# Multi Layer Neural Network

Jyoti Aggarwal

Malaviya National Institute of Technology

*2016pcp5285@mnit.ac.in*

*Under The Supervision of : Dr. Neeta Nain*

March 21, 2018

# Overview I

# Overview II

# The Perceptron I

Perceptron's were developed in the 1950s and 1960s by the scientist Frank Rosenblatt. It consists of :-

1. n input nodes $x_1, x_2, .., x_n$ and one output node O.
2. An n-vector X applied at input nodes, in one-to-one mapping.
3. It takes real valued inputs and produces binary output.
   $y = w_1 * x_1 + w_2 * x_2 + w_3 * x_3$

# The Perceptron II

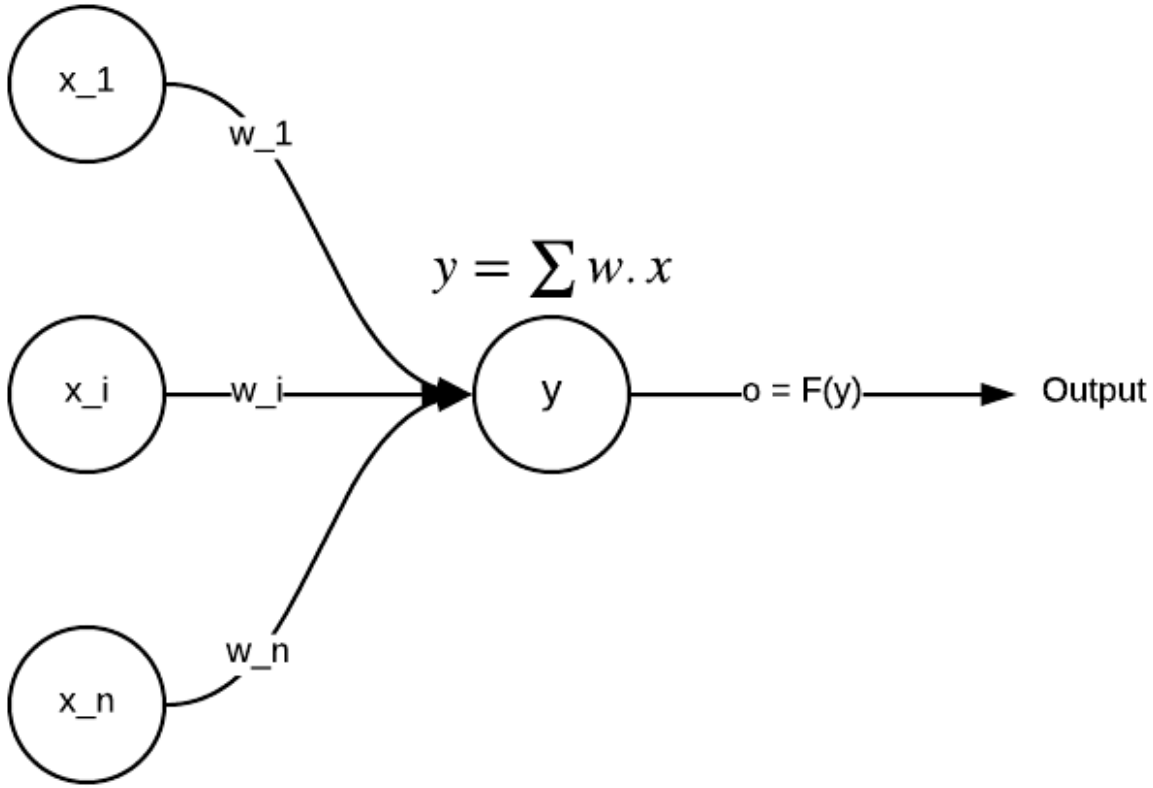


Figure 1: perceptron with 3 inputs (Single Layer Perceptron)

# Multi Layer Perceptron I

A multi-layer perceptron (MLP) has the same structure of a single layer perceptron with one or more hidden layers.

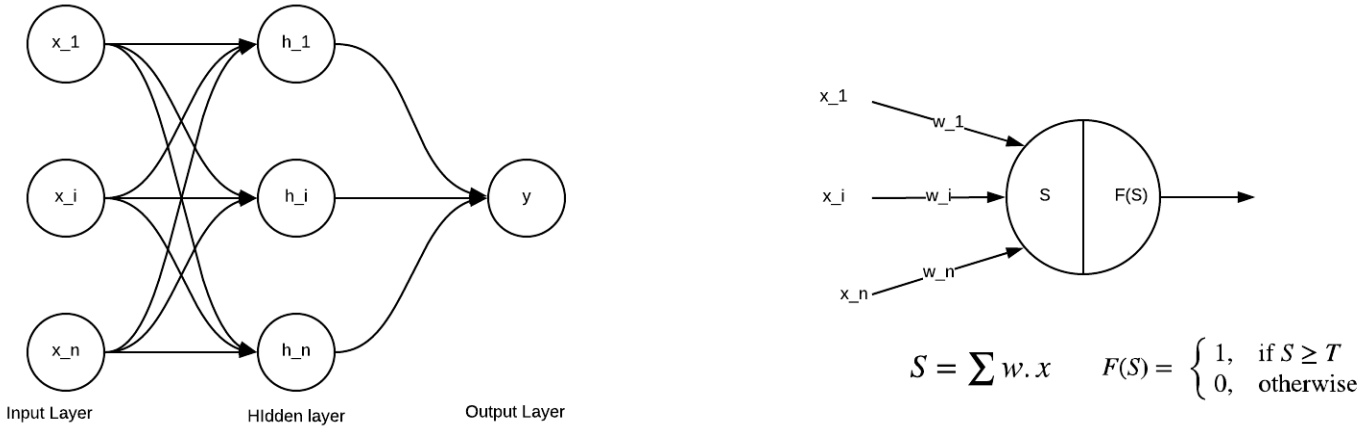


Figure 2: Single Layer Perceptron

# Limitaions of Perceptron I

1. It can classify a set of patterns into two linearly separable classes only
2. A simple perceptron is very limited in its capacity

# The Basic Neuron I

Simple mathematical model presented by McCulloch and Pitts (1943). It consists of :-

1. n input nodes $x_1, x_2, .., x_n$ and one output node O.
2. An n-vector X applied at input nodes, in one-to-one mapping.
3. It takes real valued inputs and produces real valued output.
4. Each neuron is associated to an activation function which transforms the input to a output value.

# The Basic Neuron II

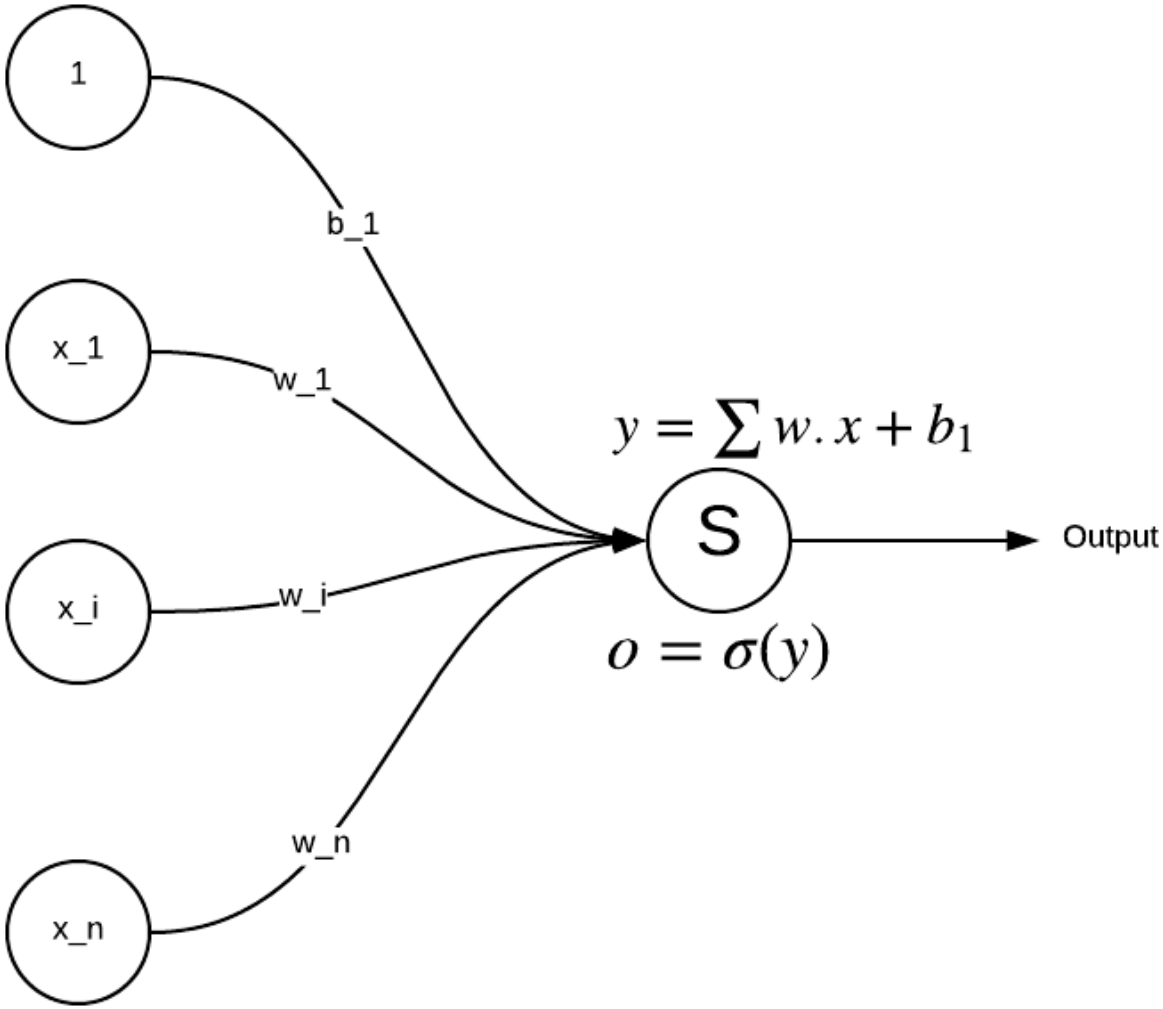


Figure 3: neuron with bias

# Activation Functions I

They are also called as Transfer function. mainly they are divided into 2 types:

1. Linear Activation Function
2. Non- Linear Activation Function

**Linear or Identity Activation Function**: The function is a line or linear. Therefore, the output of the functions will not be confined between any range.

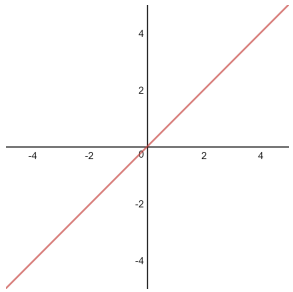


Figure 4: Linear Function

**The Nonlinear Activation Functions are mainly divided on the basis of their range or curves-**

1. Sigmoid or Logistic Function

# Activation Functions II

2. Tanh or hyperbolic tangent Activation Function
3. ReLU (Rectified Linear Unit) Activation Function
4. Leaky ReLU

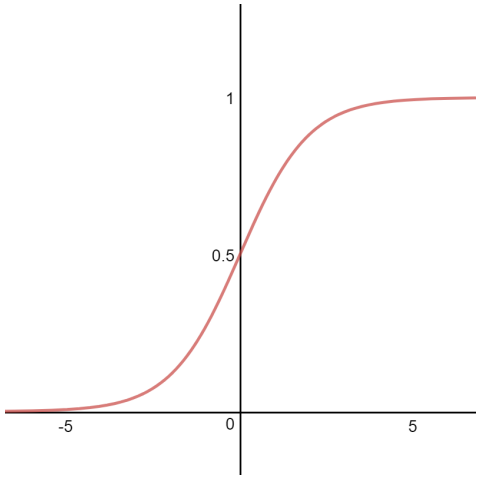


Figure 5: Logistic Function

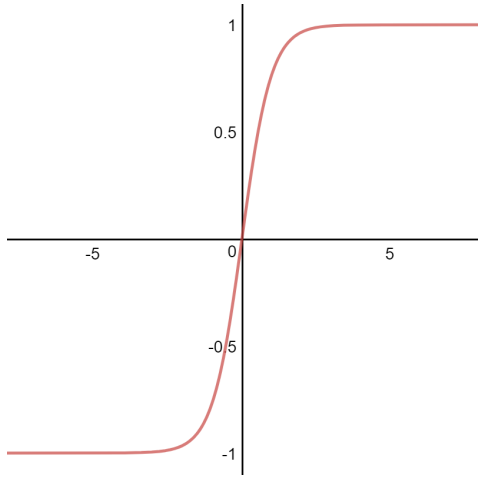


Figure 6: Tanh Function

# Activation Functions III

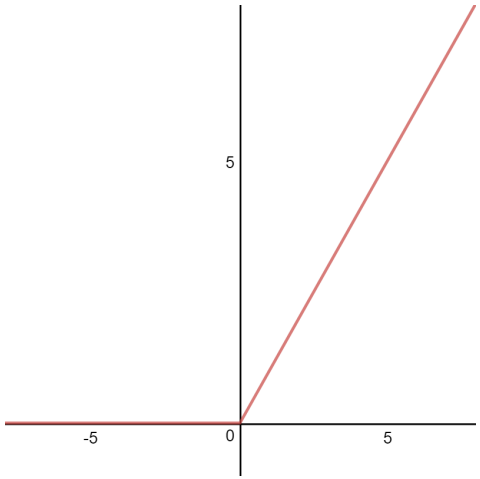


Figure 7: Rectified Linear Unit

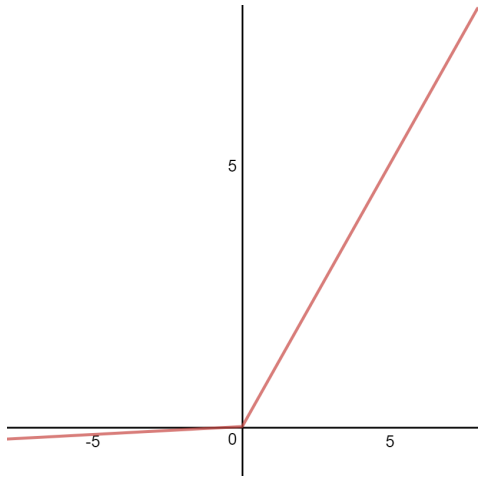


Figure 8: leaky ReLU

# Comparison Between Activation Functions I

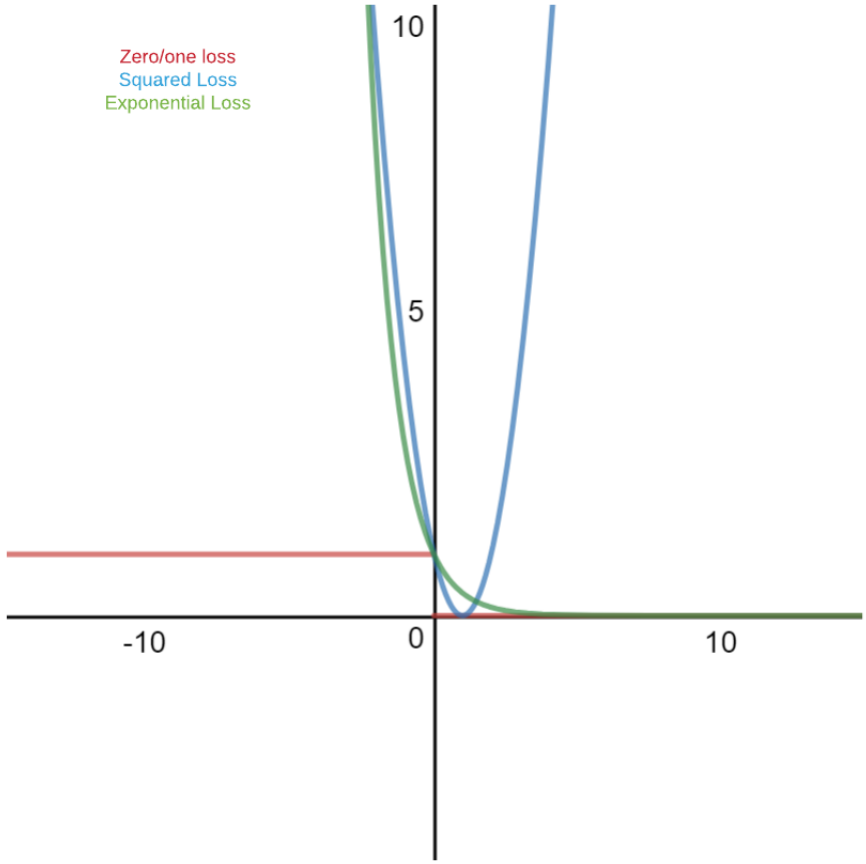


Figure 9: Activation Functions

# Comparison Between Activation Functions II

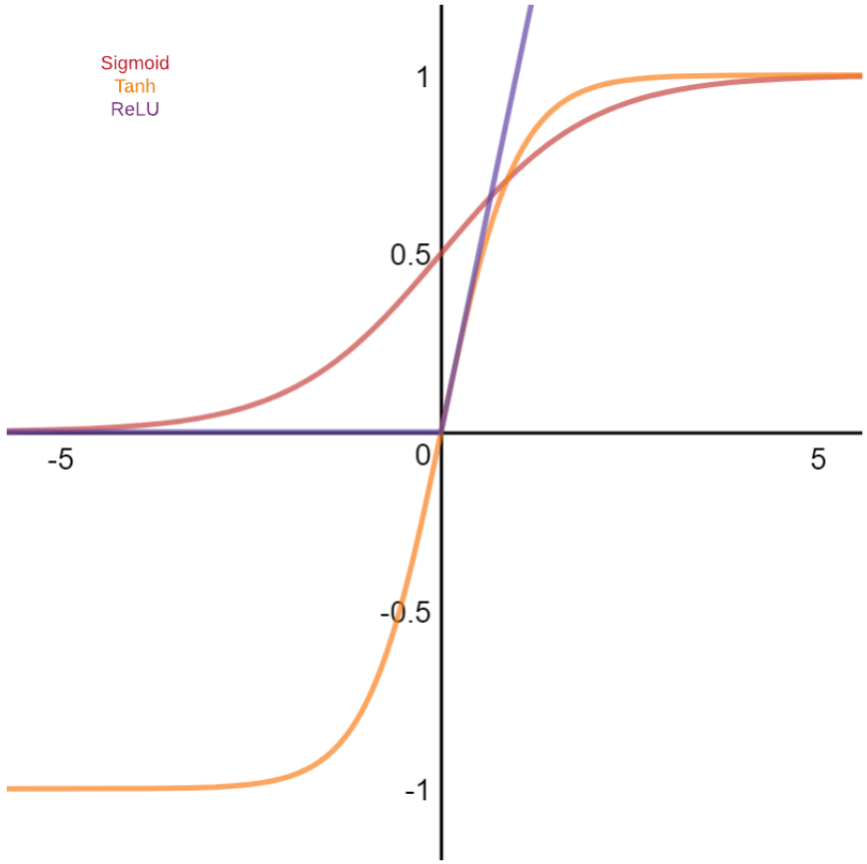


Figure 10: Activation Functions

# Chain Rule I

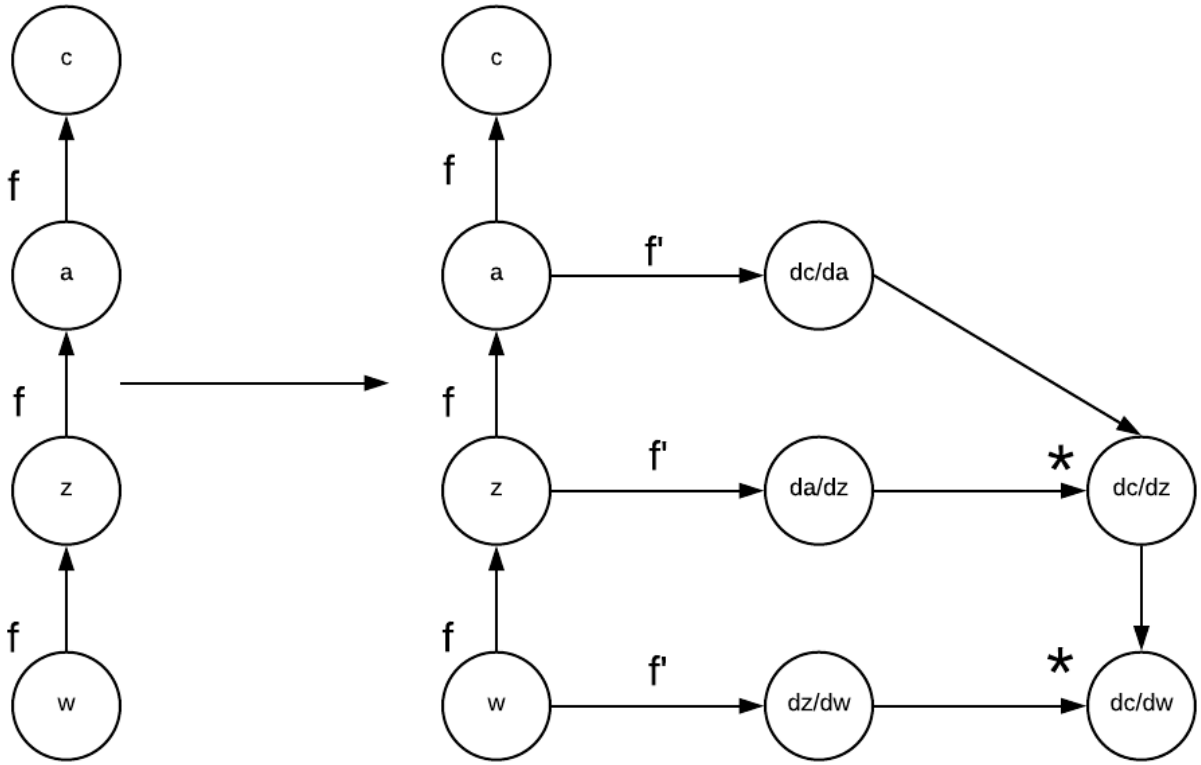


Figure 11: Chain Rule

# Introduction I

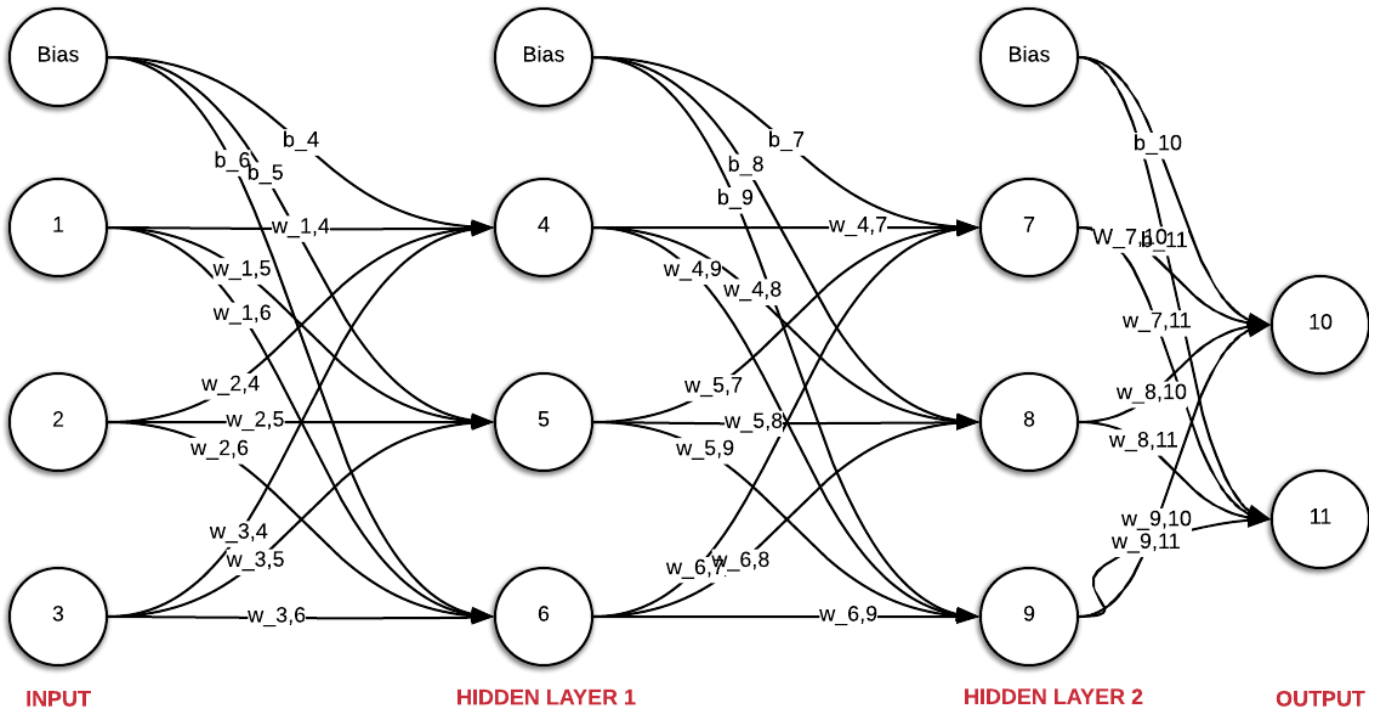


Figure 12: Neural Network with 2 hiddden layers

# Introduction II

$$\begin{bmatrix} Input \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$\begin{bmatrix} Output \end{bmatrix} = \begin{bmatrix} y_{10} \\ y_{11} \end{bmatrix}$$

# Output at First Hidden Layer I

Output at First Hidden Layer:

1. Net input of hidden layer 1

$$\begin{bmatrix} z_4 \\ z_5 \\ z_6 \end{bmatrix} = \begin{bmatrix} w_{1,4} & w_{2,4} & w_{3,4} \\ w_{1,5} & w_{2,5} & w_{3,5} \\ w_{1,6} & w_{2,6} & w_{3,6} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_4 \\ b_5 \\ b_6 \end{bmatrix}$$

2. Output of hidden layer 1

$$\begin{bmatrix} a_4 \\ a_5 \\ a_6 \end{bmatrix} = \sigma \begin{bmatrix} z_4 \\ z_5 \\ z_6 \end{bmatrix}$$

# Output at First Hidden Layer II

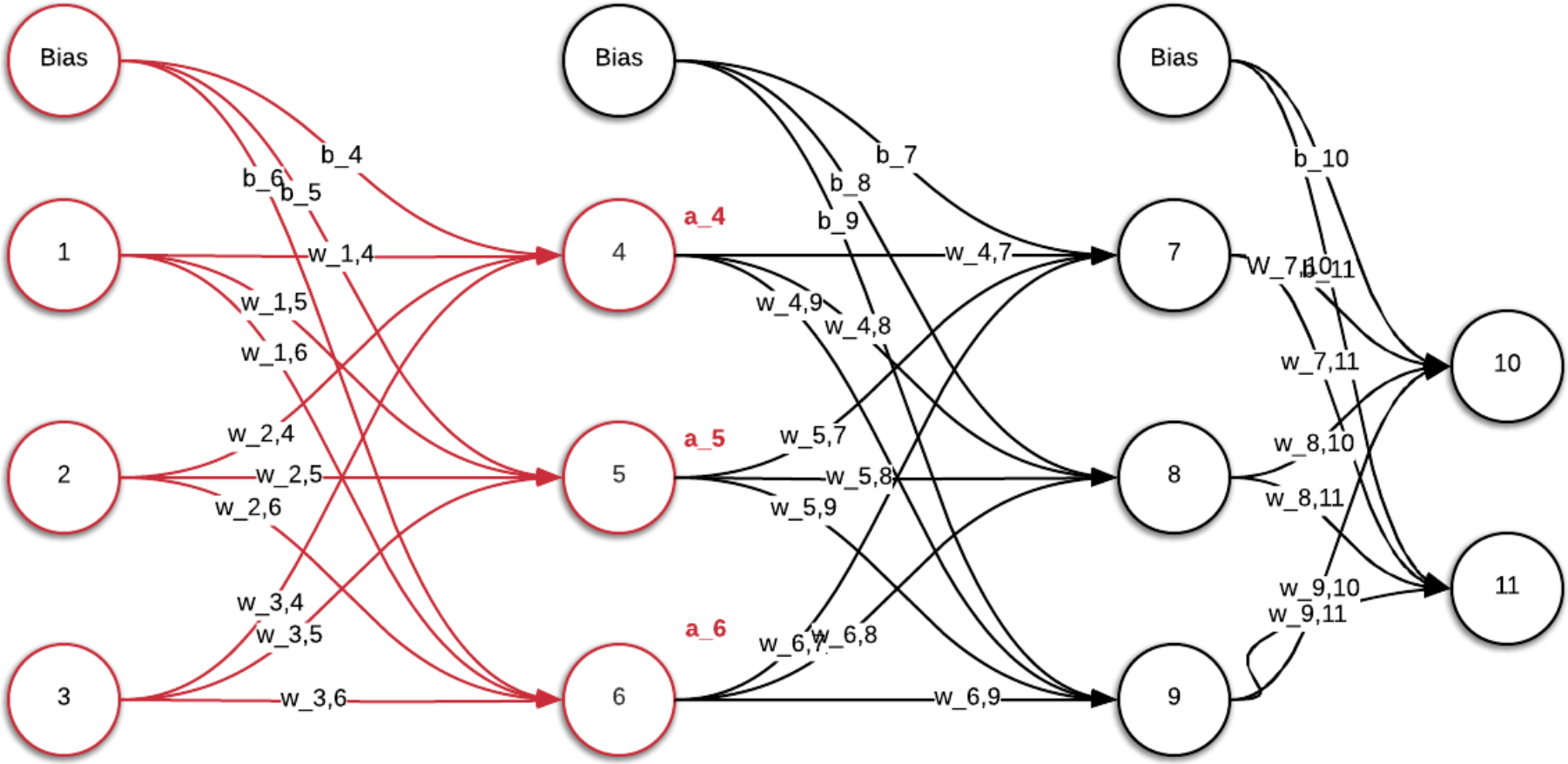


Figure 13: Output at First Hidden Layer

# Output at Second Hidden Layer I

Output at Second Hidden Layer:

1. Net input of hidden layer 2

$$\begin{bmatrix} z_7 \\ z_8 \\ z_9 \end{bmatrix} = \begin{bmatrix} w_{4,7} & w_{5,7} & w_{6,7} \\ w_{4,8} & w_{5,8} & w_{6,8} \\ w_{4,9} & w_{5,9} & w_{6,9} \end{bmatrix} \cdot \begin{bmatrix} a_4 \\ a_5 \\ a_6 \end{bmatrix} + \begin{bmatrix} b_7 \\ b_8 \\ b_9 \end{bmatrix}$$

2. Output of hidden layer 2

$$\begin{bmatrix} a_7 \\ a_8 \\ a_9 \end{bmatrix} = \sigma \begin{bmatrix} z_7 \\ z_8 \\ z_9 \end{bmatrix}$$

# Output at Second Hidden Layer II

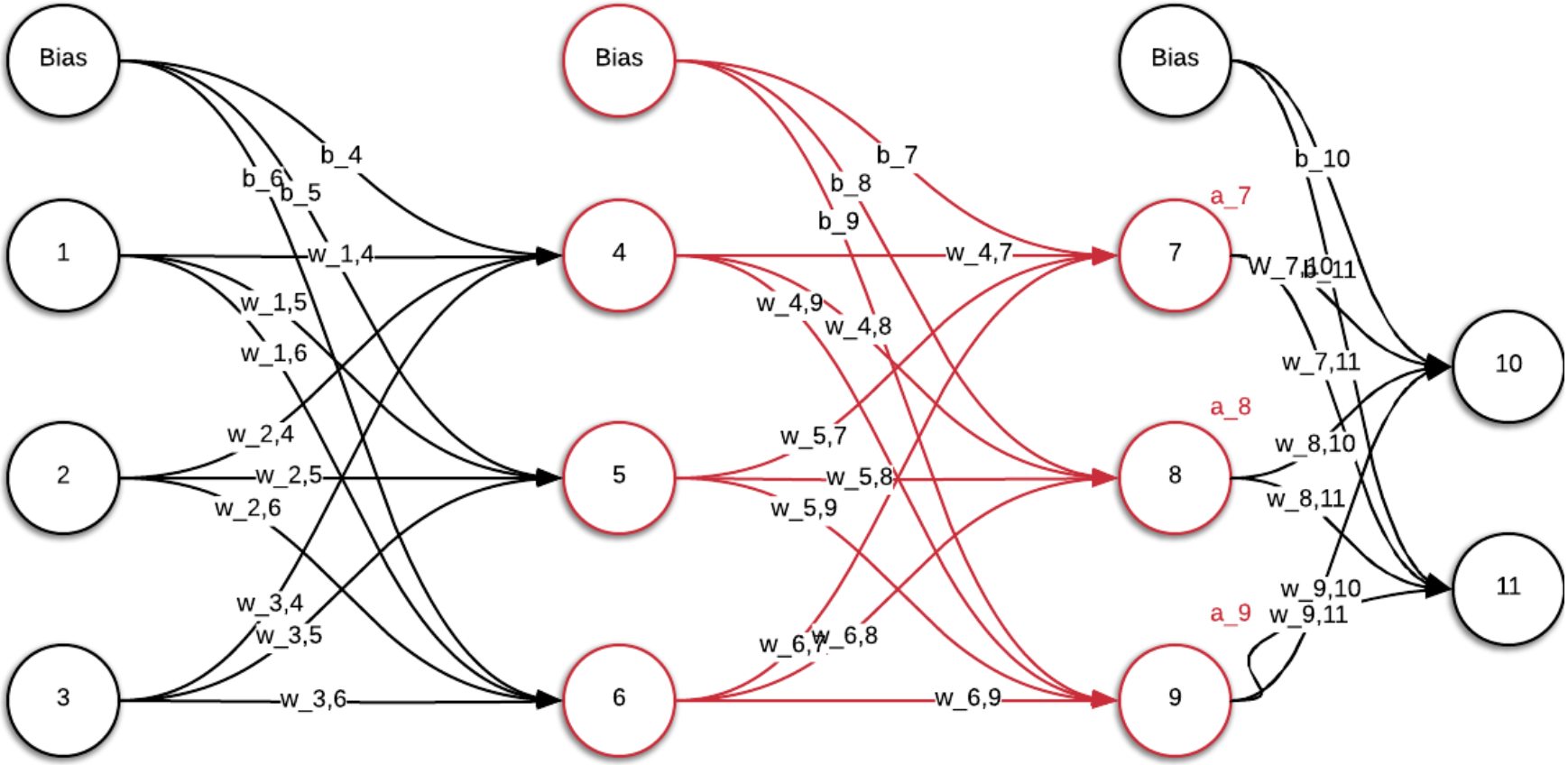


Figure 14: Output at Second Hidden Layer

# Output at Final Layer I

Output at Final Layer:

1. Net input of final layer

$$\begin{bmatrix} z_{10} \\ z_{11} \end{bmatrix} = \begin{bmatrix} w_{7,10} & w_{8,10} & w_{9,10} \\ w_{7,11} & w_{8,11} & w_{9,11} \end{bmatrix} \cdot \begin{bmatrix} a_7 \\ a_8 \\ a_9 \end{bmatrix} + \begin{bmatrix} b_{10} \\ b_{11} \end{bmatrix}$$

2. Output of final layer

$$\begin{bmatrix} a_{10} \\ a_{11} \end{bmatrix} = \sigma \begin{bmatrix} z_{10} \\ z_{11} \end{bmatrix}$$

# Output at Final Layer II

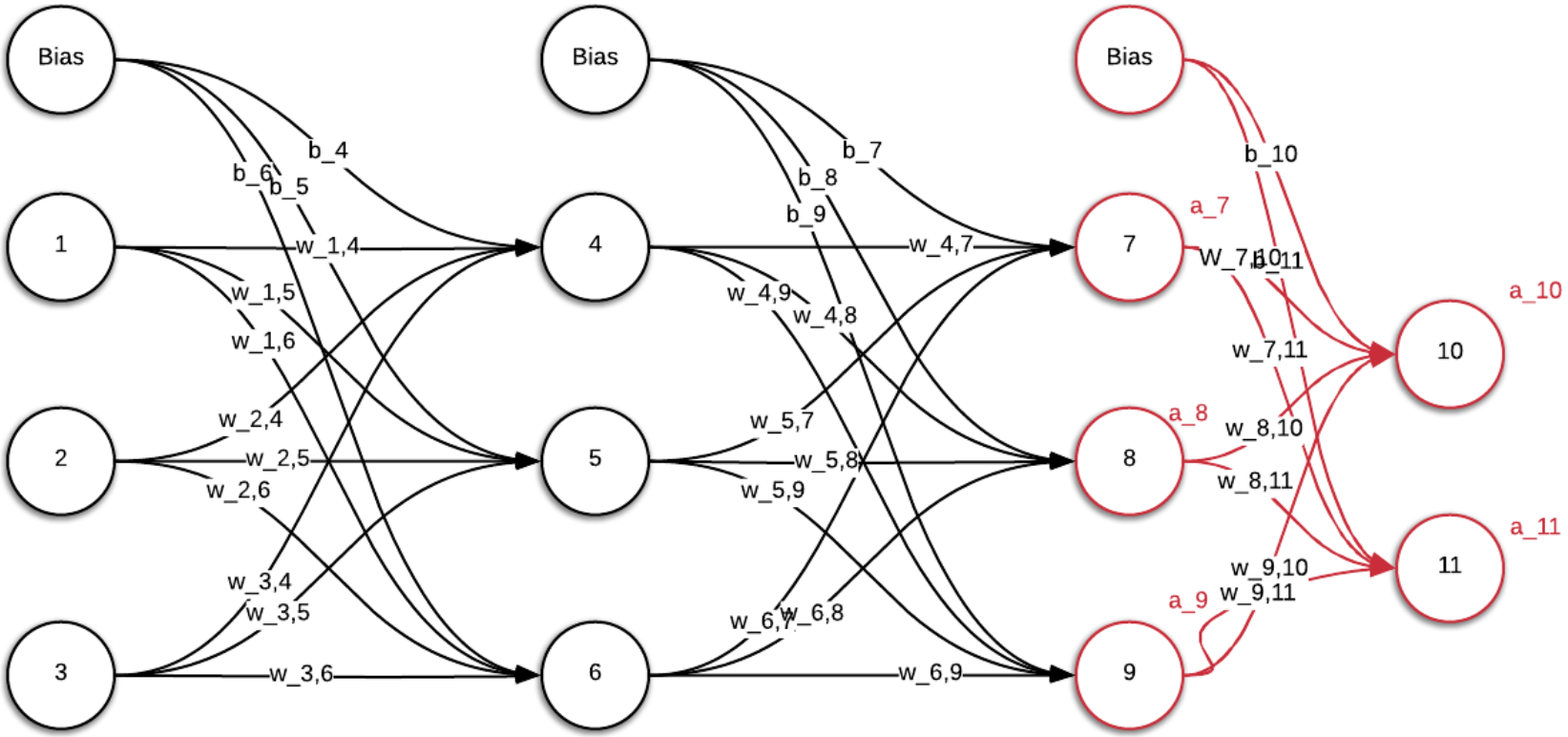


Figure 15: Output at Final Layer

# Calculating Total Error I

Total Error:

$$C_T = \sum \frac{1}{2}(target - output)^2$$

$$C_T = \frac{1}{2}(y_{10} - a_{10})^2 + \frac{1}{2}(y_{11} - a_{11})^2$$

# Error at last layer neuron I

Error at Output Neuron 10:

$$\frac{\partial C_T}{\partial w} = \frac{\partial C_{10}}{\partial a_{10}} * \frac{\partial a_{10}}{\partial z_{10}} * \frac{\partial z_{10}}{\partial w}$$

$$\frac{\partial C_T}{\partial w} = \delta_{10} * \frac{\partial z_{10}}{\partial w}$$

$$\frac{\partial C}{\partial w} = -(y_{10} - a_{10}) * a_{10} * (1 - a_{10}) * \frac{\partial z_{10}}{\partial w}$$

Change in output layer weights (dependent on neuron 10)

$$w_{7,10} = w_{7,10} - \eta * \frac{\partial C_{10}}{\partial w}$$

$$w_{7,10} = w_{7,10} + \eta * (y_{10} - a_{10}) * a_{10} * (1 - a_{10}) * \frac{\partial z_{10}}{\partial w_{7,10}}$$

$$w_{7,10} = w_{7,10} + \eta * (y_{10} - a_{10}) * a_{10} * (1 - a_{10}) * a_7$$

Similarly

$$w_{8,10} = w_{8,10} + \eta * (y_{10} - a_{10}) * a_{10} * (1 - a_{10}) * a_8$$

$$w_{9,10} = w_{9,10} + \eta * (y_{10} - a_{10}) * a_{10} * (1 - a_{10}) * a_9$$

# Error at last layer neuron II

$$b_{10} = b_{10} + \eta * (y_{10} - a_{10}) * a_{10} * (1 - a_{10})$$

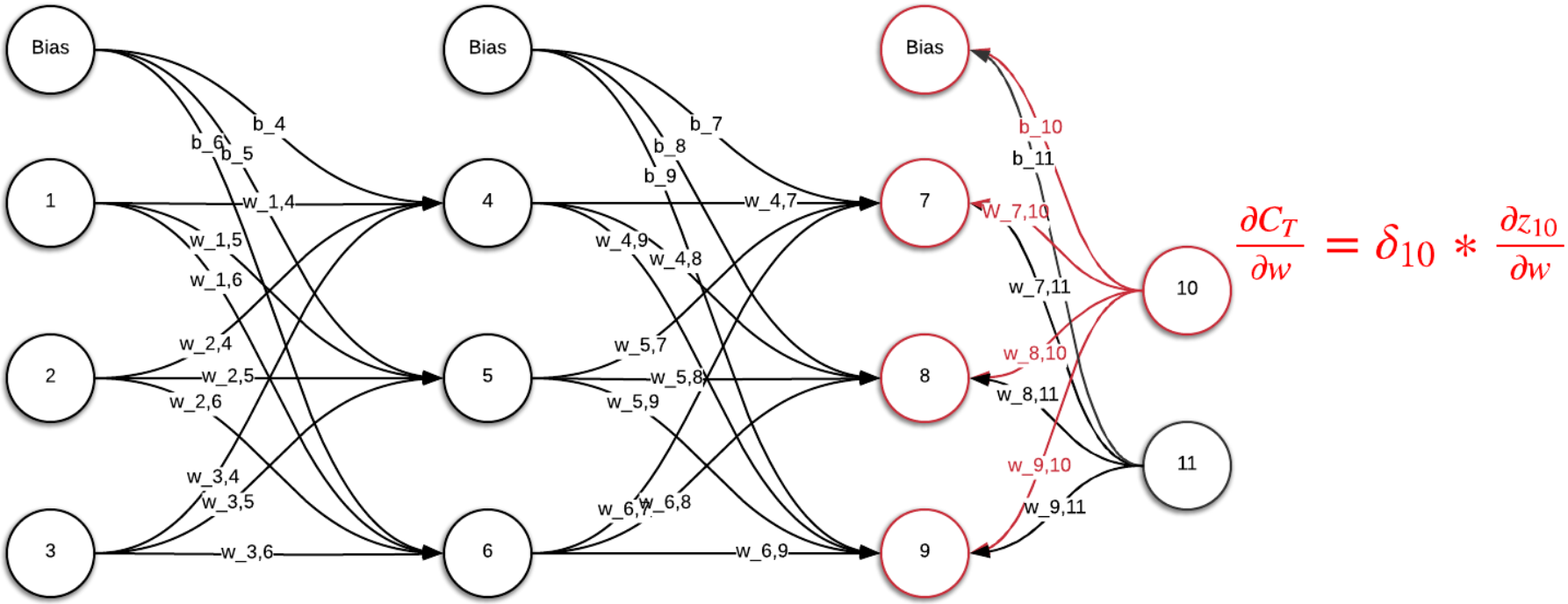


Figure 16: Error at Output Neuron 10

# Error at hidden layer 2 neuron I

Error at hidden layer neuron 7:

$$\frac{\partial C_T}{\partial w} = \frac{\partial C_T}{\partial a_7} * \frac{\partial a_7}{\partial z_7} * \frac{\partial z_7}{\partial w}$$
$$\frac{\partial C_T}{\partial w} = \left(\frac{\partial C_{10}}{\partial a_7} + \frac{\partial C_{11}}{\partial a_7}\right) * \frac{\partial a_7}{\partial z_7} * \frac{\partial z_7}{\partial w}$$
$$\frac{\partial C_T}{\partial w} = \left(\frac{\partial C_{10}}{\partial a_{10}} * \frac{\partial a_{10}}{\partial z_{10}} * \frac{\partial z_{10}}{\partial a_7} + \frac{\partial C_{11}}{\partial a_{11}} * \frac{\partial a_{11}}{\partial z_{11}} * \frac{\partial z_{11}}{\partial a_7}\right) * \frac{\partial a_7}{\partial z_7} * \frac{\partial z_7}{\partial w}$$
$$\frac{\partial C_T}{\partial w} = \left(\delta_{10} * \frac{\partial z_{10}}{\partial a_7} + \delta_{11} * \frac{\partial z_{11}}{\partial a_7}\right) * \frac{\partial a_7}{\partial z_7} * \frac{\partial z_7}{\partial w}$$
$$\frac{\partial C_T}{\partial w} = \left(\delta_{10} * w_{7,10} + \delta_{11} * w_{7,11}\right) * a_7 * (1 - a_7) * \frac{\partial z_7}{\partial w}$$
$$\frac{\partial C_T}{\partial w} = \delta_7 * \frac{\partial z_7}{\partial w}$$

Change in hidden layer weights (dependent on neuron 7)

$$w_{4,7} = w_{4,7} - \eta * \frac{\partial C_T}{\partial w}$$
$$w_{4,7} = w_{4,7} - \eta * \delta_7 * \frac{\partial z_7}{\partial w}$$
$$w_{4,7} = w_{4,7} - \eta * \delta_7 * a_4$$

# Error at hidden layer 2 neuron II

similarly

$$w_{5,7} = w_{5,7} - \eta * \delta_7 * a_5$$

$$w_{6,7} = w_{6,7} - \eta * \delta_7 * a_6$$

$$b_7 = b_7 - \eta * \delta_7$$

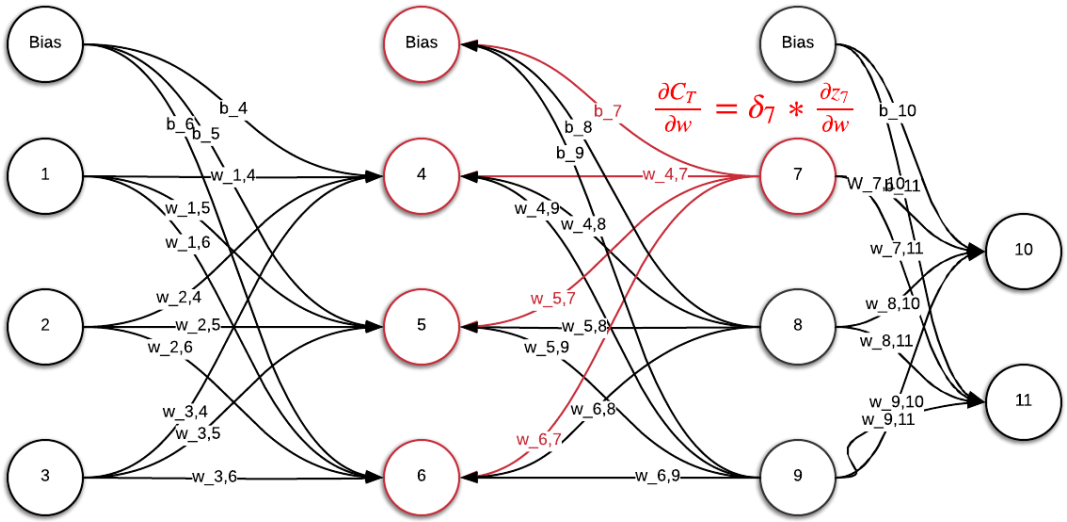


Figure 17: Error at Neuron 7

# Error at hidden layer 1 neuron I

Error at hidden layer neuron 4:

$$\frac{\partial C_T}{\partial w} = \frac{\partial C_T}{\partial a_4} * \frac{\partial a_4}{\partial z_4} * \frac{\partial z_4}{\partial w}$$
$$\frac{\partial C_T}{\partial w} = \left(\frac{\partial C_7}{\partial a_4} + \frac{\partial C_8}{\partial a_4} + \frac{\partial C_9}{\partial a_4}\right) * \frac{\partial a_4}{\partial z_4} * \frac{\partial z_4}{\partial w}$$
$$\frac{\partial C_T}{\partial w} = \left(\frac{\partial C_7}{\partial a_7} * \frac{\partial a_7}{\partial z_7} * \frac{\partial z_7}{\partial a_4} + \frac{\partial C_8}{\partial a_8} * \frac{\partial a_8}{\partial z_8} * \frac{\partial z_8}{\partial a_4} + \frac{\partial C_9}{\partial a_9} * \frac{\partial a_9}{\partial z_9} * \frac{\partial z_9}{\partial a_4}\right) * \frac{\partial a_4}{\partial z_4} * \frac{\partial z_4}{\partial w}$$
$$\frac{\partial C_T}{\partial w} = \left(\delta_7 * \frac{\partial z_7}{\partial a_4} + \delta_8 * \frac{\partial z_8}{\partial a_4} + \delta_9 * \frac{\partial z_9}{\partial a_4}\right) * \frac{\partial a_4}{\partial z_4} * \frac{\partial z_4}{\partial w}$$
$$\frac{\partial C_T}{\partial w} = \left(\delta_7 * w_{4,7} + \delta_8 * w_{4,8} + \delta_9 * w_{4,9}\right) * a_4 * (1 - a_4) * \frac{\partial z_4}{\partial w}$$
$$\frac{\partial C_T}{\partial w} = \delta_4 * \frac{\partial z_4}{\partial w}$$

Change in hidden layer weights (dependent on neuron 4)

$$w_{1,4} = w_{1,4} - \eta * \frac{\partial C_T}{\partial w}$$
$$w_{1,4} = w_{1,4} - \eta * \delta_4 * \frac{\partial z_4}{\partial w}$$
$$w_{1,4} = w_{1,4} - \eta * \delta_4 * x_1$$

# Error at hidden layer 1 neuron II

similarly

$$w_{2,4} = w_{2,4} - \eta * \delta_4 * x_2$$

$$w_{3,4} = w_{3,4} - \eta * \delta_4 * x_3$$

$$b_4 = b_4 - \eta * \delta_4$$

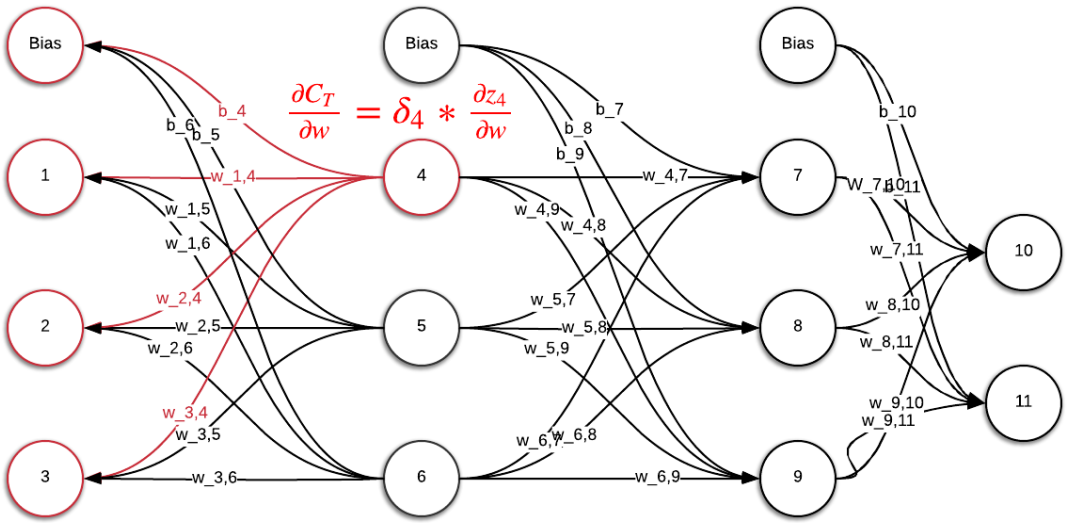


Figure 18: Error at Neuron 4

# Example I

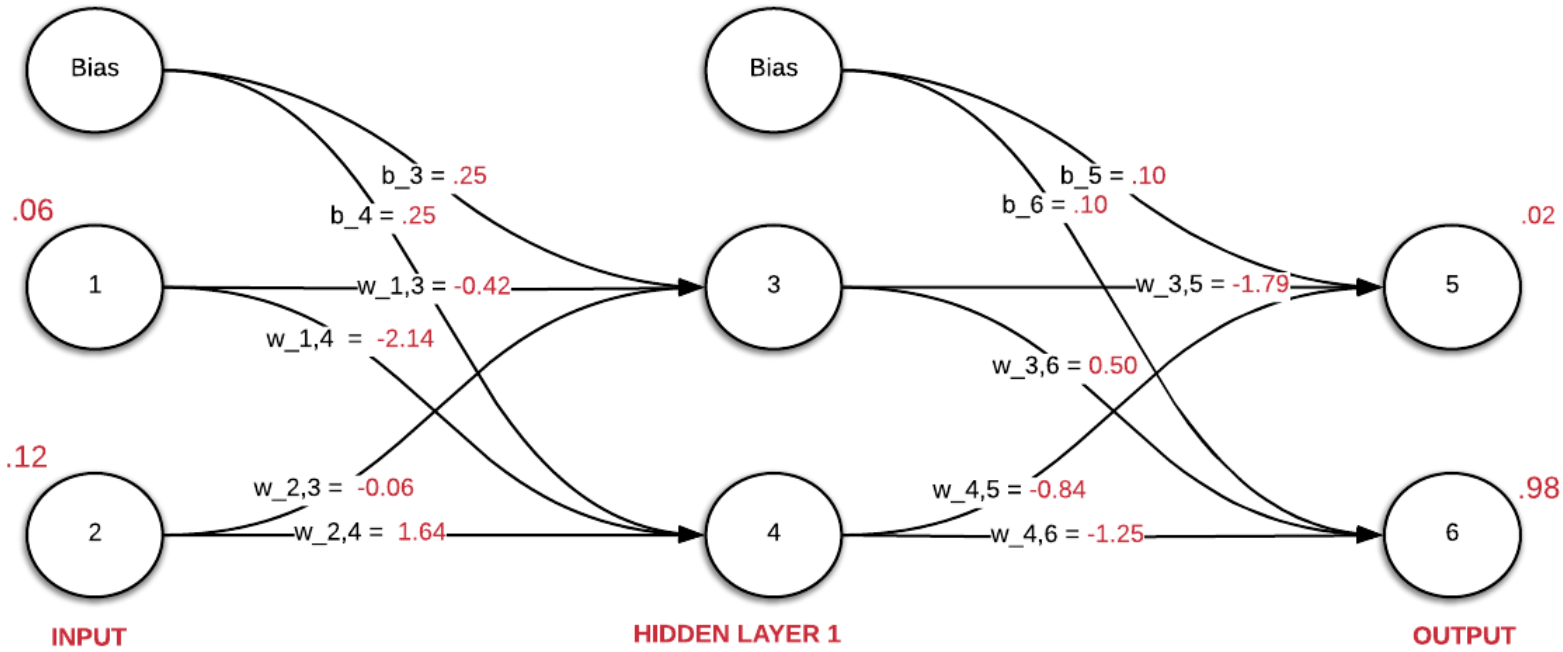


Figure 19: Example of Neural Network

# Example II

$$\begin{bmatrix} Input \\ \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} .06 \\ .12 \end{bmatrix}$$

$$\begin{bmatrix} Output \\ \end{bmatrix} = \begin{bmatrix} y_5 \\ y_6 \end{bmatrix} = \begin{bmatrix} .02 \\ .98 \end{bmatrix}$$

$\eta = .50$

$$\begin{bmatrix} z_3 \\ z_4 \end{bmatrix} = \begin{bmatrix} w_{1,3} & w_{2,3} \\ w_{1,4} & w_{2,4} \end{bmatrix} . \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_3 \\ b_4 \end{bmatrix}$$

$$\begin{bmatrix} z_3 \\ z_4 \end{bmatrix} = \begin{bmatrix} -0.42 & -0.06 \\ -2.14 & 1.64 \end{bmatrix} . \begin{bmatrix} .06 \\ .12 \end{bmatrix} + \begin{bmatrix} .25 \\ .25 \end{bmatrix} = \begin{bmatrix} 0.2176 \\ 0.3184 \end{bmatrix}$$

$$\begin{bmatrix} a_3 \\ a_4 \end{bmatrix} = \sigma \begin{bmatrix} z_3 \\ z_4 \end{bmatrix}$$

$$\begin{bmatrix} a_3 \\ a_4 \end{bmatrix} = \sigma \begin{bmatrix} 0.2176 \\ 0.3184 \end{bmatrix} = \begin{bmatrix} 0.55418636 \\ 0.57893427 \end{bmatrix}$$

# Example III

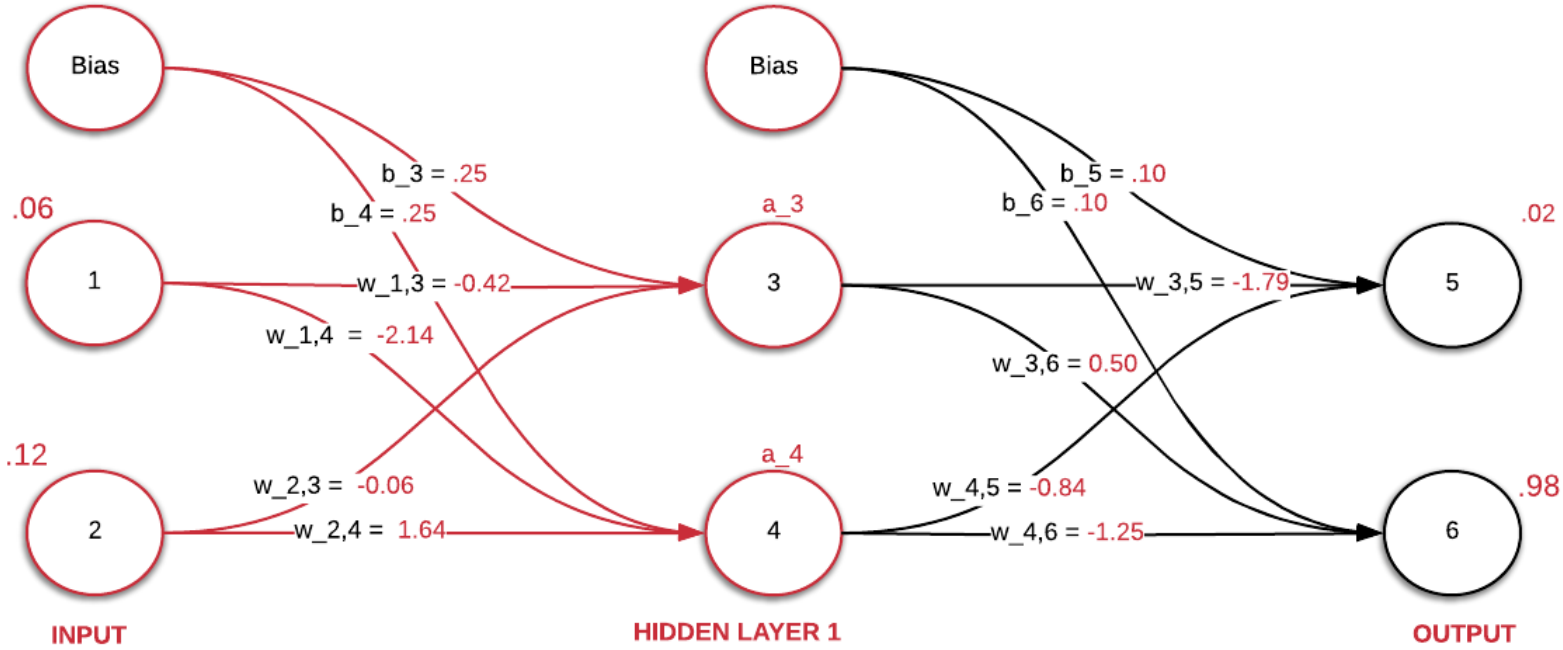


Figure 20: Example of Neural Network

# Example IV

$$\begin{bmatrix} z_5 \\ z_6 \end{bmatrix} = \begin{bmatrix} w_{3,5} & w_{4,5} \\ w_{3,6} & w_{4,6} \end{bmatrix} \cdot \begin{bmatrix} a_3 \\ a_4 \end{bmatrix} + \begin{bmatrix} b_5 \\ b_6 \end{bmatrix}$$

$$\begin{bmatrix} z_5 \\ z_6 \end{bmatrix} = \begin{bmatrix} -1.79 & -0.84 \\ 0.50 & -1.25 \end{bmatrix} \cdot \begin{bmatrix} 0.55418636 \\ 0.57893427 \end{bmatrix} + \begin{bmatrix} .10 \\ .10 \end{bmatrix} = \begin{bmatrix} -1.37829837 \\ -0.34657466 \end{bmatrix}$$

$$\begin{bmatrix} a_5 \\ a_6 \end{bmatrix} = \sigma \begin{bmatrix} z_5 \\ z_6 \end{bmatrix}$$

$$\begin{bmatrix} a_5 \\ a_6 \end{bmatrix} = \sigma \begin{bmatrix} -1.37829837 \\ -0.34657466 \end{bmatrix} = \begin{bmatrix} 0.20128243 \\ 0.4142133 \end{bmatrix}$$

# Example V

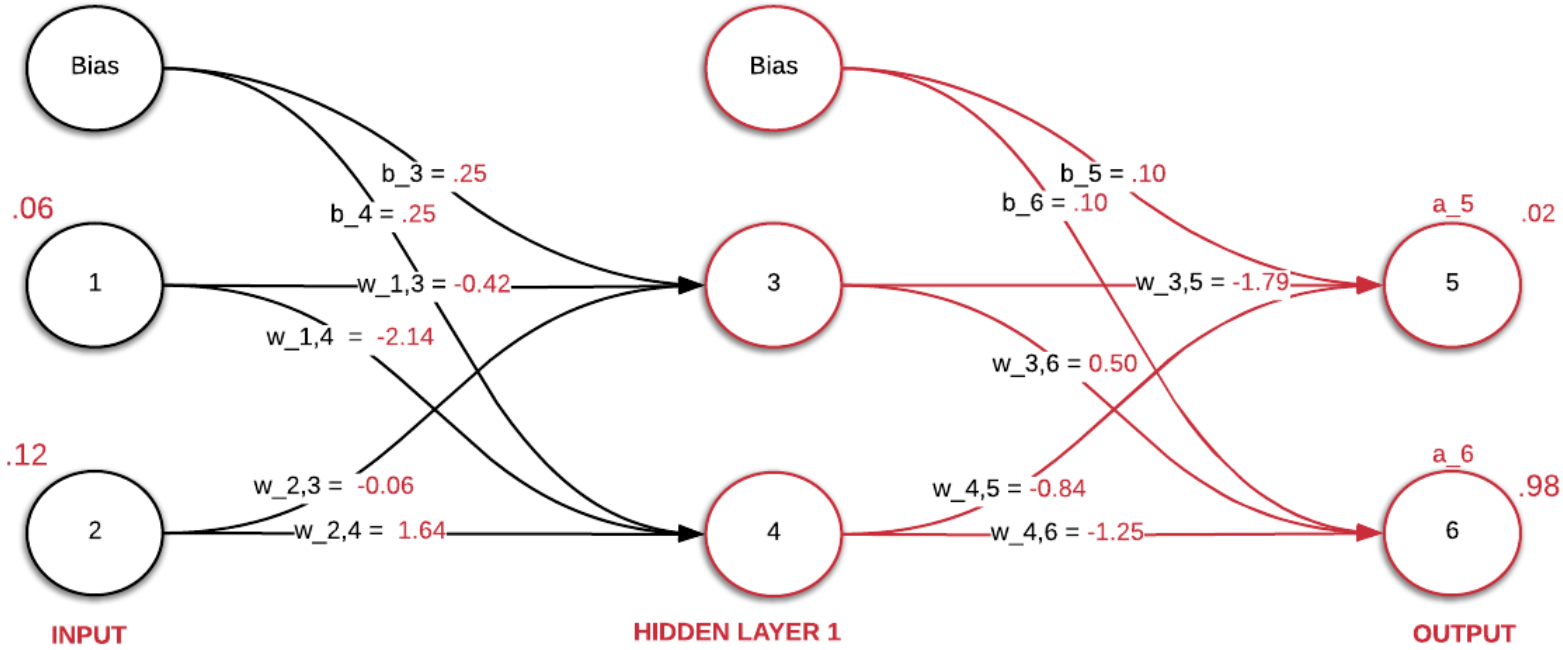


Figure 21: Example of Neural Network

# Example VI

$$C_T = \frac{1}{2}(y_5 - a_5)^2 + \frac{1}{2}(y_6 - a_6)^2$$
$$C_T = \frac{1}{2}(.02 - 0.20128243)^2 + \frac{1}{2}(.98 - 0.4142133)^2$$
$$C_T = 0.17648895466179743$$

Weight updation on final layer:

$$\delta_5 = -(y_5 - a_5) * a_5 * (1 - a_5)$$
$$w_{3,5} = w_{3,5} - \eta * \delta_5 * a_3$$
$$w_{3,5} = -1.79807571$$
$$b_5 = b_5 - \eta * \delta_5$$
$$b_5 = 0.15406923$$

Similarly:

$$w_{4,5} = -0.84843634$$
$$w_{3,6} = 0.53804014$$
$$w_{4,6} = -1.21026113$$

# Example VII

$b_6 = 0.15406923$

Weight updation on hidden layer:

$\delta_3 = (\delta_5 * w_{3,5} + \delta_6 * w_{3,6}) * a_3 * (1 - a_3)$

$w_{1,3} = w_{1,3} - \eta * \delta_3 * x_1$

$w_{1,3} = -0.4191045$

$b_3 = b_3 - \eta * \delta_3$

$b_3 = 0.24699192$

Similarly:

$w_{2,3} = -0.05820914$

$w_{1,4} = -2.14107592$

$w_{2,4} = 1.63784817$

$b_4 = 0.24699192$

# Convolutional Neural network I

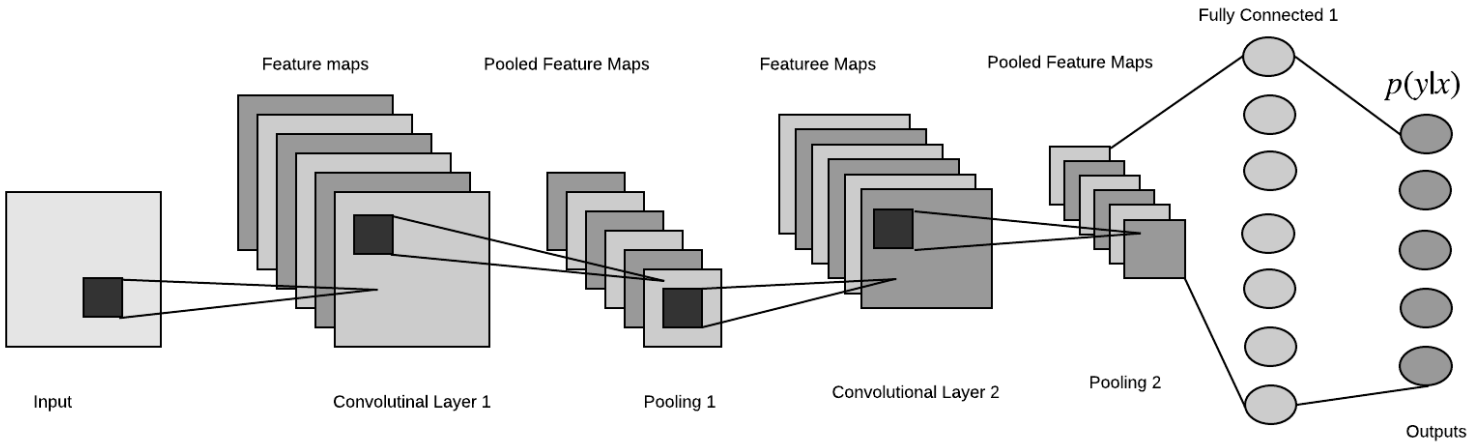


Figure 22: Convolutional Neural network

# Convolution I

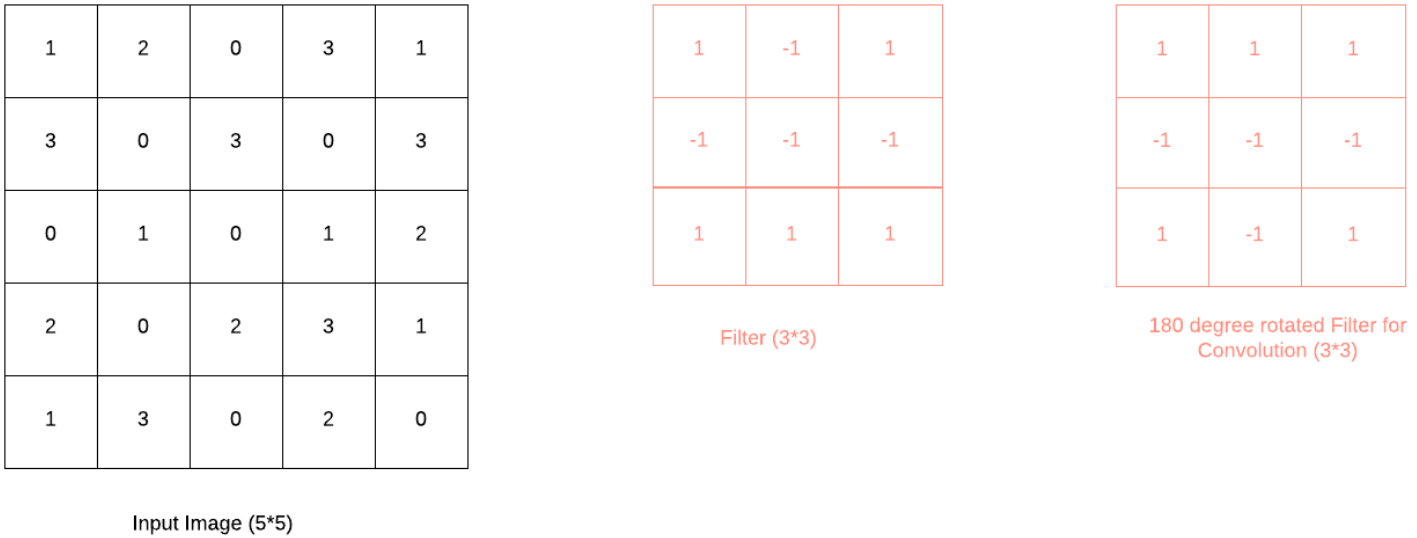


Figure 23: Input Image (size 5*5), Filter (3*3)

# Convolution II

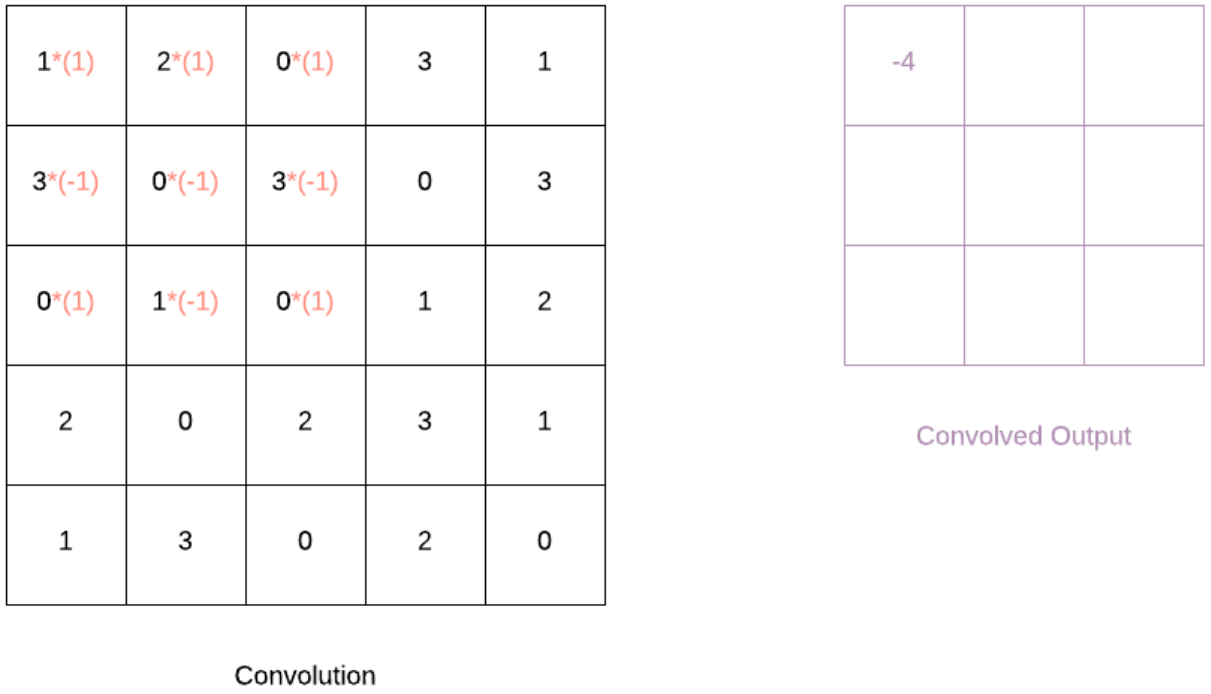


Figure 24: Convolution Operation at First Position

# Convolution III

| | | | | |
|---|---|---|---|---|
| 1 | 2*(1) | 0*(1) | 3*(1) | 1 |
| 3 | 0*(-1) | 3*(-1) | 0*(-1) | 3 |
| 0 | 1*(1) | 0*(-1) | 1*(1) | 2 |
| 2 | 0 | 2 | 3 | 1 |
| 1 | 3 | 0 | 2 | 0 |

Convolution

| | | |
|---|---|---|
| -4 | 4 | |
| | | |
| | | |

Convolved Output

Figure 25: Convolution Operation at Second Position

# Convolution IV

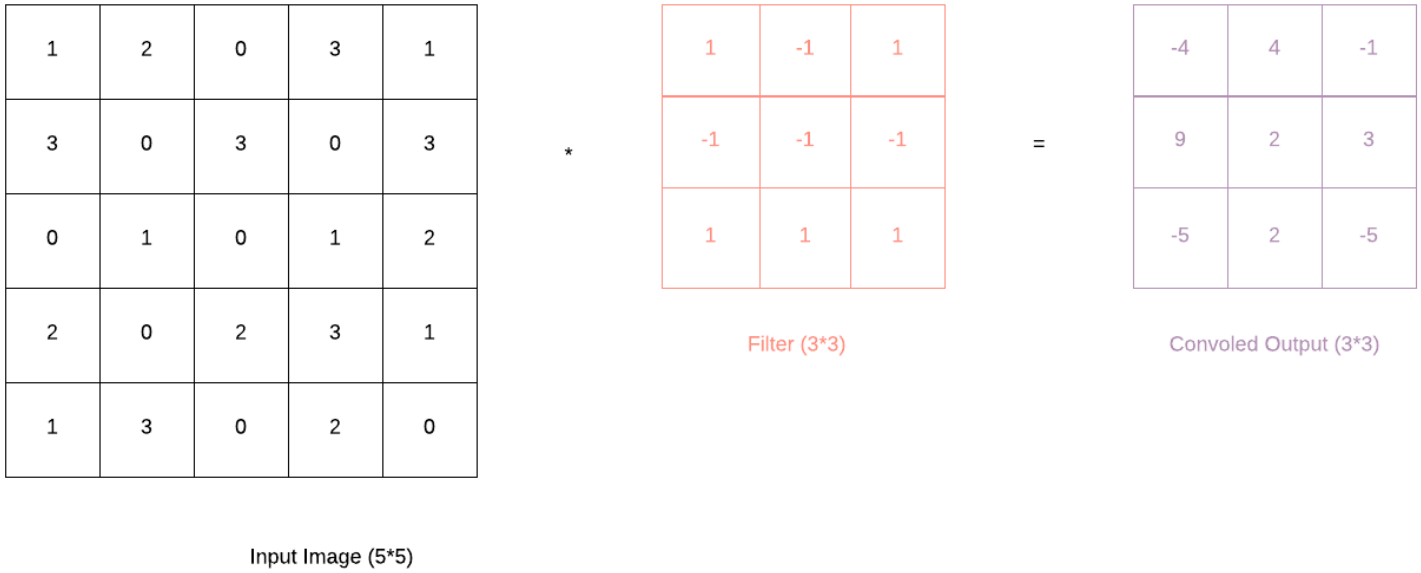


Figure 26: Convolution Output

# Rectified Output I

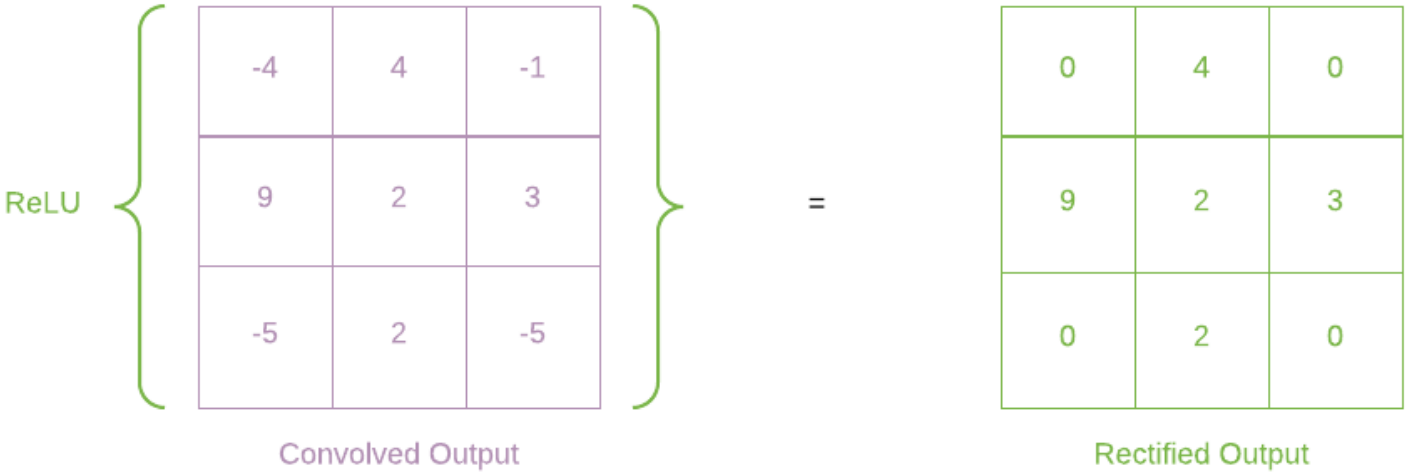


Figure 27: Rectified Output

# Pooling I

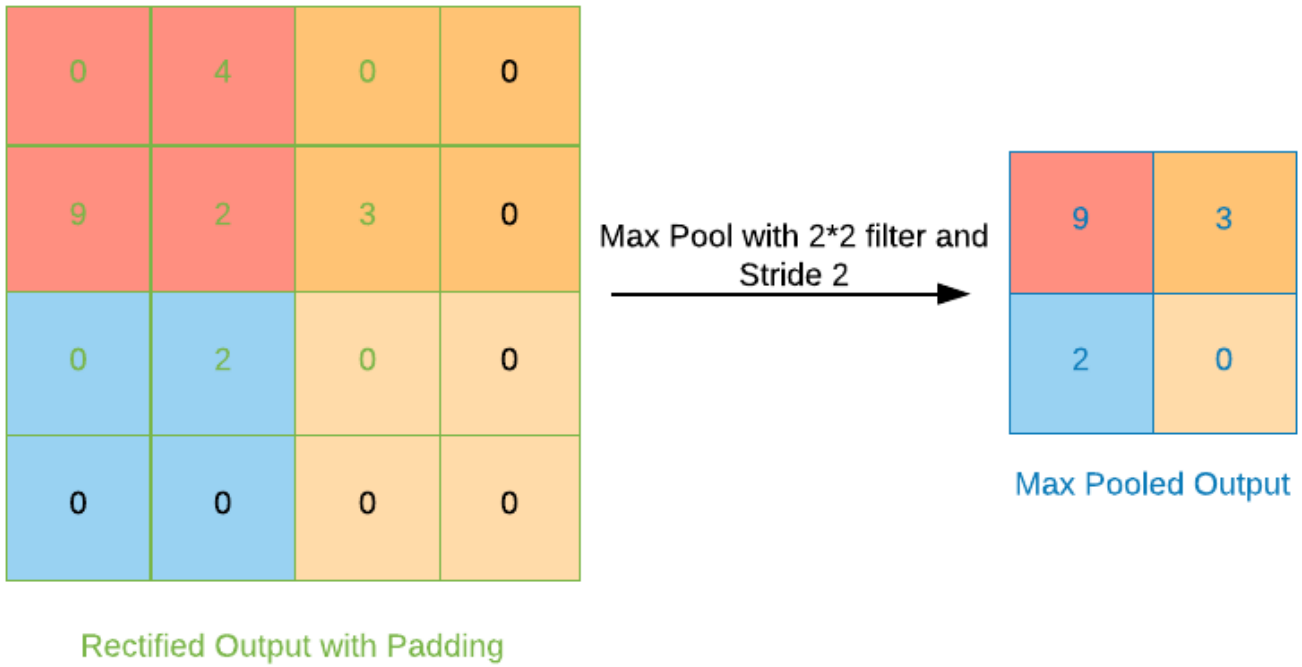


Figure 28: Max Pooled Output

# Computational Considerations I

Largest bottleneck while constructing ConvNet architecture is memory. Generally GPU have a limit of 3/4/6 GB memory, but modern GPU have 12GB memory. There are 3 major sources of memory to keep track of:

1. The intermediate volume sizes are the Raw no of activations and gradient at each layer of convNet
2. The parameter sizes are the numbers that hold the network parameters, their gradients during back propagation. So the memory to store parameter vector must be multiplied by a factor of 2 or 3
3. Every ConvNet implementation needs to maintain miscellaneous memory, such as the image data batches etc.

# Case Studies I

| Network | Year | Layers | Remark |
|---|---|---|---|
| Alexnet | 2012 | 7 | Winner of ILSVRC 2012 challenge |
| VGG-19 | 2014 | 19 | Runner up of ILSVRC 2014 challenge |
| GoogleNet | 2014 | 22 | Winner of ILSVRC 2014 challenge |
| Resnet-50 | 2015 | 50 | Winner of ILSVRC 2015 challenge |

# CNN Example I

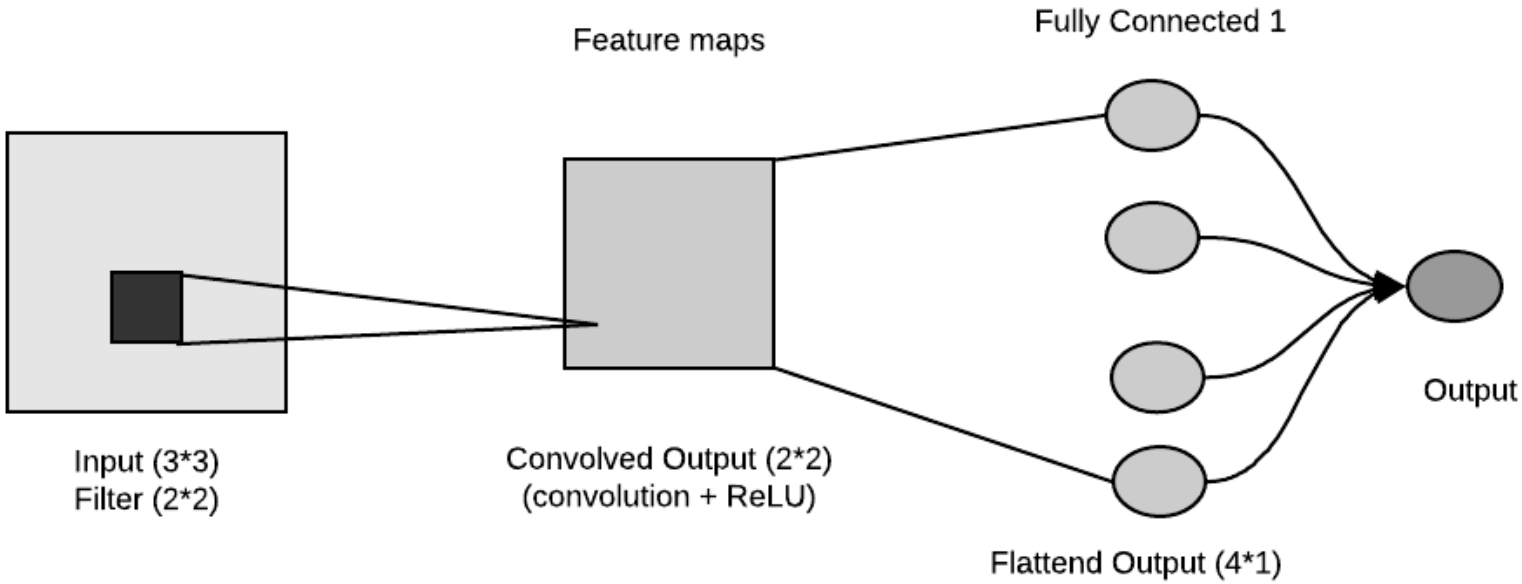


Figure 29: CNN Example (Architecture)

# Forward Pass I

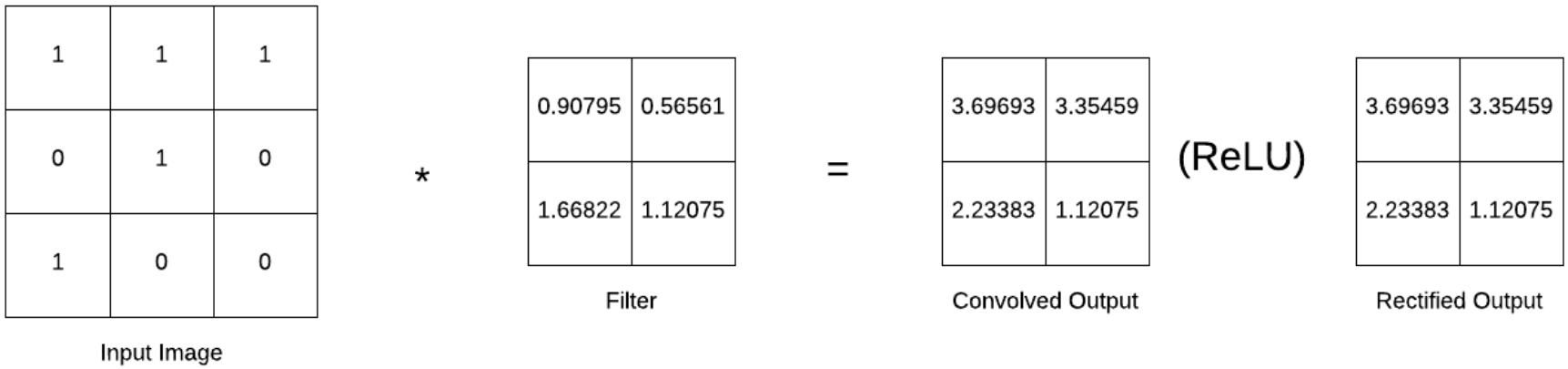


Figure 30: Output of Convolution Layer

# Forward Pass II

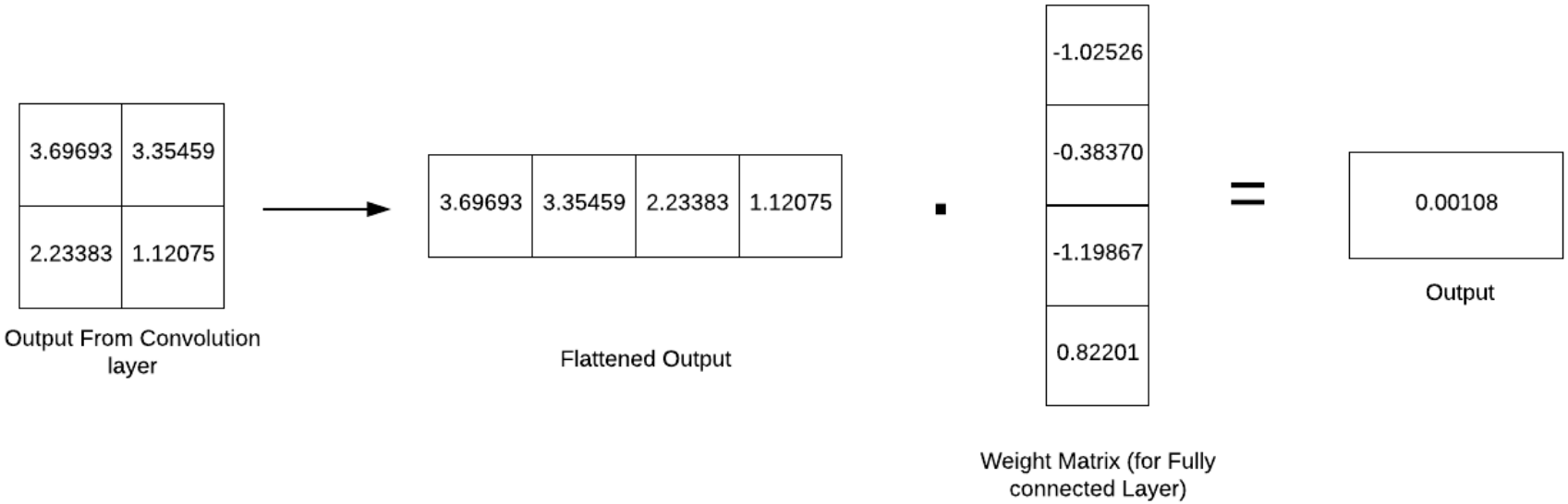


Figure 31: Output of Fully Connected Layer)

# Backward Pass I

$$delta\_fc = (output - actual\_output) * d\_sigmoid(output)$$

$$delta\_fc = (0.00108 - 0.77) * 0.00107 = -0.00083$$

$$grad\_fc = flattened\_output.T.dot(delta\_fc)$$

$$grad\_fc = \begin{bmatrix} 3.69693349 \\ 3.35458658 \\ 2.23383253 \\ 1.12075406 \end{bmatrix} * -0.00083$$

$$weight\_fc = weight\_fc - grad\_fc * (learning\_rate = 0.5)$$

$$weight\_fc = \begin{bmatrix} -1.02373 \\ -0.38231 \\ 1.19775 \\ 0.82248 \end{bmatrix}$$

Figure 32: Updated Weight Matrix of Fully Connected Layer

# Backward Pass II

$$delta\_conv = np.reshape((delta\_fc).dot(weight\_fc.T), (2, 2))$$

$$grad\_conv = delta\_conv * d\_relu(rectified\_output)$$

$$grad\_conv = np.rot90(signal.convolve2d(input, np.rot90(grad\_conv, 2), 'valid'), 2)$$

$$filter = filter - grad\_conv * (learning_rate = 0.5)$$

$$filter = \begin{bmatrix} 0.90753 & 0.56495 \\ 1.66714 & 1.12051 \end{bmatrix}$$

Figure 33: Updated Filter of Convolution Layer

# The End