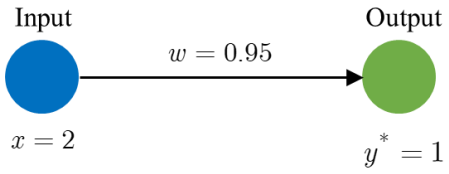
SK, 10/26/2020

Deep Learning Neural Networks

**Example: Single neural network**

Consider a very simple neural network layout consisting of an input and an output layer (no hidden layers). To simplify this example even further, we will assume that there is no bias unit and no nonlinearity involved so that the activation is just the net input. *Hence, the only variable that we can change in our network is one weight*. In order to minimize the error (objective or cost), we will use the mean square error function $J = \frac{1}{2}(y - y^*)^2$, where $y$ denotes computed values and $y^*$ is our target values. The input vector consists of a single value $x = 2$ and the output vector (target value) is also a single number $y^* = 1$. We will also assume that the learning rate in this example is $h = 0.1$ (constant during training) and that the initial value of the weight is $w = 0.95$ (obviously, the correct value of the weight $w$ is $\frac{1}{2}$).



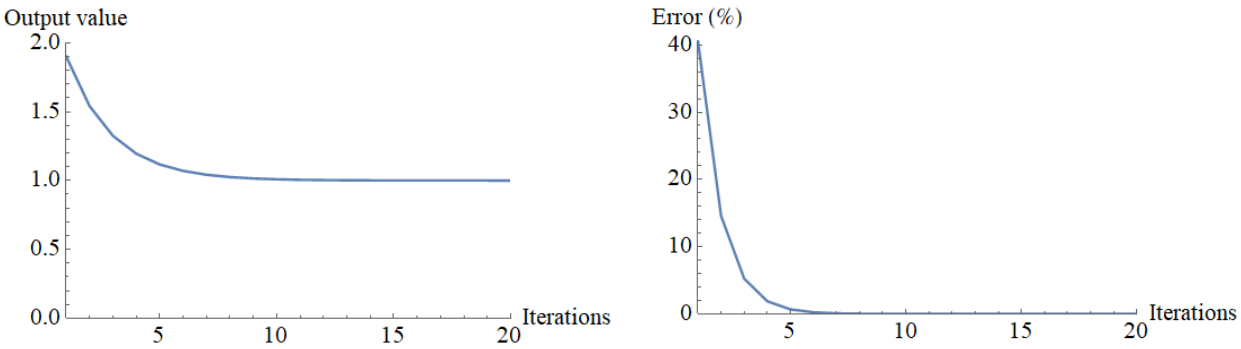

Using the given data, the computed output value is $y = x \times w = 1.9$ while the target value is $y^* = 1$. To correct the weight (train the model), we will apply the backpropagation algorithm. Thus, the weight is adjusted using the gradient descent method and learning rate:

$$w_{new} = w_{old} - h\frac{\partial J(w)}{\partial w},$$

where $\frac{\partial J(w)}{\partial w}$ represents the derivative of the objective function w.r.t. the weight $w$, computed as (using the chain rule of differentiation):

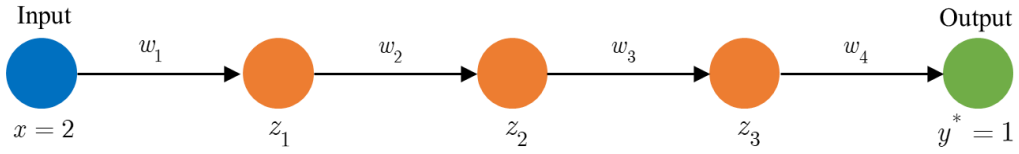
$$\frac{\partial J(w)}{\partial w} = \frac{\partial J(w)}{\partial y}\frac{\partial y}{\partial w}.$$

Applying this, we have a new value for the weight $w_{new} = 0.77$. Performing 10, 15, and 20 iterations, we obtained the weight to be $w = 0.5027$, $w = 0.5002$, and $w = 0.5000$, respectively. The following graph shows the convergence of the output value $y$ and the error function $J$.

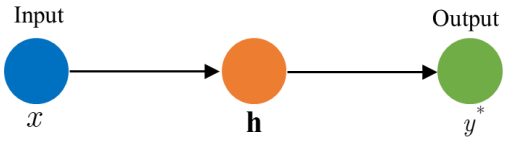



**Example: Neural network with hidden layers with a single neuron**

Consider the same problem described above (with the same input, output data, and learning rate) but with the neural network architecture as shown below. The neural network contains additional 3 hidden layers, and thus, 3 more weights are included. This means that in this example, we have 4 variables that can be changed (optimize) in order to find the desired output. All the weights are randomly initialized as $\mathbf{w} = [0.8, 0.6, 0.1, 0.5]$.




Or, in compact form:




The values of the hidden layers are computed as:

$z_1 = x \times w_1 = 1.6;\ z_2 = z_1 \times w_2 = 0.96;\ z_3 = z_2 \times w_3 = 0.096,$

while the calculated output value is $y = z_3 \times w_4 = 0.048$.

The backpropagation algorithm (searching for optimal weights) starts with computing the derivatives of the objective function (error or cost function) w.r.t. all weights $w_1$, $w_2$, $w_3$, and $w_4$.

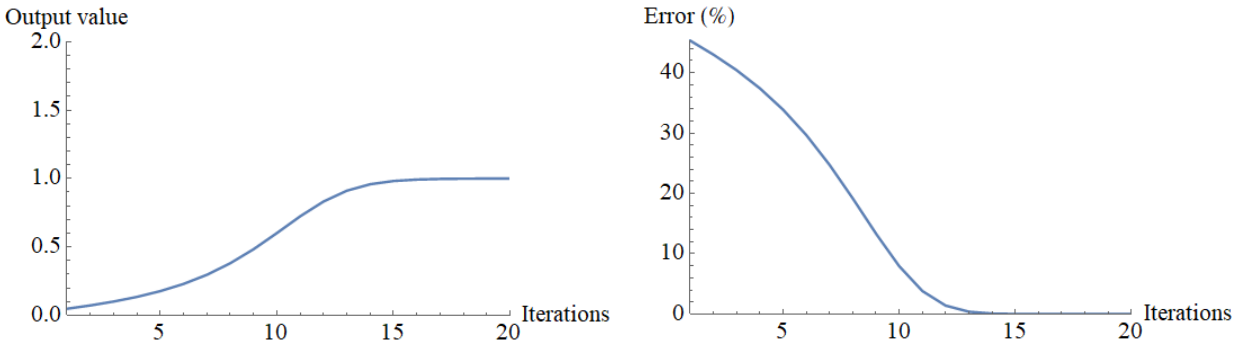
$$\frac{\partial J(\mathbf{w})}{\partial w_4} = \frac{\partial J(\mathbf{w})}{\partial y}\frac{\partial y}{\partial w_4} = (y - y^*)z_3 = d_4 z_3 \qquad \frac{\partial J(\mathbf{w})}{\partial w_3} = \frac{\partial J(\mathbf{w})}{\partial y}\frac{\partial y}{\partial z_3}\frac{\partial z_3}{\partial w_3} = d_4 w_4 z_2 = d_3 z_2$$

$$\frac{\partial J(\mathbf{w})}{\partial w_2} = \frac{\partial J(\mathbf{w})}{\partial y}\frac{\partial y}{\partial z_3}\frac{\partial z_3}{\partial z_2}\frac{\partial z_2}{\partial w_2} = d_3 w_3 z_1 = d_2 z_1 \qquad \frac{\partial J(\mathbf{w})}{\partial w_1} = \frac{\partial J(\mathbf{w})}{\partial y}\frac{\partial y}{\partial z_3}\frac{\partial z_3}{\partial z_2}\frac{\partial z_2}{\partial z_1}\frac{\partial z_1}{\partial w_1} = d_2 w_2 x = d_1 x$$

Then, all the weights will be updated as:

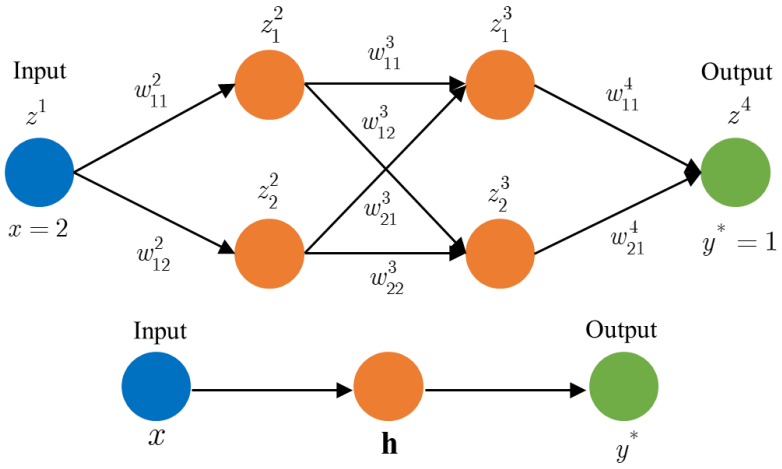
$$w_i^{new} = w_i^{old} - h\frac{\partial J(\mathbf{w})}{\partial w_i}; \quad i = 1,2,3,4.$$

The plot below shows the convergence of the output value and the error function. After 10, 15, and 20 iterations, the computed output value is $y = 0.7235$, $y = 0.992283$, and $y = 0.9999$, respectively.




**Example: Neural network with hidden layers with multiple neurons**

Consider the neural network architecture as shown below. The network contains one input layer (single neuron), 2 hidden layers (with multiple neurons), and the output layer (single neuron). For the sake of brevity, we will assume the same input and output values as in the first two examples, as well as the learning rate $h = 0.1$. All the weights are randomly initialized. However, for comparison assume the following values $w_{11}^2 = 0.7; w_{12}^2 = 0.5; w_{11}^3 = 0.4; w_{12}^3 = 0.6; w_{21}^3 = 0.5; w_{22}^3 = 0.2; w_{11}^4 = 0.1; w_{21}^4 = 0.1$.



SK, 10/26/2020

For this example, when we have multiple neurons in hidden layers, we will organize weights into matrices. There is no calculation in the input layer, so we do not have weights for this layer. Thus, the weight *matrices* are:

$$\mathbf{w}^2 = [w_{11}^2 \; w_{12}^2] \quad \mathbf{w}^3 = \begin{bmatrix} w_{11}^3 & w_{12}^3 \\ w_{21}^3 & w_{22}^3 \end{bmatrix} \quad \mathbf{w}^4 = \begin{bmatrix} w_{11}^4 \\ w_{21}^4 \end{bmatrix}.$$

**Remark:** the following notation is accepted for weights, i.e., weight $w_{ij}^l$ connects $i$ neuron from $l-1$ layer and $j$ neuron in $l$ layer. The superscripts in the above expression refer to layer number.

The values in random neuron $z_j^l$ ($l$ denotes layer and $j$ is neuron in that layer) is computed as:

$$z_j^l = \sum_i^{n^{l-1}} w_{ij}^l z_i^{l-1},$$

where $n^{l-1}$ is number of neurons in previous layer. For instance, values of neuron in the second hidden layer ($z_1^3$ and $z_2^3$) is computed as:

$$j = 1; l = 3, n^{l-1} = 2 \qquad z_1^3 = \sum_i^2 w_{ij}^l z_i^{l-1} = w_{11}^3 z_1^2 + w_{21}^3 z_2^2$$

$$j = 2; l = 3, n^{l-1} = 2 \qquad z_2^3 = \sum_i^2 w_{ij}^l z_i^{l-1} = w_{12}^3 z_1^2 + w_{22}^3 z_2^2.$$

Similar expressions can be obtained for the other neurons. The above expression can be easily written in matrix form:

$$\mathbf{z}^3 = \begin{bmatrix} z_1^3 \\ z_2^3 \end{bmatrix} = (\mathbf{w}^3)^T \mathbf{z}^2 = \begin{bmatrix} w_{11}^3 & w_{21}^3 \\ w_{12}^3 & w_{22}^3 \end{bmatrix} \begin{bmatrix} z_1^2 \\ z_2^2 \end{bmatrix}.$$

Thus, the general expression is:

$$\mathbf{z}^l = (\mathbf{w}^l)^T \mathbf{z}^{l-1}.$$

Using this expression for $\mathbf{z}^l$ we are able to compute values in each neuron in each layer, including the output layer (notice that $\mathbf{z}^1$ is the input layer so no calculation is needed). Having this, we can calculate the error. Again, we will use the mean square error function $J = \frac{1}{2}(y - y^*)^2$, where $y$ denotes computed values and $y^*$ is our target values. At this point, we have everything to start the backpropagation algorithm (to optimize weights).

To adjust our weights, we need to compute the derivatives of the objective function) w.r.t. *all* weights (for each element in all weight matrices). Before we give a general expression for this, let us go through this process step by step and do it for each weight (since the problem is very simple). As we did in the previous two examples, we start from the output layer:

$$\frac{\partial J(\mathbf{w})}{\partial w_{11}^4} = \frac{\partial J(\mathbf{w})}{\partial y}\frac{\partial y}{\partial w_{11}^4} = (y - y^*)z_1^3 = d^4 z_1^3 \quad \frac{\partial J(\mathbf{w})}{\partial w_{21}^4} = \frac{\partial J(\mathbf{w})}{\partial y}\frac{\partial y}{\partial w_{21}^4} = (y - y^*)z_2^3 = d^4 z_2^3.$$

So, for the last layer (output layer) $d^4$ value includes only $(y - y^*)$.

**Remark:** since we have only one output value in the output vector, $d^4$ is a single number but this will not be true if we have multiple output values. Also, notice that, again, superscript in the above expression denotes layer number.

Now, let us compute the derivatives for the second hidden layer ($l = 3$).

$$\frac{\partial J(\mathbf{w})}{\partial w_{11}^3} = \frac{\partial J(\mathbf{w})}{\partial y}\frac{\partial y}{\partial z_1^3}\frac{\partial z_1^3}{\partial w_{11}^3} = (y - y^*)w_{11}^4 z_1^2 = d_1^3 z_1^2$$

$$\frac{\partial J(\mathbf{w})}{\partial w_{12}^3} = \frac{\partial J(\mathbf{w})}{\partial y}\frac{\partial y}{\partial z_2^3}\frac{\partial z_2^3}{\partial w_{12}^3} = (y - y^*)w_{21}^4 z_1^2 = d_2^3 z_1^2$$

$$\frac{\partial J(\mathbf{w})}{\partial w_{21}^3} = \frac{\partial J(\mathbf{w})}{\partial y}\frac{\partial y}{\partial z_1^3}\frac{\partial z_1^3}{\partial w_{21}^3} = (y - y^*)w_{11}^4 z_2^2 = d_1^3 z_2^2$$

$$\frac{\partial J(\mathbf{w})}{\partial w_{22}^3} = \frac{\partial J(\mathbf{w})}{\partial y}\frac{\partial y}{\partial z_2^3}\frac{\partial z_2^3}{\partial w_{22}^3} = (y - y^*)w_{21}^4 z_2^2 = d_2^3 z_2^2$$

Thus, in this layer ($l = 3$) $\boldsymbol{\delta}^3$ is a vector with components:

$$\boldsymbol{\delta}^3 = \begin{bmatrix} d_1^3 \\ d_2^3 \end{bmatrix} = \begin{bmatrix} (y - y^*)w_{11}^4 \\ (y - y^*)w_{21}^4 \end{bmatrix} = d^4 \begin{bmatrix} w_{11}^4 \\ w_{21}^4 \end{bmatrix},$$

or

$$d_j^l = \sum_k^{n^{l+1}} w_{jk}^{l+1} d_k^{l+1}$$

Similarly, for the first hidden layer ($l = 2$).

$$\frac{\partial J(\mathbf{w})}{\partial w_{11}^2} = \frac{\partial J(\mathbf{w})}{\partial y}\frac{\partial y}{\partial z_1^3}\frac{\partial z_1^3}{\partial z_1^2}\frac{\partial z_1^2}{\partial w_{11}^2} + \frac{\partial J(\mathbf{w})}{\partial y}\frac{\partial y}{\partial z_2^3}\frac{\partial z_2^3}{\partial z_1^2}\frac{\partial z_1^2}{\partial w_{11}^2},$$

which gives:

$$\frac{\partial J(\mathbf{w})}{\partial w_{11}^2} = (y - y^*) w_{11}^4 w_{11}^3 z_1^1 + (y - y^*) w_{21}^4 w_{12}^3 z_1^1 = d_1^3 w_{11}^3 z_1^1 + d_2^3 w_{12}^3 z_1^1 = d_1^2 z_1^1.$$

The same relation is obtained using the general expression:

$$d_j^l = \sum_k^{n^{l+1}} w_{jk}^{l+1} d_k^{l+1} \quad \Rightarrow \quad d_1^2 = w_{11}^3 d_1^3 + w_{12}^3 d_2^3,$$

and for the other component of $\boldsymbol{\delta}^2$ vector:

$$\frac{\partial J(\mathbf{w})}{\partial w_{12}^2} = d_2^2 z_1^1 \qquad\qquad d_2^2 = w_{21}^3 d_1^3 + w_{22}^3 d_2^3$$

Notice that for layer $l = 1$ we do not have to compute derivatives since this is the input layer.

**Remark:** The above expressions for the derivatives of the objective function w.r.t all weights can be written in a more convenient way:

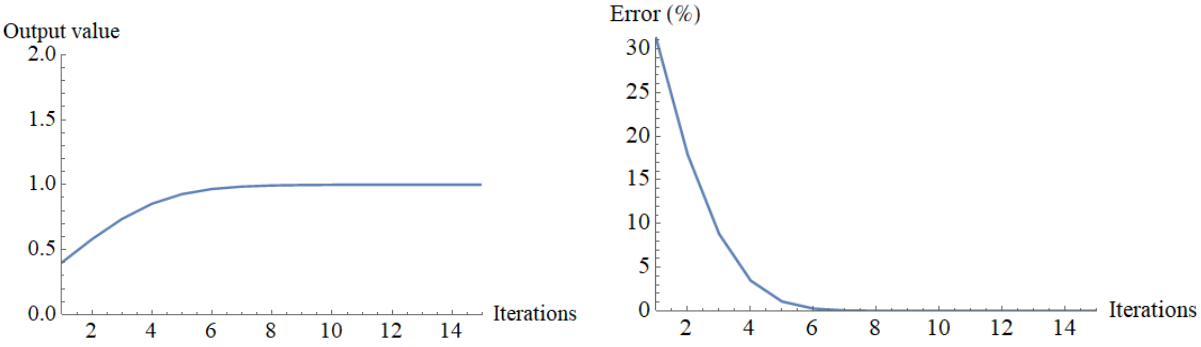
$$\frac{\partial J(\mathbf{w})}{\partial w_{ij}^l} = d_j^l z_i^{l-1}.$$

Now, we have all derivatives of the objective function w.r.t all weights so we can update the weights:

$${}^{new}w_{ij}^l = {}^{old}w_{ij}^l - \eta \frac{\partial J(\mathbf{w})}{\partial w_{ij}^l}.$$

**Remark:** Even though in this example we have 2 hidden layers with 2 neurons in each of them, the code provided for this example is designed for an arbitrary number of hidden layers and neurons.

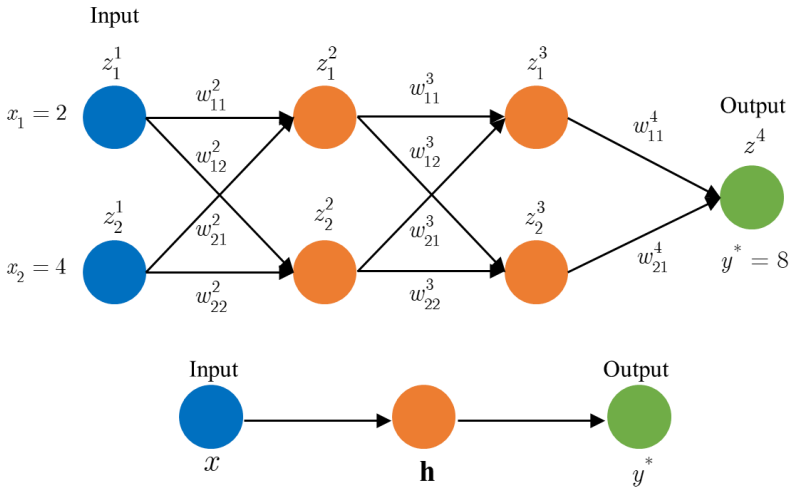
The plot below shows the convergence of the output value and the error function. After 5, 10, and 15 iterations, the computed output value is $y = 0.9272$, $y = 0.9989$, and $y = 0.9999$, respectively.



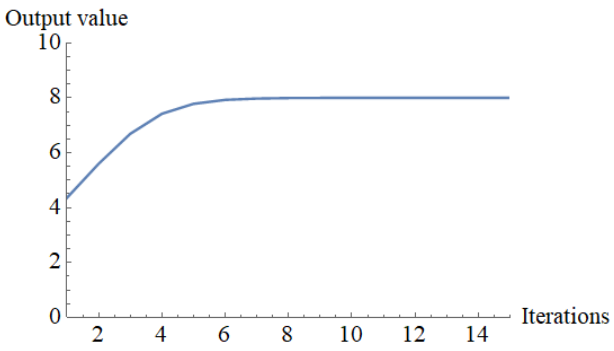
**Example: Neural network with hidden layers with multiple neurons and multiple inputs**

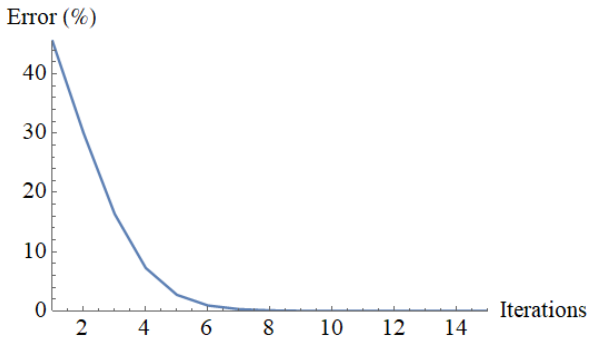
The code for the above example is valid for an arbitrary number of hidden layers and neurons. However, it is also applicable for NNs with multiple inputs. To show this, consider the neural network architecture in the figure below. The network contains one input layer (two neurons), 2 hidden layers (with 2 neurons in each layer), and the output layer (single neuron). The input vector is defined as $\mathbf{x} = [2, 4]$, while the objective (target) value is $y^* = 8$. We will assume that the learning rate is $h = 5 \times 10^{-3}$. All the weights are randomly initialized. However, for comparison assume the following values $w_{11}^2 = 0.7; w_{12}^2 = 0.5; w_{21}^2 = 0.1; w_{22}^2 = 0.9; w_{11}^3 = 0.4; w_{12}^3 = 0.6; w_{21}^3 = 0.5; w_{22}^3 = 0.2$ and

$w_{11}^4 = 0.8; w_{21}^4 = 0.4$.




The plot below shows the convergence of the output value and the error function. After 5, 10, and 15 iterations, the computed output value is $y = 7.7814$, $y = 7.9992$, and $y = 8.0$, respectively.



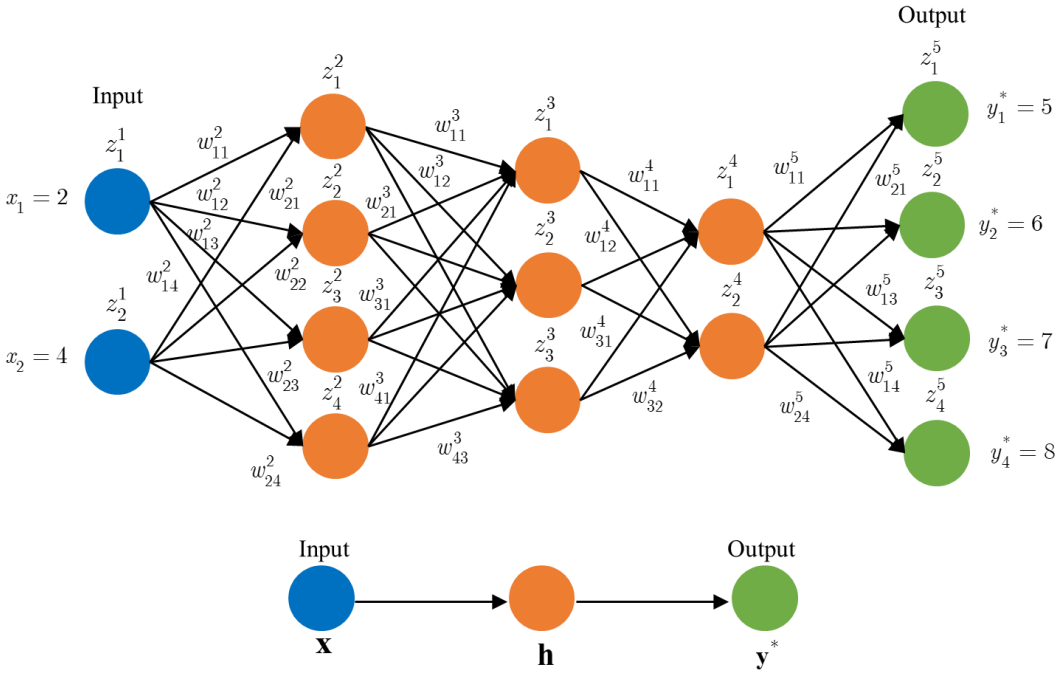
SK, 10/26/2020

**Example: Neural network with multiple outputs**

Consider the architecture neural network as shown below. The neural network contains input vector $\mathbf{x} = [2, 4]$, 3 hidden layer and output vector with multiple outputs $\mathbf{y}^* = [5, 6, 7, 8]$. We will assume that the learning rate is $h = 2 \times 10^{-3}$. All the weights are randomly initialized and the corresponding matrices are as follows:

$$\mathbf{w}^2 = \begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 & w_{14}^2 \\ w_{21}^2 & w_{22}^2 & w_{23}^2 & w_{24}^2 \end{bmatrix} \quad \mathbf{w}^3 = \begin{bmatrix} w_{11}^3 & w_{12}^3 & w_{13}^3 \\ w_{21}^3 & w_{22}^3 & w_{23}^3 \\ w_{31}^3 & w_{32}^3 & w_{33}^3 \\ w_{41}^3 & w_{42}^3 & w_{43}^3 \end{bmatrix} \quad \mathbf{w}^4 = \begin{bmatrix} w_{11}^4 & w_{12}^4 \\ w_{21}^4 & w_{22}^4 \\ w_{31}^4 & w_{32}^4 \end{bmatrix} \quad \mathbf{w}^5 = \begin{bmatrix} w_{11}^5 & w_{12}^5 & w_{13}^5 & w_{14}^5 \\ w_{21}^5 & w_{22}^5 & w_{23}^5 & w_{24}^5 \end{bmatrix}.$$

Associated values are given in the code.




The values in random neuron $z_j^l$ ($l$ denotes layer and $j$ is neuron in that layer) is computed as before:

$$\mathbf{z}^l = (\mathbf{w}^l)^T \mathbf{z}^{l-1}.$$

Using this expression, we can compute predicted values in the last layer $\mathbf{z}^5$, which is our output layer. The only difference between this example and the previous two is that now we have output *vector* instead of a single number. Thus, the error is:

SK, 10/26/2020

$$J = \tfrac{1}{2}\sum_i (y_i - y_i^*)^2,$$

where $y_i$ denotes computed values and $y_i^*$ is our target values in the output vector.

As we did before, the backpropagation algorithm starts from the last layer and we take derivatives w.r.t. *all* weights:

$$\frac{\partial J(\mathbf{w})}{\partial w_{11}^5} = \frac{\partial J(\mathbf{w})}{\partial y_1}\frac{\partial y_1}{\partial w_{11}^5} = (y_1 - y_1^*)z_1^4 = d_1^5 z_1^4 \quad \frac{\partial J(\mathbf{w})}{\partial w_{12}^5} = \frac{\partial J(\mathbf{w})}{\partial y_2}\frac{\partial y_2}{\partial w_{12}^5} = (y_2 - y_2^*)z_1^4 = d_2^5 z_1^4$$

$$\frac{\partial J(\mathbf{w})}{\partial w_{13}^5} = \frac{\partial J(\mathbf{w})}{\partial y_3}\frac{\partial y_3}{\partial w_{13}^5} = (y_3 - y_3^*)z_1^4 = d_3^5 z_1^4 \quad \frac{\partial J(\mathbf{w})}{\partial w_{14}^5} = \frac{\partial J(\mathbf{w})}{\partial y_4}\frac{\partial y_4}{\partial w_{14}^5} = (y_4 - y_4^*)z_1^4 = d_4^5 z_1^4$$

$$\frac{\partial J(\mathbf{w})}{\partial w_{21}^5} = \frac{\partial J(\mathbf{w})}{\partial y_1}\frac{\partial y_1}{\partial w_{21}^5} = (y_1 - y_1^*)z_2^4 = d_1^5 z_2^4 \quad \frac{\partial J(\mathbf{w})}{\partial w_{22}^5} = \frac{\partial J(\mathbf{w})}{\partial y_2}\frac{\partial y_2}{\partial w_{22}^5} = (y_2 - y_2^*)z_2^4 = d_2^5 z_2^4$$

$$\frac{\partial J(\mathbf{w})}{\partial w_{23}^5} = \frac{\partial J(\mathbf{w})}{\partial y_3}\frac{\partial y_3}{\partial w_{23}^5} = (y_3 - y_3^*)z_2^4 = d_3^5 z_2^4 \quad \frac{\partial J(\mathbf{w})}{\partial w_{24}^5} = \frac{\partial J(\mathbf{w})}{\partial y_4}\frac{\partial y_4}{\partial w_{24}^5} = (y_4 - y_4^*)z_2^4 = d_4^5 z_2^4$$

Comparing this with the previous examples (see remark in *Neural network with hidden layers with multiple neurons*) we can see that only $d$ vector is now extended and have more elements. The general expression for $d$ vector in the last layer is:

$$d_j^L = (y_j - y_j^*) \qquad \boldsymbol{\delta}^L = (\mathbf{y} - \mathbf{y}^*),$$

where $j$ denotes element and $L$ stands for the last layer.

After this step (computing derivatives for the last layer), everything will be the same as we did before. The other $d$ vectors are computed as:

$$d_j^l = \sum_k^{n^{l+1}} w_{jk}^{l+1} d_k^{l+1}$$

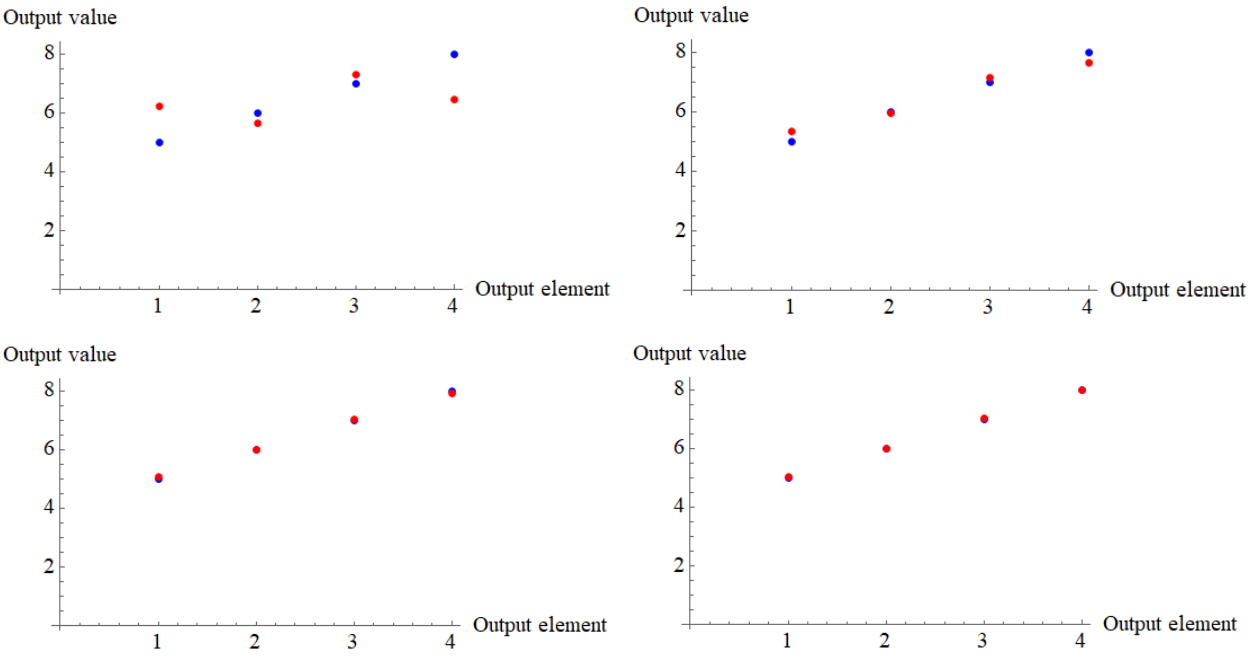
and the derivatives of the objective function w.r.t. all weights are:

$$\frac{\partial J(\mathbf{w})}{\partial w_{ij}^l} = d_j^l z_i^{l-1}.$$

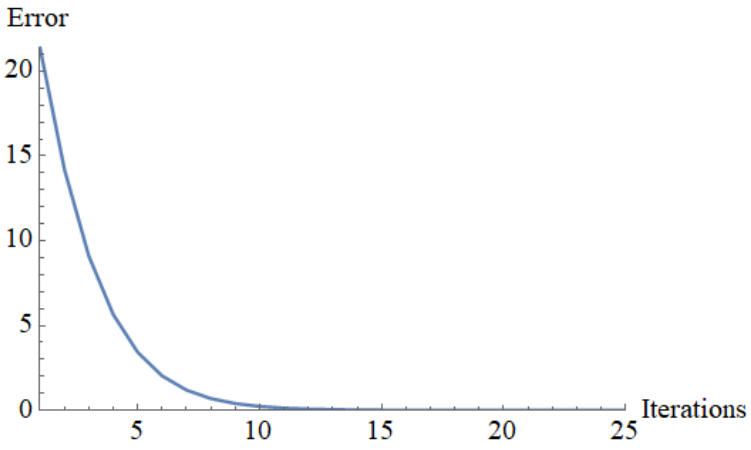
Finally, the weights are update as:

$$^{new}\mathbf{w} = {}^{old}\mathbf{w} - h\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}.$$

The figure below shows the corresponding element in the output vector and target values (blue dots) and predicted values (red dots) for 5, 10, 15, and 20 iterations, respectively. Animation for this is given in the code.



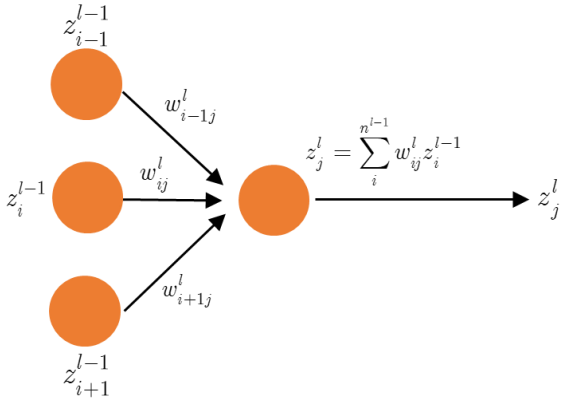

The graph below gives the error function versus iterations for 25 iterations in total. The predicted output vector for 25 iterations is $\mathbf{y} = [5.0038, 5.9990, 7.0032, 7.9952]$.



**Example: Neural network with biases and activation functions**

In all the previous examples we have only weights as contributions for neuron input and outputs. Moreover, we use only linear activation action (transfer function between neurons), meaning that we use just the *net input*. This is the simplest algorithm for artificial neural networks. We calculate this net input as:

$$z_j^l = \sum_i^{n^{l-1}} w_{ij}^l z_i^{l-1} \qquad \mathbf{z}^l = (\mathbf{w}^l)^T \mathbf{z}^{l-1}.$$

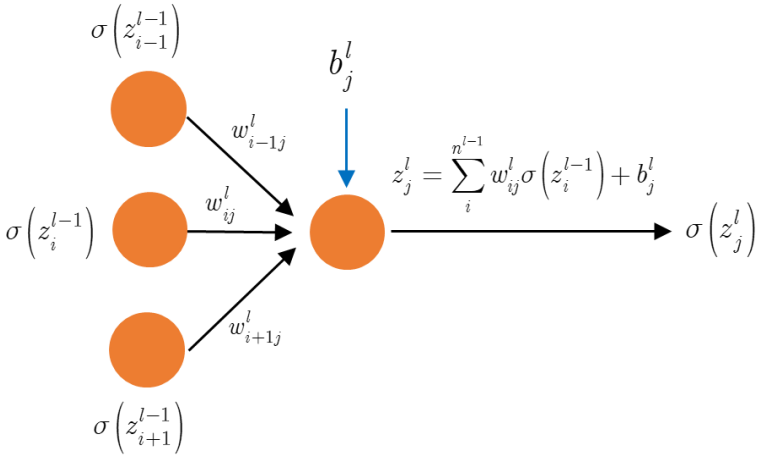



Our next goal is to introduce *biases* and different *activation functions*. Having this we will have more complex models, but we will be able to tackle more challenging problems. In this case, the *input of each neuron* is calculated as:

$$z_j^l = \sum_i^{n^{l-1}} w_{ij}^l s\left(z_i^{l-1}\right) + b_j^l \qquad \mathbf{z}^l = (\mathbf{w}^l)^T s\left(\mathbf{z}^{l-1}\right) + \mathbf{b}^l,$$

while *the actual output* of each neuron (a.k.a. activation of each neuron) is calculated applying an activation function to the neuron input:

$$s\left(z_j^l\right) \qquad s\left(\mathbf{z}^l\right).$$



In the previous expressions, $s$ is a given activation function while $b_j^l$ is a *bias* for $j$ neuron in $l$ layer.

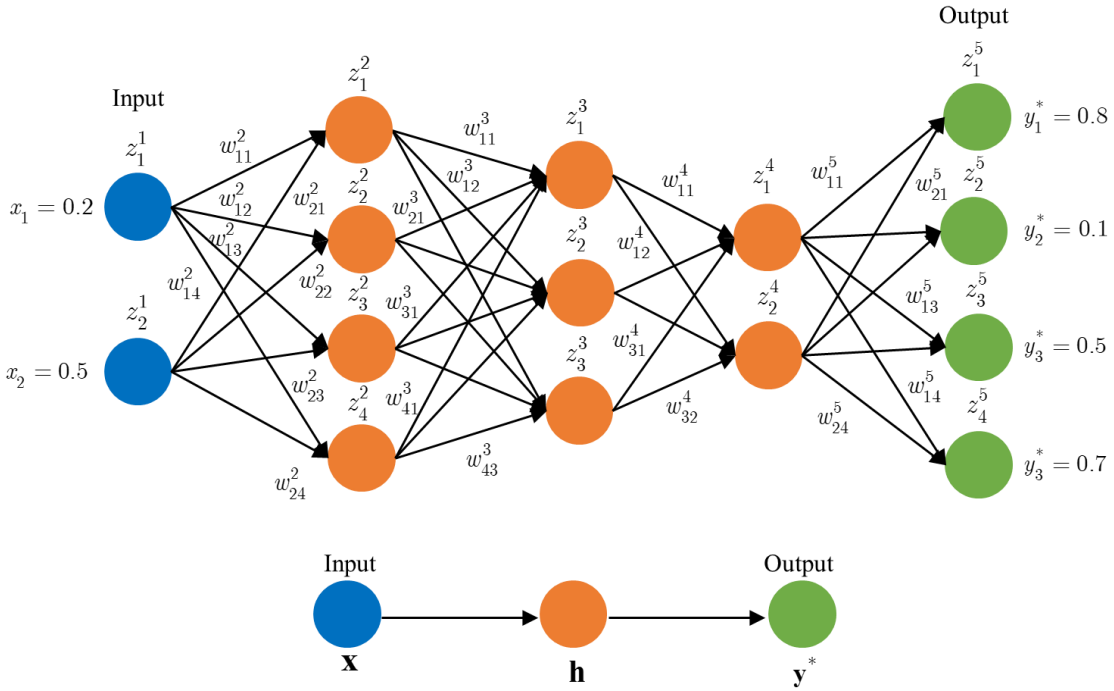
*Special case*: Notice that for linear activation function $s(z) = z$ we get the following expressions:

$$z_j^l = \sum_i^{n^{l-1}} w_{ij}^l z_i^{l-1} + b_j^l \qquad \mathbf{z}^l = (\mathbf{w}^l)^T \mathbf{z}^{l-1} + \mathbf{b}^l ,$$

which is the same as we had before in all previous examples if $\mathbf{b} = 0$ (no biases).

**Remark:** Notice now that we have new vectors for biases ($\mathbf{b}$, randomly initialized) for each hidden layer and the output layer (the last layer). Moreover, after collecting all contributions from weights and biases for a specific neuron, we apply an activation function $s(z)$ in order to get the actual output for that neuron. This applies for all hidden layer as well as for the output layer.

Using these expressions above, we can perform forward propagation and computed predicted values. Then, next step is to find derivatives of the objective function (now w.r.t. all weights and biases) and update weights and biases. In order to do that, consider the same architecture neural network that we have in the previous example. The input vector for this case will be $\mathbf{x} = [0.2, 0.5]$, while the output vector is $\mathbf{y}^* = [0.8, 0.1, 0.5, 0.7]$. We will assume that the learning rate is $h = 0.1$. Associated weight and bias values are given in the code.




The backpropagation algorithm starts from the last layer and we take derivatives w.r.t. *all* weights:

$$\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial w_{11}^5} = \frac{\partial J(\mathbf{w},\mathbf{b})}{\partial y_1}\frac{\partial y_1}{\partial z_1^5}\frac{\partial z_1^5}{\partial w_{11}^5} = (y_1 - y_1^*)s'(z_1^5)s(z_1^4) = d_1^5 s(z_1^4)$$

$$\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial w_{12}^5} = \frac{\partial J(\mathbf{w},\mathbf{b})}{\partial y_2}\frac{\partial y_2}{\partial z_2^5}\frac{\partial z_2^5}{\partial w_{12}^5} = (y_2 - y_2^*)s'(z_2^5)s(z_1^4) = d_2^5 s(z_1^4)$$

$$\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial w_{13}^5} = \frac{\partial J(\mathbf{w},\mathbf{b})}{\partial y_3}\frac{\partial y_3}{\partial z_3^5}\frac{\partial z_3^5}{\partial w_{13}^5} = (y_3 - y_3^*)s'(z_3^5)s(z_1^4) = d_3^5 s(z_1^4)$$

$$\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial w_{14}^5} = \frac{\partial J(\mathbf{w},\mathbf{b})}{\partial y_4}\frac{\partial y_4}{\partial z_4^5}\frac{\partial z_4^5}{\partial w_{14}^5} = (y_4 - y_4^*)s'(z_4^5)s(z_1^4) = d_4^5 s(z_1^4)$$

$$\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial w_{21}^5} = \frac{\partial J(\mathbf{w},\mathbf{b})}{\partial y_1}\frac{\partial y_1}{\partial z_1^5}\frac{\partial z_1^5}{\partial w_{21}^5} = (y_1 - y_1^*)s'(z_1^5)s(z_2^4) = d_1^5 s(z_2^4)$$

$$\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial w_{22}^5} = \frac{\partial J(\mathbf{w},\mathbf{b})}{\partial y_2}\frac{\partial y_2}{\partial z_2^5}\frac{\partial z_2^5}{\partial w_{22}^5} = (y_2 - y_2^*)s'(z_2^5)s(z_2^4) = d_2^5 s(z_2^4)$$

$$\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial w_{23}^5} = \frac{\partial J(\mathbf{w},\mathbf{b})}{\partial y_3}\frac{\partial y_3}{\partial z_3^5}\frac{\partial z_3^5}{\partial w_{23}^5} = (y_3 - y_3^*)s'(z_3^5)s(z_2^4) = d_3^5 s(z_2^4)$$

$$\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial w_{24}^5} = \frac{\partial J(\mathbf{w},\mathbf{b})}{\partial y_4}\frac{\partial y_4}{\partial z_4^5}\frac{\partial z_4^5}{\partial w_{24}^5} = (y_4 - y_4^*)s'(z_4^5)s(z_2^4) = d_4^5 s(z_2^4)$$

Thus, the general expression for $d$ vector in the last layer is:

$$d_j^L = (y_j - y_j^*)s'(z_j^L) \qquad \boldsymbol{\delta}^L = (\mathbf{y} - \mathbf{y}^*)s'(\mathbf{z}^L) \qquad y_j = s(z_j^L),$$

where $j$ denotes element and $L$ stands for the last layer.

**Remarks:** Notice that for linear activation function $s(z) = z$ the derivative $s'(z) = 1$ and we have the same expression as before.

Similarly, the other $d$ vectors (for other layers) can be computed using the following expression:

$$d_j^l = \sum_k^{n^{l+1}} w_{jk}^{l+1} d_k^{l+1} s'(z_j^l) \qquad \boldsymbol{\delta}^l = \mathbf{w}^{l+1}\boldsymbol{\delta}^{l+1} \circ s'(\mathbf{z}^l).$$

**Exercise:** Prove that this expression is correct. Use the same approach I did for the last layer.

Having all this we can obtain the derivatives of the objective function w.r.t. all weights are:

$$\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial w_{ij}^{l}} = d_j^l s(z_i^{l-1}) \qquad \frac{\partial J(\mathbf{w},\mathbf{b})}{\partial \mathbf{w}^{l}} = s(\mathbf{z}^{l-1})(\boldsymbol{\delta}^l)^T .$$

**Exercise:** Prove that this expression is correct.

Finally, the weights are update as:

$$^{new}\mathbf{w} = {}^{old}\mathbf{w} - \eta \frac{\partial J(\mathbf{w},\mathbf{b})}{\partial \mathbf{w}} .$$

Now, since we have also biases for each neuron, we need to take derivatives w.r.t. *all* biases:

$$\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial b_1^5} = \frac{\partial J(\mathbf{w},\mathbf{b})}{\partial y_1}\frac{\partial y_1}{\partial z_1^5}\frac{\partial z_1^5}{\partial b_1^5} = (y_1 - y_1^*)s'(z_1^5) = d_1^5$$

$$\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial b_2^5} = \frac{\partial J(\mathbf{w},\mathbf{b})}{\partial y_2}\frac{\partial y_2}{\partial z_2^5}\frac{\partial z_2^5}{\partial b_2^5} = (y_2 - y_2^*)s'(z_2^5) = d_2^5$$

$$\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial b_3^5} = \frac{\partial J(\mathbf{w},\mathbf{b})}{\partial y_3}\frac{\partial y_3}{\partial z_3^5}\frac{\partial z_3^5}{\partial b_3^5} = (y_3 - y_3^*)s'(z_3^5) = d_3^5$$

$$\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial b_4^5} = \frac{\partial J(\mathbf{w},\mathbf{b})}{\partial y_4}\frac{\partial y_4}{\partial z_4^5}\frac{\partial z_4^5}{\partial b_4^5} = (y_4 - y_4^*)s'(z_4^5) = d_4^5$$

Thus, the general expression for $\mathbf{b}$ vector in the last layer is:

$$\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial b_j^L} = d_j^L \qquad \frac{\partial J(\mathbf{w},\mathbf{b})}{\partial \mathbf{b}^L} = \boldsymbol{\delta}^L ,$$

where $j$ denotes element and $L$ stands for the last layer.

The same expression is valid for all other layer:

$$\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial b_j^l} = d_j^l \qquad \frac{\partial J(\mathbf{w},\mathbf{b})}{\partial \mathbf{b}^l} = \boldsymbol{\delta}^l$$

**Exercise:** Prove that this expression is correct. Use the same approach I did for the last layer.

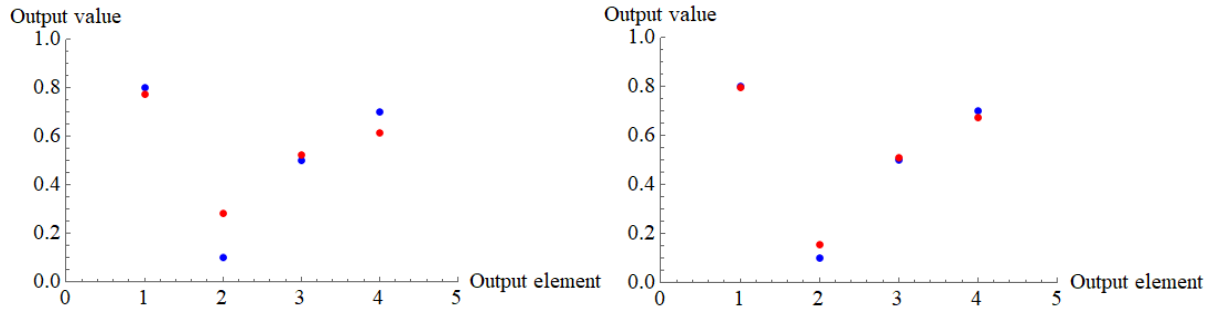
*Activation functions*: For this example, we will introduce some activation functions. Apart from linear activation function $s(z) = z$ we will start with the following:

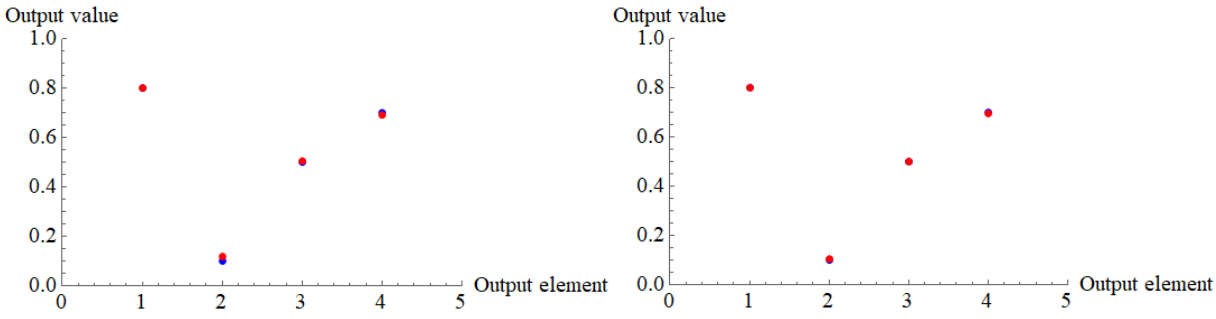
- Sigmoid (logistic) activation function $s(z) = \frac{1}{1 + e^{-z}}$. This function limits the data range $D \in (0,1)$ and thus it is suitable for probability analysis.
- ReLU (Rectified Linear Unit) activation function $s(z) = \begin{cases} 0 & z < 0 \\ z & z \geq 0 \end{cases}$. This function does not allow negative values for neuron activation, so the data range is $D \in [0,\infty)$.
- Hyperbolic tangent activation function $s(z) = \text{Tanh}(z) = \frac{2}{1 + e^{-2z}} - 1$. This function limits the data range $D \in (-1,1)$ or it can be re-written for the data range $D \in (0,1) \rightarrow s(z) = \frac{1}{1 + e^{-2z}}$.

All the above-mentioned activation functions are given and implemented in the code.

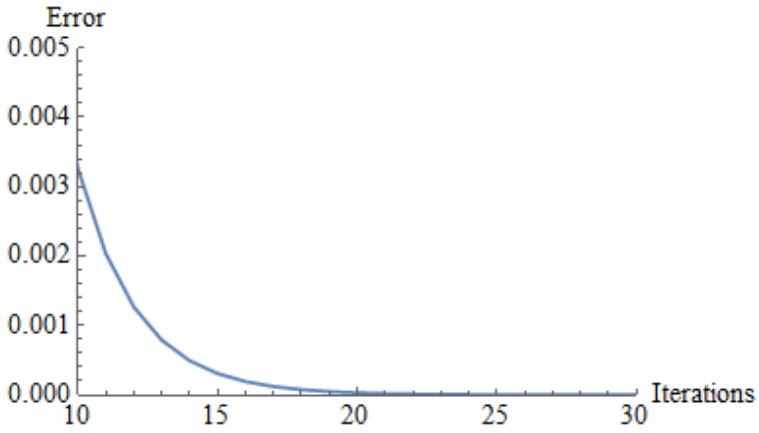
**Exercise:** Plot all four activation functions (linear, sigmoid, ReLU, and hyperbolic tangent) on one graph and see the differences. Do the same for their derivatives. What can you conclude based on these two plots? For the same initial conditions and learning rate which activation function is the fastest and which one is the slowest one (convergence based)? Why? Prove your finding with the code. By your intuition which function would give the biggest error for nonlinear data and why?

The problem described in this example can be solved with all four activation functions. Some of the obtained results are given below. The figure below shows the corresponding element in the output vector and target values (blue dots) and predicted values (red dots) obtained using linear activation function for 5, 10, 15, and 20 iterations, respectively. Animation for this is given in the code.





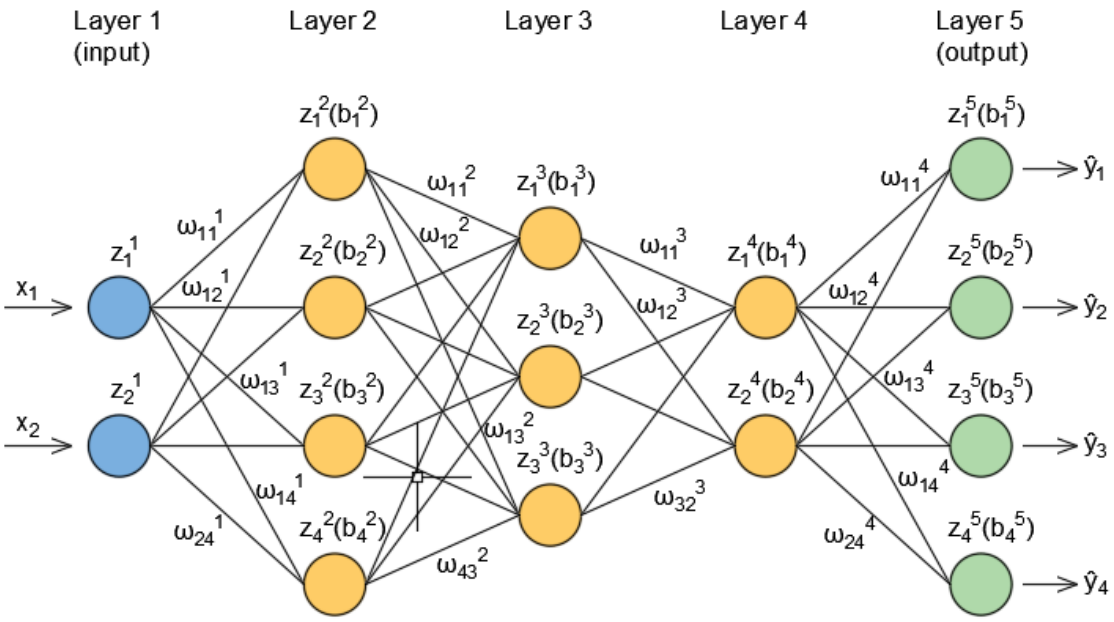

The graph below gives the error function versus iterations for 30 iterations in total. The predicted output vector for 30 iterations is $\mathbf{y} = [0.8002, 0.1005, 0.5001, 0.6997]$.




**Exercise:** The same results are obtained with ReLU activation function. Explain why this is correct? Then, try the other two activation functions. What can you conclude? Which activation function is better and why (for this problem)?

SK, 10/26/2020

**Example: Neural network with biases and activation functions (Math behind Python algorithm)**




Input layer in matrix form for one sample is a row vector:

$$X = \begin{bmatrix} x_1 & x_2 \end{bmatrix}$$

Weights could be written as:

$$W^1 = \begin{bmatrix} \omega_{11}^1 & \omega_{12}^1 & \omega_{13}^1 & \omega_{14}^1 \\ \omega_{21}^1 & \omega_{22}^1 & \omega_{23}^1 & \omega_{24}^1 \end{bmatrix} \qquad W^2 = \begin{bmatrix} \omega_{11}^2 & \omega_{12}^2 & \omega_{13}^2 \\ \omega_{21}^2 & \omega_{22}^2 & \omega_{23}^2 \\ \omega_{31}^2 & \omega_{32}^2 & \omega_{33}^2 \\ \omega_{41}^2 & \omega_{42}^2 & \omega_{43}^2 \end{bmatrix}$$

$$W^3 = \begin{bmatrix} \omega_{11}^3 & \omega_{12}^3 \\ \omega_{21}^3 & \omega_{22}^3 \\ \omega_{31}^3 & \omega_{32}^3 \end{bmatrix} \qquad W^4 = \begin{bmatrix} \omega_{11}^4 & \omega_{12}^4 & \omega_{13}^4 & \omega_{14}^4 \\ \omega_{21}^4 & \omega_{22}^4 & \omega_{23}^4 & \omega_{24}^4 \end{bmatrix}$$

SK, 10/26/2020

Output layer in matrix form as a row vector:

$$\hat{Y} = \begin{bmatrix} \hat{y}_1 & \hat{y}_2 & \hat{y}_3 & \hat{y}_4 \end{bmatrix}$$

Biases are given as:

$$B^2 = \begin{bmatrix} b_1^2 & b_2^2 & b_3^2 & b_4^2 \end{bmatrix} \qquad B^3 = \begin{bmatrix} b_1^3 & b_2^3 & b_3^3 \end{bmatrix}$$

$$B^4 = \begin{bmatrix} b_1^4 & b_2^4 \end{bmatrix} \qquad B^5 = \begin{bmatrix} b_1^5 & b_2^5 & b_3^5 & b_4^5 \end{bmatrix}$$

FEED FORWARD

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \cdot \begin{bmatrix} \omega_{11}^1 & \omega_{12}^1 & \omega_{13}^1 & \omega_{14}^1 \\ \omega_{21}^1 & \omega_{22}^1 & \omega_{23}^1 & \omega_{24}^1 \end{bmatrix} + \begin{bmatrix} b_1^2 & b_2^2 & b_3^2 & b_4^2 \end{bmatrix} = \begin{bmatrix} z_1^2 & z_2^2 & z_3^2 & z_4^2 \end{bmatrix}$$

$$X \cdot W^1 + B^2 = Z^2$$

$$\begin{bmatrix} \sigma\left(z_1^2\right) & \sigma\left(z_2^2\right) & \sigma\left(z_3^2\right) & \sigma\left(z_4^2\right) \end{bmatrix} = \begin{bmatrix} a_1^2 & a_2^2 & a_3^2 & a_4^2 \end{bmatrix}$$

$$\sigma\left(Z^2\right) = A^2$$

$$\begin{bmatrix} a_1^2 & a_2^2 & a_3^2 & a_4^2 \end{bmatrix} \cdot \begin{bmatrix} \omega_{11}^2 & \omega_{12}^2 & \omega_{13}^2 \\ \omega_{21}^2 & \omega_{22}^2 & \omega_{23}^2 \\ \omega_{31}^2 & \omega_{32}^2 & \omega_{33}^2 \\ \omega_{41}^2 & \omega_{42}^2 & \omega_{43}^2 \end{bmatrix} + \begin{bmatrix} b_1^3 & b_2^3 & b_3^3 \end{bmatrix} = \begin{bmatrix} z_1^3 & z_2^3 & z_3^3 \end{bmatrix}$$

$$A^2 \cdot W^2 + B^3 = Z^3$$

$$\begin{bmatrix} \sigma\left(z_1^3\right) & \sigma\left(z_2^3\right) & \sigma\left(z_3^3\right) \end{bmatrix} = \begin{bmatrix} a_1^3 & a_2^3 & a_3^3 \end{bmatrix}$$

$$\sigma\left(Z^3\right) = A^3$$

$$\begin{bmatrix} a_1^3 & a_2^3 & a_3^3 \end{bmatrix} \cdot \begin{bmatrix} \omega_{11}^3 & \omega_{12}^3 \\ \omega_{21}^3 & \omega_{22}^3 \\ \omega_{31}^3 & \omega_{32}^3 \end{bmatrix} + \begin{bmatrix} b_1^4 & b_2^4 \end{bmatrix} = \begin{bmatrix} z_1^4 & z_2^4 \end{bmatrix}$$

$$A^3 \cdot W^3 + B^4 = Z^4$$

$$\begin{bmatrix} \sigma\left(z_1^4\right) & \sigma\left(z_2^4\right) \end{bmatrix} = \begin{bmatrix} a_1^4 & a_2^4 \end{bmatrix}$$

$$\sigma\left(Z^4\right) = A^4$$

SK, 10/26/2020

$$\begin{bmatrix} a_1^4 & a_2^4 \end{bmatrix} \cdot \begin{bmatrix} \omega_{11}^4 & \omega_{12}^4 & \omega_{13}^4 & \omega_{14}^4 \\ \omega_{21}^4 & \omega_{22}^4 & \omega_{23}^4 & \omega_{24}^4 \end{bmatrix} + \begin{bmatrix} b_1^5 & b_2^5 & b_3^5 & b_4^5 \end{bmatrix} = \begin{bmatrix} z_1^5 & z_2^5 & z_3^5 & z_4^5 \end{bmatrix}$$

$$A^4 \cdot W^4 + B^5 = Z^5$$

$$\begin{bmatrix} \sigma\left(z_1^5\right) & \sigma\left(z_2^5\right) & \sigma\left(z_3^5\right) & \sigma\left(z_4^5\right) \end{bmatrix} = \begin{bmatrix} \hat{y}_1 & \hat{y}_2 & \hat{y}_3 & \hat{y}_4 \end{bmatrix}$$

$$\sigma\left(Z^5\right) = \hat{Y}$$

Target values are denoted as:

$$Y = \begin{bmatrix} y_1 & y_2 & y_3 & y_4 \end{bmatrix}$$

COST FUNCTION

Cost function is used for evaluating an error at the end of feed forward and it is defined as:

$$J = \frac{1}{2} \sum_i \left( \hat{y}_i - y_i \right)$$

or in matrix form for one sample:

$$J = \frac{1}{2} \sum_i \left( \hat{Y} - Y \right)$$

and multiple samples:

$$J = \frac{1}{2} \sum_i \sum_j \left( \hat{Y} - Y \right)$$

where $i$ and $j$ are used for rows and columns respectively.

SK, 10/26/2020

BACKPROPAGATION

**Layer 5(output)**

$$\hat{y}_1 = \sigma\left(z_1^5\right); \qquad z_1^5 = a_1^4\omega_{11}^4 + a_2^4\omega_{21}^4 + b_1^5$$
$$\hat{y}_2 = \sigma\left(z_2^5\right); \qquad z_2^5 = a_1^4\omega_{12}^4 + a_2^4\omega_{22}^4 + b_2^5$$
$$\hat{y}_3 = \sigma\left(z_3^5\right); \qquad z_3^5 = a_1^4\omega_{13}^4 + a_2^4\omega_{23}^4 + b_3^5$$
$$\hat{y}_4 = \sigma\left(z_4^5\right); \qquad z_4^5 = a_1^4\omega_{14}^4 + a_2^4\omega_{24}^4 + b_4^5$$

Derivatives w.r.t. the weights:

$$\frac{\partial J(w,b)}{\partial w_{11}^4} = \frac{\partial J(w,b)}{\partial \hat{y}_1}\frac{\partial \hat{y}_1}{\partial z_1^5}\frac{\partial z_1^5}{\partial w_{11}^4} = \left(\hat{y}_1 - y_1\right)\sigma'\left(z_1^5\right)a_1^4 = \delta_1^5 a_1^4$$
$$\frac{\partial J(w,b)}{\partial w_{12}^4} = \frac{\partial J(w,b)}{\partial \hat{y}_2}\frac{\partial \hat{y}_2}{\partial z_2^5}\frac{\partial z_2^5}{\partial w_{12}^4} = \left(\hat{y}_2 - y_2\right)\sigma'\left(z_2^5\right)a_1^4 = \delta_2^5 a_1^4$$
$$\frac{\partial J(w,b)}{\partial w_{13}^4} = \frac{\partial J(w,b)}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial z_3^5}\frac{\partial z_3^5}{\partial w_{13}^4} = \left(\hat{y}_3 - y_3\right)\sigma'\left(z_3^5\right)a_1^4 = \delta_3^5 a_1^4$$
$$\frac{\partial J(w,b)}{\partial w_{14}^4} = \frac{\partial J(w,b)}{\partial \hat{y}_4}\frac{\partial \hat{y}_4}{\partial z_4^5}\frac{\partial z_4^5}{\partial w_{14}^4} = \left(\hat{y}_4 - y_4\right)\sigma'\left(z_4^5\right)a_1^4 = \delta_4^5 a_1^4$$

$$\frac{\partial J(w,b)}{\partial w_{21}^4} = \frac{\partial J(w,b)}{\partial \hat{y}_1}\frac{\partial \hat{y}_1}{\partial z_1^5}\frac{\partial z_1^5}{\partial w_{21}^4} = \left(\hat{y}_1 - y_1\right)\sigma'\left(z_1^5\right)a_2^4 = \delta_1^5 a_2^4$$
$$\frac{\partial J(w,b)}{\partial w_{22}^4} = \frac{\partial J(w,b)}{\partial \hat{y}_2}\frac{\partial \hat{y}_2}{\partial z_2^5}\frac{\partial z_2^5}{\partial w_{22}^4} = \left(\hat{y}_2 - y_2\right)\sigma'\left(z_2^5\right)a_2^4 = \delta_2^5 a_2^4$$
$$\frac{\partial J(w,b)}{\partial w_{23}^4} = \frac{\partial J(w,b)}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial z_3^5}\frac{\partial z_3^5}{\partial w_{23}^4} = \left(\hat{y}_3 - y_3\right)\sigma'\left(z_3^5\right)a_2^4 = \delta_3^5 a_2^4$$
$$\frac{\partial J(w,b)}{\partial w_{24}^4} = \frac{\partial J(w,b)}{\partial \hat{y}_4}\frac{\partial \hat{y}_4}{\partial z_4^5}\frac{\partial z_4^5}{\partial w_{24}^4} = \left(\hat{y}_4 - y_4\right)\sigma'\left(z_4^5\right)a_2^4 = \delta_4^5 a_2^4$$

SK, 10/26/2020

The general expressions for the last(output) layer could be written as:

$$\delta_j^L = \left(\hat{y}_j - y_j\right)\sigma'\left(z_j^L\right); \qquad \hat{y}_j = \sigma\left(z_j^L\right)$$

$$\frac{\partial J(w,b)}{\partial w_{ij}^{L-1}} = \delta_j^L a_i^{L-1}$$

where L stands for the last layer.

Previous expression for $L=5$ could be written in matrix form as:

$$\frac{\partial J(w,b)}{\partial W^4} = \begin{bmatrix} \frac{\partial J(w,b)}{\partial w_{11}^4} & \frac{\partial J(w,b)}{\partial w_{12}^4} & \frac{\partial J(w,b)}{\partial w_{13}^4} & \frac{\partial J(w,b)}{\partial w_{14}^4} \\ \frac{\partial J(w,b)}{\partial w_{21}^4} & \frac{\partial J(w,b)}{\partial w_{22}^4} & \frac{\partial J(w,b)}{\partial w_{23}^4} & \frac{\partial J(w,b)}{\partial w_{24}^4} \end{bmatrix} = \begin{bmatrix} a_1^4 \\ a_2^4 \end{bmatrix} \cdot \begin{bmatrix} \delta_1^5 & \delta_2^5 & \delta_3^5 & \delta_4^5 \end{bmatrix}$$

$$\frac{\partial J(w,b)}{\partial W^4} = \left(A^4\right)^T \cdot \delta^5$$

Derivatives w.r.t the biases:

$$\frac{\partial J(w,b)}{b_1^5} = \frac{\partial J(w,b)}{\partial \hat{y}_1}\frac{\partial \hat{y}_1}{\partial z_1^5}\frac{z_1^5}{b_1^5} = \left(\hat{y}_1 - y_1\right)\sigma'\left(z_1^5\right) = \delta_1^5$$

$$\frac{\partial J(w,b)}{b_2^5} = \frac{\partial J(w,b)}{\partial \hat{y}_2}\frac{\partial \hat{y}_2}{\partial z_2^5}\frac{z_2^5}{b_2^5} = \left(\hat{y}_2 - y_2\right)\sigma'\left(z_2^5\right) = \delta_2^5$$

$$\frac{\partial J(w,b)}{b_3^5} = \frac{\partial J(w,b)}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial z_3^5}\frac{z_3^5}{b_3^5} = \left(\hat{y}_3 - y_3\right)\sigma'\left(z_3^5\right) = \delta_3^5$$

$$\frac{\partial J(w,b)}{b_4^5} = \frac{\partial J(w,b)}{\partial \hat{y}_4}\frac{\partial \hat{y}_4}{\partial z_4^5}\frac{z_4^5}{b_4^5} = \left(\hat{y}_4 - y_4\right)\sigma'\left(z_4^5\right) = \delta_4^5$$

written in the general expression for the last layer:

$$\frac{\partial J(w,b)}{\partial b_j^L} = \delta_j^L$$

also in a matrix form:

$$\frac{\partial J(w,b)}{\partial B^5} = \begin{bmatrix} \frac{\partial J(w,b)}{\partial b_1^5} & \frac{\partial J(w,b)}{\partial b_2^5} & \frac{\partial J(w,b)}{\partial b_3^5} & \frac{\partial J(w,b)}{\partial b_4^5} \end{bmatrix} = \begin{bmatrix} \delta_1^5 & \delta_2^5 & \delta_3^5 & \delta_4^5 \end{bmatrix}$$

$$\frac{\partial J(w,b)}{\partial B^5} = \delta^5$$

SK, 10/26/2020

**Layer 4**

$$a_1^4 = \sigma\left(z_1^4\right); \qquad z_1^4 = a_1^3\omega_{11}^3 + a_2^3\omega_{21}^3 + a_3^3\omega_{31}^3 + b_1^4$$
$$a_2^4 = \sigma\left(z_2^4\right); \qquad z_2^4 = a_1^3\omega_{12}^3 + a_2^3\omega_{22}^3 + a_3^3\omega_{32}^3 + b_2^4$$

Derivatives w.r.t. the weights:

$$\frac{\partial J(w,b)}{\partial w_{11}^3} = \frac{\partial J(w,b)}{\partial \hat{y}_1}\frac{\partial \hat{y}_1}{\partial z_1^5}\frac{\partial z_1^5}{\partial a_1^4}\frac{\partial a_1^4}{\partial z_1^4}\frac{\partial z_1^4}{\partial w_{11}^3} + \frac{\partial J(w,b)}{\partial \hat{y}_2}\frac{\partial \hat{y}_2}{\partial z_2^5}\frac{\partial z_2^5}{\partial a_1^4}\frac{\partial a_1^4}{\partial z_1^4}\frac{\partial z_1^4}{\partial w_{11}^3}$$
$$\frac{\partial J(w,b)}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial z_3^5}\frac{\partial z_3^5}{\partial a_1^4}\frac{\partial a_1^4}{\partial z_1^4}\frac{\partial z_1^4}{\partial w_{11}^3} + \frac{\partial J(w,b)}{\partial \hat{y}_4}\frac{\partial \hat{y}_4}{\partial z_4^5}\frac{\partial z_4^5}{\partial a_1^4}\frac{\partial a_1^4}{\partial z_1^4}\frac{\partial z_1^4}{\partial w_{11}^3}$$
$$\frac{\partial J(w,b)}{\partial w_{11}^3} = \delta_1^5 w_{11}^4\sigma'\left(z_1^4\right)a_1^3 + \delta_2^5 w_{12}^4\sigma'\left(z_1^4\right)a_1^3 + \delta_3^5 w_{13}^4\sigma'\left(z_1^4\right)a_1^3 + \delta_4^5 w_{14}^4\sigma'\left(z_1^4\right)a_1^3$$

$$\frac{\partial J(w,b)}{\partial w_{12}^3} = \frac{\partial J(w,b)}{\partial \hat{y}_1}\frac{\partial \hat{y}_1}{\partial z_1^5}\frac{\partial z_1^5}{\partial a_2^4}\frac{\partial a_2^4}{\partial z_2^4}\frac{\partial z_2^4}{\partial w_{12}^3} + \frac{\partial J(w,b)}{\partial \hat{y}_2}\frac{\partial \hat{y}_2}{\partial z_2^5}\frac{\partial z_2^5}{\partial a_2^4}\frac{\partial a_2^4}{\partial z_2^4}\frac{\partial z_2^4}{\partial w_{12}^3}$$
$$\frac{\partial J(w,b)}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial z_3^5}\frac{\partial z_3^5}{\partial a_2^4}\frac{\partial a_2^4}{\partial z_2^4}\frac{\partial z_2^4}{\partial w_{12}^3} + \frac{\partial J(w,b)}{\partial \hat{y}_4}\frac{\partial \hat{y}_4}{\partial z_4^5}\frac{\partial z_4^5}{\partial a_2^4}\frac{\partial a_2^4}{\partial z_2^4}\frac{\partial z_2^4}{\partial w_{12}^3}$$
$$\frac{\partial J(w,b)}{\partial w_{12}^3} = \delta_1^5 w_{21}^4\sigma'\left(z_2^4\right)a_1^3 + \delta_2^5 w_{22}^4\sigma'\left(z_2^4\right)a_1^3 + \delta_3^5 w_{23}^4\sigma'\left(z_2^4\right)a_1^3 + \delta_4^5 w_{24}^4\sigma'\left(z_2^4\right)a_1^3$$

The same procedure for the other derivatives from $W^3$:

$$\frac{\partial J(w,b)}{\partial w_{21}^3} = \delta_1^5 w_{11}^4\sigma'\left(z_1^4\right)a_2^3 + \delta_2^5 w_{12}^4\sigma'\left(z_1^4\right)a_2^3 + \delta_3^5 w_{13}^4\sigma'\left(z_1^4\right)a_2^3 + \delta_4^5 w_{14}^4\sigma'\left(z_1^4\right)a_2^3$$
$$\frac{\partial J(w,b)}{\partial w_{22}^3} = \delta_1^5 w_{21}^4\sigma'\left(z_2^4\right)a_2^3 + \delta_2^5 w_{22}^4\sigma'\left(z_2^4\right)a_2^3 + \delta_3^5 w_{23}^4\sigma'\left(z_2^4\right)a_2^3 + \delta_4^5 w_{24}^4\sigma'\left(z_2^4\right)a_2^3$$
$$\frac{\partial J(w,b)}{\partial w_{31}^3} = \delta_1^5 w_{11}^4\sigma'\left(z_1^4\right)a_3^3 + \delta_2^5 w_{12}^4\sigma'\left(z_1^4\right)a_3^3 + \delta_3^5 w_{13}^4\sigma'\left(z_1^4\right)a_3^3 + \delta_4^5 w_{14}^4\sigma'\left(z_1^4\right)a_3^3$$
$$\frac{\partial J(w,b)}{\partial w_{32}^3} = \delta_1^5 w_{21}^4\sigma'\left(z_2^4\right)a_3^3 + \delta_2^5 w_{22}^4\sigma'\left(z_2^4\right)a_3^3 + \delta_3^5 w_{23}^4\sigma'\left(z_2^4\right)a_3^3 + \delta_4^5 w_{24}^4\sigma'\left(z_2^4\right)a_3^3$$

where:

$$\delta_1^4 = \sum_k^4 \delta_k^5 w_{1k}^4 \cdot \sigma'\left(z_1^4\right); \qquad \delta_2^4 = \sum_k^4 \delta_k^5 w_{2k}^4 \cdot \sigma'\left(z_2^4\right);$$

SK, 10/26/2020

Delta values could be written in matrix form:

$$\begin{bmatrix} \delta_1^4 & \delta_2^4 \end{bmatrix} = \begin{bmatrix} \delta_1^5 & \delta_2^5 & \delta_3^5 & \delta_4^5 \end{bmatrix} \cdot \begin{bmatrix} w_{11}^4 & w_{21}^4 \\ w_{12}^4 & w_{22}^4 \\ w_{13}^4 & w_{23}^4 \\ w_{14}^4 & w_{24}^4 \end{bmatrix} \odot \begin{bmatrix} \sigma'\left(z_1^4\right) & \sigma'\left(z_2^4\right) \end{bmatrix}$$

$$\delta^4 = \delta^5 \cdot \left(W^4\right)^T \odot \sigma'\left(Z^4\right)$$

Derivatives could be also written as:

$$\frac{\partial J(w,b)}{\partial w_{ij}^3} = \delta_j^4 a_i^3$$

and in matrix form:

$$\frac{\partial J(w,b)}{\partial W^3} = \begin{bmatrix} \frac{\partial J(w,b)}{\partial w_{11}^3} & \frac{\partial J(w,b)}{\partial w_{12}^3} \\ \frac{\partial J(w,b)}{\partial w_{21}^3} & \frac{\partial J(w,b)}{\partial w_{22}^3} \\ \frac{\partial J(w,b)}{\partial w_{31}^3} & \frac{\partial J(w,b)}{\partial w_{32}^3} \end{bmatrix} = \begin{bmatrix} a_1^3 \\ a_2^3 \\ a_3^3 \end{bmatrix} \cdot \begin{bmatrix} \delta_1^4 & \delta_2^4 \end{bmatrix}$$

$$\frac{\partial J(w,b)}{\partial W^3} = \left(A^3\right)^T \cdot \delta^4$$

Derivatives w.r.t. the biases:

$$\frac{\partial J(w,b)}{\partial b_1^4} = \frac{\partial J(w,b)}{\partial \hat{y}_1} \frac{\partial \hat{y}_1}{\partial z_1^5} \frac{\partial z_1^5}{\partial a_1^4} \frac{\partial a_1^4}{\partial z_1^4} \frac{\partial z_1^4}{\partial b_1^4} + \frac{\partial J(w,b)}{\partial \hat{y}_2} \frac{\partial \hat{y}_2}{\partial z_2^5} \frac{\partial z_2^5}{\partial a_1^4} \frac{\partial a_1^4}{\partial z_1^4} \frac{\partial z_1^4}{\partial b_1^4}$$
$$\frac{\partial J(w,b)}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial z_3^5} \frac{\partial z_3^5}{\partial a_1^4} \frac{\partial a_1^4}{\partial z_1^4} \frac{\partial z_1^4}{\partial b_1^4} + \frac{\partial J(w,b)}{\partial \hat{y}_4} \frac{\partial \hat{y}_4}{\partial z_4^5} \frac{\partial z_4^5}{\partial a_1^4} \frac{\partial a_1^4}{\partial z_1^4} \frac{\partial z_1^4}{\partial b_1^4}$$

$$\frac{\partial J(w,b)}{\partial b_1^4} = \delta_1^5 w_{11}^4 \sigma'\left(z_1^4\right) + \delta_2^5 w_{12}^4 \sigma'\left(z_1^4\right) + \delta_3^5 w_{13}^4 \sigma'\left(z_1^4\right) + \delta_4^5 w_{14}^4 \sigma'\left(z_1^4\right)$$

$$\frac{\partial J(w,b)}{\partial b_2^4} = \frac{\partial J(w,b)}{\partial \hat{y}_1}\frac{\partial \hat{y}_1}{\partial z_1^5}\frac{\partial z_1^5}{\partial a_2^4}\frac{\partial a_2^4}{\partial z_2^4}\frac{\partial z_2^4}{\partial b_2^4} + \frac{\partial J(w,b)}{\partial \hat{y}_2}\frac{\partial \hat{y}_2}{\partial z_2^5}\frac{\partial z_2^5}{\partial a_2^4}\frac{\partial a_2^4}{\partial z_2^4}\frac{\partial z_2^4}{\partial b_2^4}$$
$$\frac{\partial J(w,b)}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial z_3^5}\frac{\partial z_3^5}{\partial a_2^4}\frac{\partial a_2^4}{\partial z_2^4}\frac{\partial z_2^4}{\partial b_2^4} + \frac{\partial J(w,b)}{\partial \hat{y}_4}\frac{\partial \hat{y}_4}{\partial z_4^5}\frac{\partial z_4^5}{\partial a_2^4}\frac{\partial a_2^4}{\partial z_2^4}\frac{\partial z_2^4}{\partial b_2^4}$$
$$\frac{\partial J(w,b)}{\partial b_2^4} = \delta_1^5 w_{21}^4 \sigma'\left(z_2^4\right) + \delta_2^5 w_{22}^4 \sigma'\left(z_2^4\right) + \delta_3^5 w_{23}^4 \sigma'\left(z_2^4\right) + \delta_4^5 w_{24}^4 \sigma'\left(z_2^4\right)$$

written in a matrix form:

$$\frac{\partial J(w,b)}{\partial B^4} = \begin{bmatrix} \frac{\partial J(w,b)}{\partial b_1^4} & \frac{\partial J(w,b)}{\partial b_2^4} \end{bmatrix} = \begin{bmatrix} \delta_1^4 & \delta_2^4 \end{bmatrix}$$
$$\frac{\partial J(w,b)}{\partial B^4} = \delta^4$$

**General expressions in backpropagation**

There is no general expession for derivatives w.r.t. all weights or biases but at least we can create three groups of general expressions.

First step (derivatives w.r.t. the weights between last to layers and biases in the last layer):

$$\delta_j^L = \left(\hat{y}_j - y_j\right)\sigma'\left(z_j^L\right); \qquad \delta^L = \left(\hat{Y}_j - \hat{Y}_j\right)\sigma'\left(z^L\right)$$

$$\frac{\partial J(w,b)}{\partial w_{ij}^{L-1}} = \delta_j^L a_i^{L-1}; \qquad \frac{\partial J(w,b)}{\partial W^{L-1}} = \left(A^{L-1}\right)^T \cdot \delta^L$$

$$\frac{\partial J(w,b)}{\partial b_j^L} = \delta_j^L; \qquad \frac{\partial J(w,b)}{\partial B^L} = \delta^L$$

Middle steps (derivatives w.r.t. all other weights and biases except the first one):

$$\delta_j^l = \sum_{k=1}^{n^{l+1}} \delta_k^{l+1} w_{jk}^l \cdot \sigma'\left(z_j^l\right); \qquad \delta^l = \delta^{l+1} \cdot \left(W^l\right)^T \square\ \sigma'\left(Z^l\right)$$

$$\frac{\partial J(w,b)}{\partial w_{ij}^{l-1}} = \delta_j^l a_i^{l-1}; \qquad \frac{\partial J(w,b)}{\partial W^{l-1}} = \left(A^{l-1}\right)^T \cdot \delta^l$$

$$\frac{\partial J(w,b)}{\partial b_j^l} = \delta_j^l; \qquad \frac{\partial J(w,b)}{\partial B^l} = \delta^l$$

SK, 10/26/2020

The last step (derivatives w.r.t. the weights between first two layers and biases of the second one):

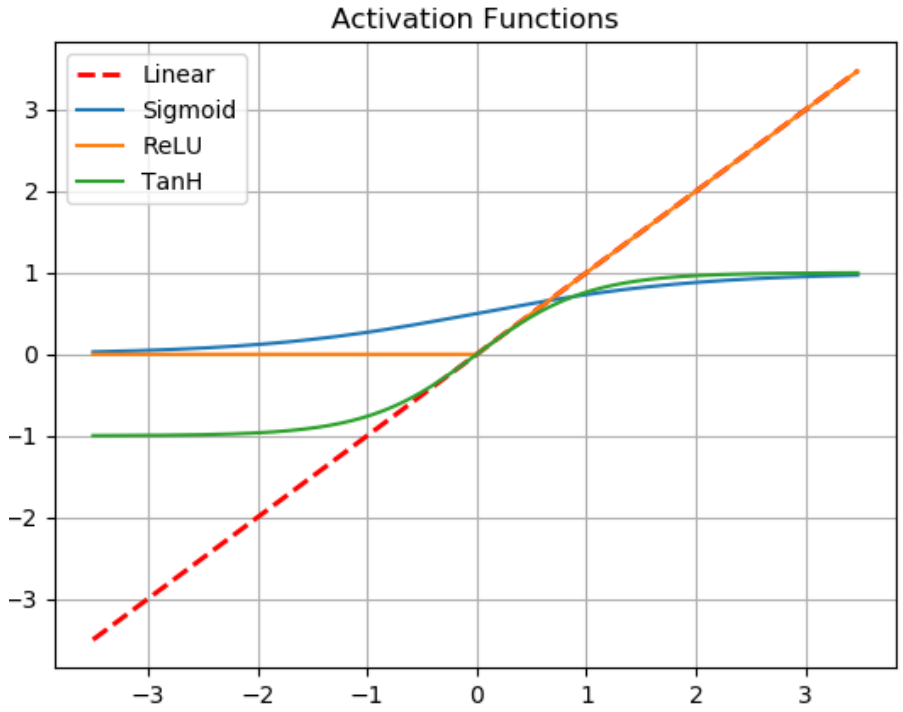
$$\delta_j^2 = \sum_{k=1}^{n^3} \delta_k^3 w_{jk}^2 \cdot \sigma'\left(z_j^2\right); \qquad \delta^2 = \delta^3 \cdot \left(W^2\right)^T \square \ \sigma'\left(Z^2\right)$$

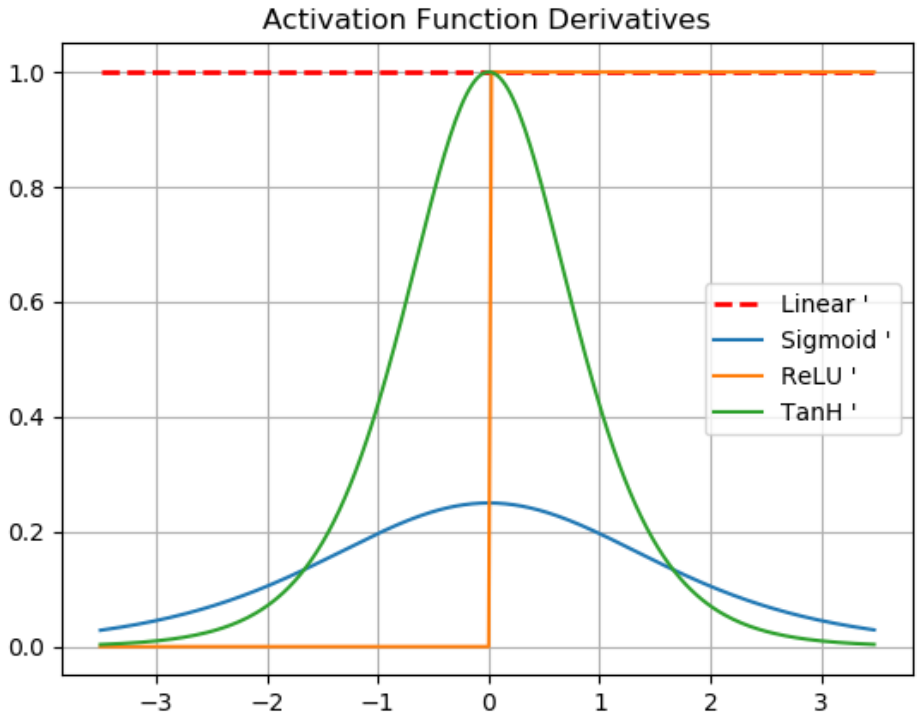
$$\frac{\partial J(w,b)}{\partial w_{ij}^1} = \delta_j^2 x_i; \qquad \frac{\partial J(w,b)}{\partial W^1} = X^T \cdot \delta^2$$

$$\frac{\partial J(w,b)}{\partial b_j^2} = \delta_j^2; \qquad \frac{\partial J(w,b)}{\partial B^2} = \delta^2$$

SK, 10/26/2020

**Exercise:** Activation Functions



Based on the figures above some conclusions could be made:

1. **Linear** function: Input and output of a neuron are the same values. It is good for the problems where a linear function as output is expected. On the other side, it is almost useless for non-linear problems. Its derivative is constant and does not make contribution in backpropagation. It is obvious from the above written general expressions. Output domain: $(-\infty, +\infty)$.
2. **Sigmoid** function: Non-linear activation function with output domain $(0,1)$. Unlike linear function derivative of Sigmoid is another non-constant function that makes contribution in backpropagation. Sigmoid reduces large values close to 1 and small values close to 0. For enough large and small values derivative goes to zero (vanishing gradient problem). Derivative of Sigmoid is even function.
3. **ReLU** function: In range $(-\infty, 0)$ function is constant and equal to 0 and in range $(0, +\infty)$ is linear function where output is equal to input neuron values. This function turns off neurons with negative input value. For the same initial inputs (weights, biases), if all input values of the neurons stay positive during bacpropagation, linear and ReLU function give the same output. Since this function is not smooth, its derivative is not defined at 0.
4. **TanH** function: Everything written for Sigmoid function is the same for this function except domain. TanH output domain is $(-1,1)$. Derivative is a bit steeper than derivative of Sigmoid function with maximum value of 1 when function is equal to 0. Function itself is odd.

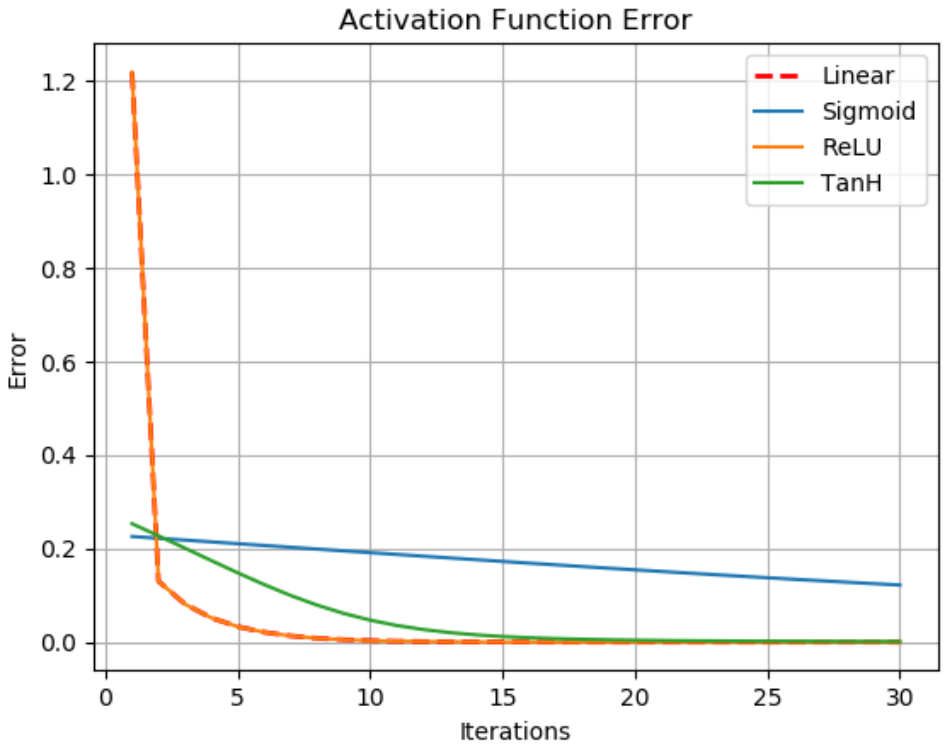


Figure above shows convergence of the output for same initial values (weights, biases) for the first 30 iterations. All positive initial values and linear mapping make linear and ReLU functions give the same output. Convergency based these two are the fastest functions and Sigmoid is the slowest one. Fast convergence of the linear and ReLU function could be explained with the fact that both are linear and used to describe the linear problem. Non-linear function could be used for linear problems but not so efficiently as linear activation functions. On the figure “Activation Functions” could be noticed that TanH function is close to linear when input values are close to 0 and have the same slope when input is 0. That probably makes it better choise for linear problems than Sigmoid.

SK, 10/26/2020

**Example: Neural network with multiple samples**

All examples that we have considered so far are mainly for one data sample. We have derived different models with multiple hidden layers, multiple input elements (features), and multiple output values. We have also introduced some activation functions and biases. Then, the next step is to deal with multiple samples. This would conclude all examples and we will start dealing with more complex problems.

In general, everything that we have derived before will be the same, all expressions and derivations will stay unchanged. The only difference is that now we have multiple samples that we need to pass through neural networks. Suppose that we have $n$ *testing* data samples and we need to train a model that gives good predictions for all testing data samples.

Before we start let us introduce some notation. When all testing data go through the neural network, we say that we have completed one *epoch*. Next, we want to know how we will update weights and biases if we have more samples. At this moment, we will consider the following three approaches:

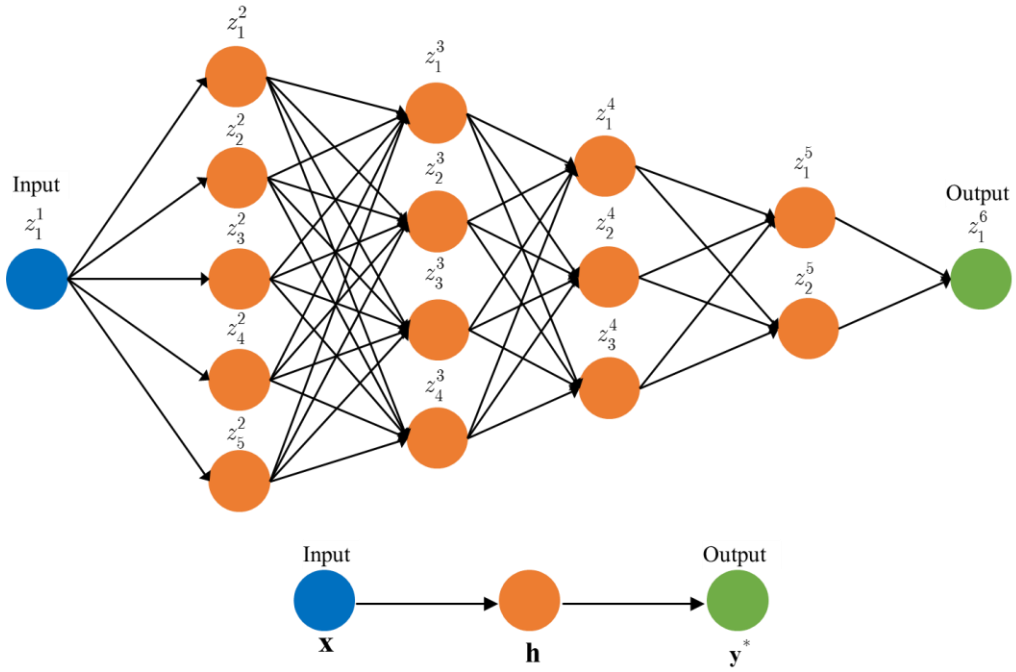
- Batch Gradient Descent (BGD): in this approach all the training data is taken into consideration. We pass all our data through the neural network, compute the gradients for all weights and biases for all the training data, then take the average of the gradients of all the training data and use that mean gradient to update our parameters.
- Stochastic Gradient Descent (SGD): in this approach we consider just one data sample at a time to take a single step. Then we compute the gradient for this data sample and update our parameters. We continue with another data sample and again we update the weights and biases. Basically, we update the weights and biases after each data sample. *For this case, the cost (objective) function will fluctuate over the training data sample and it will decrease with fluctuations*.
- Mini Batch Gradient Descent (MBGD): this approach combines the previous two. We use a batch of a fixed number of training data samples, which is less than the actual dataset. The batch size is the number of samples that are passed to the network at once. For instance, if we have $n = 1000$ data samples, then the size of mini batch can be $4,\ 8,\ 16,\ 32$, or something else. The key difference is that we update our parameters for each mini batch whereas for the BGD approach we update parameters after all date samples, or in case of SGD we update after each data sample. The MBGD approach will also fluctuate over the training data but it will decrease.

SK, 10/26/2020

All three approaches are implemented in the code and they can be described with one code. First, we will loop over the number of epochs, then loop over the number of batches and then, finally, loop over the number of samples in each batch. Thus, if the number of batches is $1$ then we have BGD. If the number of batches is equal to the total number of the training data, then we have SGD. Otherwise, we have MBGD.

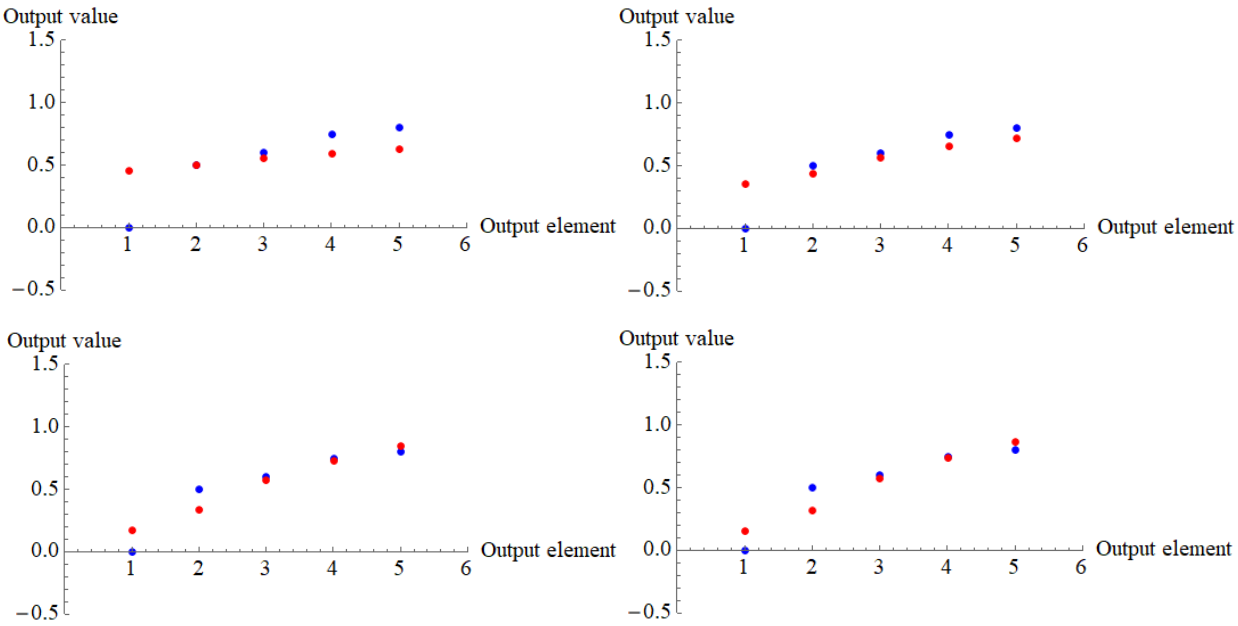
Some results using an artificial neural network with multiple inputs are given below. The considered architecture of the network includes the input layer (one element), 4 hidden layers and the output layer (one element). The learning rate is defined as $h = 0.3$. In this example we have 5 data samples as defined in the table below.

| Input data $x$ | Output data $y$ |
|---|---|
| 0 | 0 |
| 0.20 | 0.50 |
| 0.50 | 0.60 |
| 0.70 | 0.75 |
| 0.85 | 0.80 |



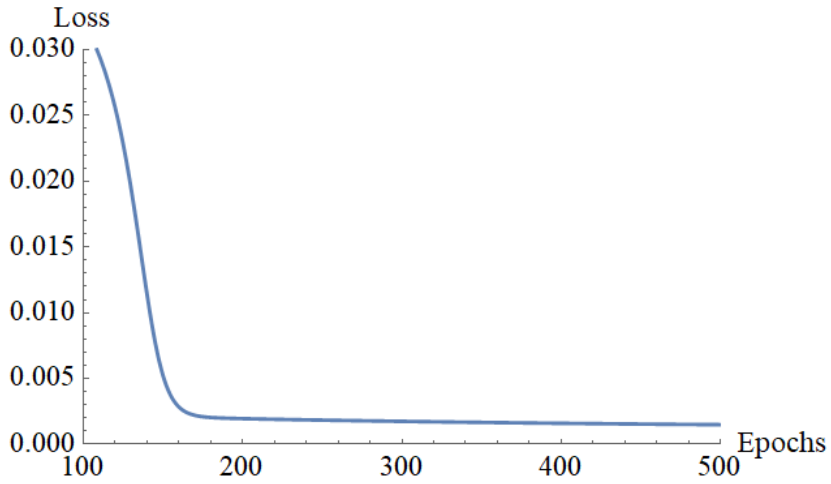

The figure below shows all training data samples in the output vector (blue dots) and predicted values (red dots) obtained using linear activation function for 20, 50, 100, and 200 iterations, respectively. Animation for this is given in the code.

SK, 10/26/2020




The graph below gives the error function versus epochs for 500 iterations in total for hyperbolic tangent activation function. The predicted output vector for 500 iterations is

$\mathbf{y} = [0.0295, 0.4255, 0.6867, 0.7470, 0.7706]$.



SK, 10/26/2020

**Example: Neural network with multiple samples (Math behind Python algorithm)**