ECS 171 Machine Learning

Lecture7: ANN-Back Propagation, Termination Criteria, Activation Functions

Instructor: Dr. Setareh Rafatirad

How to learn a NN?

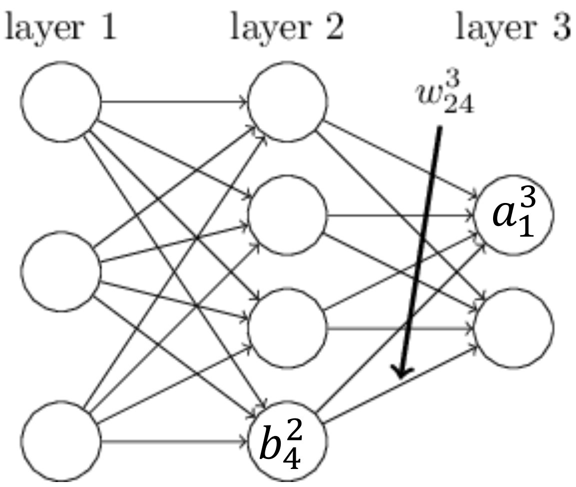
Back Propagation

Back-propagation is a efficient technique for evaluating the gradient of an error function for a FFNN. This technique uses a message-passing scheme in which information is sent alternately forwards and backwards through the network.

Back-Propagation Definition

- Back-propagation is a efficient technique for evaluating the gradient of an error function for a FFNN.
- This technique is achieved through a message-passing scheme in which information is sent alternately forwards and backwards through the network.
- It is also called *error propagation*, or *backprop*.

NN Revisited



w^l_{jk} is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer

N= number of samples

Error function

$$E(w) = \sum_{n=1}^{N} E_n(w)$$

E_n: Error evaluation for the n^{th} sample

k: number of output nodes

$$E_n = \frac{1}{2}\sum_k (\hat{y}_{nk} - y_{nk})^2$$

Sum of Squared Errors

$$SSE = \sum_{records} \sum_{output\ nodes} (actual - predicted)^2$$

$$predicted\ output:\ \hat{y}_k = \sum_i w_{ki} x_i$$

$$actual\ output:\quad y_{nk} = y_k(x_n; w)$$

b^l_j : the bias of the j^{th} neuron in the l^{th} layer.

a^l_j : the activation of the j^{th} neuron in the l^{th} layer.

$$a^l_j = \sigma(\textstyle\sum_k w^l_{jk} z^{l-1}_k + b^l_j)$$

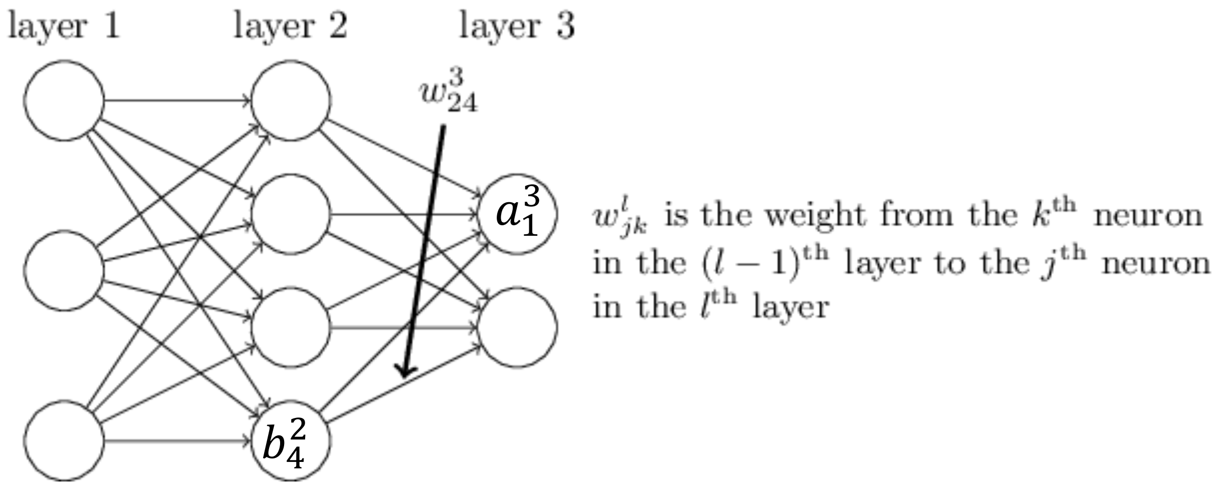
Sigmoid activation function

Gradient of the error function : $\dfrac{\partial E_n}{\partial w_{jk}} = (\hat{y}_{nj} - y_{nj}) x_{nj}$

Matrix Form

$$a_j^l = \sigma(\sum_k w_{jk}^l a_k^{l-1} + b_j^l)$$

- $a^l = activation\ vector for\ the\ l^{th}\ layer$



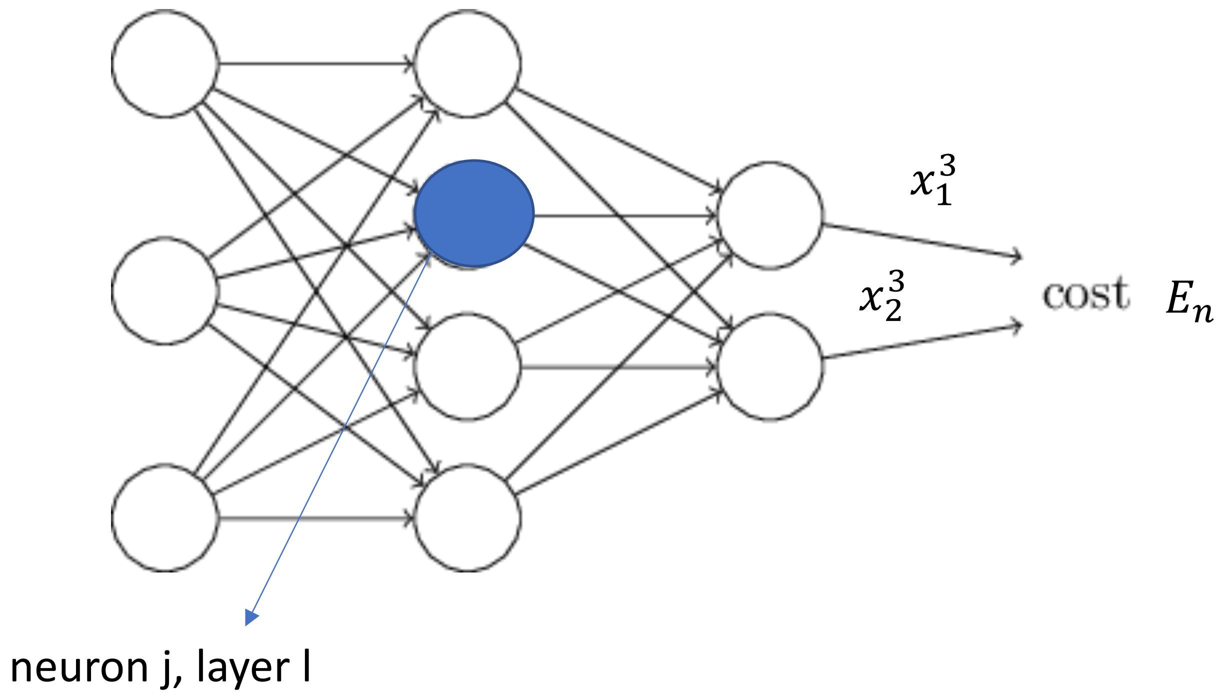
$$a^l = \sigma(w^l a^{l-1} + b^l)$$

Backpropagation

k: number of output neurons

$$E_n = \frac{1}{2}\sum_k (\hat{y}_{nk} - y_{nk})^2$$

This is the error for the n^{th} sample.



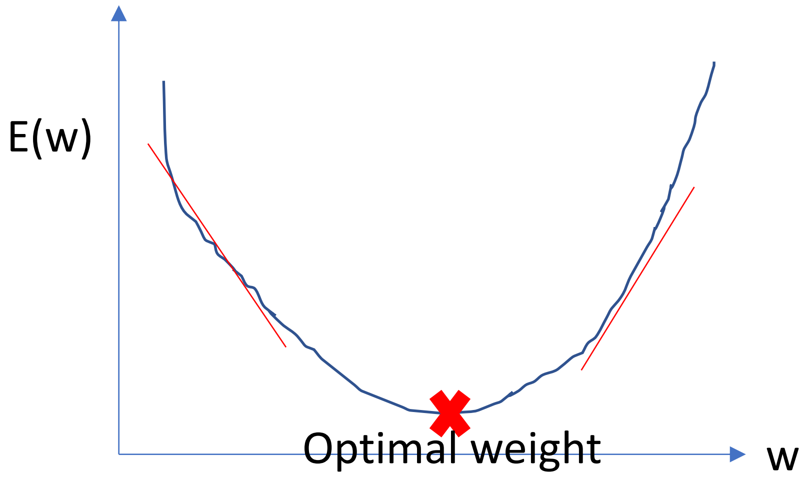
GD Update Rule:

$$w_{new} = w_{current} + \Delta w_{current}$$

A little change to the current value of $w_{current}$ to get closer to the optimal weight

$$\frac{\partial E_n}{\partial w_{jk}} = (\hat{y}_{nj} - y_{nj})x_{nj}$$

$$\frac{\partial E_n}{\partial w_{jk}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{jk}}$$ Chain rule

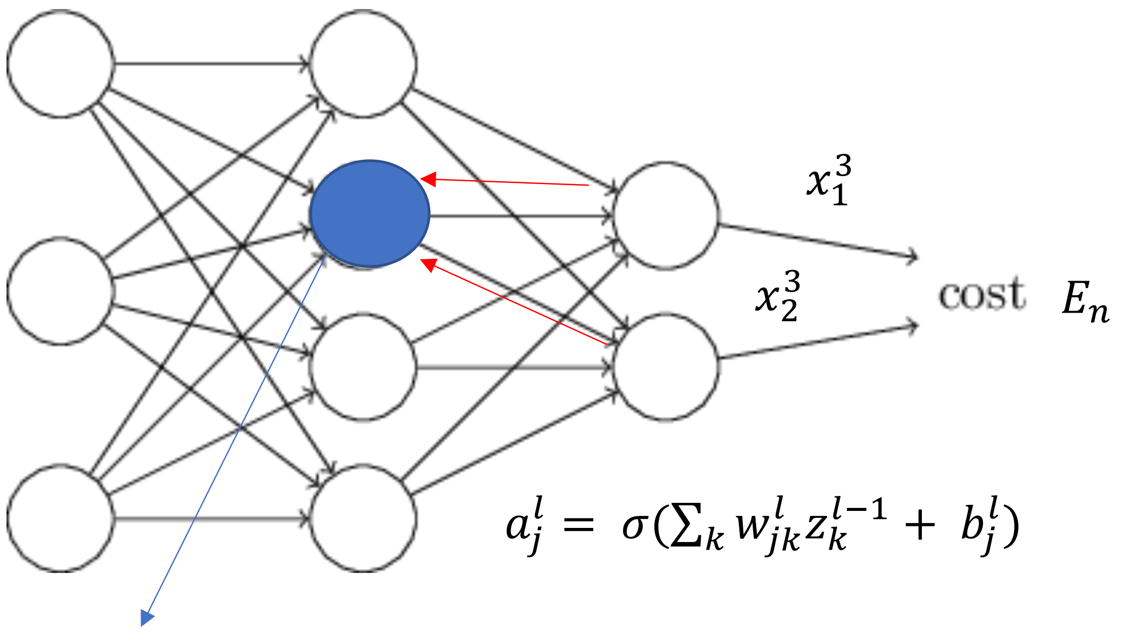


a_j : the activation of the j^{th} neuron in the layer.

Backpropagation

$$E_n = \frac{1}{2}\sum_k (\hat{y}_{nk} - y_{nk})^2$$

$$\frac{\partial E_n}{\partial w_{jk}} = (\hat{y}_{nj} - y_{nj})x_{nj}$$



$a_j^l = \sigma(\sum_k w_{jk}^l z_k^{l-1} + b_j^l)$

neuron j, layer l

$$\frac{\partial E_n}{\partial w_{jk}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{jk}}$$

z_j: activation of the neuron from the previous layer wrt. the edge (weight) connected to node j in the current layer

error of neuron j : δ_j

$$\delta_j = \begin{cases} output_j(1 - output_j)(actual_j - output_j) & \textit{For output layer node j} \\ output_j(1 - output_j)\sum w_{jk}\delta_j & \textit{For hidden layer node j} \end{cases}$$

Weighted sum of the error responsibilities for the nodes downstream from the particular hidden layer node.

$w_{new} = w_{current} + \Delta w_{current}$

$\Delta w_{current} \rightarrow \eta\delta_j z_j$

η : *learning rate*

$output_j$: output of node j

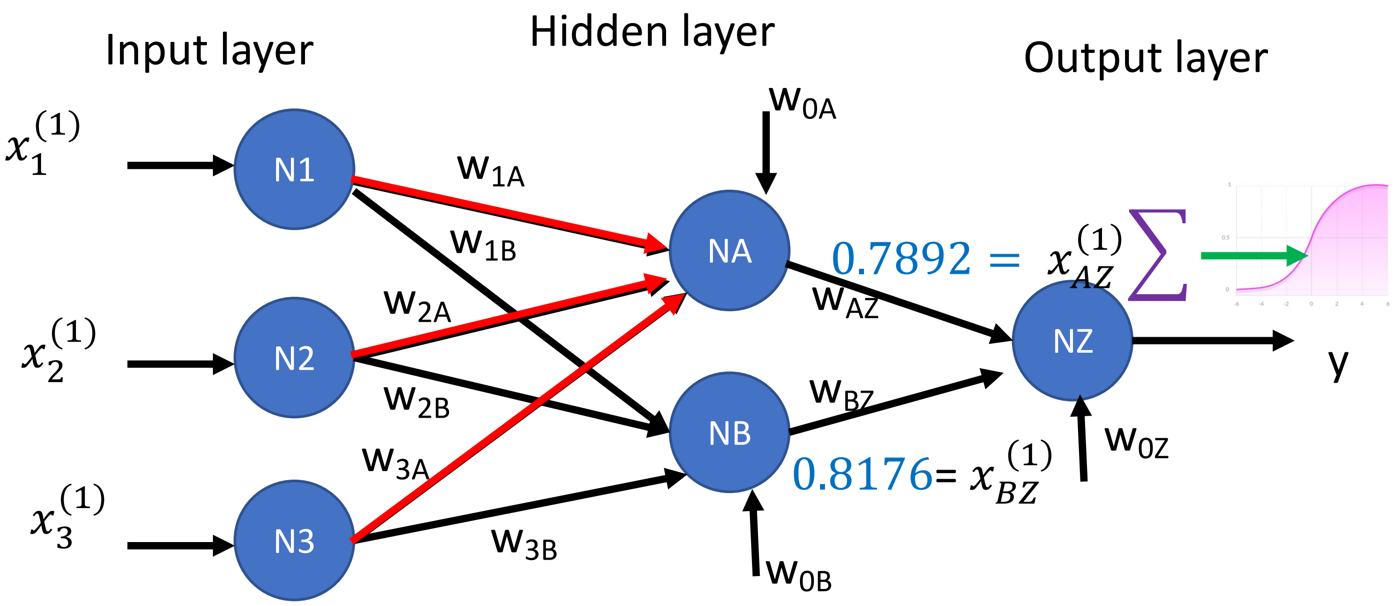
Lecture 5, Slide 18

Basic Example of FFNN: Output Node

$$net^{(1)}{}_Z = \sum_k w_{kZ} x_{kZ}^{(1)} = \omega_{0Z} + \omega_{AZ} x_{AZ}^{(1)} + \omega_{BZ} x_{BZ}^{(1)} = 0.5 + 0.9(0.7892) + 0.9(0.8176) = 1.9461$$

Input to the activation function

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

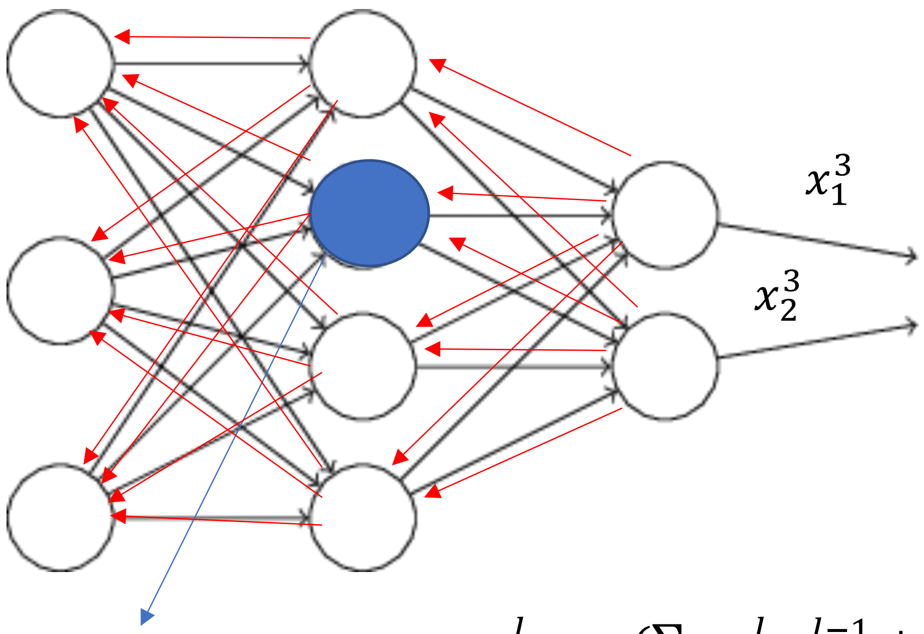


$$f(net_Z) = \frac{1}{1 + e^{-(1.9461)}} = 0.8750$$

Output from the NN for pass 1 through the network , and is the predicted value for the first observation in the dataset D.

Source: Discovering Knowledge in Data D. Larose

Forward-pass Backpropagation



$$a_j^l = \sigma\left(\sum_k w_{jk}^l z_k^{l-1} + b_j^l\right)$$

$$E_n = \frac{1}{2}\sum_k (\hat{y}_{nk} - y_{nk})^2$$

Gradient of the error function

$$\frac{\partial E_n}{\partial w_{jk}} = (\hat{y}_{nj} - y_{nj})x_{nj}$$

$$\frac{\partial E_n}{\partial w_{jk}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{jk}}$$

z_j

$error\ of\ neuron\ j:\ \delta_j$

$$\delta_j = \begin{cases} output_j(1 - output_j)(actual_j - output_j) & \textit{For output layer node} \\ output_j(1 - output_j)\underline{\sum w_{jk}\delta_j} & \textit{For hidden layer nodes} \end{cases}$$

Weighted sum of the error responsibilities for the nodes downstream from the particular hidden layer node.

Update Rule:

$$w_{new} = w_{current} + \Delta w_{current}$$

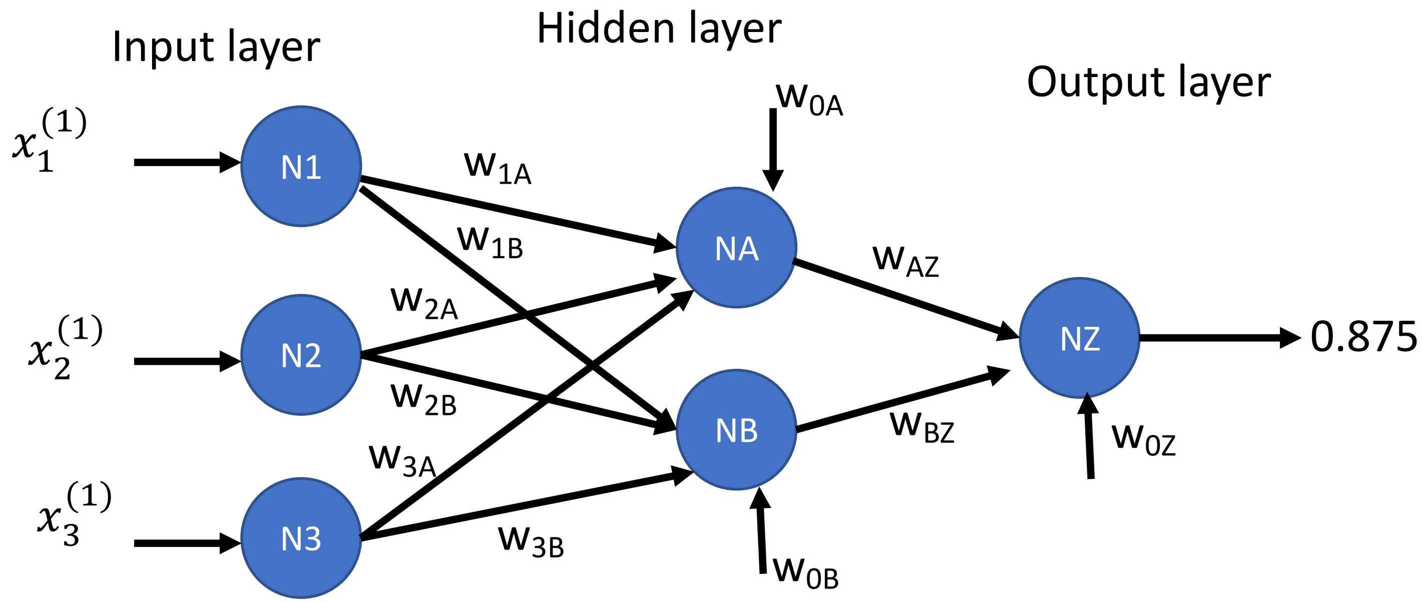
$\eta\delta_j z_j$

$\eta : learning\ rate\ ; 0 \leq \eta \leq 1$

MLP: Backpropagation Example

In this MLP which is a three layer ANN, a simple feed forward network learning happens in 2 phases, a forward pass, and a **backward pass**.

MLP: Multi-Layer Perceptron



w_{ij}: the weight associated with the i^{th} input to node j

x_{ij}: i^{th} input to node j

w_{0A} =0.5	w_{0B} =0.7	w_{0Z}=0.5
w_{1A} = 0.6	w_{1B} = 0.9	w_{AZ}=0.9
w_{2A}=0.8	w_{2B}=0.8	w_{BZ}=0.9
w_{3A}=0.6	w_{3B}=0.4	

N_1 =0.4	N_A =0.7892
N_2 =0.2	N_B =0.8176
N_3 =0.7	N_z =0.875

Source: Discovering Knowledge in Data D. Larose

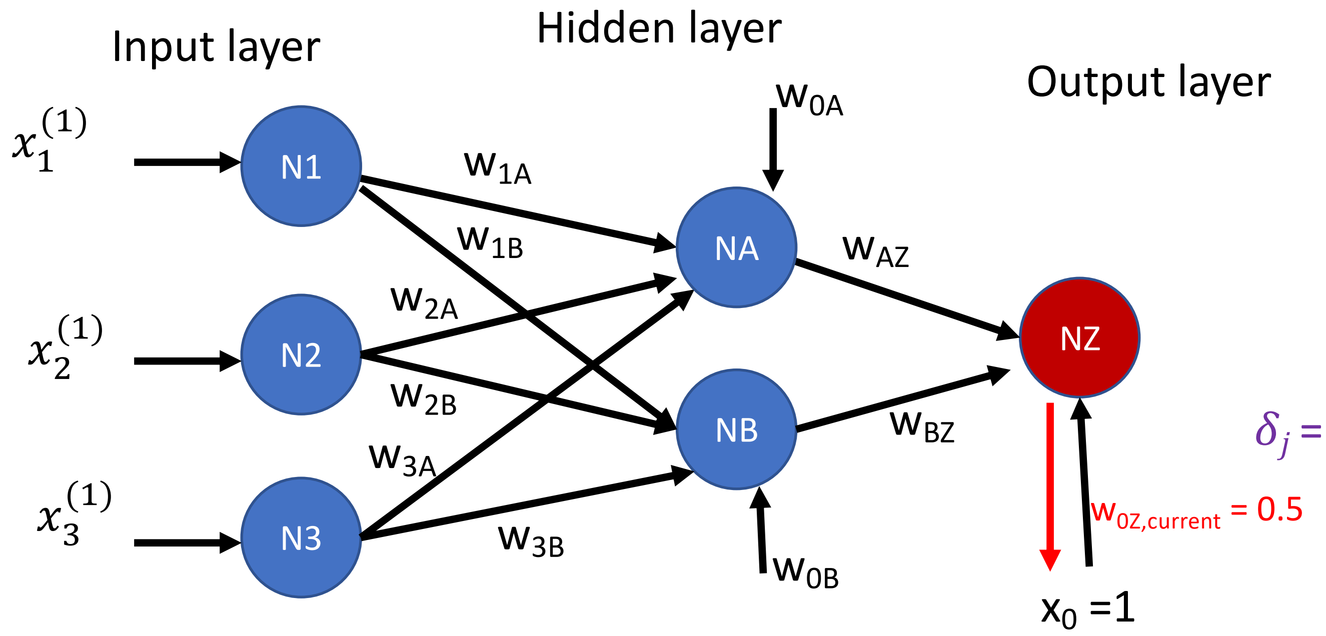
Assume actual y= 0.8 → residual error = 0.8 – 0.875 = -0.075

$$net^{(i)}{}_j = \sum_k w_{kj} x_{kj}^{(i)} \qquad N_j = \frac{1}{1+e^{-x}} = f(net_j)$$

actual$_z$= 0.8 → residual error = 0.8 – 0.875 = -0.075

MLP: Backpropagation Example

N_1 =0.4	N_A =0.7892
N_2 =0.2	N_B =0.8176
N_3 =0.7	N_z =0.875



Source: Discovering Knowledge in Data D. Larose

residual error = 0.8 – 0.875 = -0.075

$$learning\ rate\ ;0\ \leq \eta \leq 1$$
$$Assume\ :\ \eta = 0.1$$

For output layer node

$$\delta_j = \begin{cases} output_j(1-output_j)(actual_j - output_j) \\ output_j(1-output_j) \sum w_{jk}\delta_j \end{cases}$$

For hidden layer nodes

Weighted sum of the error responsibilities for the nodes downstream from the particular hidden layer node.

$$\delta z = N_z(1-N_z)(actual_z - N_z) =$$
0.875 (1-0.875) (-0.075) = -0.0082

Update Rule:

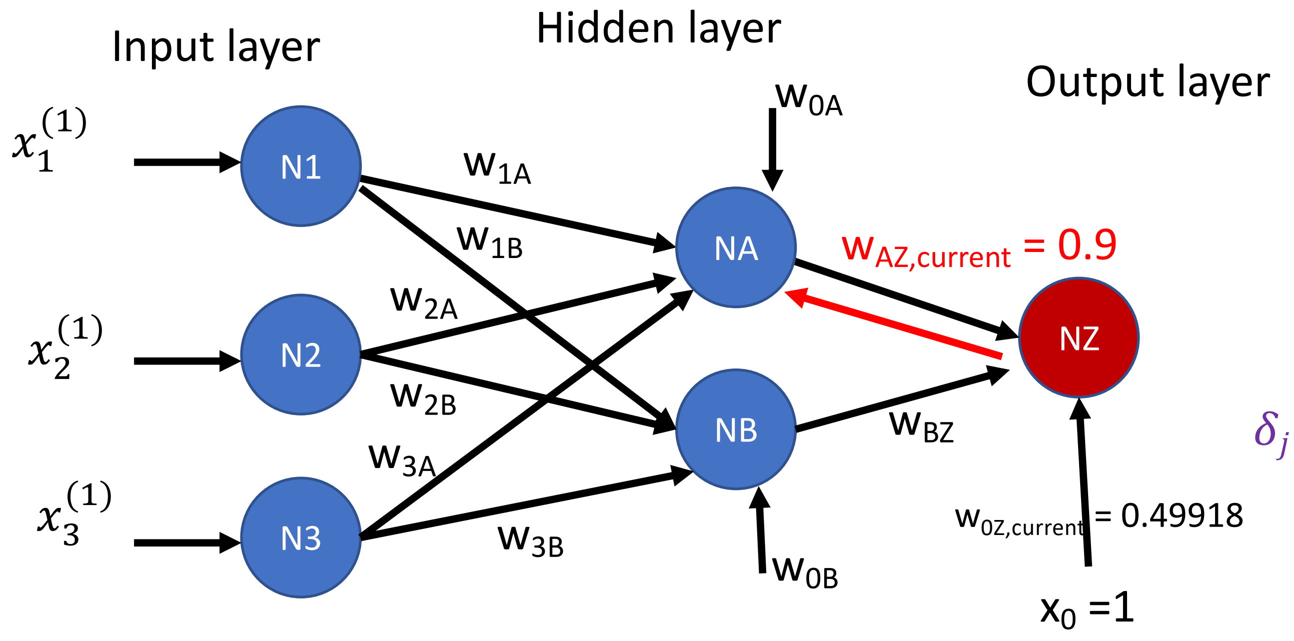
$$w_{0Z,new} = w_{0Z,current} + \Delta w_{0Z} = 0.49918$$

0.5 $\qquad$ $\eta\delta_z x_0$ = 0.1 (-0.0082) 1 = -0.00082

actual$_z$= 0.8 → residual error = 0.8 – 0.875 = -0.075

MLP: Backpropagation Example

N_1 =0.4	N_A =0.7892
N_2 =0.2	N_B =0.8176
N_3 =0.7	N_z =0.875



residual error = 0.8 – 0.875 = -0.075

$$learning\ rate\ ; 0\ \leq \eta \leq 1$$
$$Assume\ :\ \eta = 0.1$$

For output layer node

$$\delta_j = \begin{cases} output_j(1 - output_j)(actual_j - output_j) \\ output_j(1 - output_j)\ \underline{\sum w_{jk}\delta_j} \end{cases}$$

For hidden layer nodes

Weighted sum of the error responsibilities for the nodes downstream from the particular hidden layer node.

Source: Discovering Knowledge in Data D. Larose

Update Rule:

$$w_{AZ,new} = w_{AZ,current} + \Delta w_{AZ} = 0.899353$$

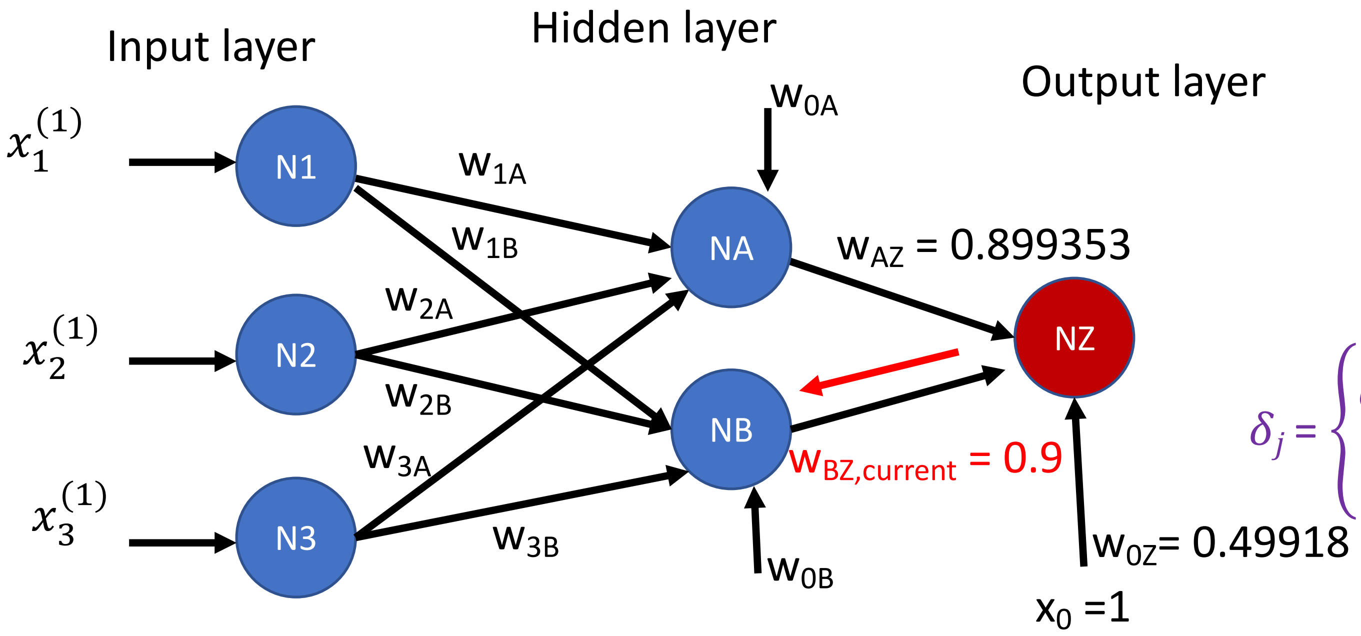
0.9

$\eta\delta_Z N_A$= 0.1 (- 0.0082) (0.7892) = -0.000647

actual$_z$= 0.8 → residual error = 0.8 – 0.875 = -0.075

MLP: Backpropagation Example

N_1 =0.4	N_A =0.7892
N_2 =0.2	N_B =0.8176
N_3 =0.7	N_z =0.875



residual error = 0.8 – 0.875 = -0.075

$$learning\ rate\ ;0\ \leq \eta \leq 1$$
$$Assume\ :\ \eta = 0.1$$

For output layer node

$$\delta_j = \begin{cases} output_j(1 - output_j)(actual_j - output_j) \\ output_j(1 - output_j)\ \underline{\sum w_{jk}\delta_j} \end{cases}$$

For hidden layer nodes

Weighted sum of the error responsibilities for the nodes downstream from the particular hidden layer node.

Source: Discovering Knowledge in Data D. Larose

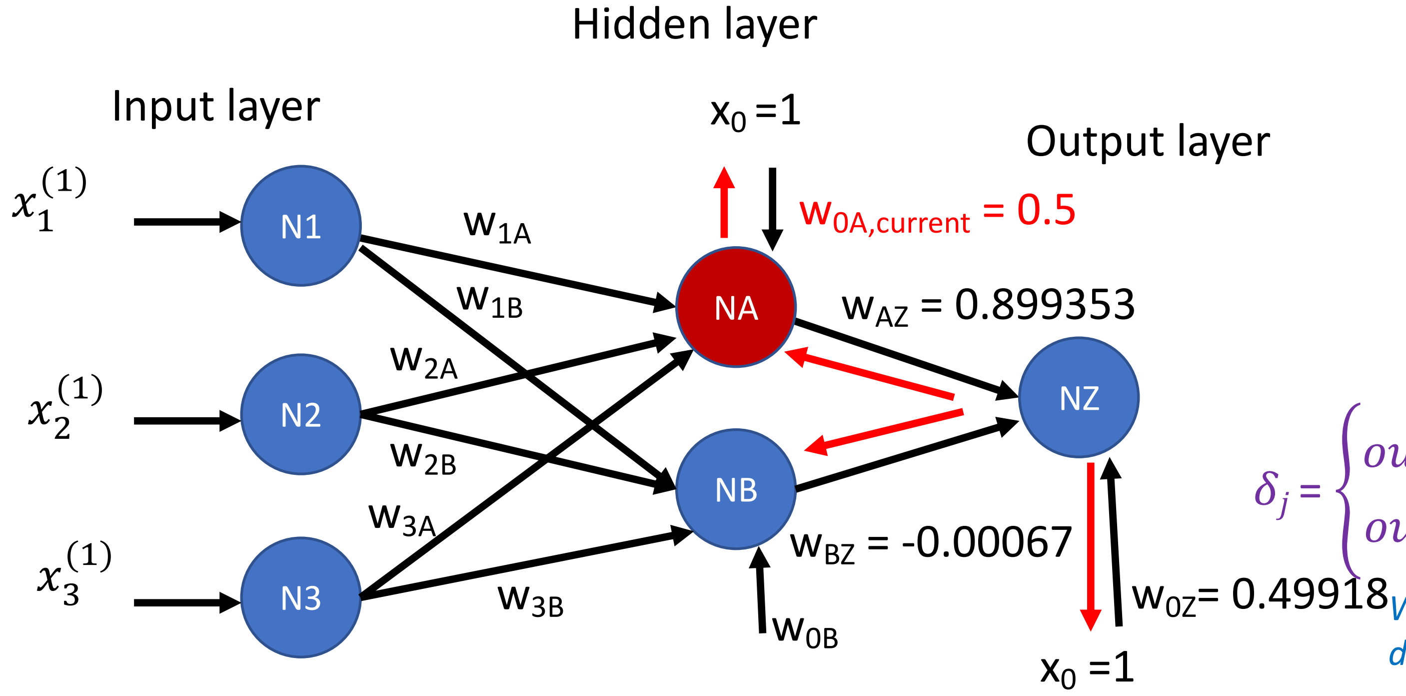
Update Rule:

$$w_{BZ,new} = w_{BZ,current} + \Delta w_{BZ} = 0.899353$$

0.9

$\eta\delta_Z N_B$= 0.1 (- 0.0082) (0.8176) = -0.00067

MLP: Backpropagation Example



Source: Discovering Knowledge in Data D. Larose

actual$_z$= 0.8 → residual error = 0.8 – 0.875 = -0.075

$N_1 = 0.4$	$N_A = 0.7892$
$N_2 = 0.2$	$N_B = 0.8176$
$N_3 = 0.7$	$N_z = 0.875$

residual error = 0.8 – 0.875 = -0.075

$$learning\ rate\ ;0 \le \eta \le 1$$
$$Assume:\ \eta = 0.1$$

For output layer node

$$\delta_j = \begin{cases} output_j(1 - output_j)(actual_j - output_j) \\ output_j(1 - output_j)\ \underline{\sum w_{jk}\delta_j} \end{cases}$$

For hidden layer nodes

Weighted sum of the error responsibilities for the nodes downstream from the particular hidden layer node.

The only node downstream from N_A is N_Z.

$\delta_A = N_A(1 - N_A)(w_{AZ}\delta_Z) =$
0.7892 (1-0.7892) (0.9)(-0.0082) = -0.00123

Update Rule:

$$w_{0A,new} = w_{0A,current} + \Delta w_{0A} = 0.499877$$

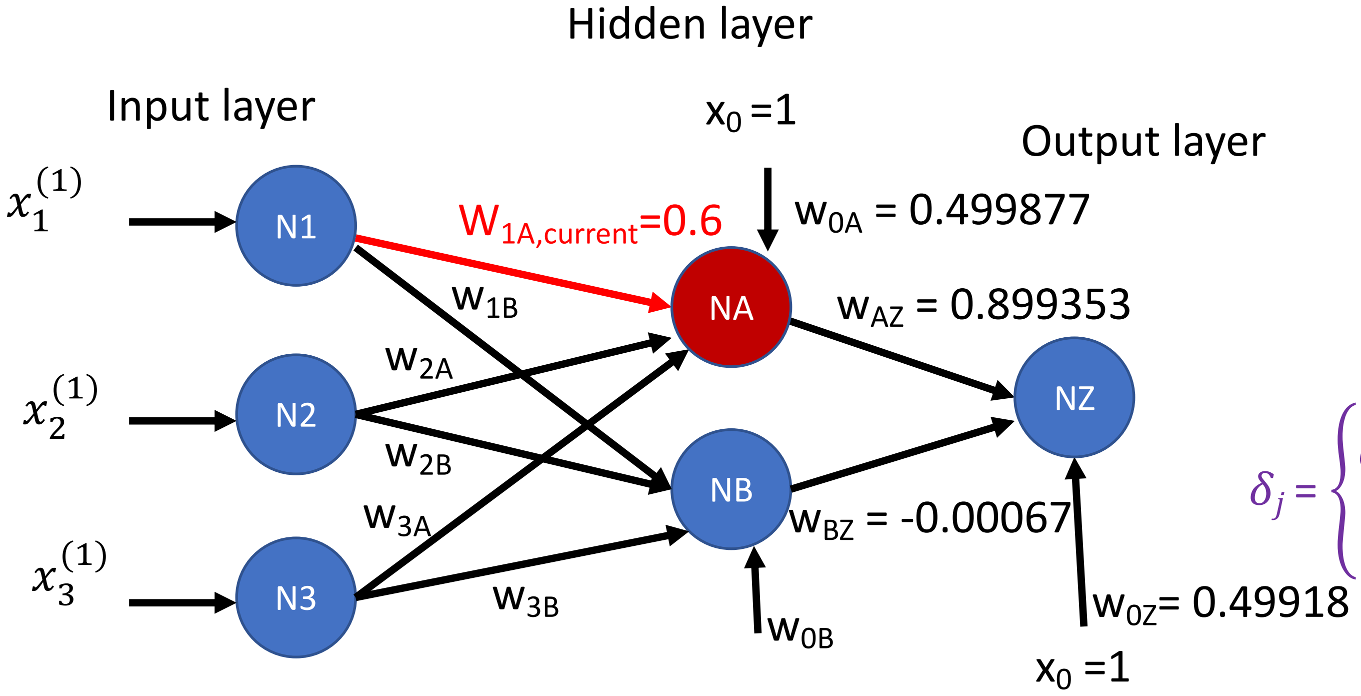
0.5

$\eta\delta_A x_0$= 0.1 (-0.00123) (1) = -0.000123

MLP: Backpropagation Example

actual$_z$= 0.8 → residual error = 0.8 – 0.875 = -0.075

N_1 =0.4	N_A =0.7892
N_2 =0.2	N_B =0.8176
N_3 =0.7	N_z =0.875



residual error = 0.8 – 0.875 = -0.075

$learning\ rate\ ; 0 \leq \eta \leq 1$

$Assume: \ \eta = 0.1$

For output layer node

$$\delta_j = \begin{cases} output_j(1 - output_j)(actual_j - output_j) \\ output_j(1 - output_j) \underline{\sum w_{jk}\delta_j} \end{cases}$$

For hidden layer nodes

Weighted sum of the error responsibilities for the nodes downstream from the particular hidden layer node.

Source: Discovering Knowledge in Data D. Larose

Update Rule:

$$w_{1A,new} = w_{1A,current} + \Delta w_{1A} = 0.5999508$$

Recall: δ_A = -0.00123

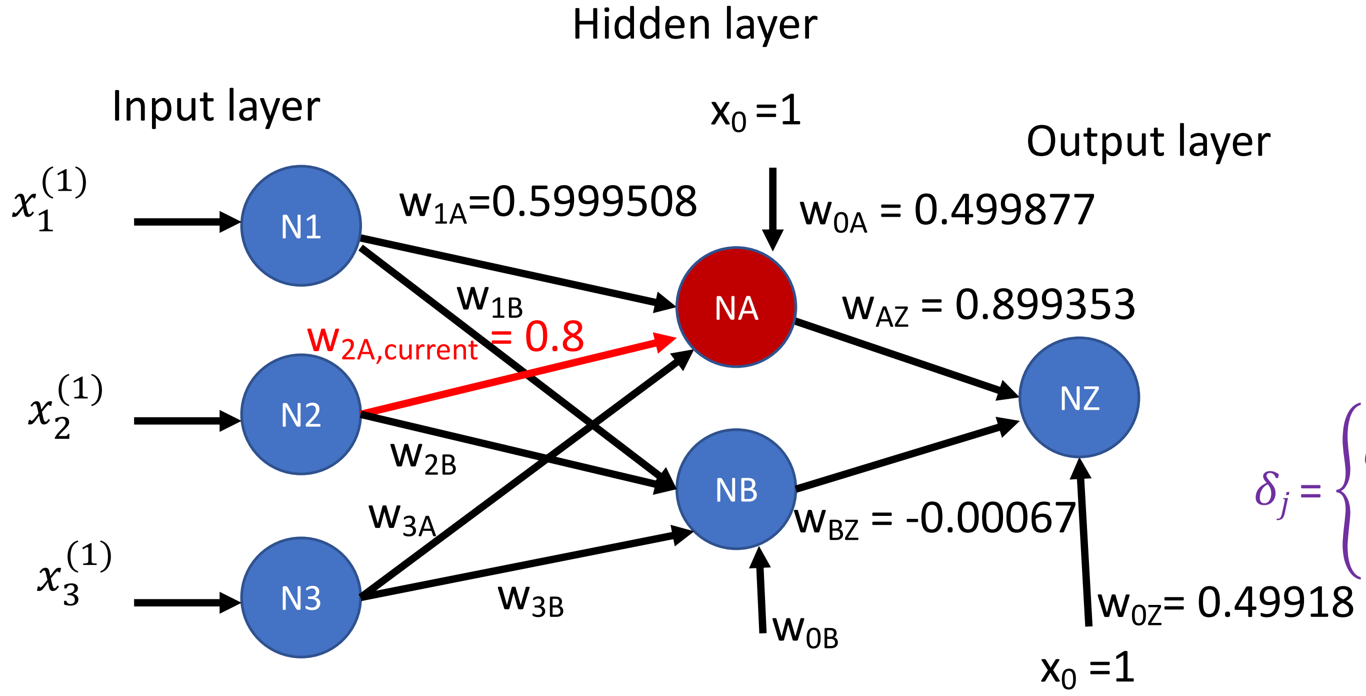
0.6

$\eta\delta_A N_1$= 0.1 (-0.00123) (0.4) = -0.0000492

$actual_z$= 0.8 → residual error = 0.8 – 0.875 = -0.075

MLP: Backpropagation Example

N_1 =0.4	N_A =0.7892
N_2 =0.2	N_B =0.8176
N_3 =0.7	N_z =0.875



residual error = 0.8 – 0.875 = -0.075

$learning\ rate\ ; 0 \leq \eta \leq 1$

$Assume : \eta = 0.1$

For output layer node

$$\delta_j = \begin{cases} output_j(1 - output_j)(actual_j - output_j) \\ output_j(1 - output_j) \underline{\sum w_{jk}\delta_j} \end{cases}$$

For hidden layer nodes

Weighted sum of the error responsibilities for the nodes downstream from the particular hidden layer node.

Source: Discovering Knowledge in Data D. Larose

Update Rule:

$$w_{2A,new} = w_{2A,current} + \Delta w_{2A} = 0.7999754$$

Recall: δ_A = -0.00123

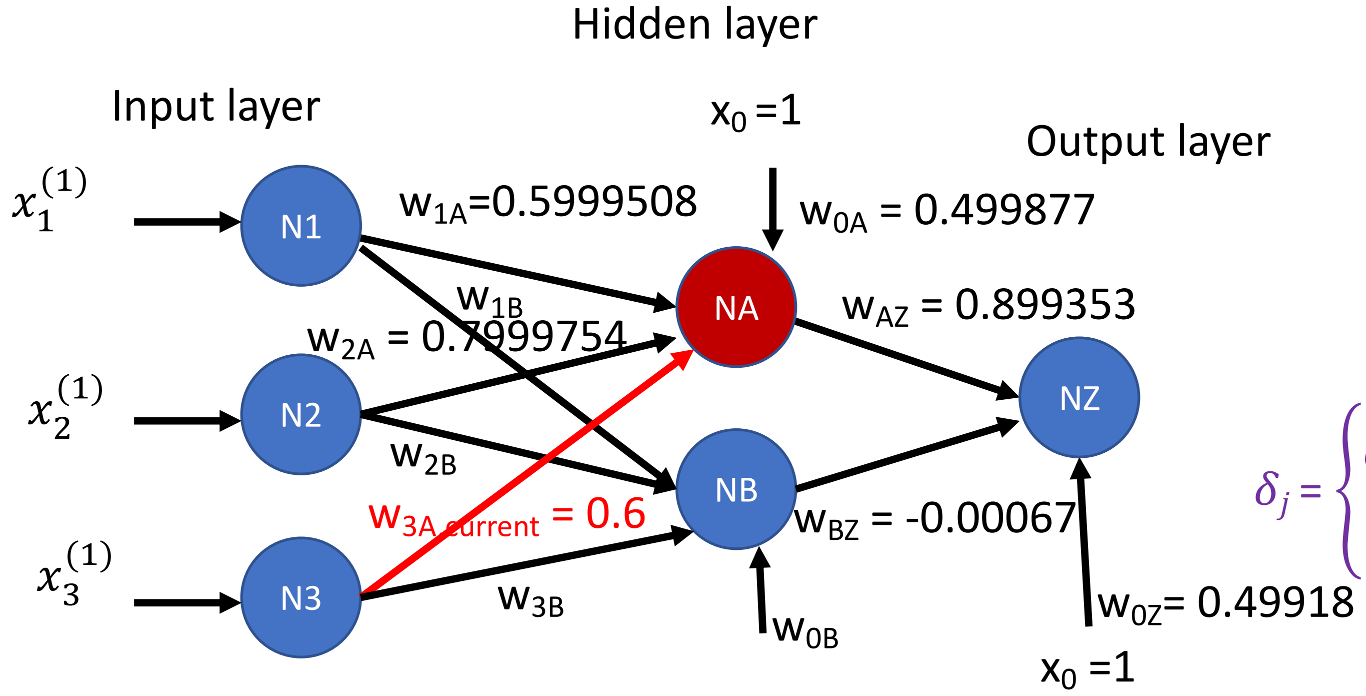
0.8

$\eta\delta_A N_2$= 0.1 (-0.00123) (0.2) = -0.0000246

MLP: Backpropagation Example

actual$_z$= 0.8 → residual error = 0.8 – 0.875 = -0.075

N_1 =0.4	N_A =0.7892
N_2 =0.2	N_B =0.8176
N_3 =0.7	N_z =0.875



residual error = 0.8 – 0.875 = -0.075

$learning\ rate\ ; 0 \leq \eta \leq 1$

$Assume: \ \eta = 0.1$

$$\delta_j = \begin{cases} output_j(1-output_j)(actual_j - output_j) & \textit{For output layer node} \\ output_j(1-output_j)\ \underline{\sum w_{jk}\delta_j} & \textit{For hidden layer nodes} \end{cases}$$

Weighted sum of the error responsibilities for the nodes downstream from the particular hidden layer node.

Source: Discovering Knowledge in Data D. Larose

Update Rule:

$$w_{3A,new} = w_{3A,current} + \Delta w_{3A} = 0.5999139$$

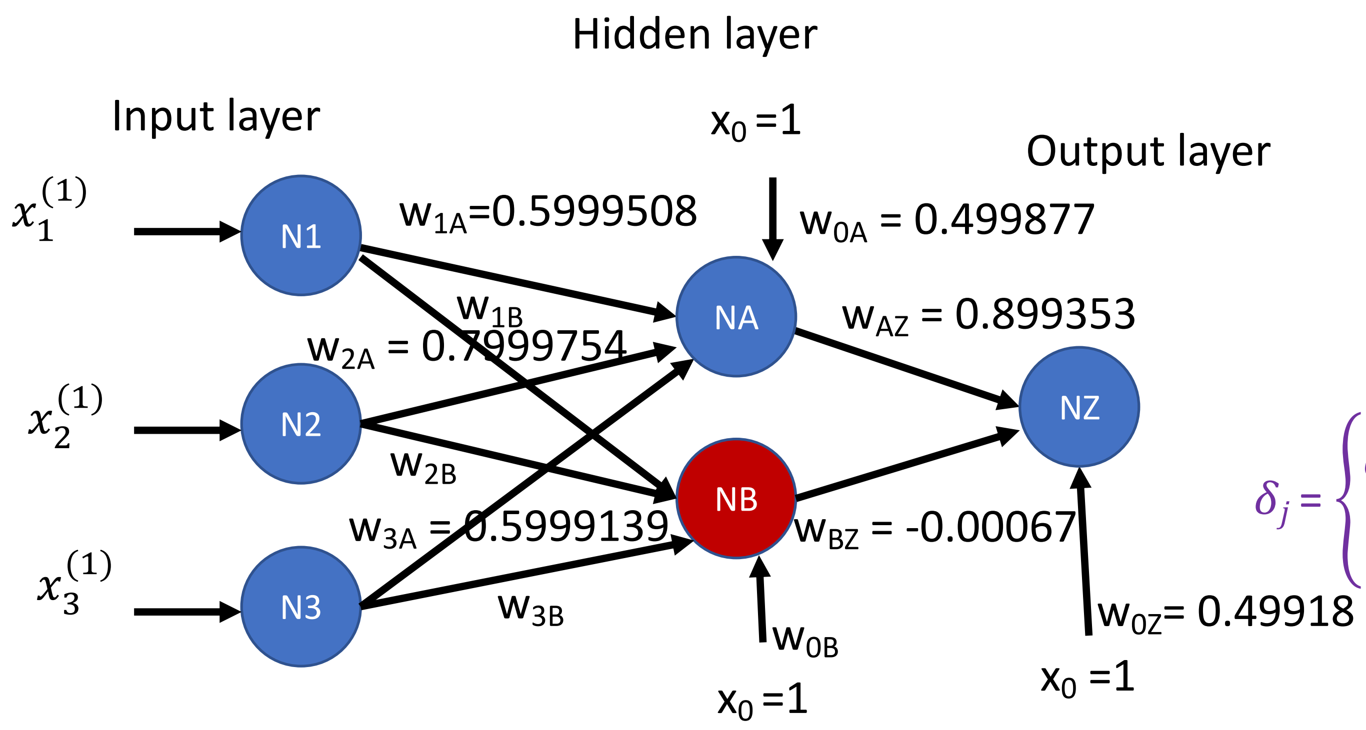
Recall: δ_A = -0.00123

$w_{3A,current}$ → 0.6

Δw_{3A} → $\eta\delta_A N_3$= 0.1 (-0.00123) (0.7) = -0.0000861

actual$_z$= 0.8 → residual error = 0.8 – 0.875 = -0.075

MLP: Backpropagation Example



Source: Discovering Knowledge in Data D. Larose

N_1 =0.4	N_A =0.7892
N_2 =0.2	N_B =0.8176
N_3 =0.7	N_z =0.875

residual error = 0.8 – 0.875 = -0.075

$learning\ rate\ ; 0 \leq \eta \leq 1$

$Assume\ :\ \eta = 0.1$

For output layer node

$$\delta_j = \begin{cases} output_j(1 - output_j)(actual_j - output_j) \\ output_j(1 - output_j) \underline{\sum w_{jk}\delta_j} \end{cases}$$

For hidden layer nodes

Weighted sum of the error responsibilities for the nodes downstream from the particular hidden layer node.

The only node downstream from N_B is N_Z.

$\delta_B = N_B(1 - N_B)(w_{BZ}\delta_Z) =$
0.8176(1-0.8176) (0.9)(-0.0082) = -0.0011

Update Rule:

$$w_{0B,new} = w_{0B,current} + \Delta w_{0B}$$

$$\Delta w_{0B} \rightarrow \eta\delta_B x_j$$

Which weights to update?
Do this on your own as a practice at home.

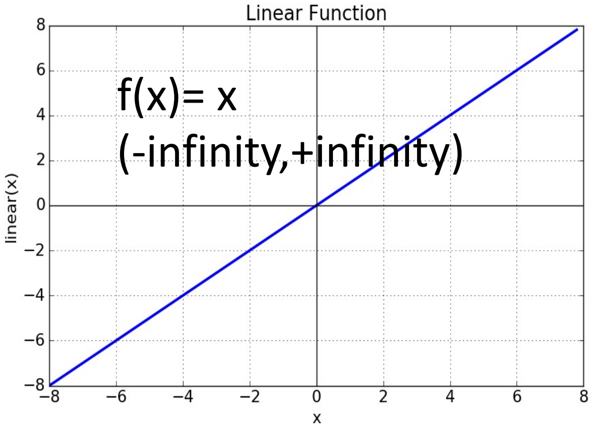
Termination Criteria

- Training time
  - Risk: Degradation in model performance
- Reaching threshold for the prediction error
  - Risk: Overfitting
- Cross validation termination procedure
  - Use part of the dataset as a validation set
  - Train the NN using feedforward and backpropagation and update the weights
  - Apply the weights learned from the training data on validation data
  - Supervise two sets of weights, 1) current set of weights produced by the training data b) best set of weights measured by the lowest prediction error so far on validation data
  - Terminate the algorithm when the current set of weights have significantly greater prediction error (SSE) than the best set of weights

Regardless of the termination criteria used, the NN is not guaranteed to arrive at the global minimum for the SSE as it may become stuck in a local minimum which still represents a good , if not optimal solution.

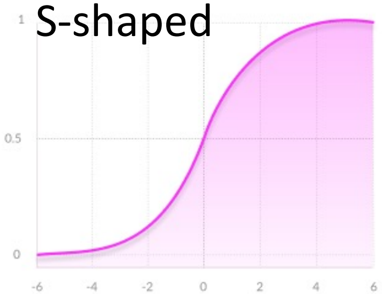
Selection of an Activation Function

- Activation functions determine the output of a NN (determine whether a neuron should ”fire”.
- They can normalize the output of each neuron.
- Computationally efficient
- Increasingly use non-linear functions to learn complex data and provide accurate predictions.
- Linear function doesn’t help with the complexity of data
- Non-Linear activation functions help the model to generalize or adapt with the variety of data
- Examples:
    - Linear function (linear line)
    - Binary step function
    - Non-linear activation functions
    - Sigmoid/logistic, Relu, Parametric Relu, Tanh/hyperbolic tangent , softmax, Swish
    - Swish outperforms Relu in terms of classification accuracy.

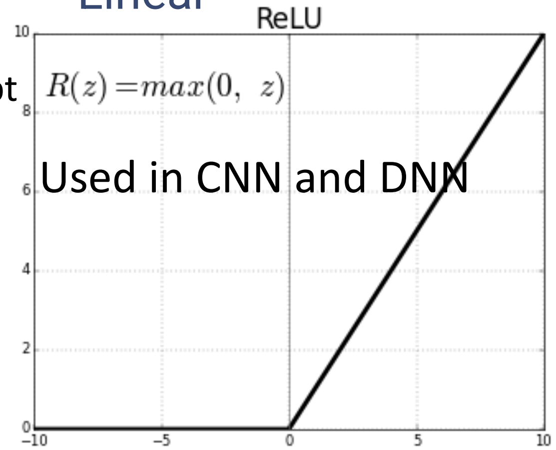


Linear

Output range: [0,1]
S-shaped

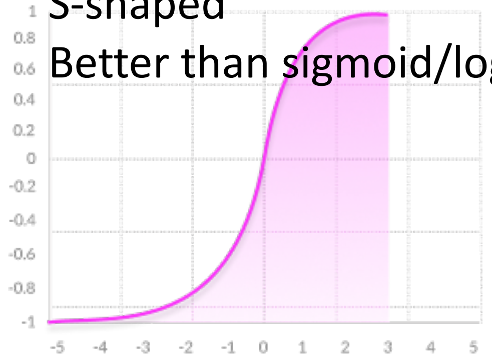


Sigmoid / Logistic



ReLU

Output range: [-1,1]
S-shaped
Better than sigmoid/logistic

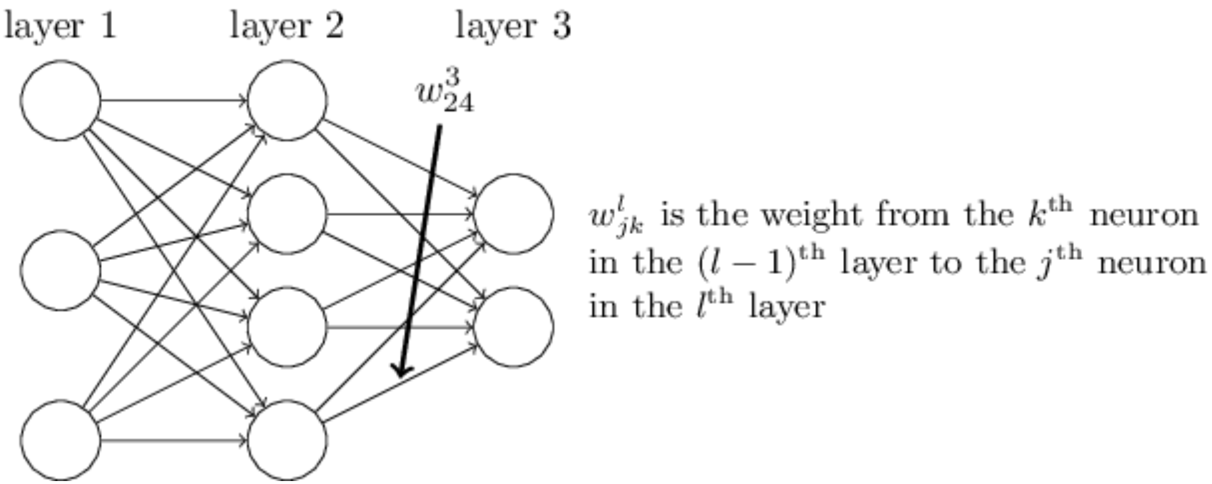


TanH / Hyperbolic Tangent

ANN Applications

- Natural Language Processing
- Computer Vision
- AI

Neural networks increase in accuracy with the number of hidden layers.



A shallow neural network