- For nonseparable data, single layer perceptron model is not going to do the job.
- Example: Data $(0,0), -1, (0,1), +1, (1,0), +1, (1,1), -1$ (This is the Boolean exclusive OR function).
- Suppose that there is some learning parameters: w_0, w_1, w_2 for this model:

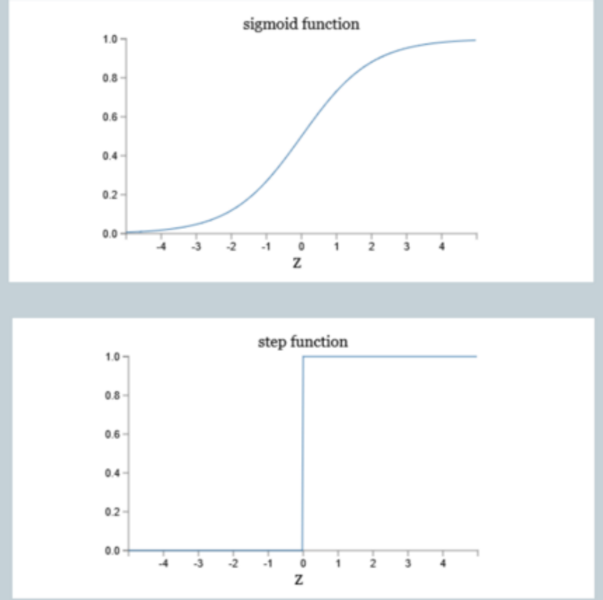
$$\begin{aligned} w_0 &< 0 \\ w_0 + w_2 &> 0 \\ w_0 + w_1 &> 0 \\ w_0 + w_1 + w_2 &> 0. \end{aligned}$$

- This system of inequalities does not have a solution.

- Perceptron and Adaline use the activation function g(z)=sign(z) or the thresh-hold function.
- These function are not differentiable, hard for computation.
- Sigmoid neurons: uses a "smooth" step activation function that returns a number between 0 and 1.

$$g(\langle w, x\rangle) = \frac{1}{1 + e^{-\langle w,x\rangle}}$$

- Sigmoid neurons correspond to the input-output mapping similar to logistic regression.


sigmoid function
1.0
0.8
0.6
0.4
0.2
0.0
-4
-3
-2
-1
0
1
2
3
4
Z
step function
1.0
0.8
0.6
0.4
0.2
0.0
-4
-3
-2
-1
0
1
2
3
4
Z

- Hyperbolic tangent activation function:

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

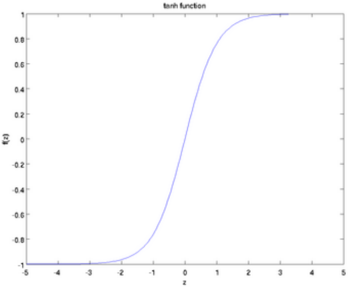
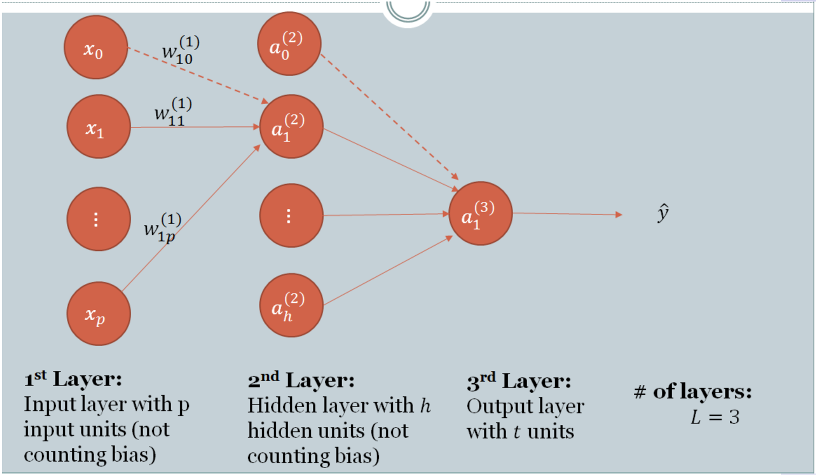
- 

Figure: ufldl.stanford.edu

- Multi-layer network: stringing many layers of neurons together.
- First layer: input layer, last layer: output layer, layers in between: hidden layers.
- Notation: Input coming out of layer l node i: $x_i^{(l)}$.
- Wieght $w_{ij}^{(l)}$: weight of the link between node i of layer l-1 and node j of layer l.
- $s_i^{(l)}$: signal coming into node i of layer l:

$$s_i^{(l)} = \sum_{j-=1}^{p} w_{ji}^{(l)} x_j^{(l-1)},$$

  where p is the number of nodes in layer l-1.
- Except of the first layer: $x_i^{(l)} = g(s_i^{(l)})$, where g is the activation function.


x_0
x_1
$\vdots$
x_p
$w_{10}^{(1)}$
$w_{11}^{(1)}$
$w_{1p}^{(1)}$
$a_0^{(2)}$
$a_1^{(2)}$
$\vdots$
$a_h^{(2)}$
$a_1^{(3)}$
$\hat{y}$
1st Layer:
Input layer with p input units (not counting bias)
2nd Layer:
Hidden layer with h hidden units (not counting bias)
3rd Layer:
Output layer with t units
# of layers:
$L = 3$

- Forward feed: Given all the weights of the connection link in the network, we can feed input pair (x,y) into the network as following:
- Second layer: $s_1^{(2)} = \sum_{j=1}^{p} w_{j1}^{(2)} x_j^{(1)}$, $s_2^{(2)} = \sum_{j=1}^{p} w_{j2}^{(2)} x_j^{(1)}, \cdots \; x_1^{(2)} = g(s_1^{(2)}, x_2^{(2)} = g(s_2^{(2)}$
- 3rd layer: $s_1^{(3)} = \sum_{j=1}^{p} w_{j1}^{(3)} x_j^{(2)}, x_1^{(3)} = g(s_1^{(3)}) \rightarrow output$ (denote by o)
- Given the training data, we need to find the parameters w that minimize the difference between the output o and the response y.

$$J(w) = (y - o)^2.$$

To avoid overfitting,

$$J(w) = (y - o)^2 + \lambda \|w\|^2.$$

- Backward propagation: The objective function, due to the layers of neurons, is non-convex.
- Gradient descent method with very small step length can achieve a local minima solution.
- Need to evaluate : $\frac{\partial J(w)}{\partial w_{ij}^{(l)}} = \frac{\partial J(w)}{\partial s_j^{(l)}} \frac{\partial s_j^{(l)}}{\partial w_{ij}^{(l)}}$
- Since $s_j^{(l)} = \sum_i w_{ij}^{(l)} x_i^{(l-1)}$, we have $\frac{\partial s_j^{(l)}}{\partial w_{ij}^{(l)}} = x_i^{(l-1)}$
- Denote $\delta_i^{(l)} = \frac{\partial J(w)}{\partial s_i^{(l)}}$

- It can be shown that:

$$\delta_j^{(l-1)} = \frac{\partial J(w)}{\partial s_i^{(l-1)}}$$
$$= \sum_j \frac{\partial J(w)}{\partial s_j^{(l)}} \frac{\partial s_j^{(l)}}{\partial x_i^{(l-1)}} \frac{\partial x_i^{(l-1)}}{\partial s_j^{(l-1)}}$$
$$= \sum_j \delta_j^{(l)} w_{ij}^{(l)} g'(s_j^{(l-1)})$$

- Using the relation above we can calculate any $\frac{\partial J(w)}{\partial w_{ij}^{(l)}}$.
- Once the gradient is evaluated, we can use gradient descent method.

- Stochastic gradient descent method: When the data set has many observation in the training (can be millions of them), calculating such gradient might take a long time.
- A popular choice is using an online version of gradient descent method: stochastic gradient descent.
- Instead of calculating the gradient based on the sum of the accumulated gradient for each observation x_i:

$$w = w - \nu \sum_{i=1}^{n} \frac{\partial J(w, x_i)}{\partial w}$$

- We can update the weight w using a smaller number of observation chosen at random (or you can make a pass through all observation one by one)

$$w = w - \nu \frac{\partial J(w, x_i)}{\partial w}$$

- Stochastic gradient descent converges slower than the full version, but we save the time from computing the full gradient.