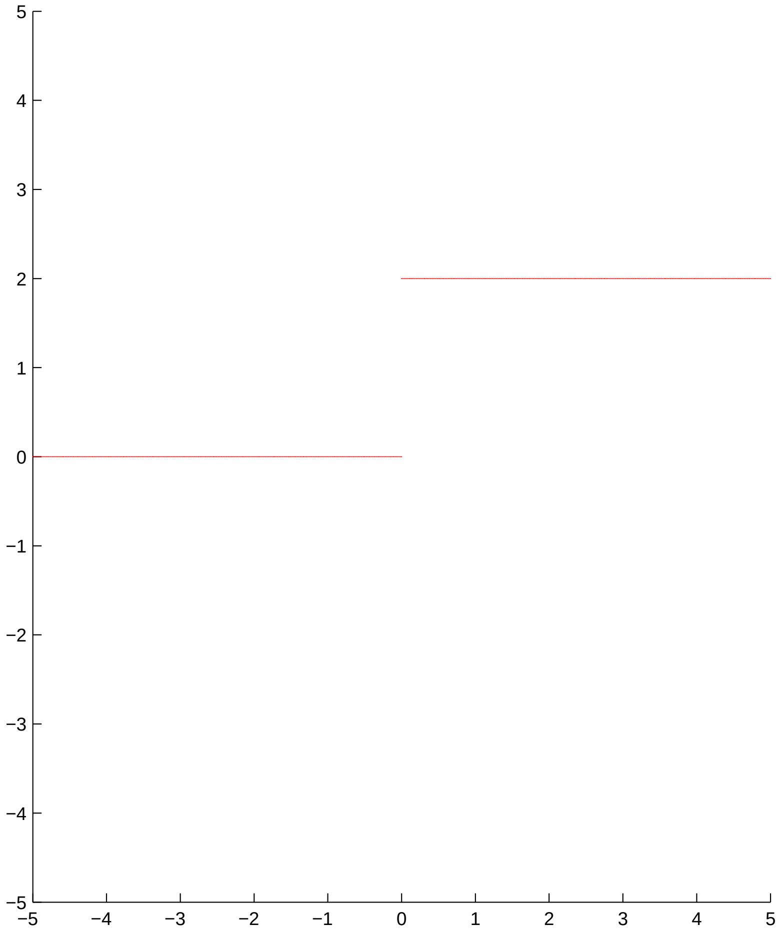

Figure 2: Step Transfer Function

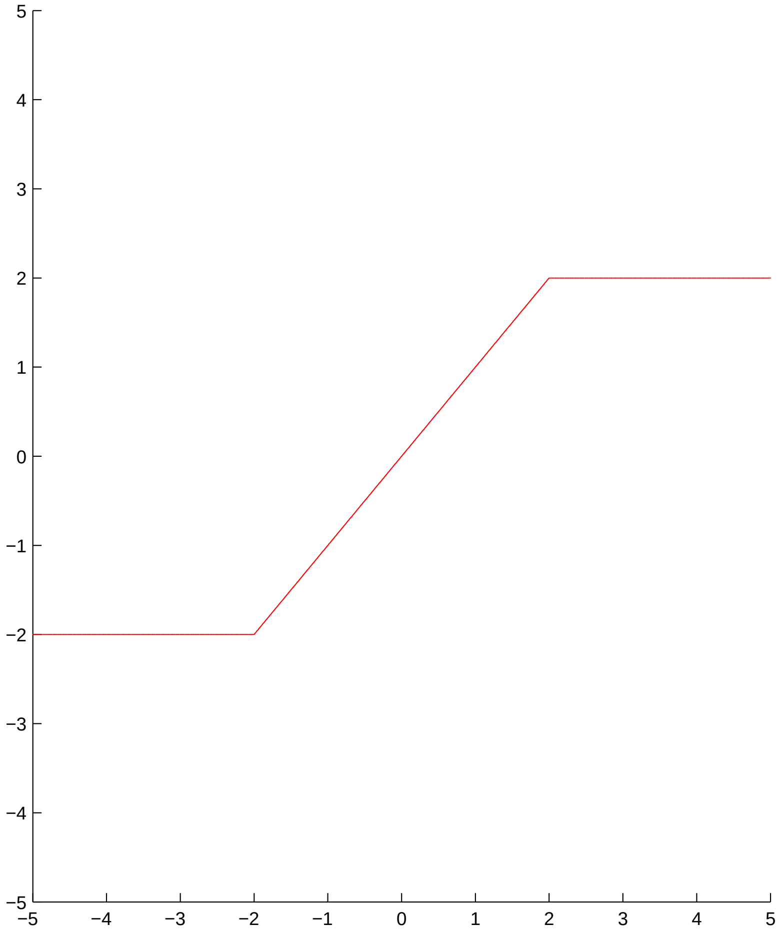

Figure 3: Ramp Transfer Function

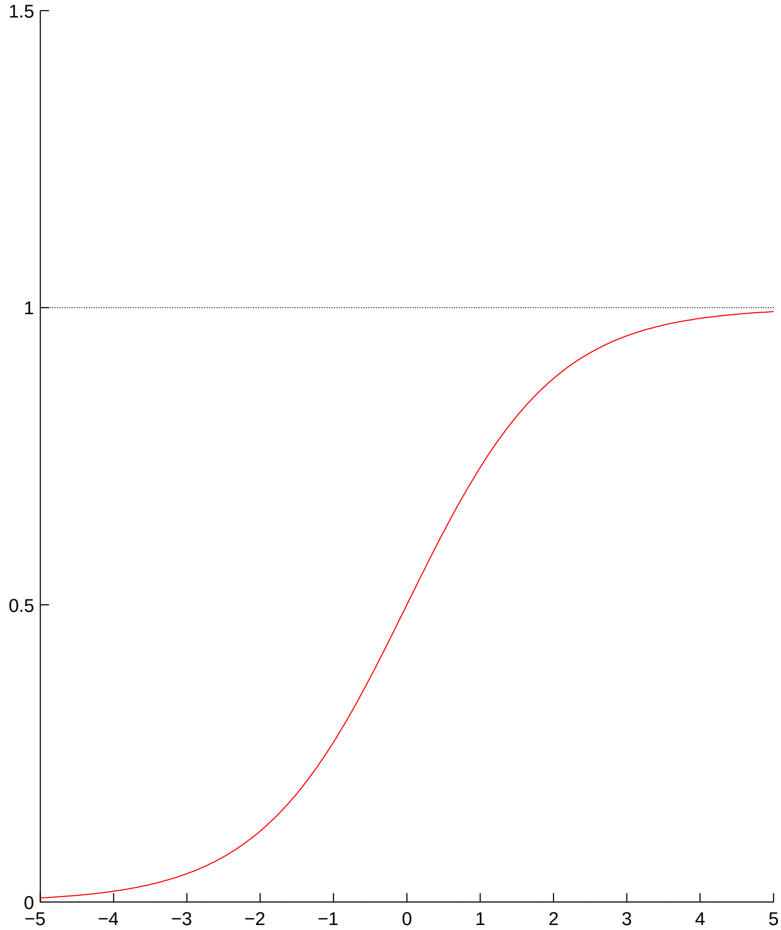

Figure 4: 0-1 Sigmoid Transfer Function

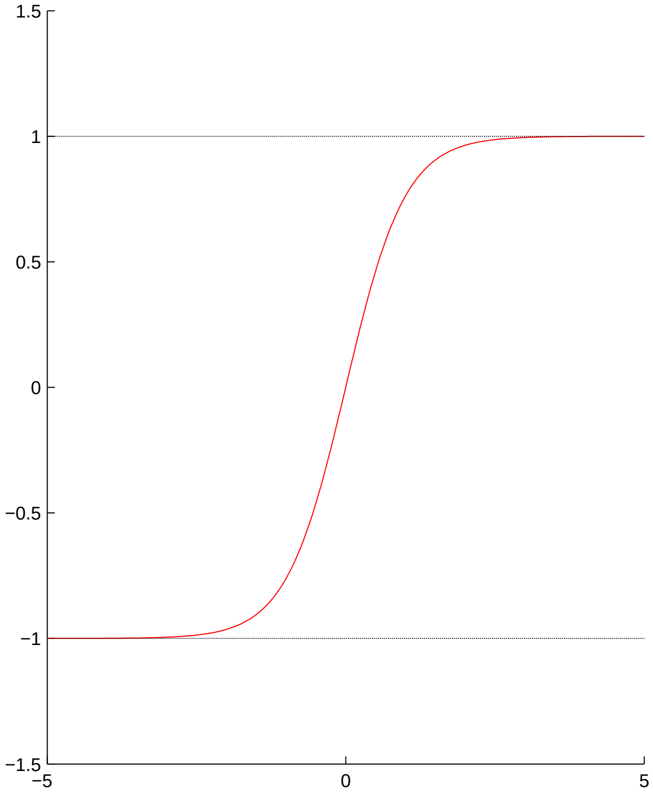

Figure 5: -1 - 1 Sigmoid Transfer Function

For the 0-1 sigmoid function

$$f(x) = \frac{1}{1 + e^x}$$

and

$$f'(x) = (1 - f(x))\, f(x)$$

For the -1 -1 sigmoid function

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{x^x + e^{-x}}$$

and

$$f'(x) = (1 - f(x))^2$$

The Back Propagation Algorithm

Gradient Minimization Methods

Definitions: Let $f : \Re^n \rightarrow \Re$ be a twice continuously differentiable function. The *gradient* of f at a point $\mathbf{x}^0$ is the vector of partial derivatives at $\mathbf{x}^0$:

$$\nabla f(\mathbf{x}^0) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(\mathbf{x}^0) \\ \vdots \\ \frac{\partial f}{\partial x_n}(\mathbf{x}^0) \end{bmatrix}$$

Definition: $\mathbf{x}^0$ is a **local minimum** of f if there is an $\epsilon > 0$ such that

$$f(\mathbf{x}) \geq f(\mathbf{x}^0)$$

for all $\mathbf{x}$ such that $|| \, \mathbf{x} - \mathbf{x}^0 \, || \leq \epsilon$.

Theorem: If $\mathbf{x}^0$ is a local minimum of f then $\nabla f(\mathbf{x}^0) = \mathbf{0}$.

Definition: $\mathbf{d} \in \Re^n$ is a **descent direction** at a point $\mathbf{x}^0$ if there is an $\alpha^* > 0$ such that

$$f(\mathbf{x}^0 + \alpha \, \mathbf{d}) < f(\mathbf{x}^0)$$

for all $\alpha \in (0, \alpha^*)$.

Theorem: $\mathbf{d}$ is a descent direction at $\mathbf{x}^0$ if

$$\nabla f(\mathbf{x}^0)^t\,\mathbf{d} = \sum_{j=1}^{n} \frac{\partial f}{\partial x_j}(\mathbf{x}^0)\, d_j < 0.$$

Theorem: $\mathbf{d} = -\nabla f(\mathbf{x}^0)$ is a descent direction of f at $\mathbf{x}^0$.

$$\nabla f(\mathbf{x}^0)^t\,(-\nabla f(\mathbf{x}^0)) = -\sum_{j} \frac{\partial f}{\partial x_j}(\mathbf{x}^0)^2$$

Back-tracking Descent Algorithm

1. Choose $\mathbf{x}^0$, and set $k = 0$.
2. Choose $\mathbf{d}^k = -\nabla f(\mathbf{x}^k)$, a descent direction of f at $\mathbf{x}^k$.
3. Find M, the smallest nonnegative integer such that

$$f(\mathbf{x}^k + 2^{-M}\,\mathbf{d}^k) < f(\mathbf{x}^k)$$

4. Set $\mathbf{x}^{k+1} = \mathbf{x}^k + 2^{-M}\,\mathbf{d}^k$
5. Set $k = k + 1$ and return to step (2).

Theorem: Suppose $\{\mathbf{x} : f(\mathbf{x}) < f(\mathbf{x}^0)\}$ is bounded and $\mathbf{x}^k$ is generated by the descent algorithm. If $\mathbf{x}^*$ is any limit point of $\{\mathbf{x}^k\}$ then $\nabla f(\mathbf{x}^*) = \mathbf{0}$.

Definition: If in the backtracking algorithm $\mathbf{d}^k$ is chosen as $-\nabla f(\mathbf{x}^k)$ then the algorithm is called the *steepest descent* algorithm.

Recall

$T^k: \ k = 1, \ldots, N$ – training sets.

$$T^k = [x_1^0, x_2^0, \ldots, x_p^0]$$

$C^k: \ k = 1, \ldots, N$ – correct responses for training sets.

$$C^k = [c_1, c_2, \ldots, C_q]$$

where $q = |V^2|$.

$E^{0,1}$ and $E^{1,2}$ – edges from input-to-intermediate nodes and from output-to-intermediate nodes.

$w_{i,j}^m, \ m = 0, 1$ – Weights for arcs.

$x_j^m, \ m = 0, 1, 2$ – Output flow from node j in layer m.

$b_j^m, \ m = 1, 2$ – Bias for node j in layer m.

$y_j^m, \ m = 1, 2$ – Activation at node j in layer m.

$$y_j^m = \sum \, [w_{i,j}^{m-1} \, x_j^{i-1}] + b_j^m$$

$f(x), g(x)$ – Transfer functions;

$$\begin{array}{rcl} x_j^1 & = & f(y_j^1), \quad \text{intermediate,} \\ & = & \\ x_j^2 & = & g(y_j^2), \quad \text{output} \end{array}$$

Back Propagation Algorithm

In the following algorithm we will use the (0,1) sigmoid function f for the intermediate layer transfer function and the identity for the output transfer function.

Note that the output function at node $j, j = 1, \ldots, q$ can be written

$$\begin{aligned} y_j^2 &= \left(\sum \, [w_{i,j}^1 * x_i^1]\right) + b_j^2 \\ &= \left(\sum \; w_{i,j}^1 \; f([\sum \; w_{s,i}^0 \, x_s^0] + b_i^1 \,)\right) + b_m^2 \end{aligned}$$

Thus y_m^2 is a function of the weights and biases in the model.

Given an input $\mathbf{x}^0 = T^k$ for some training vector k we can compute the output for each $j, m = 1, \ldots, |V^2|$ and we obtain the (squared) error for this input

$$\mathcal{E}^k = [\,(c_1 - g(y_1^2))^2 + (c_2 - g(y_2^2))^2 + \cdots + (c_q - g(y_q^2))^2$$

where $q = |V^2|$

A *steepest descent* step for decreasing this error as a function of the weights and biases is a step of the form

$$w_{i,j}^m \leftarrow w_{i,j}^m - \gamma \frac{\partial \mathcal{E}^k}{\partial w_{i,j}^m}, \quad m = 0, 1$$

and

$$b_j^m = \leftarrow b_j^m - \gamma \frac{\partial \mathcal{E}^k}{\partial b_j^m}, \quad m = 1, 2$$

for all $w_{i,j}^m$ and b_j^m that are involved in the calculation of $\mathcal{E}^k$. The positive *step length parameter* γ is chosen so that the change actually decreases $\mathcal{E}^k$; which means that it is generally restricted to be small.

To change the weights according to the above formula requires the computation of the partial derivatives of $\mathcal{E}^k$. The partial derivatives with respect to biases on the output node are easy because only one of them is involved, i.e.,

$$\begin{aligned}\frac{\partial \mathcal{E}^k}{\partial b_j^2} &= -2 \sum_{n=1}^{q} (c_n - g(y_n^2))\, g'(y_n^2)\, \frac{\partial y_n^2}{\partial b_j^2} \\ &= -2\,(c_j - g(y_j^2))\, g'(y_j^2)\end{aligned}$$

where $q = |V^2|$.

The partial derivatives with respect to the weights from the intermediate layer to the output layer are also easily computed. If $(i,j) \in E^{1,2}$ then

$$\frac{\partial \mathcal{E}^k}{\partial w_{i,j}^1} = -2\,(c_j - g(y_j^2))\, x_i^1\, g'(y_j^2)$$

The partial derivatives with respect to the biases and on the weights from the input nodes to the hidden nodes depend upon the derivative of the function f and the previously computed changes. If $(i,j) \in E^{0,1}$ then

$$\begin{aligned}\frac{\partial \mathcal{E}^k}{\partial w_{i,j}^0} &= -2 \sum_{n=1}^{q} (c_n - g(y_n^2))\, \frac{\partial y_n^2}{\partial w_{i,j}^0}\, g'(y_n^2) \\ &= -2 \sum_{n=1}^{q} (c_n - g(y_n^2))\, g'(y_n^2)\, f'(y_m^1)\, \frac{\partial y_m^1}{\partial w_{i,j}^0} \\ &= -2 \sum_{\{n:(j,n)\in E^{1,2}\}} (c_n - g(y_n^2))\, g'(y^2 n)\, f'(y_j^1)\, x_i^0\end{aligned}$$

Similarly,

$$\frac{\partial \mathcal{E}^k}{\partial b_j^1} = -\,2 \sum_{(j,n)\in V^1} (c_n - g(y_n^2))\, f'(y_j^1)\, g'(y^1, n)\, x_n^0$$

The computations now depend on the type of transfer function to be used. Rather than do all of them we will assume that the the 0-1 sigmoid is used for both. Then

$$g' = g\,(1-g) \quad \text{and} \quad f' = f\,(1-f)$$

So the formulas become

$$\frac{\partial \mathcal{E}^k}{\partial w^1_{i,j}} = -\,2\,(c_j - x^2_j)\,x^2_j\,(1-x^2_j)\,x^1_i$$

$$\frac{\partial \mathcal{E}^k}{\partial w^0_{i,j}} = -\,2 \sum_{\{n:(j,n)\in E^{1,2}\}} (c_n - x^2_n))\,x^2_n\,(1-x^2_n)\,x^1_j\,(1-x^1_j)\,x^0_i$$

$$\frac{\partial \mathcal{E}^k}{\partial b^2_j} = -\,2\,(c_j - x^2_j)\,x^2_j\,(1-x^2_j)$$

$$\frac{\partial \mathcal{E}^k}{\partial b^1_j} = -\,2 \sum_{(j,n)\in V^1} (c_n - x^2_n)\,x^1_j\,(1-x^1_j))\,x^1_n\,(1-x^1_n)\,x^0_n$$

If we denote

$$\delta_n^2 = (c_n - y_n^2)\, x_n^2\, (1 - x_n^2)$$

then the partial derivatives have the form

$$\begin{aligned}
\frac{\partial \mathcal{E}^k}{\partial b_j^2} &= -\,2\,\delta_j^2 \\
\frac{\partial \mathcal{E}^k}{\partial w_{i,j}^1} &= -\,2\,\delta_j^2\, x_i^1 \\
\frac{\partial \mathcal{E}^k}{\partial b_j^1} &= -\,2 \sum_{(j,n)\in V^1} \delta_n^2\, x_j^1\, (1 - x_j^1) \\
\frac{\partial \mathcal{E}^k}{\partial w_{j,i}^0} &= -\,2 \sum_{\{n:(j,n)\in E^{1,2}\}} \delta_n^2\, x_j^1 * (1 - x_j^1) * x_i^0.
\end{aligned}$$

The basic single step one-layer back propagation algorithm

1. Choose an initial set of weights and biases and a sequence of **step sizes** ρ_t. Set $t = 1$.

2. Choose a training vector from the training set and determine the initial flows and correct responses

$$x_i^0,\ i = 1, \ldots, p.$$

   and

$$C = [C_1, \ldots, C_q]$$

3. Compute the activations at the intermediate nodes for $n = 1, \ldots, s$

$$y_n^1 = [w_{i,n}^0 * x_i^0] + b_n^1$$

4. Compute the flows from the intermediate nodes according to

$$x_n^1 = f(y_n^1),\ n = 1, \ldots, s$$

5. Compute the activations and responses at the output nodes for $j = 1, \ldots, q$

$$y_j^2 = \sum_{\{n:W_{n,j}^1 \neq 0\}} [w_{n,j}^1 * x_n^1] + b_j^2$$

6. Compute the responses at the output nodes according to
$$x_j^2 = g(y_j^2), \ j = 1, \ldots, q$$

7. Compute the change factors, δ_j^2, for each output node,

8. Compute the change factors for each intermediate node, $n = 1, \ldots, s$
$$\delta_n^1 = \sum \delta_j^2 * x_n^1 * (1 - x_n^1)$$

9. Change the intermediate-to-output weights and output biases by the rules
$$\begin{aligned} w_{n,j}^1 &= w_{n,j}^1 + \rho_t * \delta_j^2 * x_n^1, \ j = 1, \ldots, q, \\ &= \text{if } W_(^1 n, j) \neq 0 \\ b_j^2 &= b_j^2 + \rho_t * \delta_j^2, \ j = 1, \ldots, q \end{aligned}$$

10. Change the input-to-intermediate weights and intermediate biases by the rules
$$w_{i,n}^0 = w_{i,n}^0 + \rho_t * \delta_n^1 * x_i^0$$
and
$$b_n^1 = b_n^1 + \rho_t * \delta_n^1,$$

11. Set $t = t + 1$ and return to Step 2.