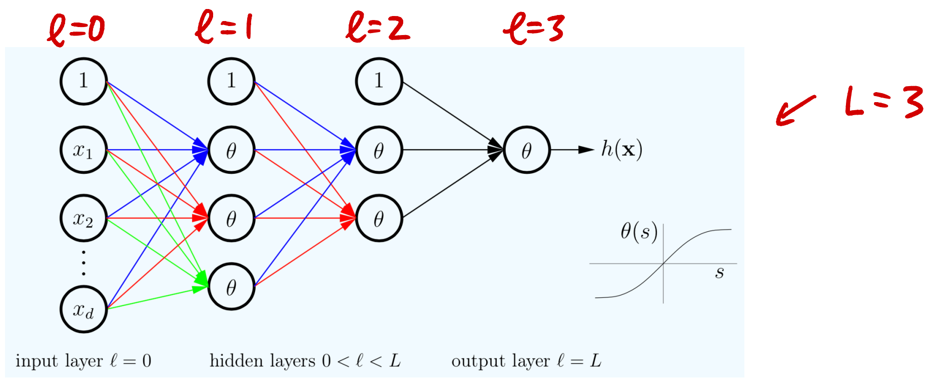
Neural Networks: Forward Propagation[1]

The neural network is our ‘softened’ Multilayer Perceptron (MLP). At the highest level, the algorithm consists of three simple steps. First, present one or more training examples to the neural network. Second, compare the output of the neural network to the desired value. Finally, adjust the weights to make the output get closer to the desired value. It is as simple as that! It is exactly what we did in the perceptron learning algorithm, and we used gradient descent to determine how much to adjust the weights.

Let’s begin with a graph representation of a feed-forward neural network.



While this graphical view is aesthetic and intuitive, with information *flowing* from the inputs on the far left, along links and through hidden nodes, ultimately to the output $h(\mathbf{x})$ on the far right, it will be necessary to algorithmically describe the function being computed. Things are going to get messy, and this calls for a very systematic notation; bear with me.

1. Notation

There are layers labeled by $l = 0, 1, 2, \cdots, L$. In our example above, $L = 3$, i.e. we have three layers (the input layer $l = 0$ is usually not considered a layer and is meant for feeding in the inputs). The layer $l = L$ is the output layer, which determines the value of the function. The layers in between, $0 < l < L$, are the hidden layers.

We will use superscript$^{(l)}$ to refer to a particular layer. Each layer l has ‘dimension’ $d^{(l)}$, which means that it has $d^{(l)} + 1$ nodes, labeled $0, 1, 2, \ldots, d^{(l)}$. Every layer has one special node, which is called the bias node (labeled 0).

$d^{(0)}+1$

[1] Major contents are adapted from dynamic e-chapters of Abu-Mostafa, Y S, Magdon-Ismail, M., Lin, H-T (2012) *Learning from Data*, AMLbook.com.

Every arrow represents a **weight** from a node in a layer to a node in the next higher layer. Notice that the bias nodes have no incoming weights. A node with an incoming weight indicates that some signal is fed into this node. Every such node with an input has an activation function θ.

Activation Function θ

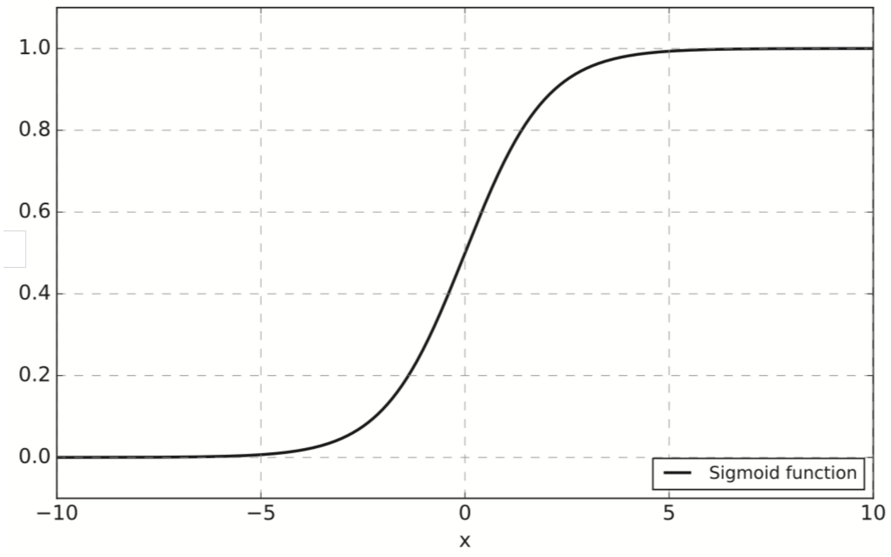
In choosing activation function, we often use a soft threshold or sigmoid in neural networks. A version of sigmoid is defined as:

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

Or sometimes express as:

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

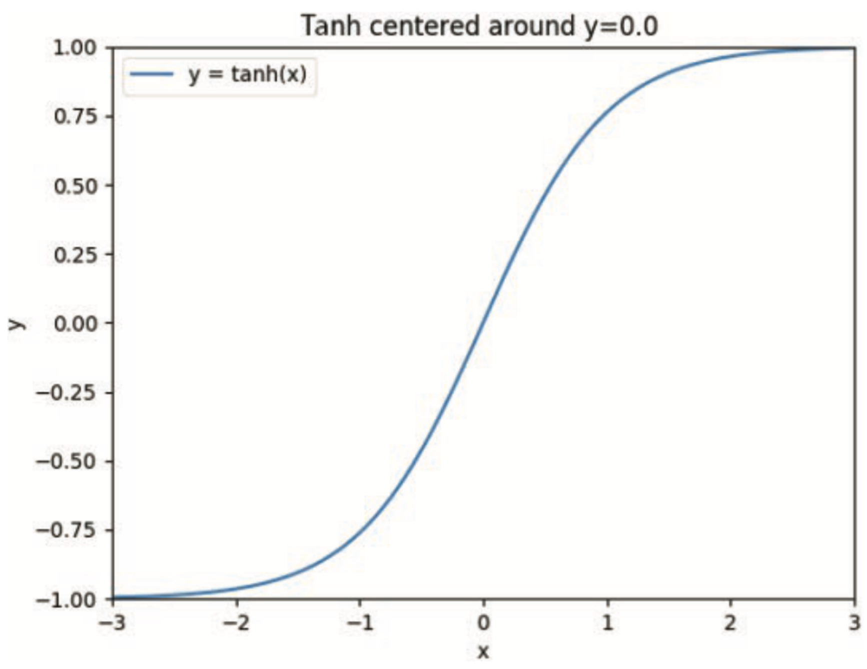
A plot of this function is given in the figure below.



Remark: The sigmoid function maps x from $(-\infty, +\infty)$ to y in the range of $[0, 1]$. It is a soft version of step function we used for perceptron.

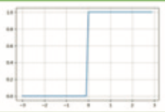
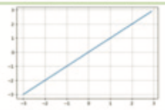
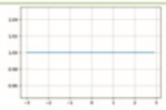
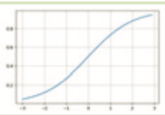
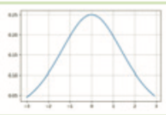
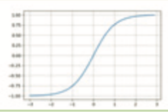
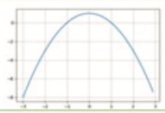
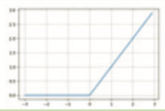
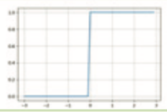
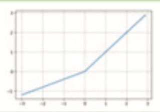
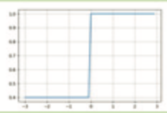
Another popular soft threshold is the hyperbolic tangent

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



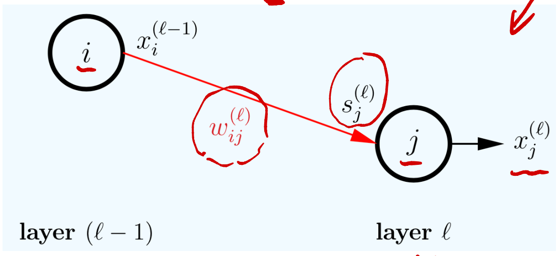
The hyperbolic tangent function maps x from $(-\infty,+\infty)$ to y in the range of $[-1,1]$. It is a soft version of bipolar step function we used for perceptron.

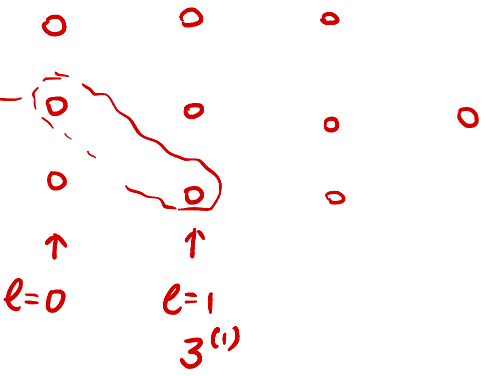
The following table lists a few common activation functions.

Name	Equation	Derivative	1-D Graph	1-D Graph(derivative)
Binary Step	$\sigma(x)=\begin{cases}1, & x>0\\ 0.5, & x=0\\ 0, & x<0\end{cases}$	$\sigma'(x)=\begin{cases}0, & x\neq 0\\ ?, & x=0\end{cases}$		
Identity	$\sigma(x)=x$	$\sigma'(x)=1$		
Sigmoid	$\sigma(x)=\dfrac{1}{1+e^{-x}}$	$\sigma'(x)$ $=\sigma(x)(1-\sigma(x))$		
Tanh	$\sigma(x)=\dfrac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	$\sigma'(x)=1-\sigma(x)^2$		
Rectified Linear (ReLU)	$\sigma(x)=\max(0,x)$	$\sigma'(x)=\begin{cases}1, & x\geq 0\\ 0, & x<0\end{cases}$		
Leaky ReLU	$\sigma(x)=\begin{cases}x, & x\geq 0\\ \alpha x, & x<0\end{cases}$	$\sigma'(x)=\begin{cases}1, & x\geq 0\\ \alpha, & x<0\end{cases}$		

The neural network model $\mathcal{H}$ is specified once we determine the architecture of the neural network, that is the dimension of each layer $\mathbf{d} = [d^{(0)}, d^{(1)}, d^{(2)} \dots, , d^{(L)}]$ (L is the number of layers). A hypothesis $h \in \mathcal{H}$ is specified by selecting weights for the links. Let's zoom into a node in hidden layer l, to see what weights need to be specified.

$x_i^{(l-1)}$

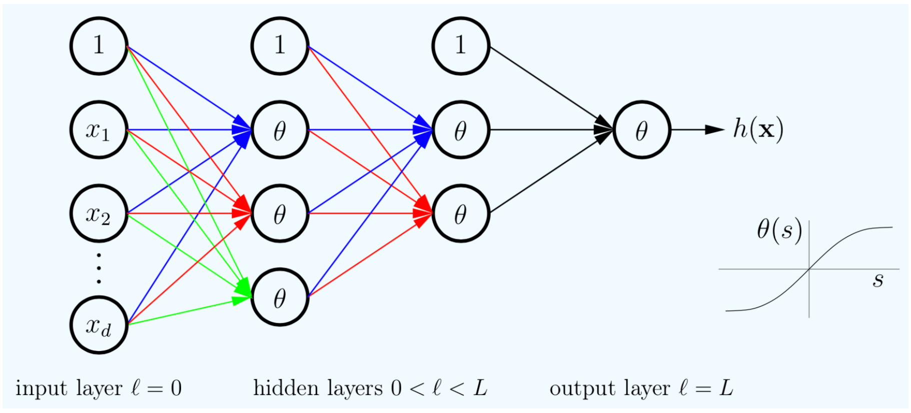




A node has an incoming signal s and an output x. The weights on links into the node from the previous layer are $w^{(l)}$, so the weights are indexed by the layer into which they go. Thus, the output of the nodes in layer $l-1$ is multiplied by weights $w^{(l)}$. We use subscripts to index the nodes in a layer. So, $w_{ij}^{(l)}$ is the weight into node j in layer l from node i in the previous layer, the signal going into node j in layer l is $s_j^{(l)}$, and the output of this node is $x_j^{(l)}$.

Remark: There are some special nodes in the network.

1. The zero nodes in every layer are constant nodes, set to output 1. They have no incoming weight, but they have an outgoing weight (bias).
2. The nodes in the input layer $l = 0$ are for the input values, and have no incoming weight or activation function.



For the most part, we only need to deal with the network on a layer by layer basis, so we introduce vector and matrix notation for that. We collect all the input signals to nodes $1, 2, \ldots, d^{(l)}$ in layer l in the vector $\mathbf{s}^{(l)}$. Similarly, collect the output from nodes $0, 1, 2, \ldots, d^{(l)}$ in the vector $\mathbf{x}^{(l)}$.

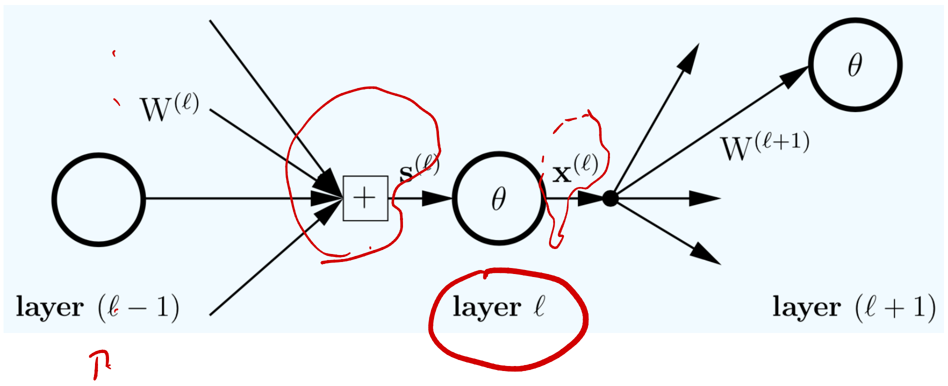
$s_j^{(\ell)}$

There are links (weights) connecting the outputs of all nodes in the previous layer to the inputs of layer l. So, into layer l, we have a $(d^{(l-1)} + 1) \times d^{(l)}$ matrix of weights $\mathbf{W}^{(l)}$. The (i, j)-entry of $\mathbf{W}^{(l)}$ is $w_{ij}^{(l)}$.

$\longrightarrow$ training parameters

Layer l Parameters

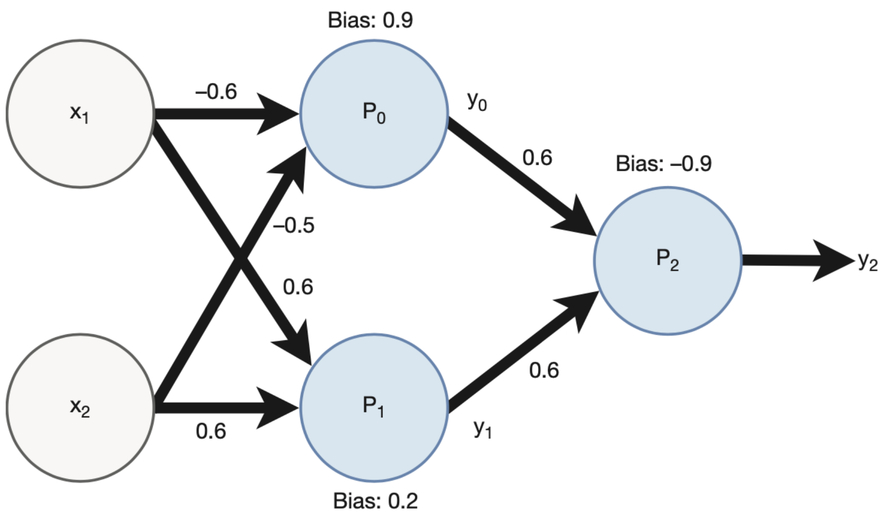
$\ell-1$ ℓ



layer ℓ parameters		
signals in	$\mathbf{s}^{(\ell)}$	$d^{(\ell)}$ dimensional input vector
outputs	$\mathbf{x}^{(\ell)}$	$d^{(\ell)}+1$ dimensional output vector
weights in	$\mathrm{W}^{(\ell)}$	$(d^{(\ell-1)}+1) \times d^{(\ell)}$ dimensional matrix
weights out	$\mathrm{W}^{(\ell+1)}$	$(d^{(\ell)}+1) \times d^{(\ell+1)}$ dimensional matrix

Remark: After you fix the weights $\mathbf{W}^{(l)}$ for $l = 1, 2, \ldots, L$, you have specified a particular neural network hypothesis $h \in \mathcal{H}$. We collect all these weight matrices into a single weight parameter $\mathbf{w} = \{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \ldots, \mathbf{W}^{(L)}\}$, and sometimes we will write $h(\mathbf{x}; \mathbf{w})$ to explicitly indicate the dependence of the hypothesis on $\mathbf{w}$.

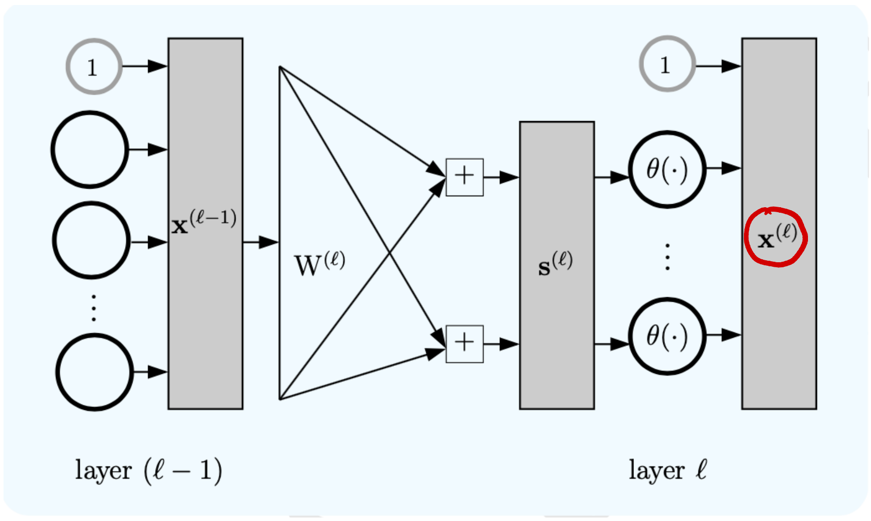
<u>Example</u>: Let us consider a simple network that solves the XOR problem:



Fun Time: The dimension for $\mathbf{W}^{(1)}$ is (1) 2x2 (2) 2x3 (3) 3x2 (4) 3x3.

Q: Write down the contents of $\mathbf{W}^{(1)}$:

2. Forward Propagation



The neural network hypothesis $h(\mathbf{x})$ is computed by the forward propagation algorithm. First observe that the inputs and outputs of a layer are related by the activation function,

$$\mathbf{x}^{(l)} = \begin{bmatrix} 1 \\ \theta(\mathbf{s}^{(l)}) \end{bmatrix}$$

w

where $\theta(\mathbf{s}^{(l)})$ is a vector whose components are $\theta(s_j^{(l)})$. To get the input vector into layer l, we compute the weighted sum of the outputs from the previous layer, with weights specified in $\mathbf{W}^{(l)}$:

$$\underline{s_j^{(l)} = \sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)}}$$

index

*

$$\underline{\mathbf{s}^{(l)} = \left(\mathbf{W}^{(l)}\right)^T \mathbf{x}^{(l-1)}}$$

矩阵

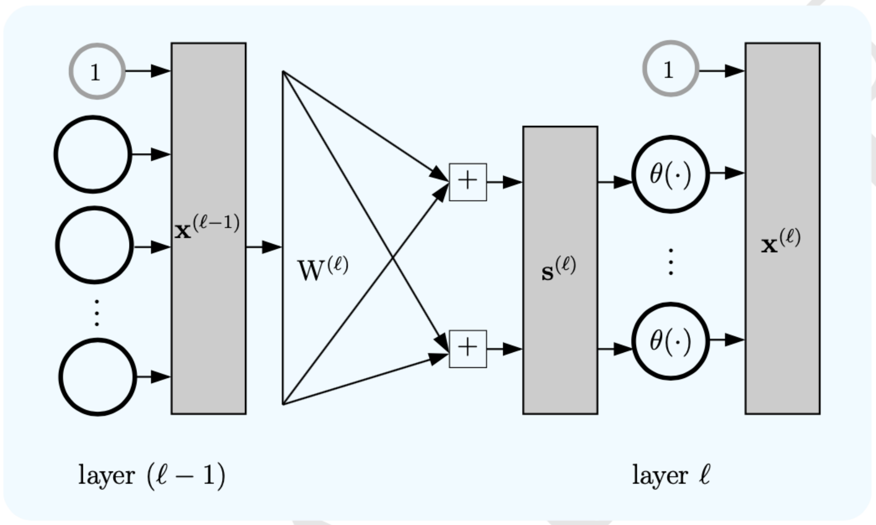
All that remains is to initialize the input layer to $\mathbf{x}^{(0)} = \mathbf{x}$ (so $d^{(0)} = d$, the input dimension) and use the above equations in the following chain,

$$\mathbf{x} = \mathbf{x}^{(0)} \xrightarrow{\mathrm{W}^{(1)}} \mathbf{s}^{(1)} \xrightarrow{\theta} \mathbf{x}^{(1)} \xrightarrow{\mathrm{W}^{(2)}} \mathbf{s}^{(2)} \xrightarrow{\theta} \mathbf{x}^{(2)} \cdots \longrightarrow \mathbf{s}^{(L)} \xrightarrow{\theta} \mathbf{x}^{(L)} = h(\mathbf{x}).$$

$\hat{y}$ y'

Forward propagation to compute $h(\mathbf{x})$:

```
1: $\mathbf{x}^{(0)} \leftarrow \mathbf{x}$                                [Initialization]
2: for $\ell = 1$ to $L$ do                           [Forward Propagation]
3:     $\mathbf{s}^{(\ell)} \leftarrow (\mathrm{W}^{(\ell)})^{\mathrm{T}}\mathbf{x}^{(\ell-1)}$
4:     $\mathbf{x}^{(\ell)} \leftarrow \begin{bmatrix} 1 \\ \theta(\mathbf{s}^{(\ell)}) \end{bmatrix}$
5: $h(\mathbf{x}) = \mathbf{x}^{(L)}$                              [Output]
```



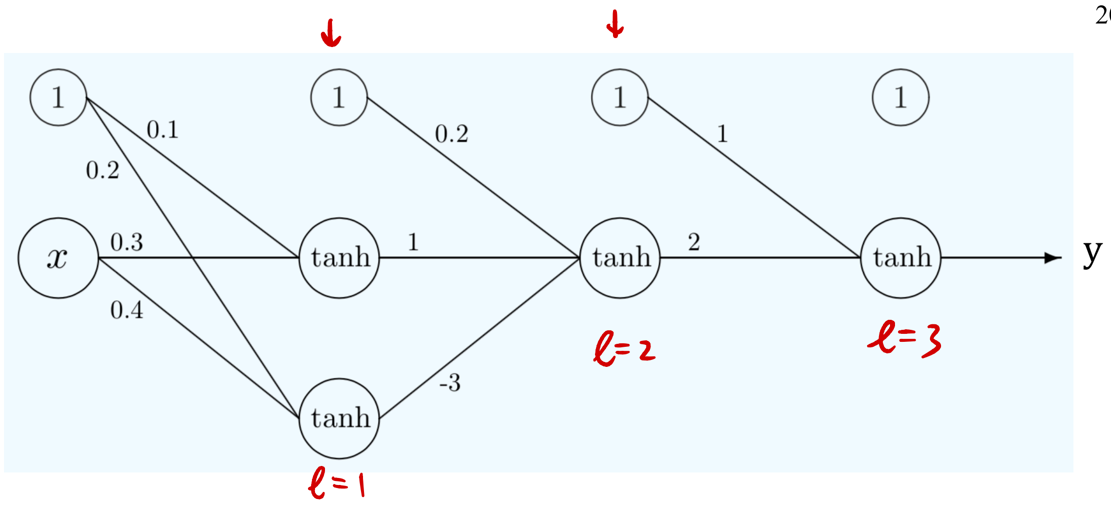
After forward propagation, the output vector $\mathbf{x}^{(l)}$ at every layer $l = 0, 1, 2, \dots, L$ has been computed.

If we want to compute the error E, all we need is $h(\mathbf{x}_n)$ and y_n. For the mean square sum (MSE) error (over the N data points $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)$):

$$E(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}(h(\mathbf{x}_n; \mathbf{w}) - y_n)^2$$

$$= \frac{1}{N}\sum_{n=1}^{N}\left(\mathbf{x}_n^{(L)} - y_n\right)^2$$

Example: Let us consider a very simple network with a single data point where the input $x = 2$ and output $y = 1$.



There is a single input, and the weight matrices are:

$$\underline{\mathbf{W}^{(1)}} = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix}, \ \underline{\mathbf{W}^{(2)}} = \begin{bmatrix} 0.2 \\ 1 \\ -3 \end{bmatrix}, \ \underline{\mathbf{W}^{(3)}} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Forward propagation gives:

$\mathbf{x}^{(0)}$	$\mathbf{s}^{(1)}$	$\mathbf{x}^{(1)}$	$\mathbf{s}^{(2)}$	$\mathbf{x}^{(2)}$	$\mathbf{s}^{(3)}$	$\mathbf{x}^{(3)}$
$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$	$\begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix}^T \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 0.7 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0.6 \\ 0.76 \end{bmatrix}$	$[-1.48]$	$\begin{bmatrix} 1 \\ -0.90 \end{bmatrix}$	$[-0.8]$	-0.66

We show above how $\mathbf{s}^{(1)} = \left(\mathbf{W}^{(1)}\right)^{\mathrm{T}} \mathbf{x}^{(0)}$ is computed.

The MSE error for this training example is $E = \left((-0.66) - 1\right)^2 = \underline{3.32}$.