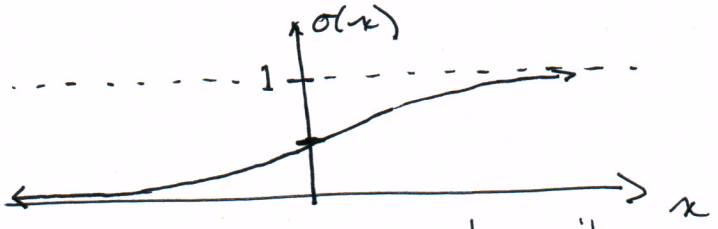
Neural Network Training With a Sigmoid

The sigmoid equation is often used as a transfer function between neurodes. It is similar to a step function, but continuous and differentiable.

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



"Sigmoid Function"

One useful property is the simplicity of computing a derivative.

$$\frac{d}{dx}\sigma(x) = \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right)$$

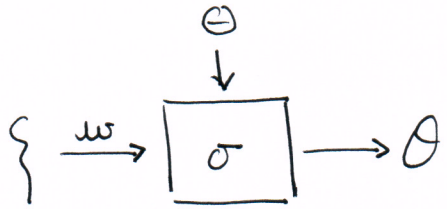
$$= \frac{e^{-x}}{(1+e^{-x})^2}$$

$$= \frac{(1+e^{-x})-1}{(1+e^{-x})^2}$$

$$= \frac{1+e^{-x}}{(1+e^{-x})^2} - \frac{1}{(1+e^{-x})^2}$$

$$= \sigma(x) - \sigma^2(x)$$

This simple form of the derivative will be useful later.



w : weight
Θ : bias

"Single Neurode"

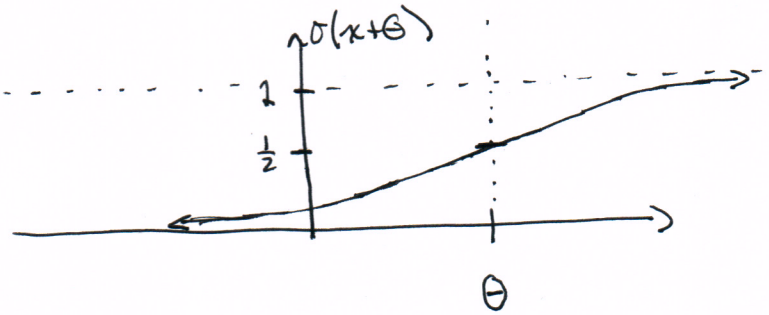
This figure represents a single neurode with a single input. The equation defining the figure is

$$\theta = \sigma(\xi w)$$

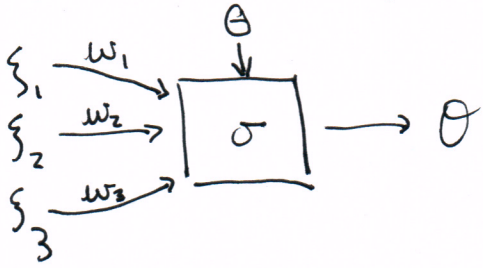
If there is a bias term Θ, the equation is

$$\theta = \sigma(\xi w + \Theta)$$

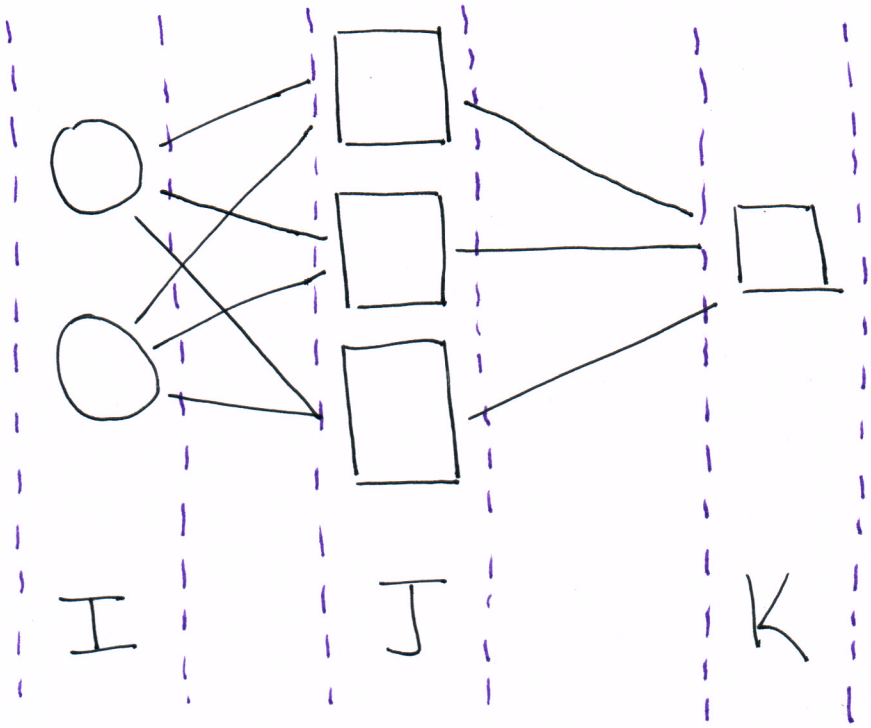
Step functions have a "threshold" (i.e. the "step"). The equivalent of this threshold for a sigmoid is Θ



Consider a Neurode w/ Multiple Inputs:



$$\theta = \sigma(w_1 \xi_1 + w_2 \xi_2 + w_3 \xi_3 + \Theta)$$



I: Input Layer

J: Hidden Layer

K: Output Layer

"Neural Network"

Notation:

x_j^l = Input to node j of layer l

W_{ij}^l = Weight from layer $l-1$ node i to layer l node j

$\sigma(x) = \frac{1}{1+e^{-x}}$ Sigmoid Transfer Function

θ_j^l = Bias of node j in layer l

Θ_j^l = Output of node j in layer l

t_j = Target value of node j of the output layer.

We provide an input, compute the output, and tell the network what we want the output to be (i.e. these t_j values). Then we use the differences between them to compute a Δ and propagate that backwards through the network to adjust it. ("train it")

The error calculation:

Given a set of training data points t_j and output layer o_j, we write the "error" as

$$E = \frac{1}{2} \sum_{k \in K} \left(o_k - t_k \right)^2$$

This measures the distance between the outputs and the desired training values summed over all output layers K. In other words, we let the error of the network for a single training iteration be denoted by E.

We want to calculate $\frac{\partial E}{\partial w_{ij}^{\ell}}$, the rate of change of the error with respect to a given connective weight so that we can minimize it. We are computing the gradient on E to iteratively minimize it

Now we consider two cases: (I) the node is an output node; (II) the node is in a hidden layer.

I. Output Node

Note that the following sum $\sum_{k \in K}$ disappears because $\frac{\partial}{\partial w_{jk}}$ is performed with respect to a specific weight — that of $k \in K$ — so all the rest are constant with respect to that weight and are zero.

$$\frac{\partial E}{\partial W_{jk}} = \frac{\partial}{\partial W_{jk}} \frac{1}{2} \sum_{k \in K} (\theta_k - t_k)^2$$

{ j: Hidden Layer Element
k: Output Layer Elemen

$$= (\theta_k - t_k) \frac{\partial}{\partial W_{jk}} \theta_k \qquad \text{(Chain Rule)}$$

$$= (\theta_k - t_k) \frac{\partial}{\partial W_{jk}} \sigma(x_k) \qquad \text{(from output definition)}$$

$$= (\theta_k - t_k)\, \sigma(x_k)(1 - \sigma(x_k)) \frac{\partial}{\partial W_{jk}} x_k$$

From the derivative on the 1st page and chain rule

Considering $\frac{\partial}{\partial W_{jk}} x_k$, x_k is the output of the j^{th} node times the weight from j to k, and so this derivative is $\frac{\partial}{\partial W_{jk}} (W_{jk}\, \sigma(x_j)) = \sigma(x_j) = \theta_j$.

$$\Rightarrow \frac{\partial E}{\partial W_{jk}} = (\theta_k - t_k)\, \theta_k (1 - \theta_k)\, \theta_j$$

Define $\delta_k \equiv (\theta_k - t_k)\, \theta_k (1 - \theta_k)$ to condense the previous expression.

$$\frac{\partial E}{\partial W_{jk}} = \theta_j \delta_k$$

Recall that we are in the output layer here and this t_k value is what we're training the neural network on. We will use δ_k to propagate the error function back through the network.

II. Hidden Layer

Now we see how to compute the back-propagation if we're on a hidden layer node. This time, we take the derivative with respect to the weight from i to j, where $i \in I$ is an element of the input layer and $j \in J$ is an element of a hidden layer.

$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial}{\partial W_{ij}} \frac{1}{2} \sum_{k \in K} (\theta_k - t_k)^2$$

Note that the weight from i to j has nothing to do directly with subscript k, but the output of node j is input for every node in layer k. Thus, we cannot drop the summation.

$$\frac{\partial E}{\partial W_{ij}} = \sum_{k \in K} (o_k - t_k) \frac{\partial}{\partial W_{ij}} o_k$$

$$= \sum_{k \in K} (o_k - t_k) \frac{\partial}{\partial W_{ij}} \sigma(x_k)$$

$$= \sum_{k \in K} (o_k - t_k) \sigma(x_k)(1 - \sigma(x_k)) \frac{\partial x_k}{\partial W_{ij}}$$

In this line, $\frac{\partial x_k}{\partial W_{ij}}$ is the problem term. Every input node k depends on output from node j. Further, the output of node j depends on the weight from i to j (W_{ij}).

$$\frac{\partial E}{\partial W_{ij}} = \sum_{k \in K} (o_k - t_k) o_k (1 - o_k) \frac{\partial x_k}{\partial o_j} \frac{\partial o_j}{\partial W_{ij}}$$

Note two things:

- $\frac{\partial x_k}{\partial o_j}$: the input to the k^{th} node with respect to the output of the j^{th} node is just the product of the output of the j^{th} node and the weight from j to k (i.e. W_{jk}).
- $\frac{\partial o_j}{\partial W_{ij}}$: This derivative no longer contains subscript k and so is no longer a part of the sum.

$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial \theta_j}{\partial W_{ij}} \sum_{k \in K} \underbrace{(\theta_k - t_k)\theta_k(1-\theta_k)}_{\delta_k} W_{jk}$$

We can again use δ_k, multiply by the weight W_{jk} that connects it to the previous layer j, add them all up (the summation over k), then move it forward.

$$\frac{\partial E}{\partial W_{ij}} = \theta_j(1-\theta_j)\frac{\partial x_j}{\partial W_{ij}} \sum_{k \in K} (\theta_k - t_k)\theta_k(1-\theta_k)W_{jk}$$

$$= \theta_j(1-\theta_j)\theta_i \sum_{k \in K} (\theta_k - t_k)\theta_k(1-\theta_k)W_{jk}$$

Now use our definition for δ_k

$$\frac{\partial E}{\partial W_{ij}} = \theta_i \theta_j (1-\theta_j) \sum_{k \in K} \delta_k W_{jk}$$

Similar to before, we will now define all terms besides the θ_i to be $\delta_j \equiv \theta_j(1-\theta_j)\sum_{k \in K} \delta_k W_{jk}$

$$\frac{\partial E}{\partial W_{ij}} = \theta_i \delta_j$$

Notice how δ_j has a similar form to δ_k even as δ_j contains δ_k. Further, for that reason, δ_j acquires the difference training terms $(\theta_k - t_k)$.

How Weights Affect the Error Terms

For an output layer node $k \in K$

$$\frac{\partial E}{\partial W_{jk}} = o_j \delta_k \quad \text{where} \quad \delta_k = o_k(1-o_k)(o_k - t_k)$$

For a hidden layer node $j \in J$

$$\frac{\partial E}{\partial W_{ij}} = o_i \delta_i \quad \text{where} \quad \delta_j = o_j(1-o_j) \sum_{k \in K} \delta_k W_{jk}$$

Bias Term:

If we incorporate the bias term Θ into the equation, we find

$$\frac{\partial \Theta}{\partial \Theta} = 1$$

This is why we view the bias term as output from a node which is always one. This holds for any layer l (output or hidden). Further, substituting this through the previous procedure yields

$$\frac{\partial E}{\partial \Theta} = \delta_k$$

The Back-Propagation Algorithm

1. Run the network forward with your input data to get the network output.
2. For each output node, compute

$$\delta_k = \theta_k(1-\theta_k)(\theta_k - t_k)$$

3. Working backwards through the hidden layers, compute

$$\delta_j = \theta_j(1-\theta_j)\sum_{k\in K}\delta_k W_{jk}$$

4. Update the weights and biases

$$\text{Given}\begin{cases}\Delta W = -\eta\delta_l\theta_{l-1}\\ \Delta\Theta = -\eta\delta_l\end{cases}$$

$$\text{Apply}\begin{cases}W+\Delta W \to W\\ \Theta+\Delta\Theta\to\Theta\end{cases}$$

where η is just some small positive number that sets the scale for the step sizes we will train by. These terms both have a negative sign because we are computing the gradient of the error function E with respect to all these weights, which points in the direction of steepest ascent in this k-dim space, but we want to minimize E, so we attach the negative sign to move in the opposite direction.