

# Building Blocks Introduction to Deep Learning

This file is meant for personal use by manju.crz@gmail.com only.

Proprietary content: ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited

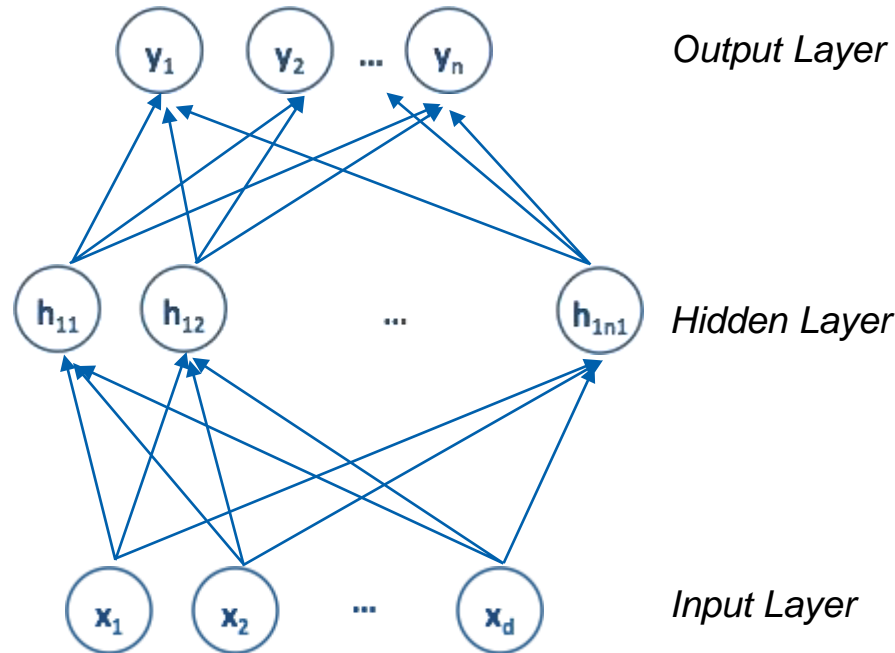
# Agenda

- Feed forward
- Back propagation
- Fully connected layer
- Activation functions
- Softmax function
- Cross-entropy loss

This file is meant for personal use by manju.crz@gmail.com only.

Proprietary content: ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited

# Refresher: Neural Network

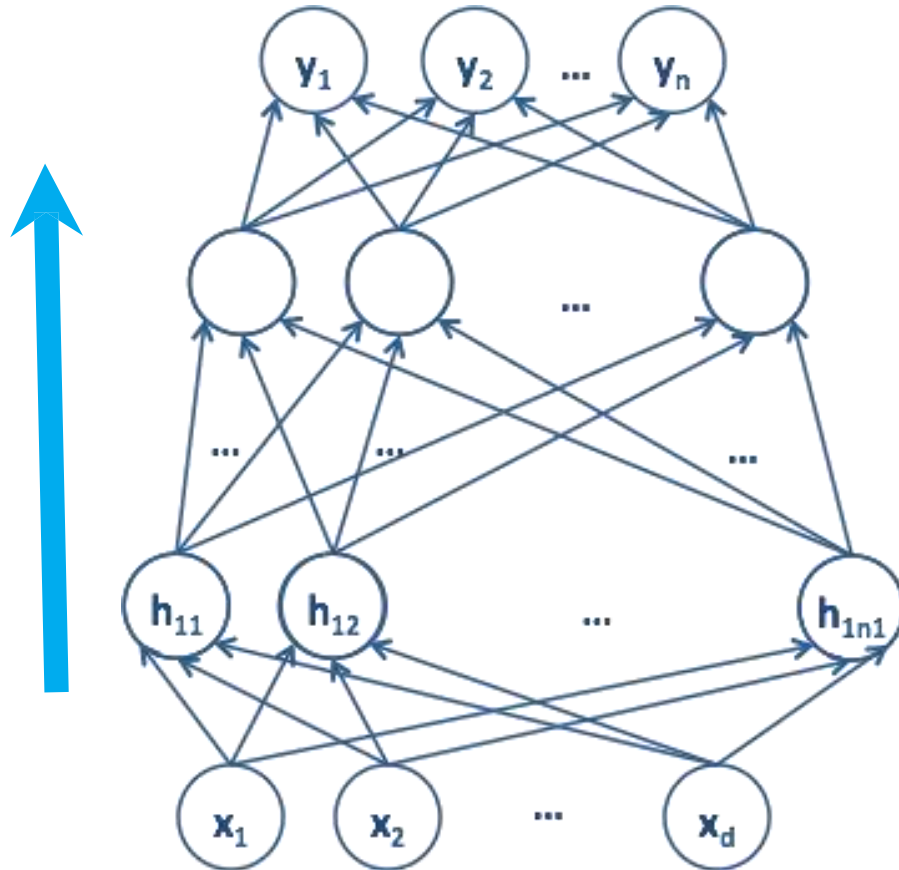


- Output layer represents the output classes (each node corresponds to a class)
- Each node in the hidden layer has an activation function that acts on the input
- Each node in the hidden layer has a particular weight (towards a weighted sum)
- Feedforward refers to a unidirectional flow of information (and no lateral/intra-layer connections) from input to output
- Predicted output is compared to the actual output during the training process, and the difference (loss) determines how much the weights should be tweaked
- Tweaking the gradients and thereby the weights is done through back-propagation

This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

# Refresher: Deep Neural Networks - Forward

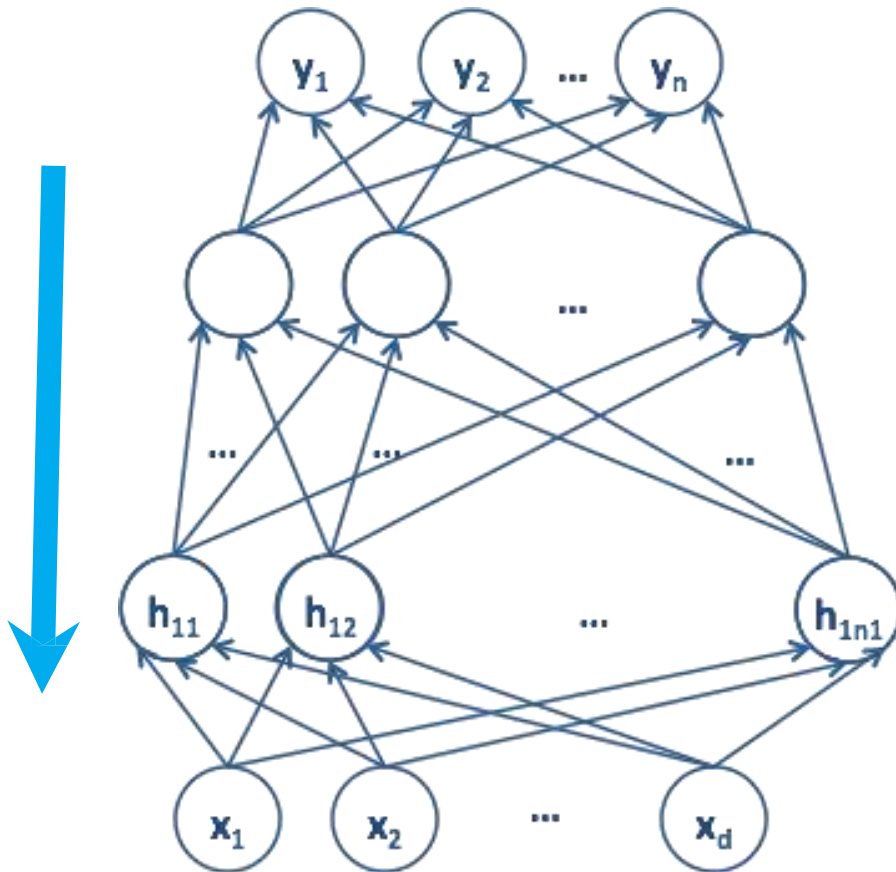


- More than one hidden layer in multi-layer (deep) neural networks, depth refers to the number of layers
- Each node in the hidden layer uses an activation function that acts on the input, and has a weight (towards a weighted sum)
- Forward pass refers to a unidirectional flow of information (and no lateral/intra-layer connections) from input to output
- When each input node is connected to all output nodes of the next layer, it is a fully connected network
- Output layer consists of nodes that represent a series of probabilities corresponding to the predicted likelihood of each class

This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

# Refresher: Deep Neural Networks - Backward



- Predicted output is typically incorrect when compared to actual value (training)
- How much the prediction is off from reality is measured using an error or loss function
- Backpropagation calculates the error gradient for all the weights and biases for all the layers (starting from output and working its way back)
- Forward pass with updated weights should lead to lower error, and over time a reduction in loss/error

# Neural Network Constructed

Now, let's review the following in turn:

- Feed forward
- Back propagation
- Fully connected layer
- Activation functions
- Softmax function
- Cross-entropy loss

... and we'll have a fully functioning network

This file is meant for personal use by manju.crz@gmail.com only.

Proprietary content: ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited

# Feed forward

This file is meant for personal use by manju.crz@gmail.com only.

Proprietary content: ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited

# Neural Network Constructed

Now, let's review the following in turn:

- Feed forward
- Back propagation
- Fully connected layer
- Activation functions
- Softmax function
- Cross-entropy loss

... and we'll have a fully functioning network

This file is meant for personal use by manju.crz@gmail.com only.

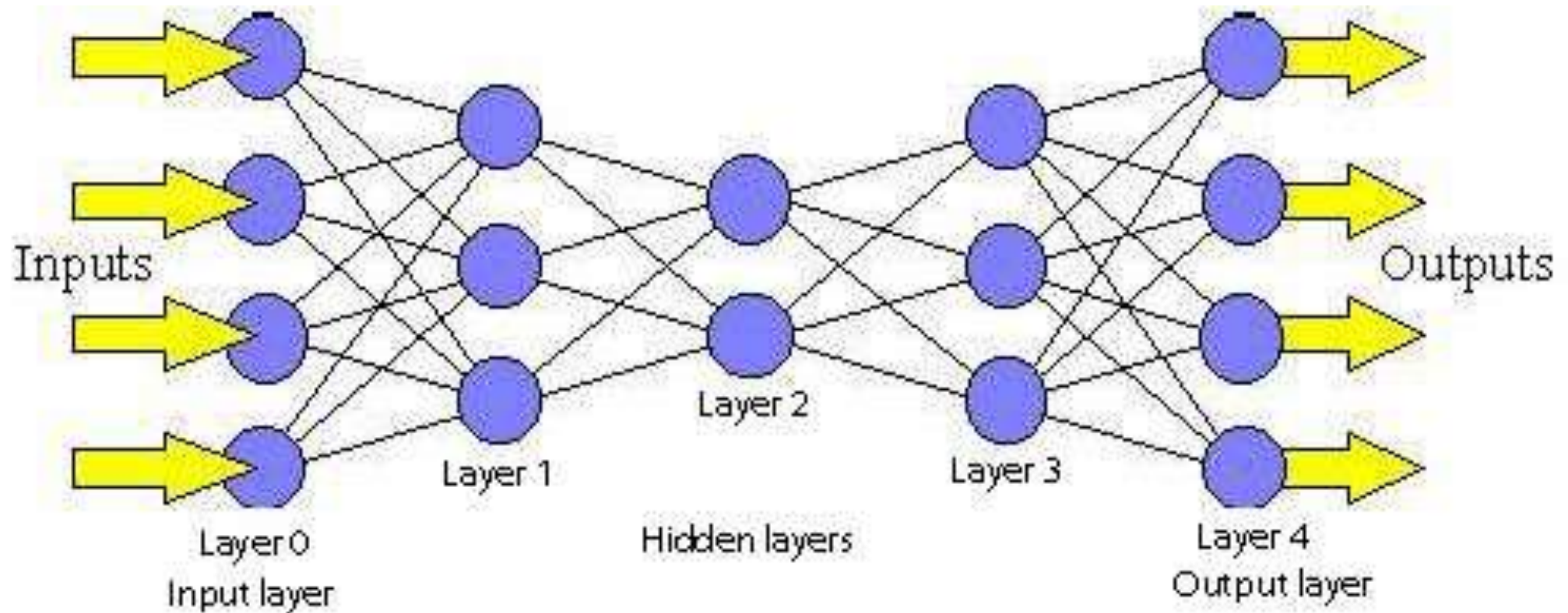
Proprietary content: ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited



# Multiple Layers – Feed Forward

- Process of calculating expected output
- Combines weights and activation functions with the inputs
- Iteratively performed over training set, and classifies test input

# Multiple Layers – Feed Forward - Composition of Functions



This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

Proprietary content: ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited

# Vector Calculus Refresher

This file is meant for personal use by manju.crz@gmail.com only.

Proprietary content: ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited

Let  $x \in R^n$  (a column vector) and let  $f : R^n \rightarrow R$ . The derivative of  $f$  with respect to  $x$  is the row vector:

$$\frac{\partial f}{\partial x} = \left( \frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_n} \right)$$

$\frac{\partial f}{\partial x}$  is called the gradient of  $f$ .

Let  $x \in R^n$  (a column vector) and let  $f : R^n \rightarrow R^m$ . The derivative of  $f$  with respect to  $x$  is the  $m \times n$  matrix:

$$\frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f(x)_1}{\partial x_1} & \dots & \frac{\partial f(x)_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f(x)_m}{\partial x_1} & \dots & \frac{\partial f(x)_m}{\partial x_n} \end{bmatrix}$$

$\frac{\partial f}{\partial x}$  is called the Jacobian matrix of  $f$ .

# Back propagation

This file is meant for personal use by manju.crz@gmail.com only.

Proprietary content: ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited

# Neural Network Constructed

Now, let's review the following in turn:

- Feed forward
- **Back propagation**
- Fully connected layer
- Activation functions
- Softmax function
- Cross-entropy loss

... and we'll have a fully functioning network

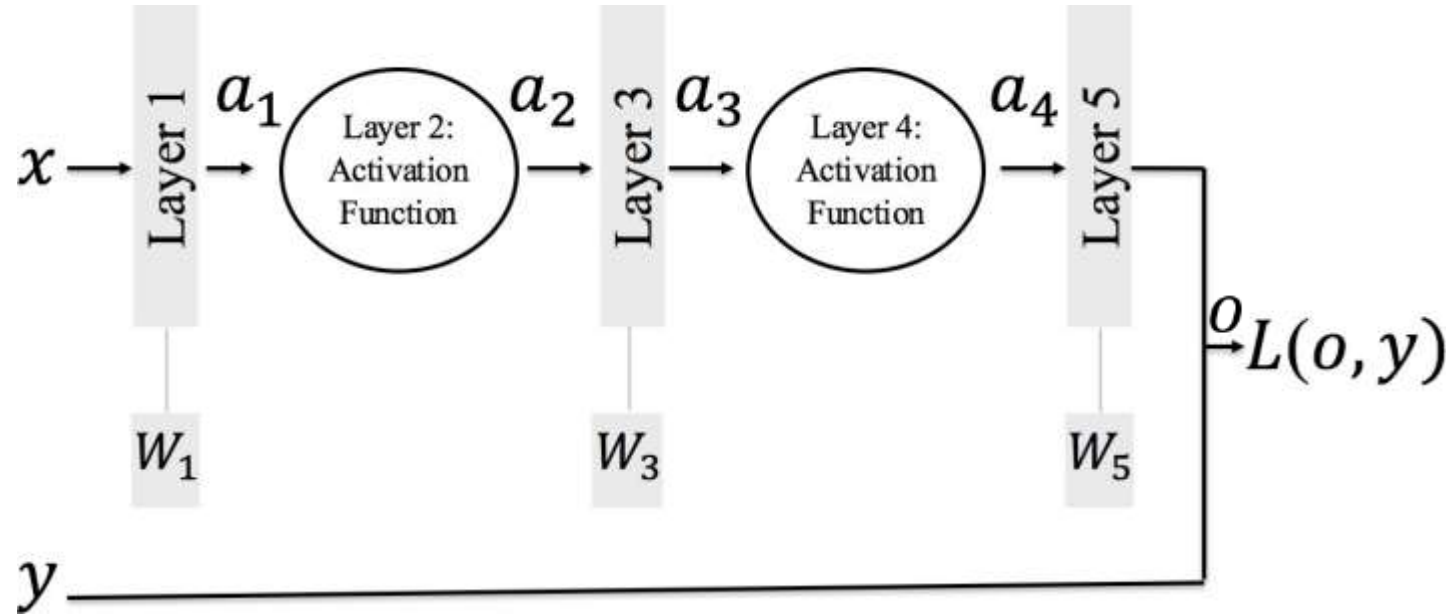
This file is meant for personal use by manju.crz@gmail.com only.

Proprietary content: ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited

# Multiple Layers – Back Prop

- At the end of each forward pass, we have a loss (difference between expected outcome and actual)
- The core of the back prop is a partial derivative of the Loss with respect to a weight - which tells us how quickly the Loss changes for any change in the weight
- Back Prop follows the chain rule of derivatives, i.e. the Loss can be computed for each and every weight in the network
- In practice, backward propagation is often abstracted away, because functions take care of it - but it's important to know how it works

# Multiple Layers – Back Prop: Chain Rule



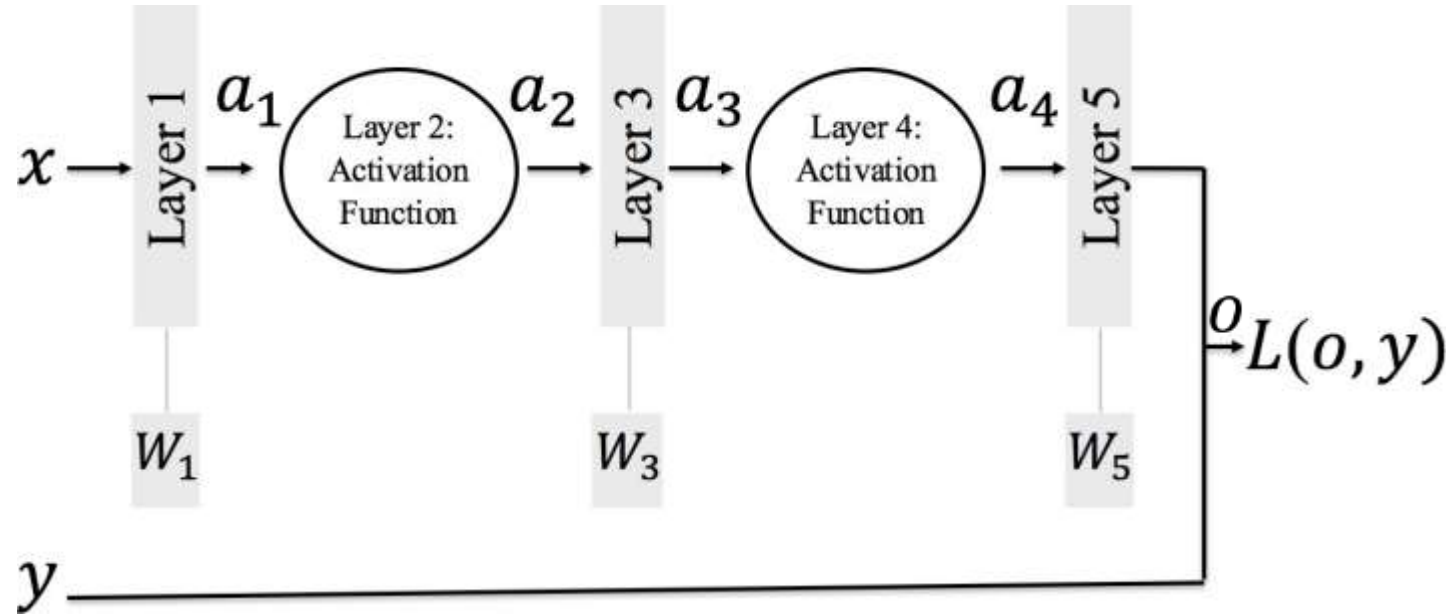
We want:  $\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_3}, \frac{\partial L}{\partial W_5}$

This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.



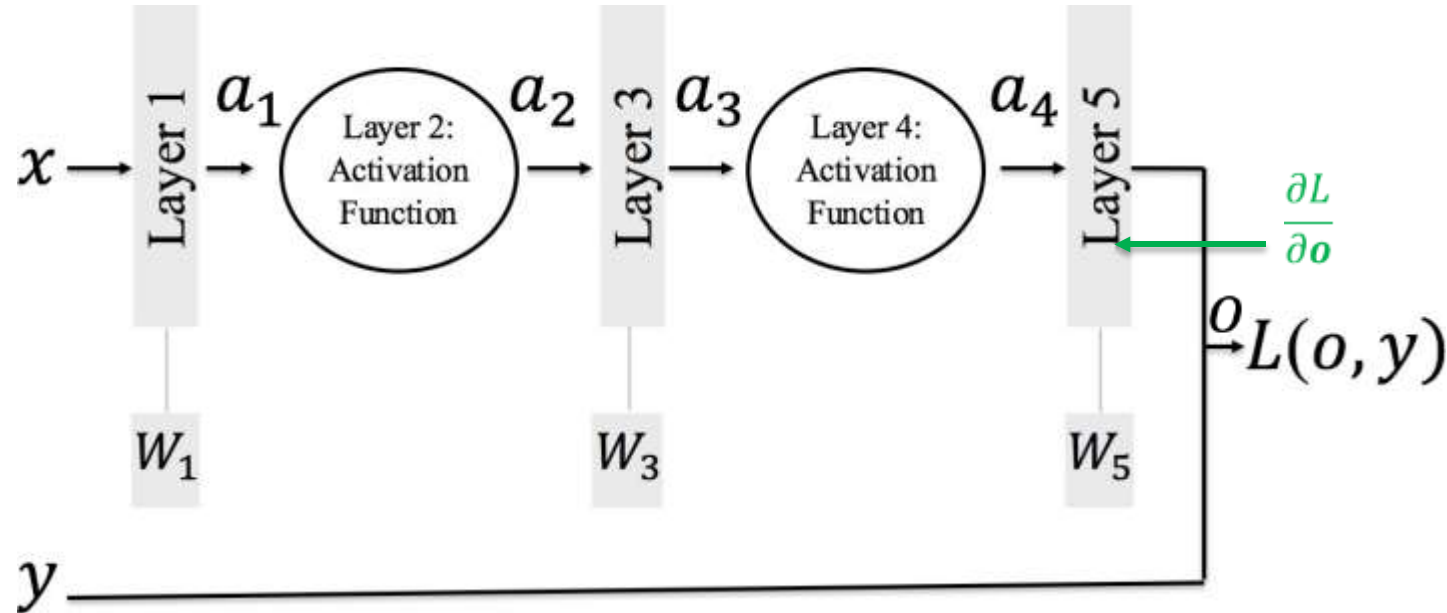
# Multiple Layers – Back Prop: Chain Rule



We want:  $\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_3}, \frac{\partial L}{\partial W_5}$       Compute:  $\frac{\partial L}{\partial o}$

This file is meant for personal use by manju.crz@gmail.com only.

# Multiple Layers – Back Prop: Chain Rule



We want:

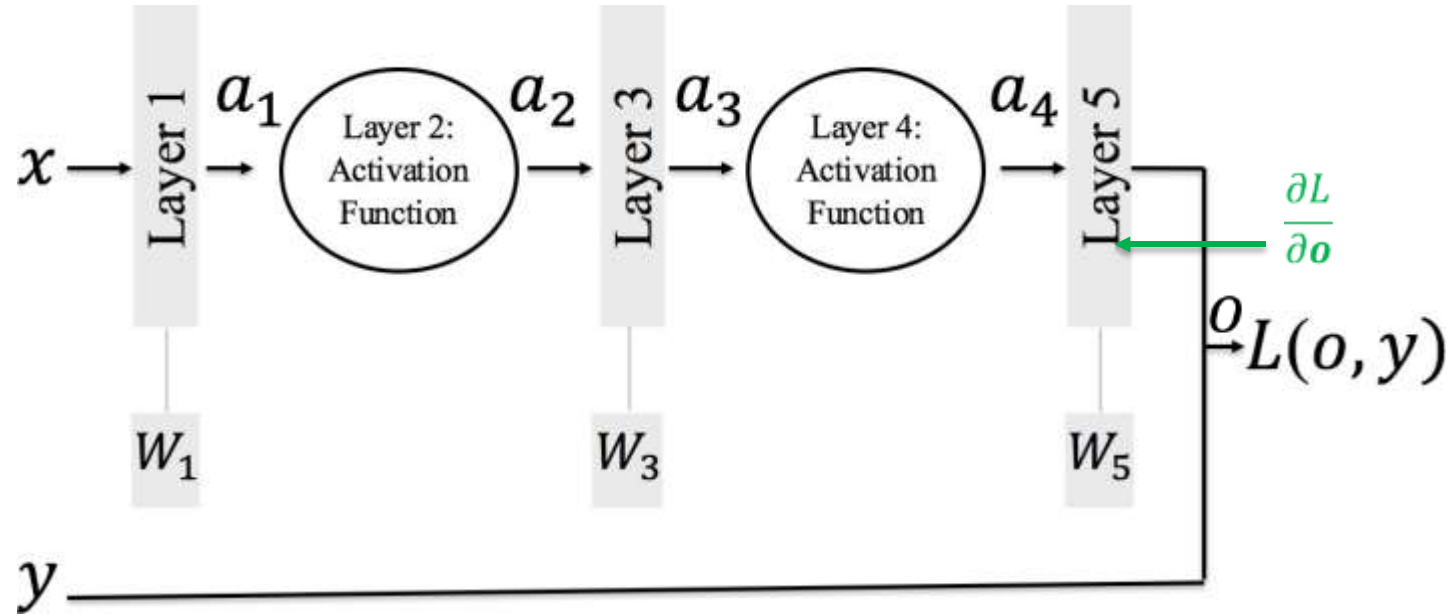
$$\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_3}, \frac{\partial L}{\partial W_5}$$

Compute:

$$\frac{\partial L}{\partial o}$$

E.g:  $L(o, y) = 1/2 \| o - y \|^2$  then:  $\frac{\partial L}{\partial o}$

# Multiple Layers – Back Prop: Chain Rule



We want:

$$\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_3}, \frac{\partial L}{\partial W_5}$$

Compute:

$$\frac{\partial L}{\partial o}$$

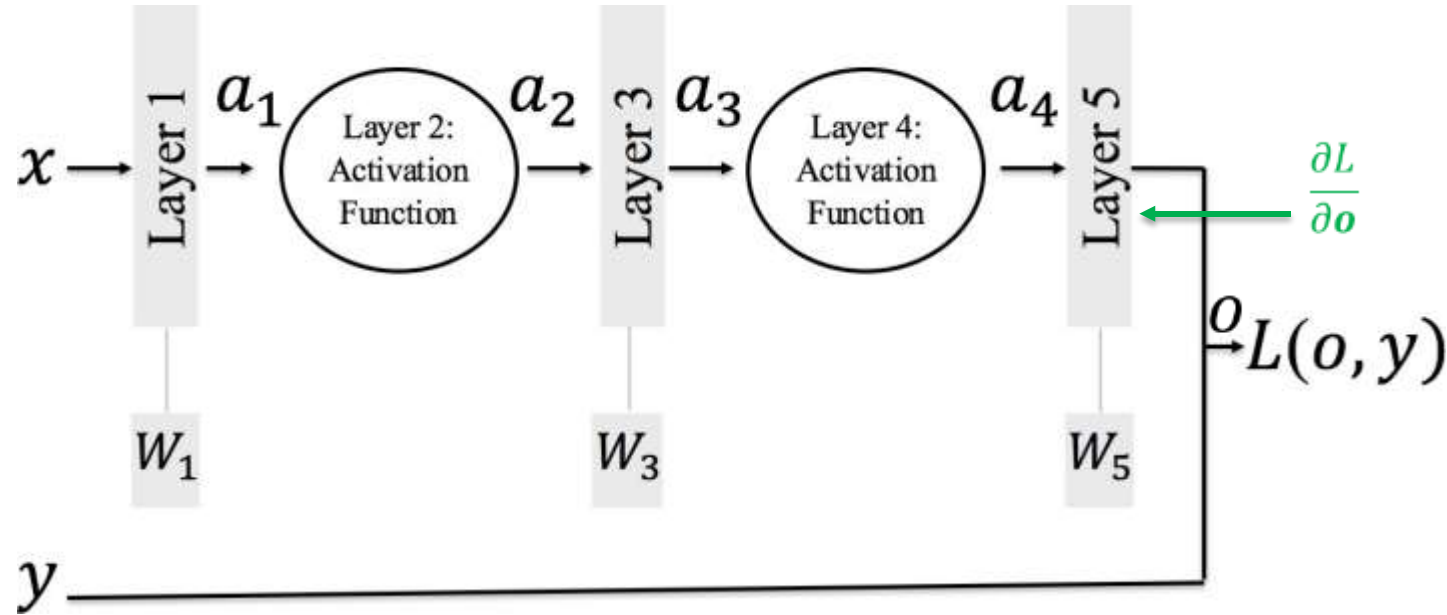
E.g:  $L(o, y) = 1/2 \| o - y \|^2$  then:  $\frac{\partial L}{\partial o} = (o - y)$

This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

Proprietary content: ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited

# Multiple Layers – Back Prop: Chain Rule



We want:

$$\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_3}, \frac{\partial L}{\partial W_5}$$

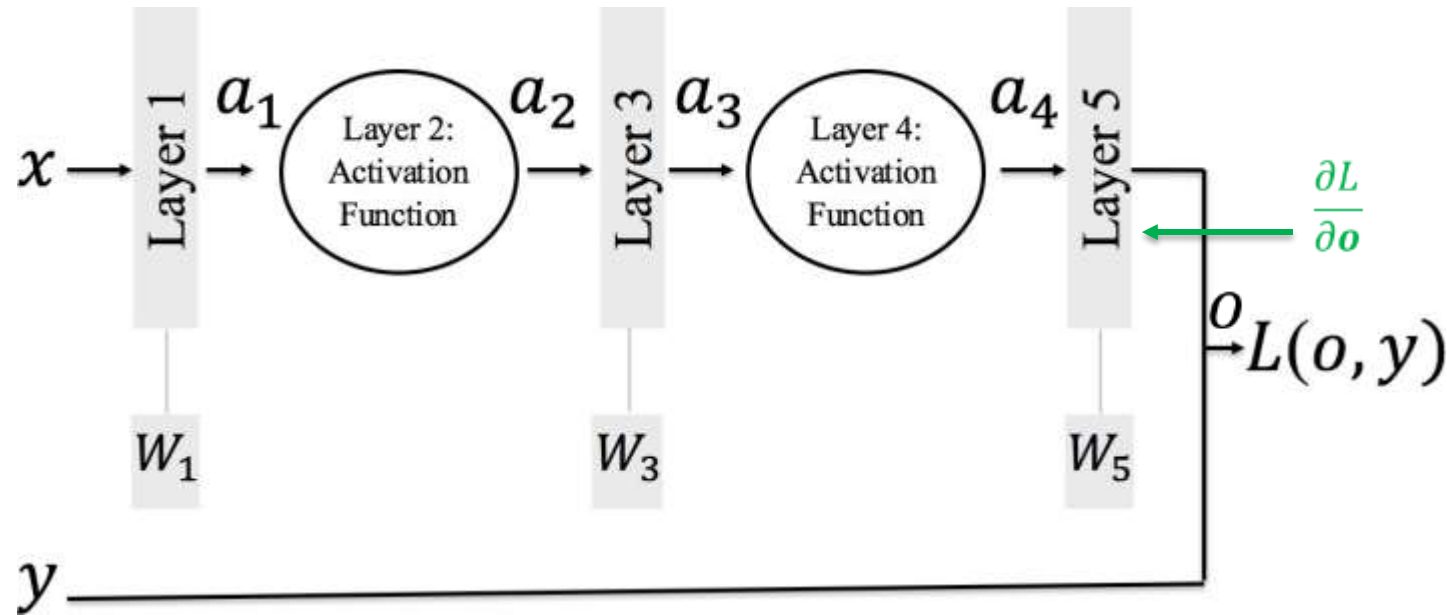
Compute:

$$\frac{\partial L}{\partial o}$$

E.g:  $L(o, y) = 1/2 \| o - y \|^2$  then:  $\frac{\partial L}{\partial o} = (o - y)$

$$\frac{\partial L}{\partial o} \in \mathbb{R}^{1 \times m}$$

# Multiple Layers – Back Prop: Chain Rule



We want:

$$\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_3}, \frac{\partial L}{\partial W_5}$$

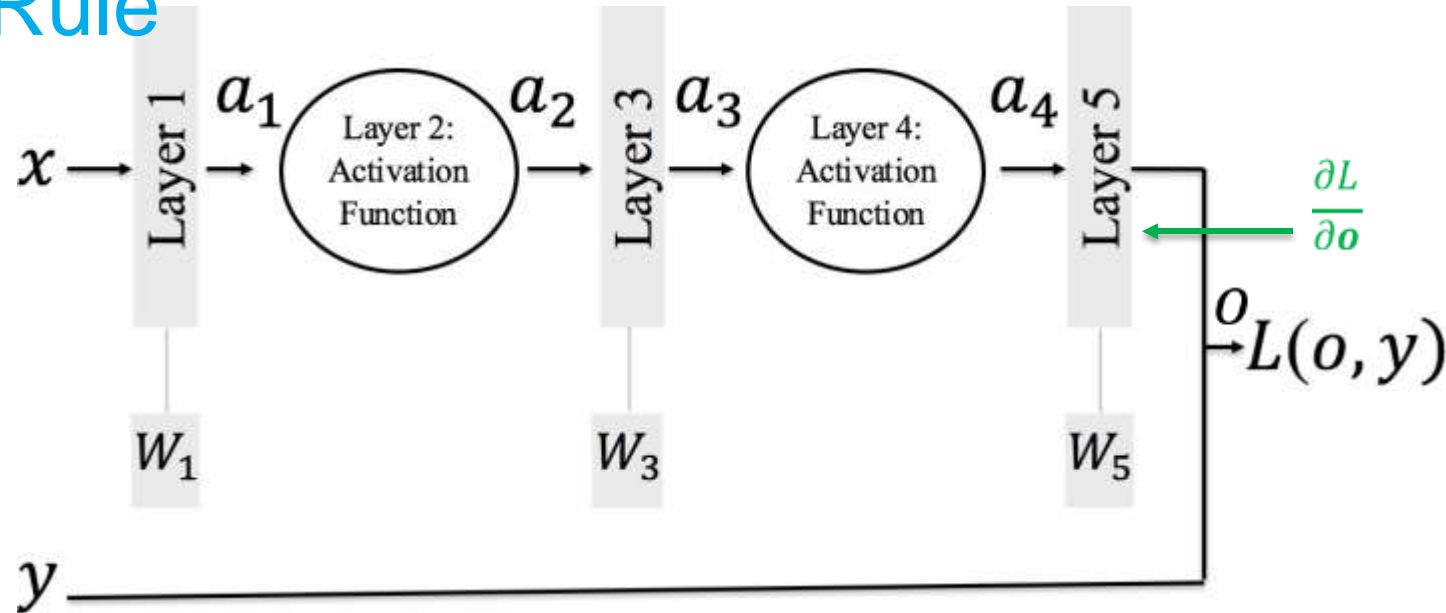
Compute:

$$\frac{\partial L}{\partial o}, \frac{\partial o}{\partial W_5}$$

This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

# Multiple Layers – Back Prop: Chain Rule



Some differentiable function with respect to  $a_4$  and  $W_5$ , so some are also calling this differentiable programming

We want:

$$\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_3}, \frac{\partial L}{\partial W_5}$$

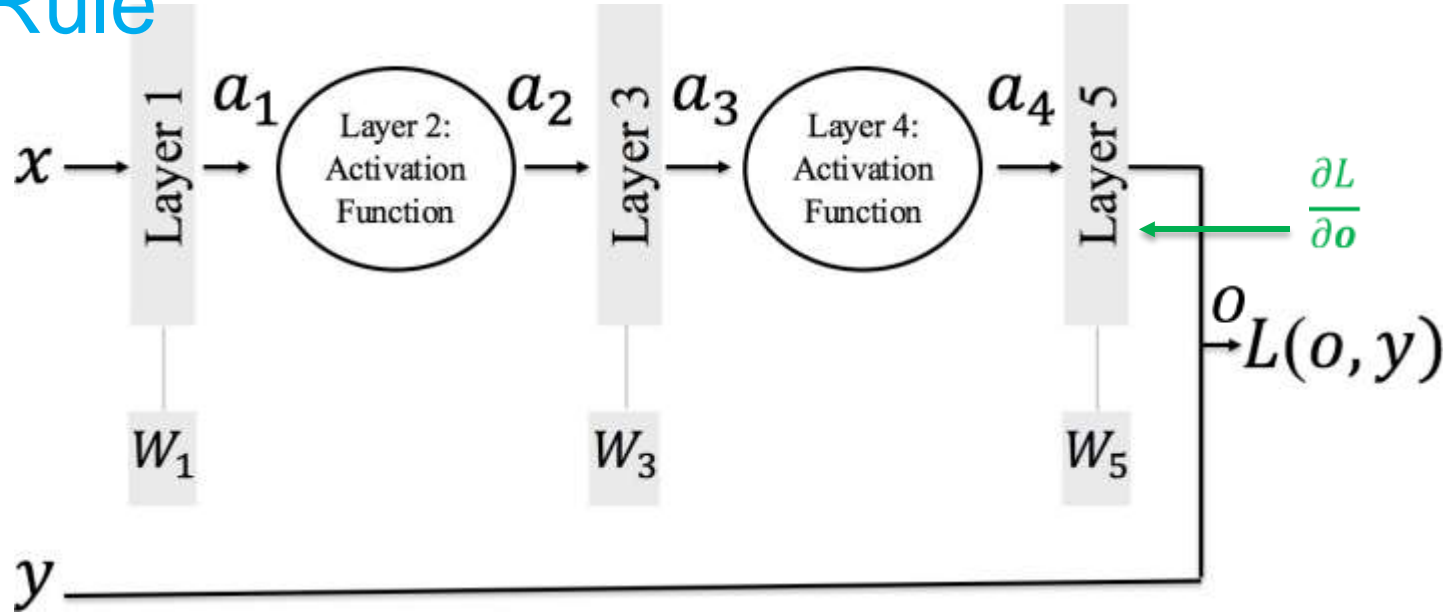
Compute:

$$\frac{\partial L}{\partial o}, \frac{\partial o}{\partial W_5}$$

since:  $o = K(a_4, W_5)$

then:  $\frac{\partial o}{\partial W_5}$  is a Jacobian of size  $\mathbb{R}^{m \times \dim(W_5)}$

# Multiple Layers – Back Prop: Chain Rule



We want:

$$\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_3}, \frac{\partial L}{\partial W_5}$$

Compute:

$$\frac{\partial L}{\partial o}, \frac{\partial o}{\partial W_5}$$

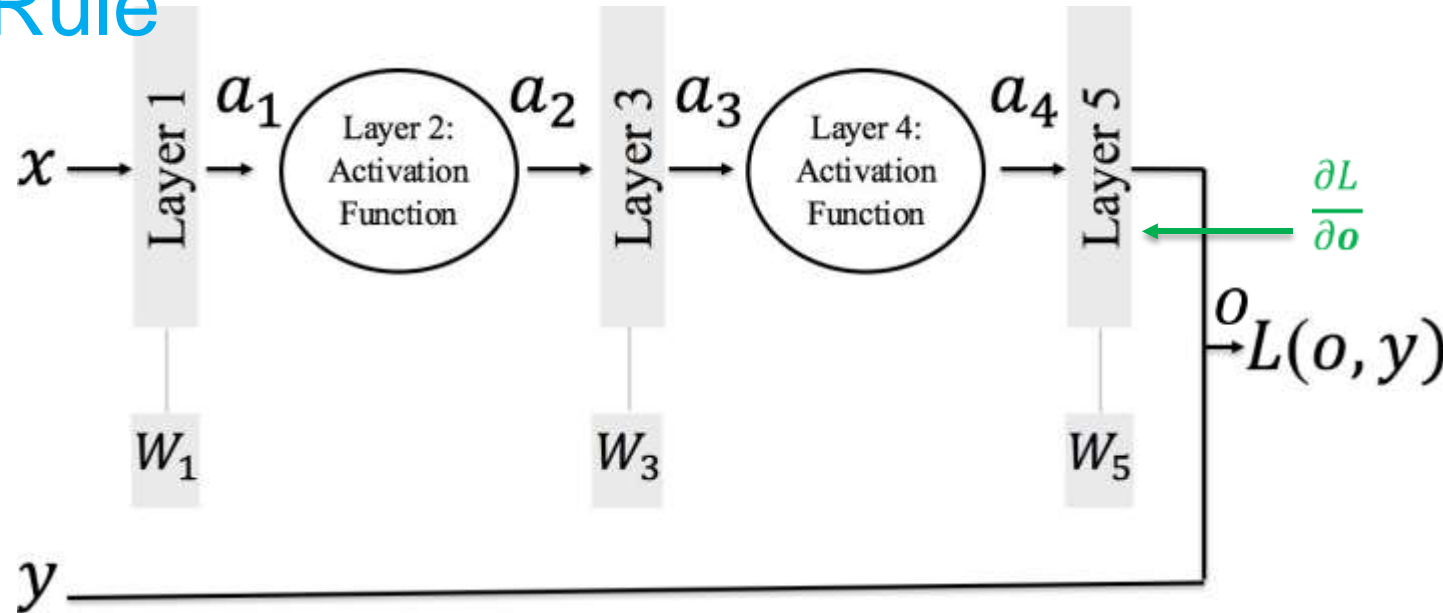
since:  $o = K(a_4, W_5)$

then: 
$$\left[ \frac{\partial o}{\partial W_5} \right]_{kl} = \frac{\partial [K(a_4, W_5)]_k}{\partial [W_5]_l}$$

This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

# Multiple Layers – Back Prop: Chain Rule



Element (k, 1) of the jacobian indicates how much the k-th output wiggles when we wiggle the 1-th weight

We want:

$$\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_3}, \frac{\partial L}{\partial W_5}$$

Compute:

$$\frac{\partial L}{\partial o}, \frac{\partial o}{\partial W_5}$$

since:  $o = K(a_4, W_5)$

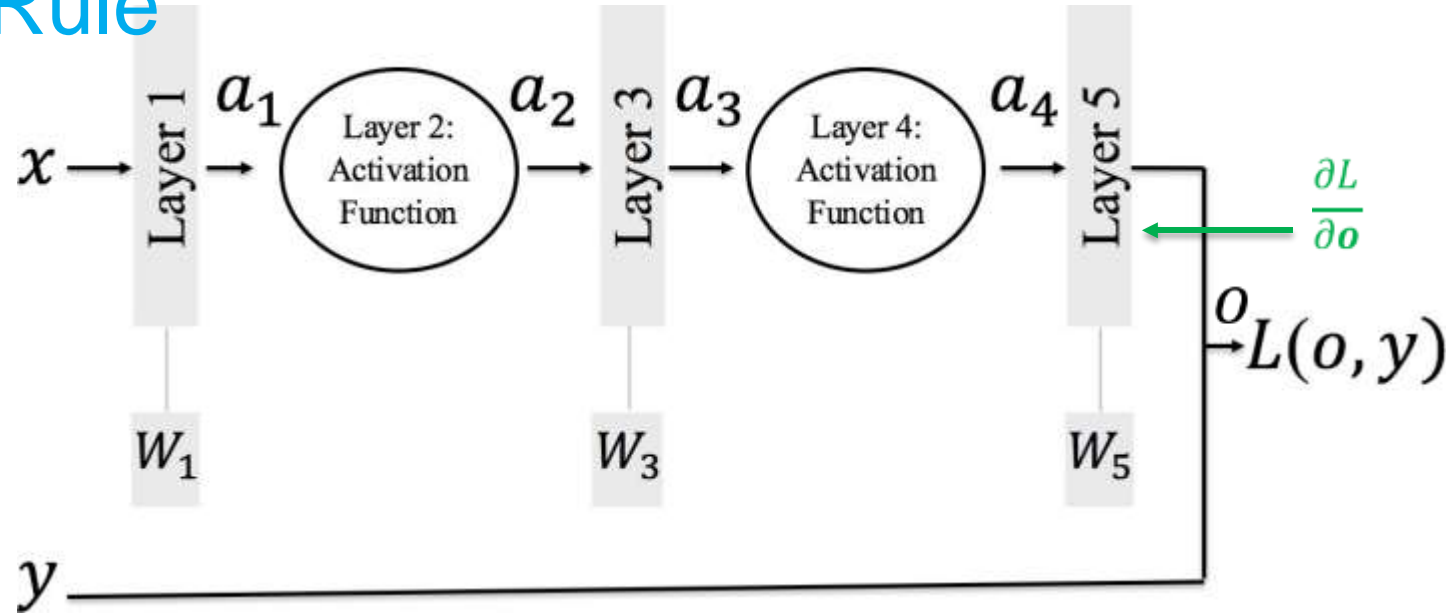
then:  $\left[ \frac{\partial o}{\partial W_5} \right]_{kl} = \frac{\partial [K(a_4, W_5)]_k}{\partial [W_5]_l}$

This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.



# Multiple Layers – Back Prop: Chain Rule



We want:

$$\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_3}, \frac{\partial L}{\partial W_5}$$

Compute:

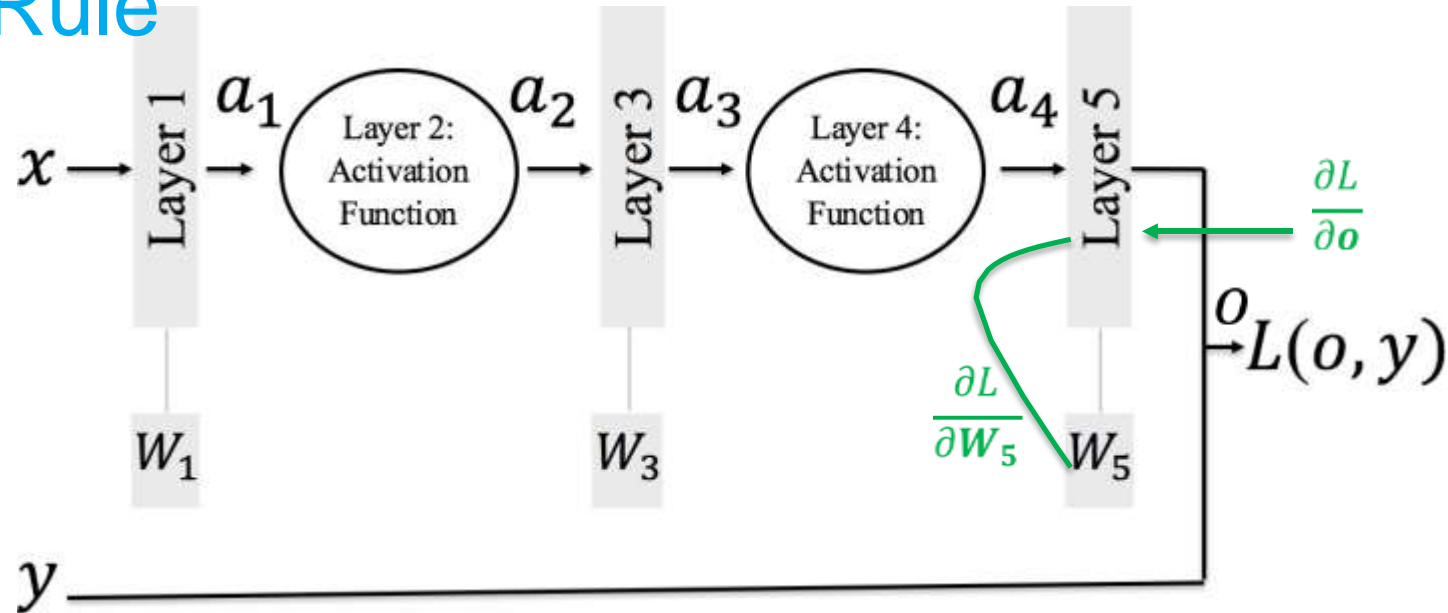
$$\frac{\partial L}{\partial W_5} = \frac{\partial L}{\partial o} \times \frac{\partial o}{\partial W_5}$$

Remember:

$$\frac{\partial L}{\partial o} \in \mathbb{R}^{1 \times m}$$

$$\frac{\partial o}{\partial W_5} \in \mathbb{R}^{1 \times \dim(W_5)}$$

# Multiple Layers – Back Prop: Chain Rule



We want:

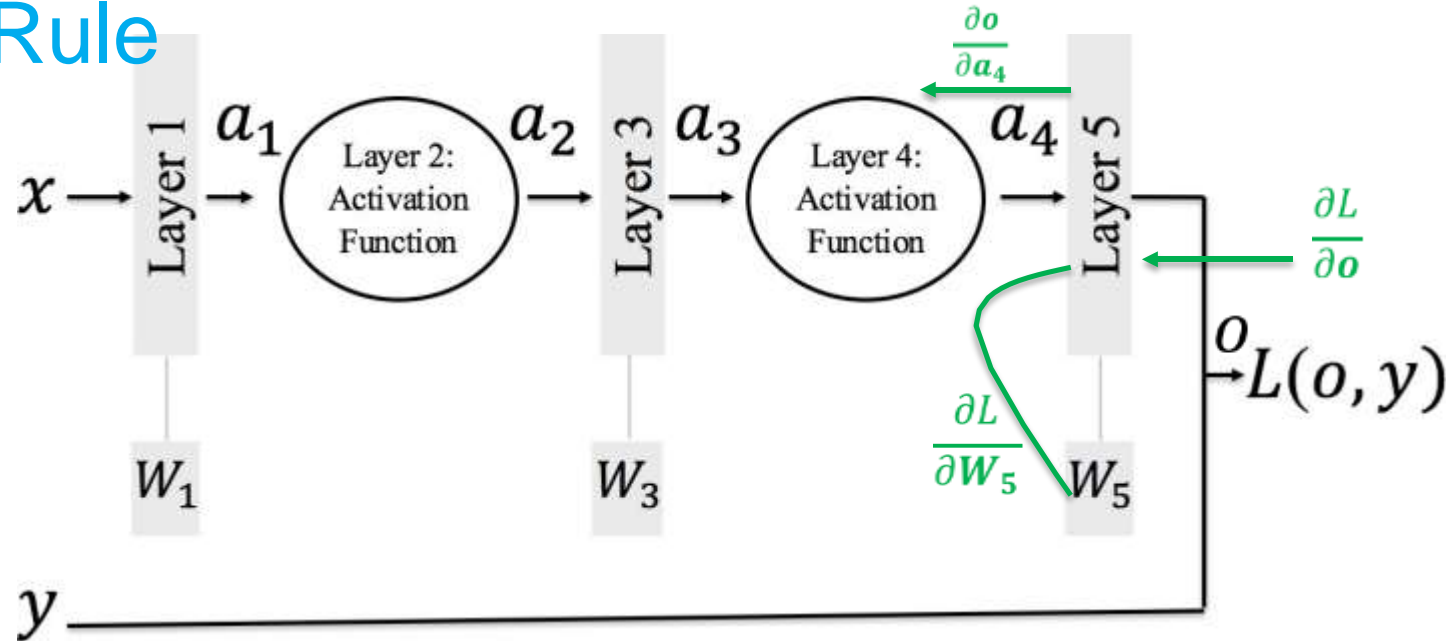
$$\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_3}, \frac{\partial L}{\partial W_5}$$

Compute:

$$\frac{\partial L}{\partial W_5} = \frac{\partial L}{\partial o} \times \frac{\partial o}{\partial W_5} \in \mathbb{R}^{1 \times \dim(W_5)}$$

This file is meant for personal use by manju.crz@gmail.com only.

# Multiple Layers – Back Prop: Chain Rule



We want:

$$\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_3}, \frac{\partial L}{\partial W_5}$$

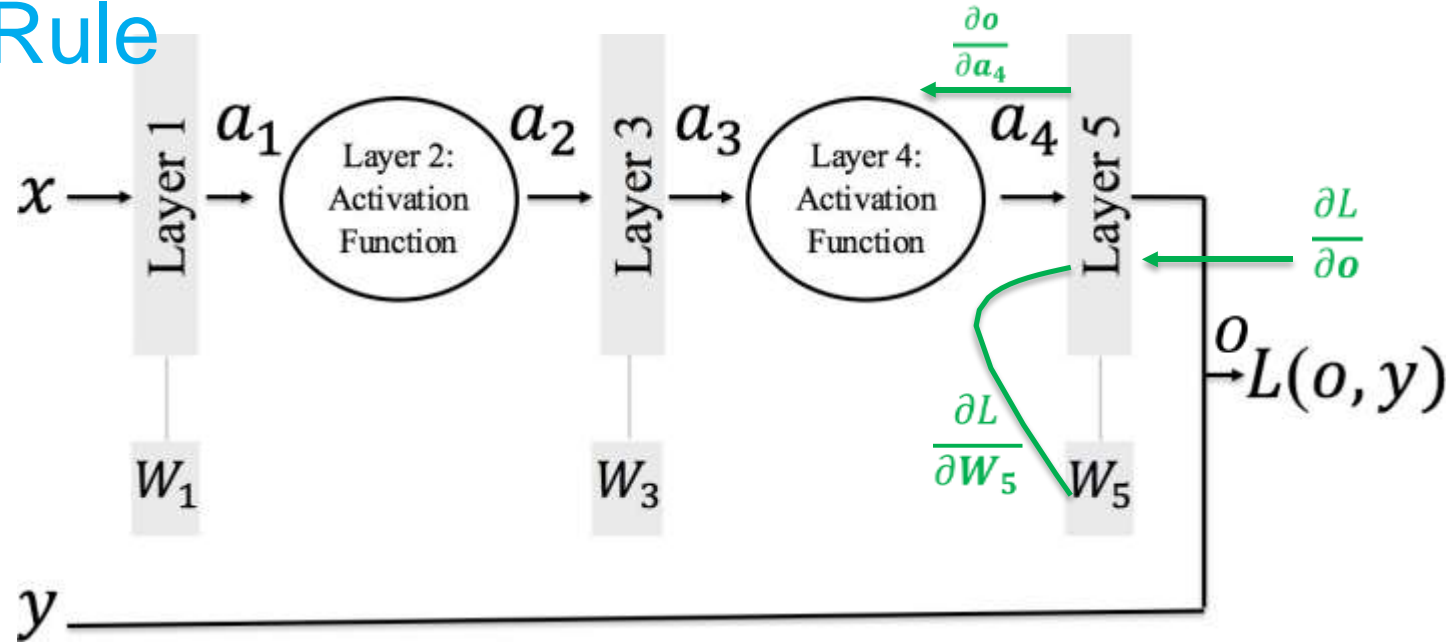
since:  $o = K(a_4, W_5)$

then:  $\frac{\partial o}{\partial a_4}$  is a Jacobian of size  $\mathbb{R}^{m \times \dim(a_4)}$

This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

# Multiple Layers – Back Prop: Chain Rule



We want:

$$\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_3}, \frac{\partial L}{\partial W_5}$$

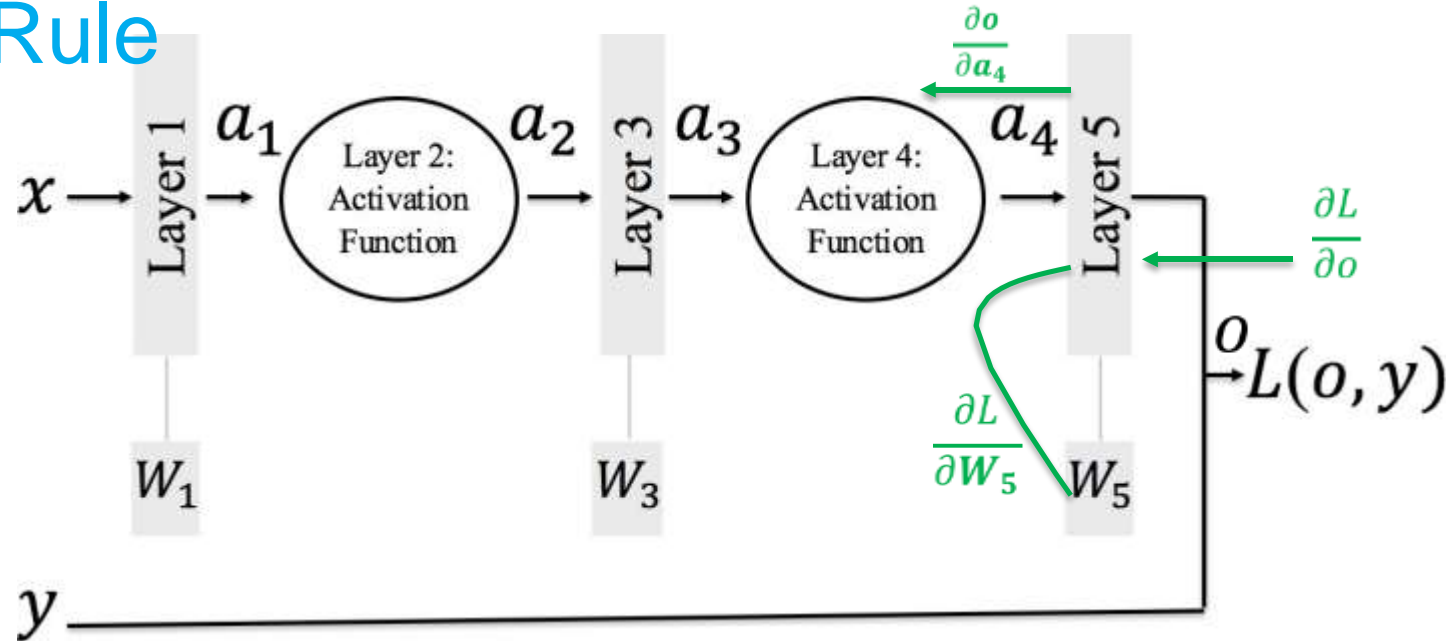
since:  $o = K(a_4, W_5)$

$$\text{then: } \left[ \frac{\partial o}{\partial a_4} \right]_{kl} = \frac{\partial [K(a_4, W_5)]_k}{\partial [a_4]_l}$$

This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

# Multiple Layers – Back Prop: Chain Rule



Element  $(k, 1)$  of the jacobian indicates how much the  $k$ -th output wiggles when we wiggle the 1-th output of the previous layer (or input to this layer)

We want:

$$\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_3}, \frac{\partial L}{\partial W_5}$$

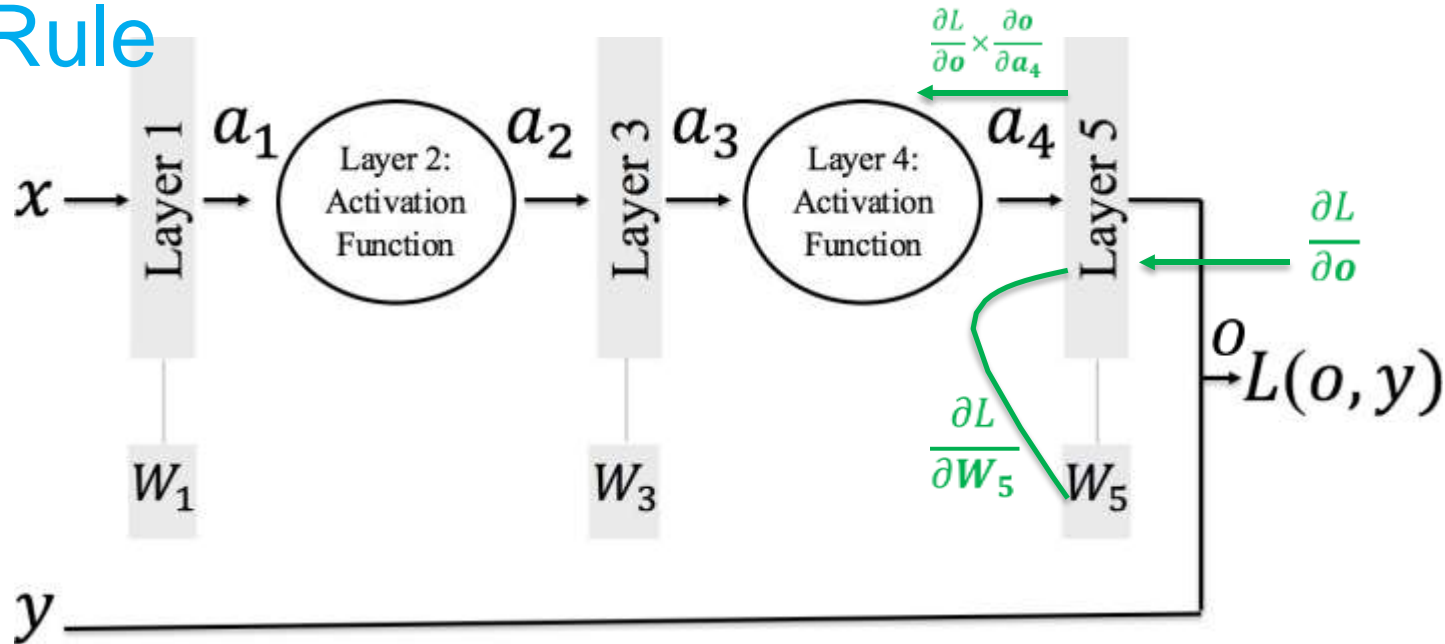
since:  $o = K(a_4, W_5)$

$$\text{then: } \left[ \frac{\partial o}{\partial a_4} \right]_{kl} = \frac{\partial [K(a_4, W_5)]_k}{\partial [a_4]_l}$$

This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

# Multiple Layers – Back Prop: Chain Rule



We want:

$$\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_3}, \frac{\partial L}{\partial W_5}$$

Backpropagate:

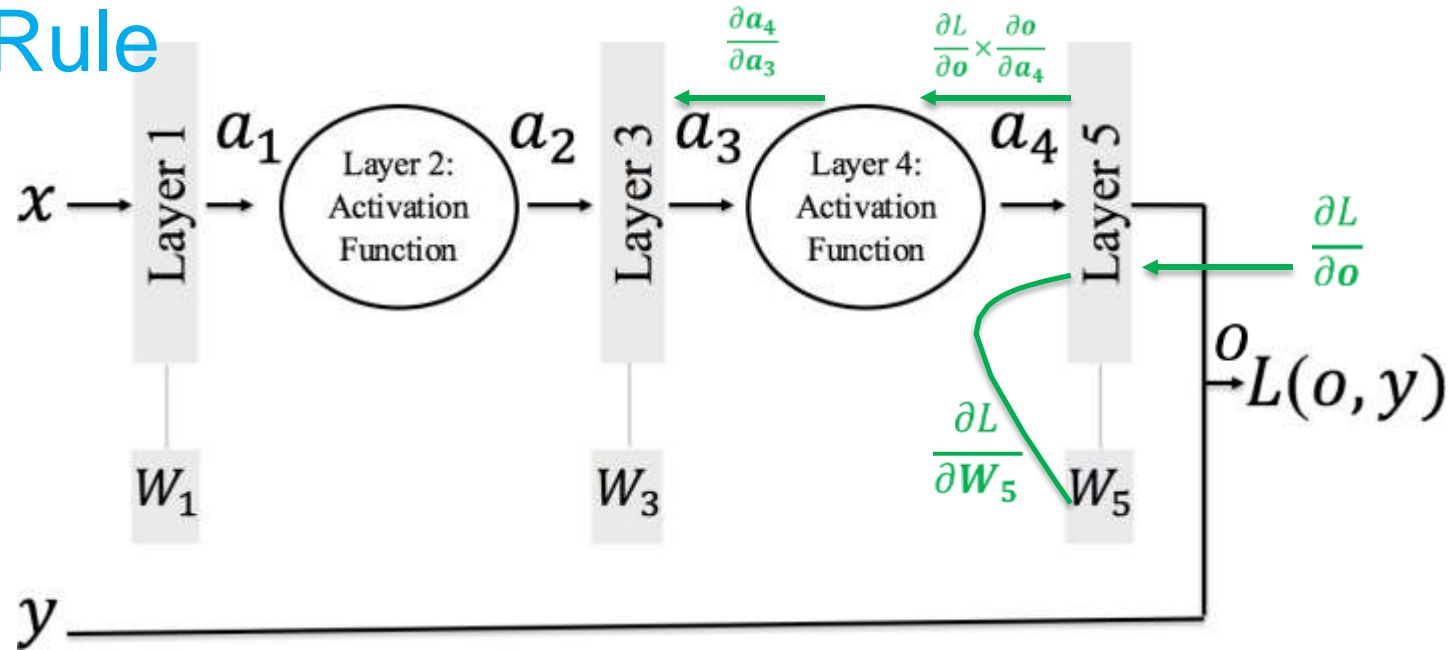
$$\frac{\partial L}{\partial o} \times \frac{\partial o}{\partial a_4} \in \mathbb{R}^{1 \times \dim(a_4)}$$

Remember:

$$\frac{\partial L}{\partial o} \in \mathbb{R}^{1 \times m}$$

$$\frac{\partial o}{\partial a_4} \in \mathbb{R}^{m \times \dim(a_4)}$$

# Multiple Layers – Back Prop: Chain Rule



We want:

$$\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_3}, \frac{\partial L}{\partial W_5}$$

since:  $a_4 = J(a_3)$

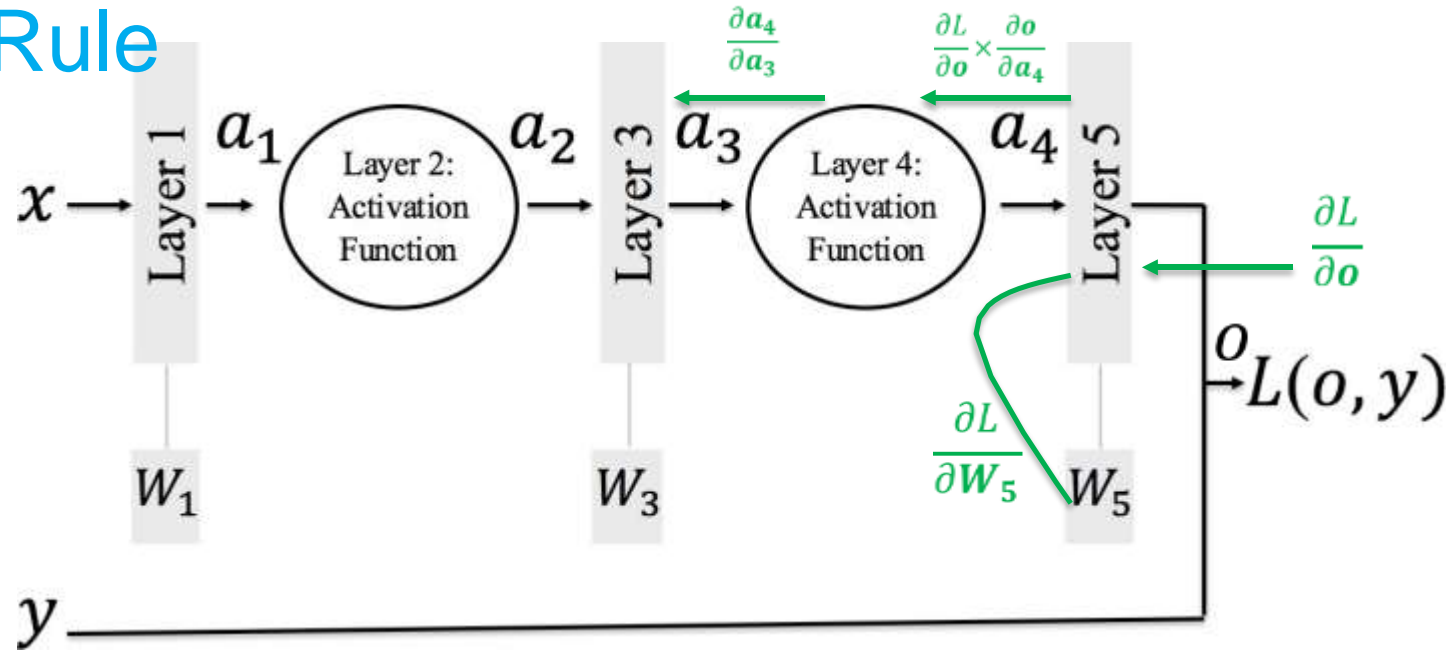
then:  $\frac{\partial a_4}{\partial a_3}$  is a Jacobian of size  $\mathbb{R}^{\dim(a_4) \times \dim(a_3)}$

This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.



# Multiple Layers – Back Prop: Chain Rule



Remember:

$$\frac{\partial o}{\partial a_4} \times \frac{\partial L}{\partial o} \in \mathbb{R}^{1 \times \dim(a_4)}$$

$$\frac{\partial a_4}{\partial a_3} \in \mathbb{R}^{\dim(a_4) \times \dim(a_3)}$$

We want:

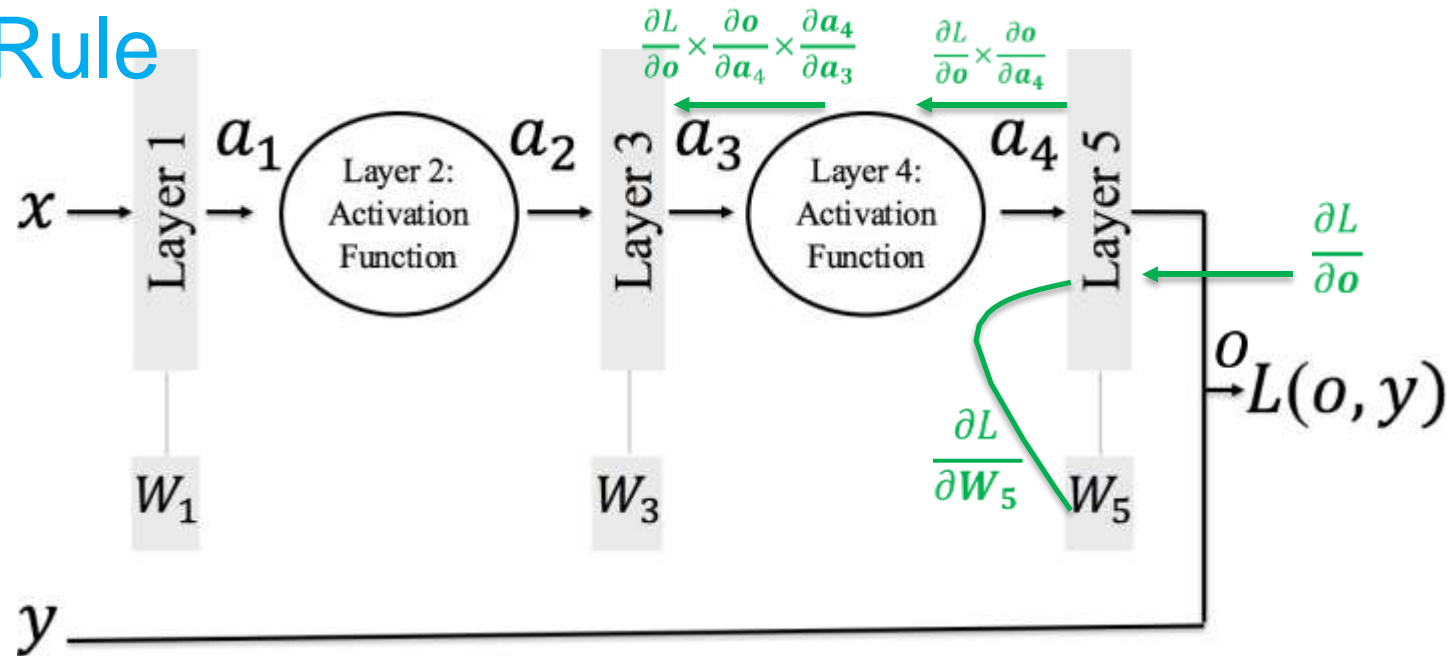
$$\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_3}, \frac{\partial L}{\partial W_5}$$

since:  $a_4 = J(a_3)$

then:  $\frac{\partial a_4}{\partial a_3}$  is a Jacobian of size  $\mathbb{R}^{\dim(a_4) \times \dim(a_3)}$



# Multiple Layers – Back Prop: Chain Rule



Remember:

$$\frac{\partial o}{\partial a_4} \times \frac{\partial L}{\partial o} \in \mathbb{R}^{1 \times \dim(a_4)}$$

$$\frac{\partial a_4}{\partial a_3} \in \mathbb{R}^{\dim(a_4) \times \dim(a_3)}$$

We want:

$$\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_3}, \frac{\partial L}{\partial W_5}$$

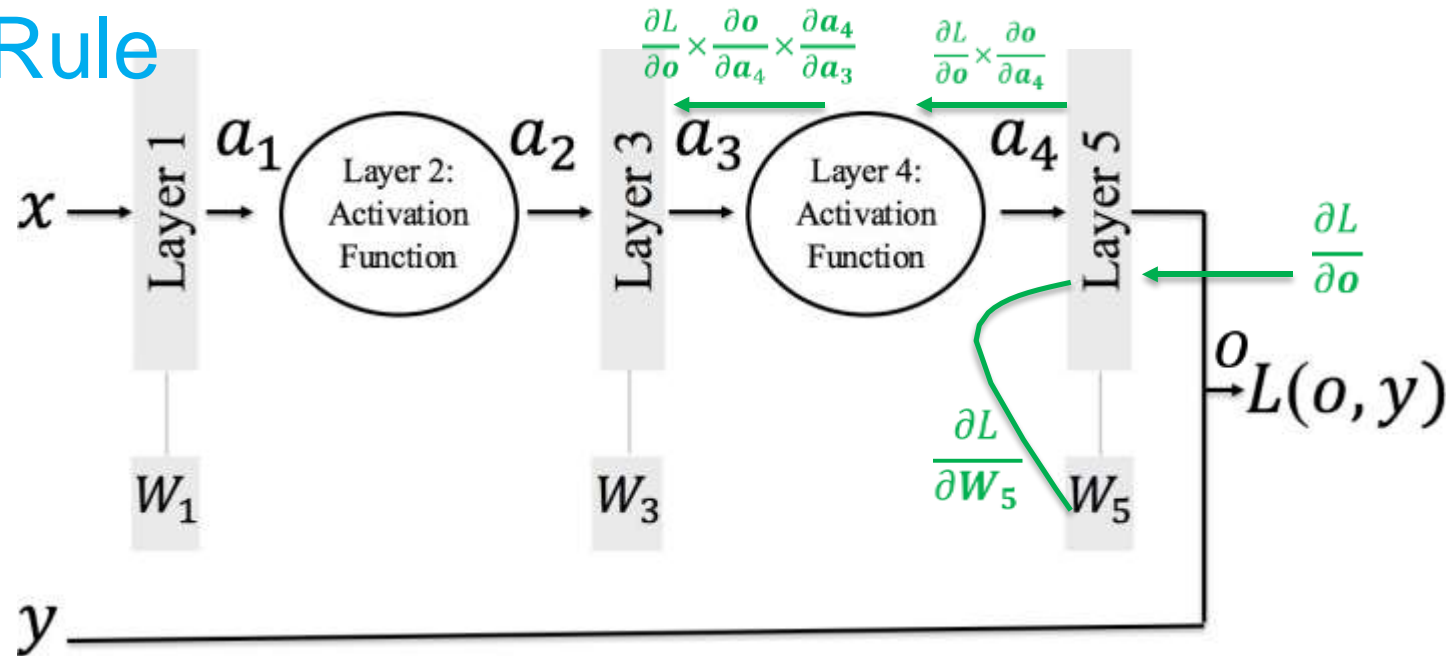
Backpropagate:

$$\frac{\partial L}{\partial o} \times \frac{\partial o}{\partial a_4} \times \frac{\partial a_4}{\partial a_3} \in \mathbb{R}^{1 \times \dim(a_3)}$$

This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

# Multiple Layers – Back Prop: Chain Rule



We want:

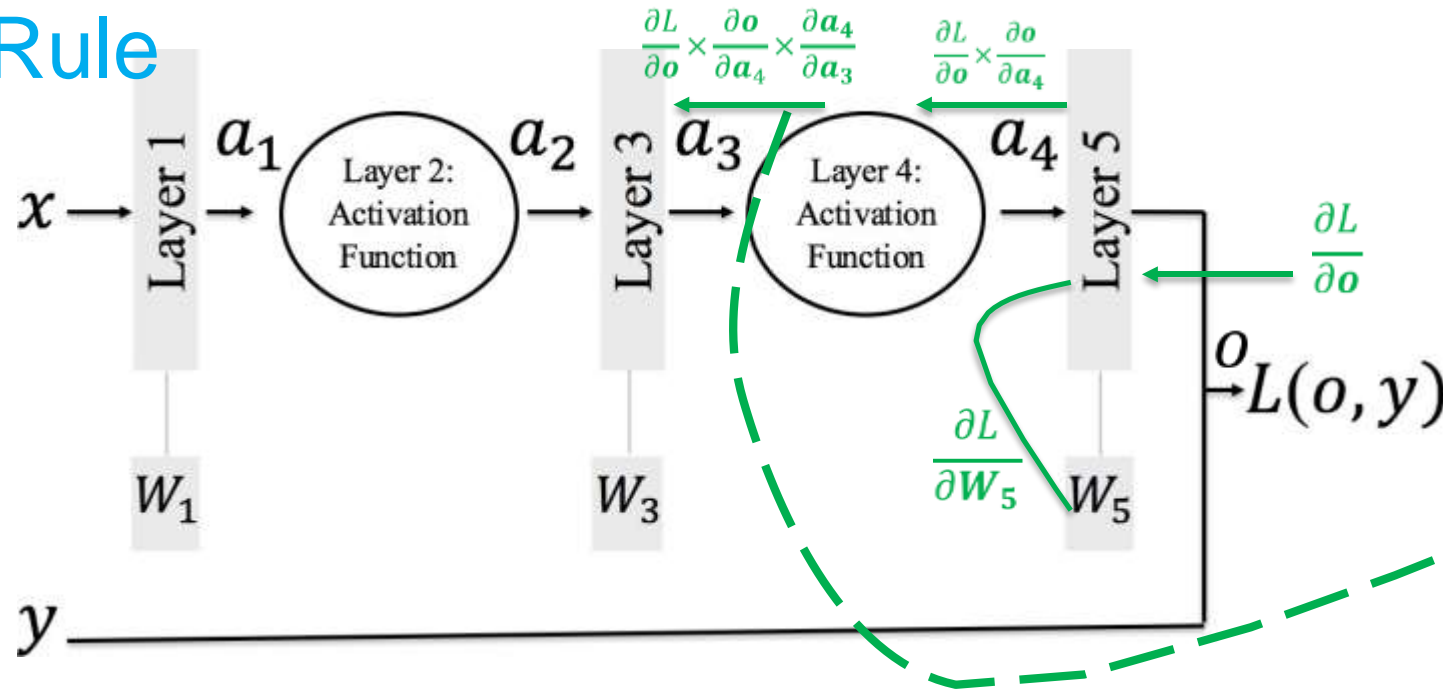
$$\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_3}, \frac{\partial L}{\partial W_5}$$

since:  $a_3 = K(a_2, W_3)$

then:  $\frac{\partial a_3}{\partial W_3}$  is a Jacobian of size  $\mathbb{R}^{\dim(a_3) \times \dim(W_3)}$

This file is meant for personal use by manju.crz@gmail.com only.

# Multiple Layers – Back Prop: Chain Rule



Remember:

$$\frac{\partial L}{\partial o} \times \frac{\partial o}{\partial a_4} \times \frac{\partial a_4}{\partial a_3} \in \mathbb{R}^{1 \times \dim(a_3)}$$

$$\frac{\partial a_3}{\partial W_3} \in \mathbb{R}^{\dim(a_3) \times \dim(W_3)}$$

We want:

$$\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_3}, \frac{\partial L}{\partial W_5}$$

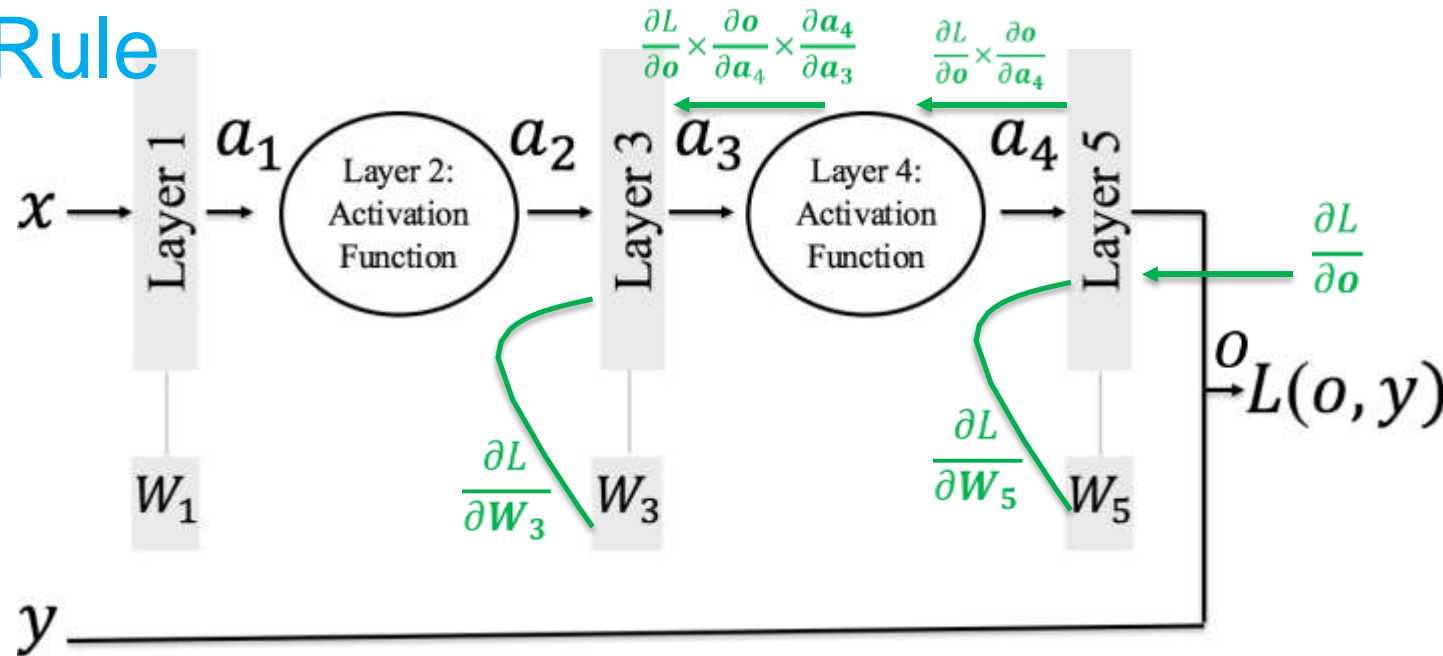
Now we can compute:

$$\frac{\partial L}{\partial W_3} = \frac{\partial L}{\partial o} \times \frac{\partial o}{\partial a_4} \times \frac{\partial a_4}{\partial a_3} \times \frac{\partial a_3}{\partial W_3}$$

This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

# Multiple Layers – Back Prop: Chain Rule



Remember:

$$\frac{\partial L}{\partial o} \times \frac{\partial o}{\partial a_4} \times \frac{\partial a_4}{\partial a_3} \in \mathbb{R}^{1 \times \dim(a_3)}$$

$$\frac{\partial a_3}{\partial W_3} \in \mathbb{R}^{\dim(a_3) \times \dim(W_3)}$$

We want:

$$\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_3}, \frac{\partial L}{\partial W_5}$$

Now we can compute:

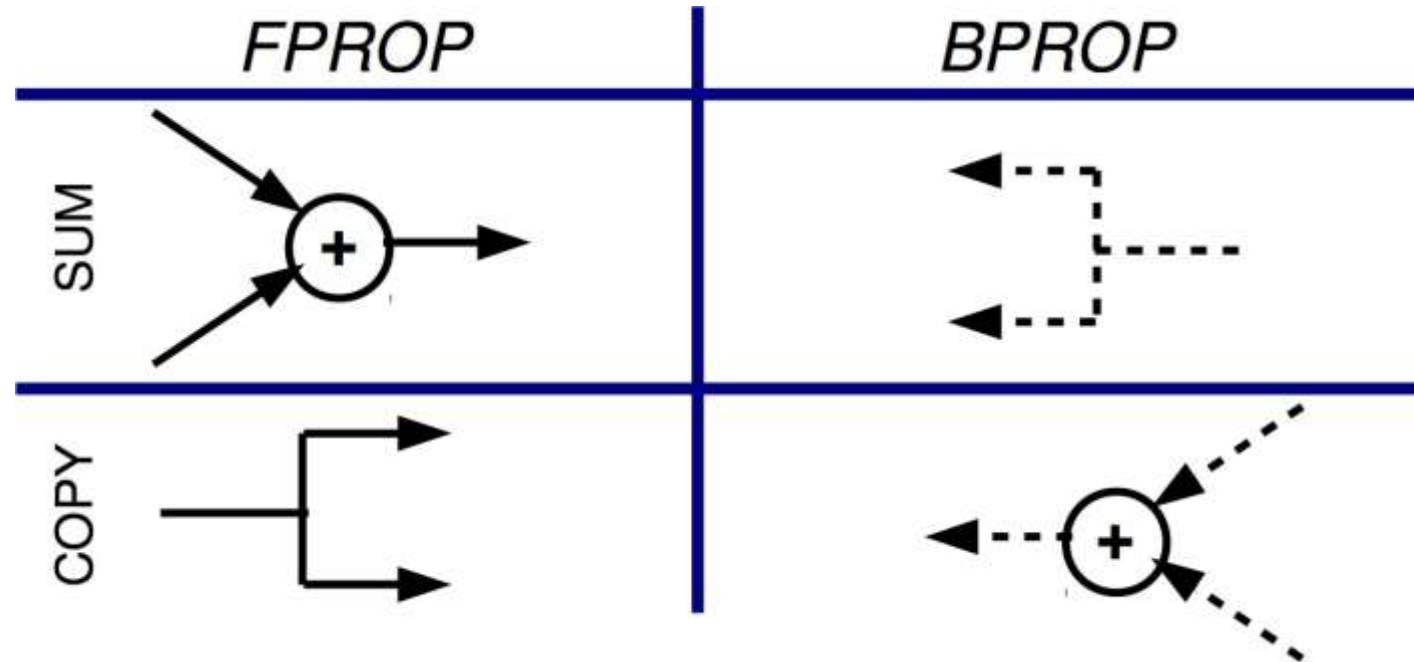
$$\frac{\partial L}{\partial W_3} = \frac{\partial L}{\partial o} \times \frac{\partial o}{\partial a_4} \times \frac{\partial a_4}{\partial a_3} \times \frac{\partial a_3}{\partial W_3}$$

This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

# Multiple Layers – Back Prop: Chain Rule

FPROP and BPROP are duals of each other:



This file is meant for personal use by manju.crz@gmail.com only.

# Fully Connected

This file is meant for personal use by manju.crz@gmail.com only.

Proprietary content: ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited

# Neural Network Constructed

Now, let's review the following in turn:

- Feed forward
- Back propagation
- Fully connected layer
- Activation functions
- Softmax function
- Cross-entropy loss

... and we'll have a fully functioning network

This file is meant for personal use by manju.crz@gmail.com only.

Proprietary content: ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited

# Fully Connected

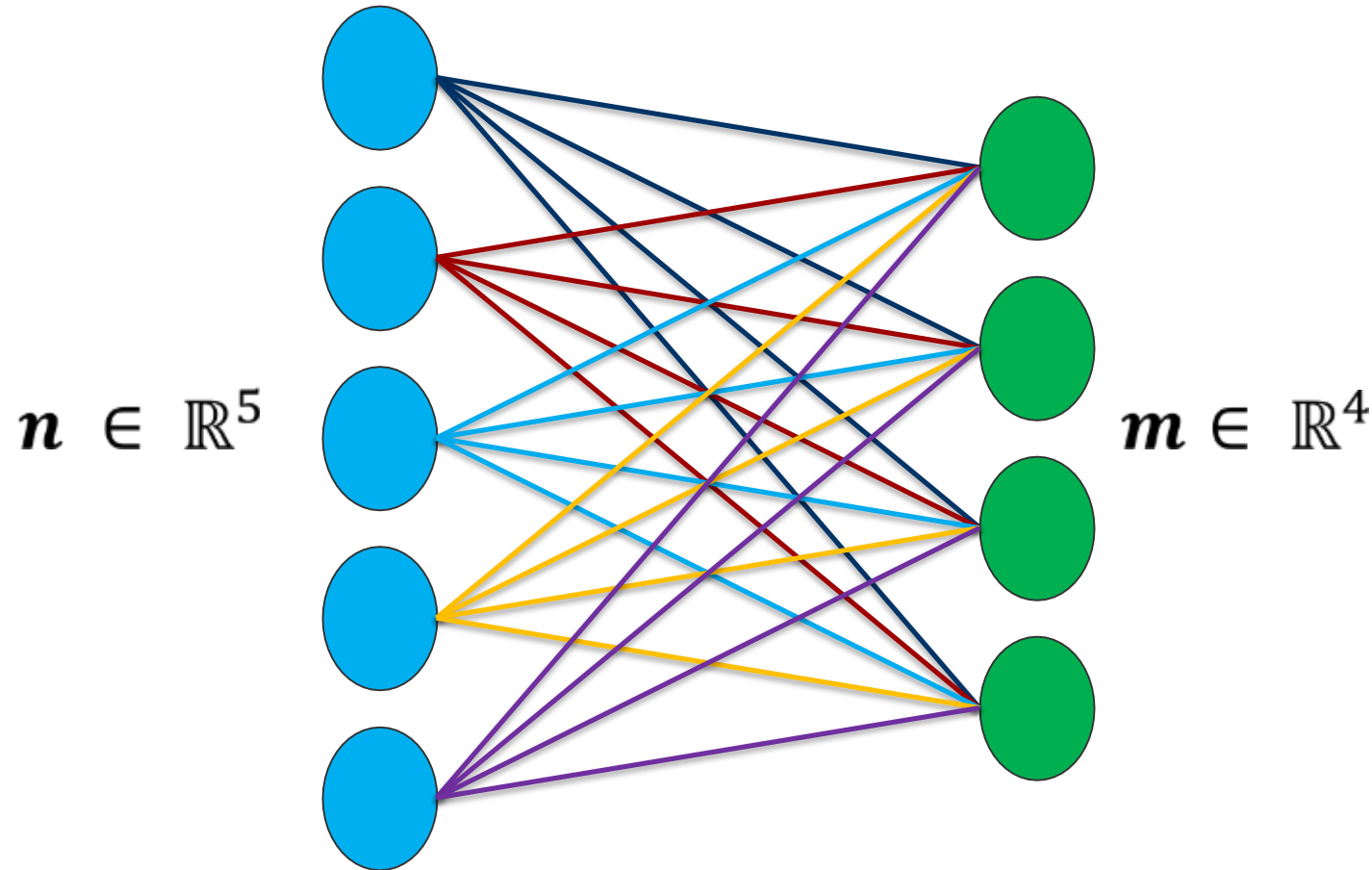
- Neurons have connections to all activations of the previous layer
- Number of connections add up very quickly due to all the combinations
- Forward pass of a fully connected layer -> one matrix multiplication followed by a bias offset and activation function (you'll soon see what we mean)

This file is meant for personal use by manju.crz@gmail.com only.

Proprietary content: ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited



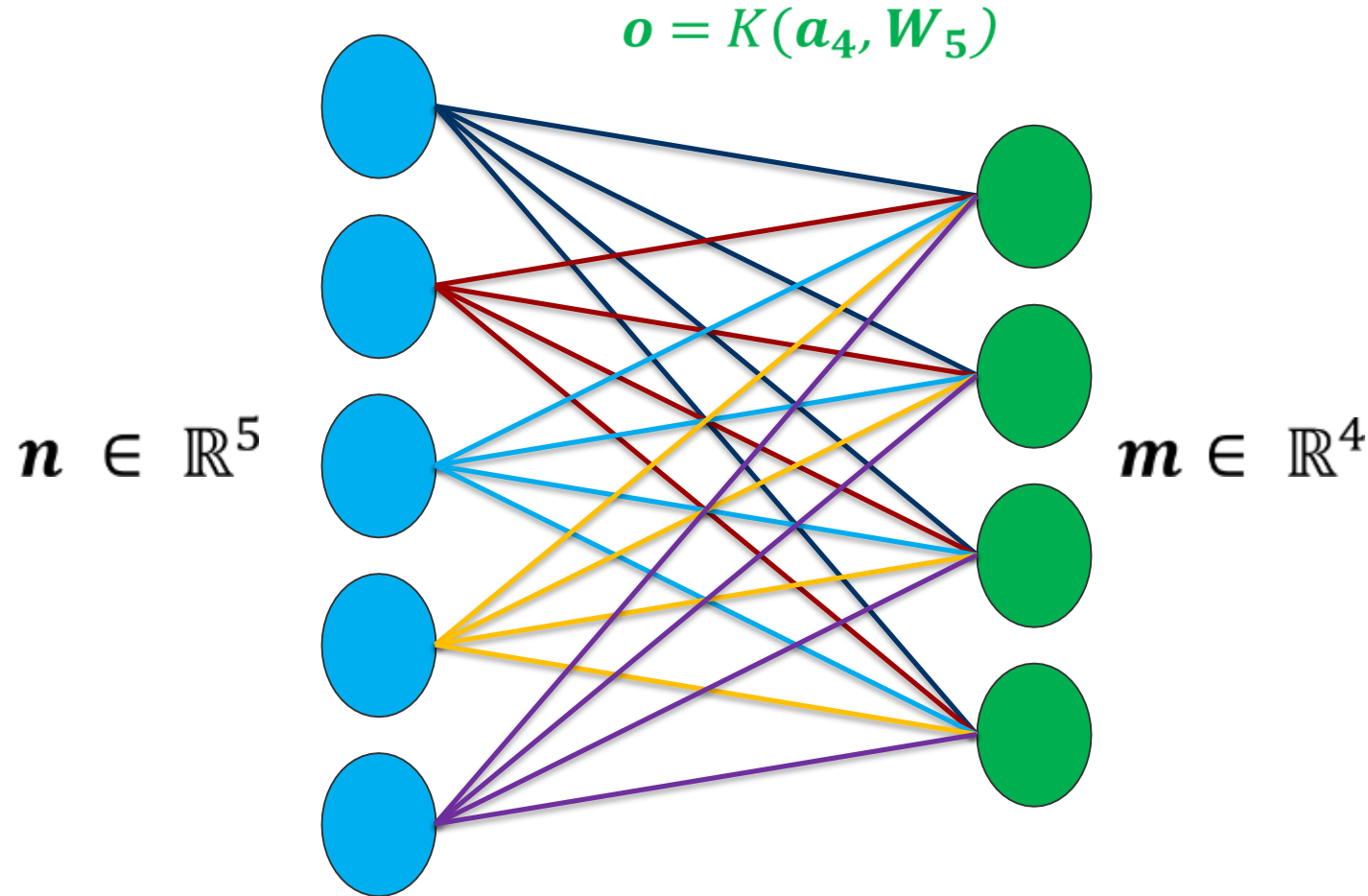
# Fully Connected



This file is meant for personal use by manju.crz@gmail.com only.

Proprietary content: ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited

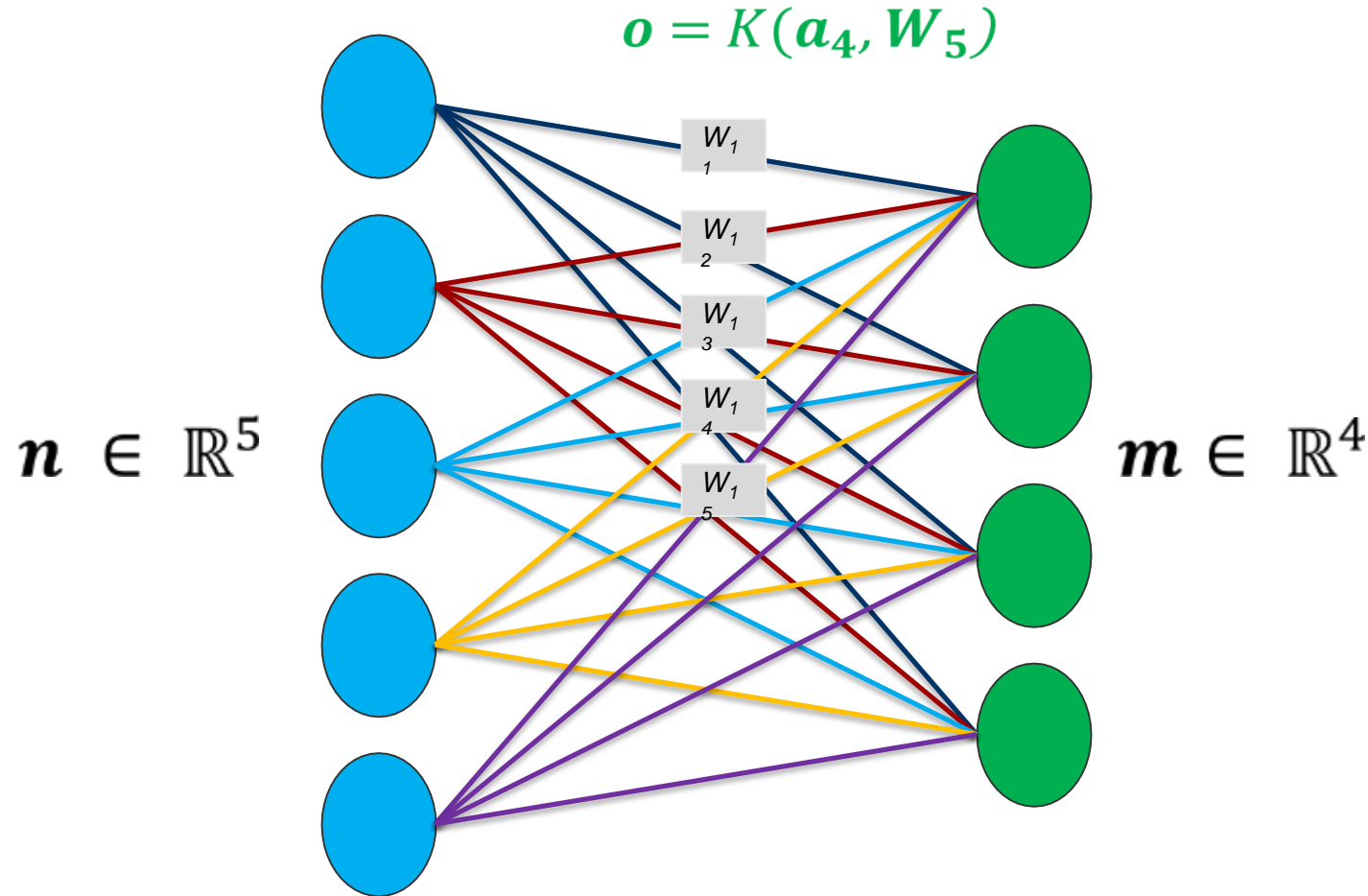
# Fully Connected



This file is meant for personal use by manju.crz@gmail.com only.

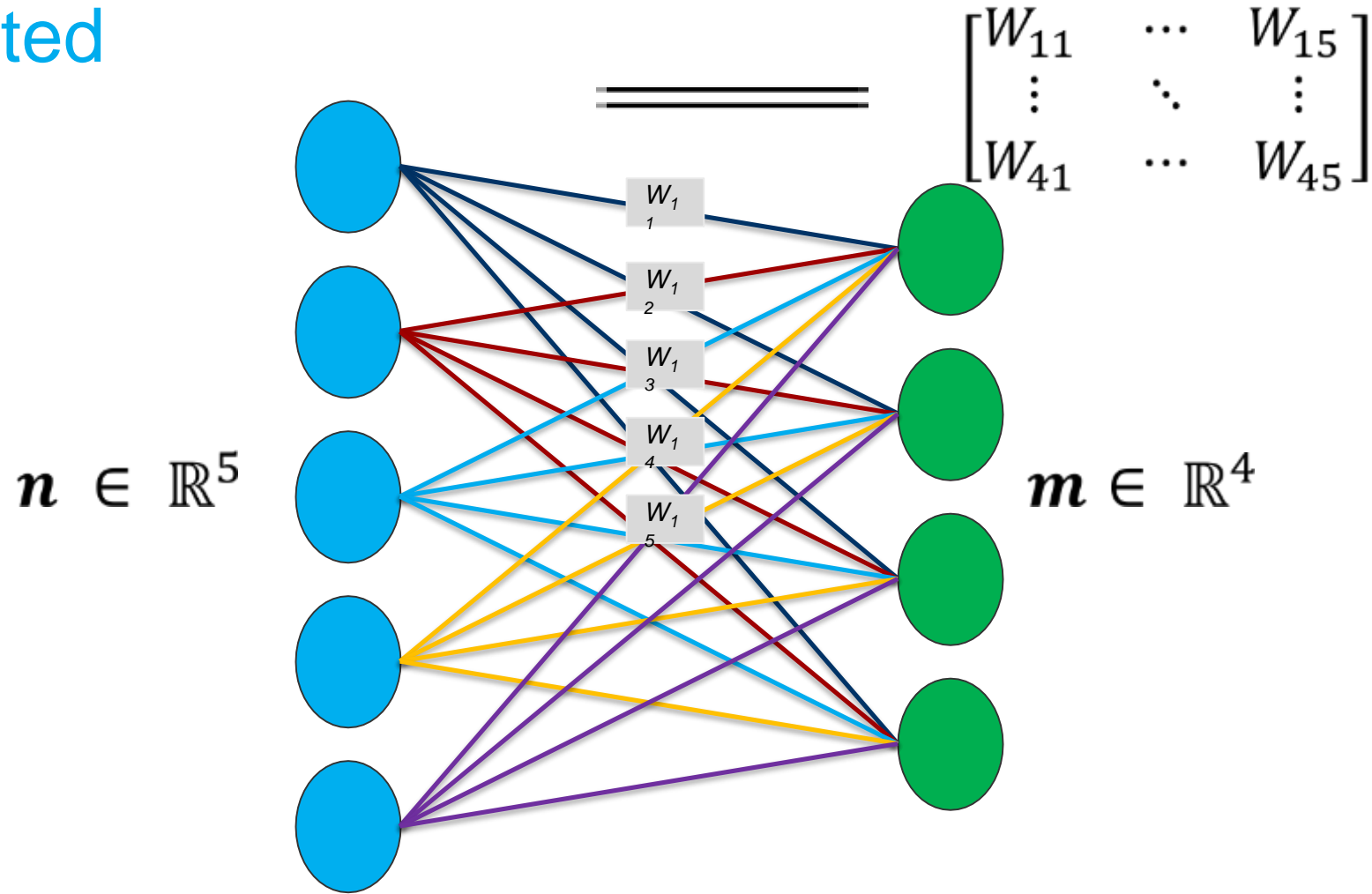
Proprietary content: ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited

# Fully Connected



This file is meant for personal use by manju.crz@gmail.com only.

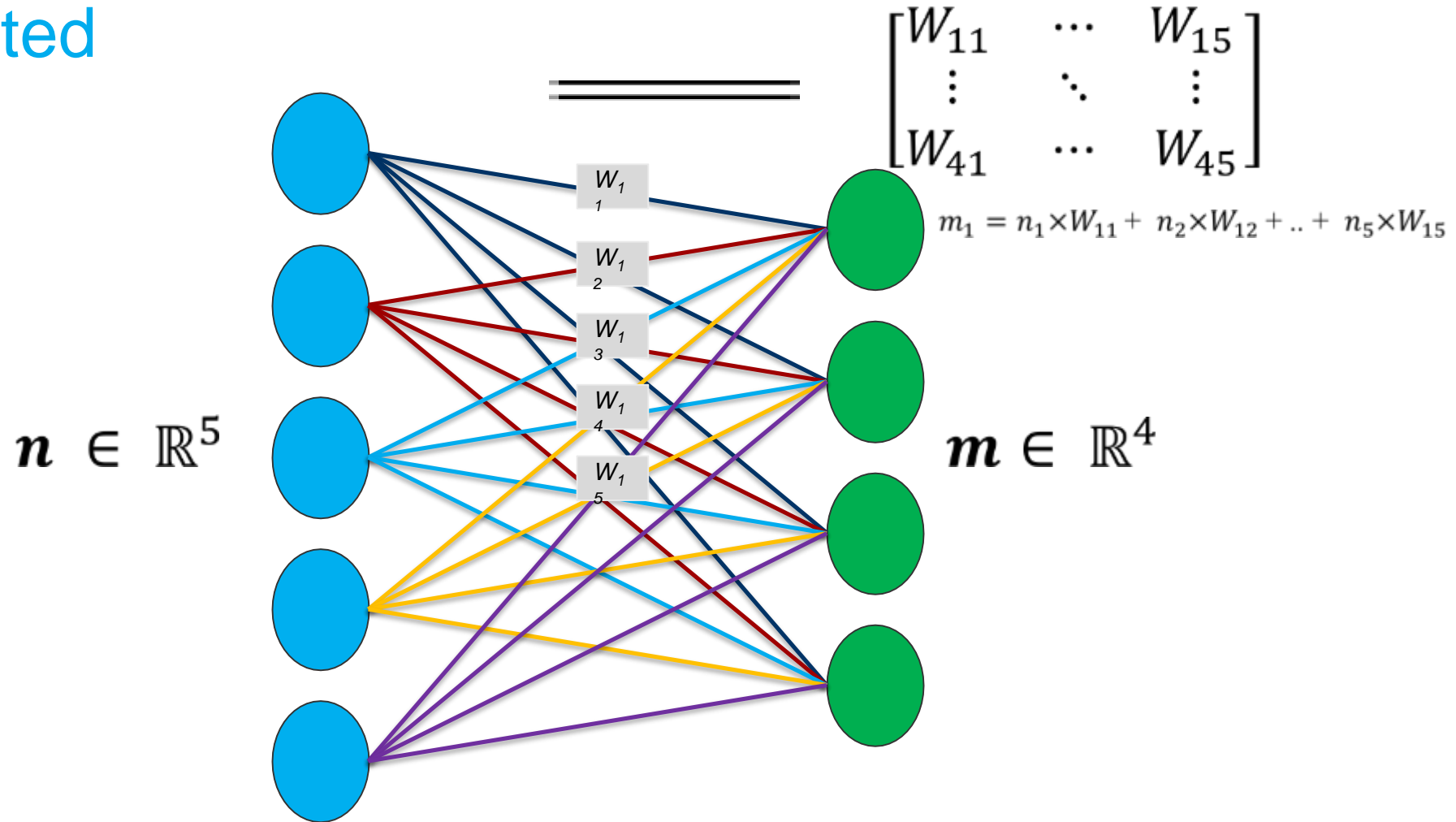
# Fully Connected



This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

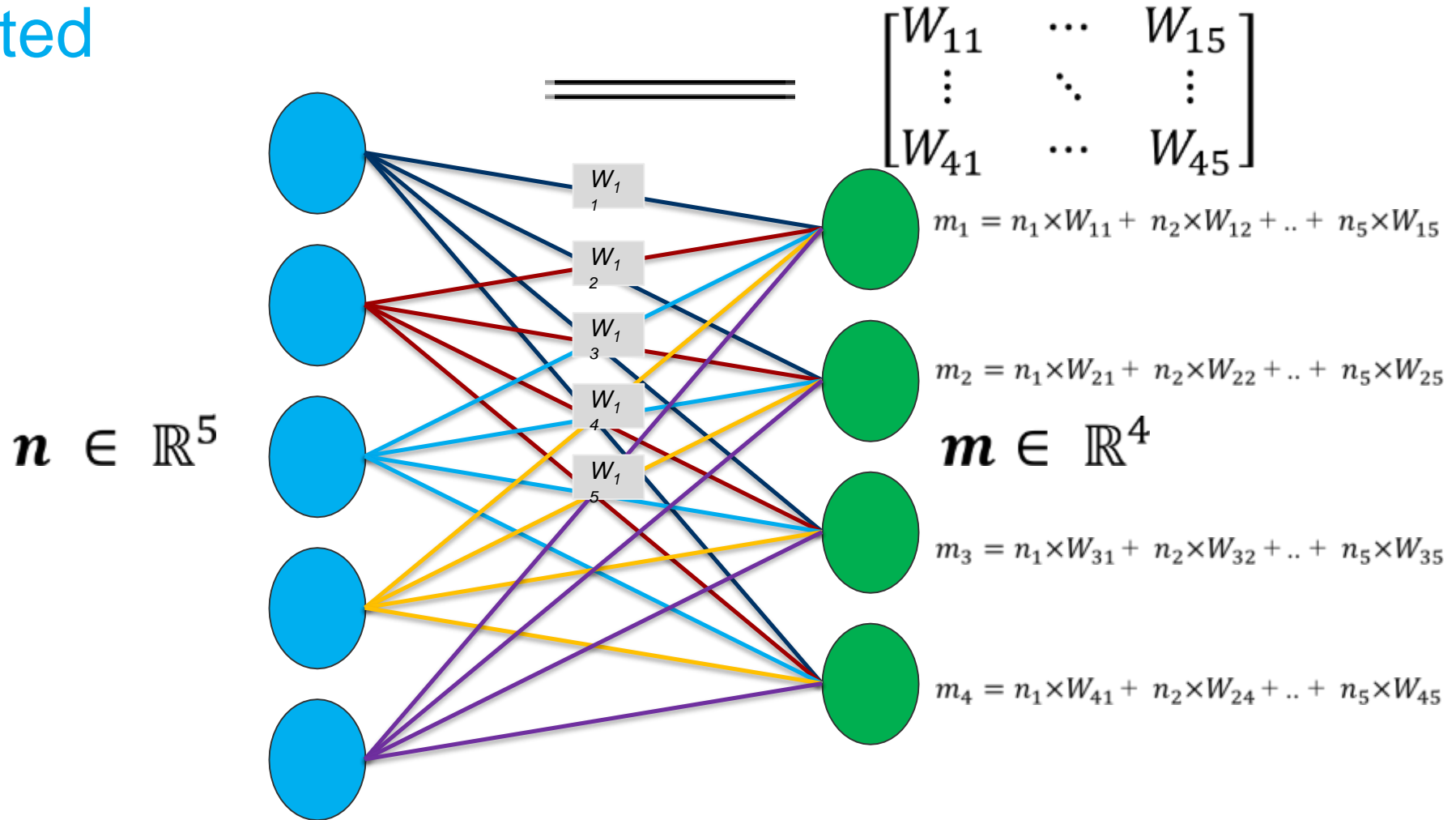
# Fully Connected



This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

# Fully Connected



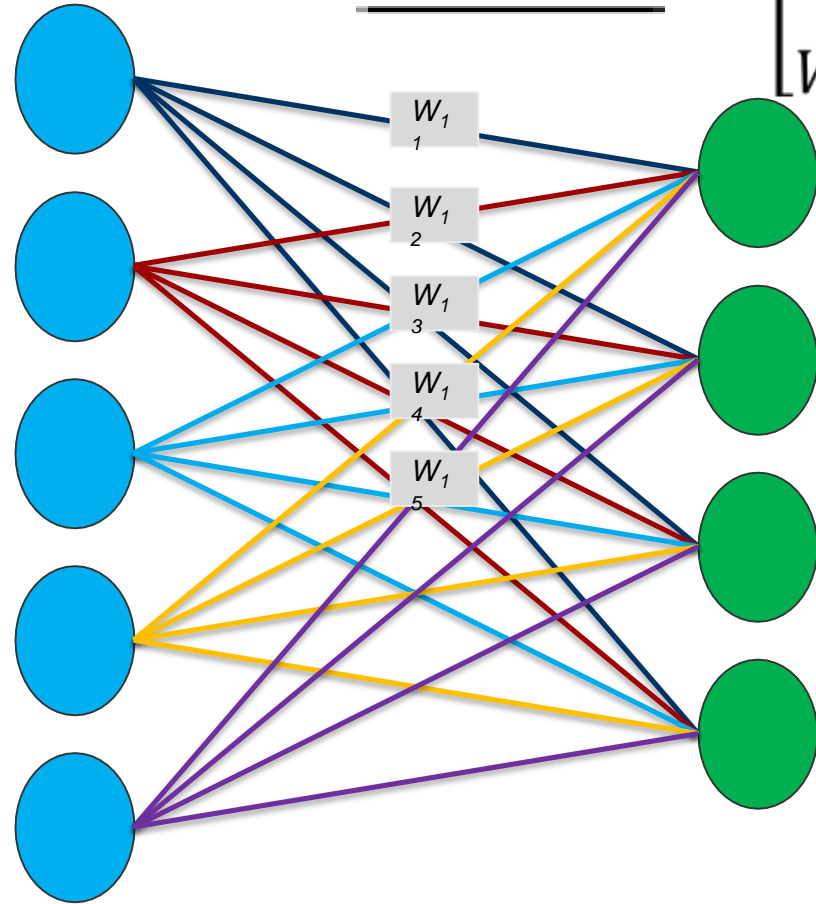
This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

# Fully Connected

Q:  
Why fully  
connected?

$$\mathbf{n} \in \mathbb{R}^5$$



$$m_1 = n_1 \times W_{11} + n_2 \times W_{12} + \dots + n_5 \times W_{15}$$

$$m_2 = n_1 \times W_{21} + n_2 \times W_{22} + \dots + n_5 \times W_{25}$$

$$\mathbf{m} \in \mathbb{R}^4$$

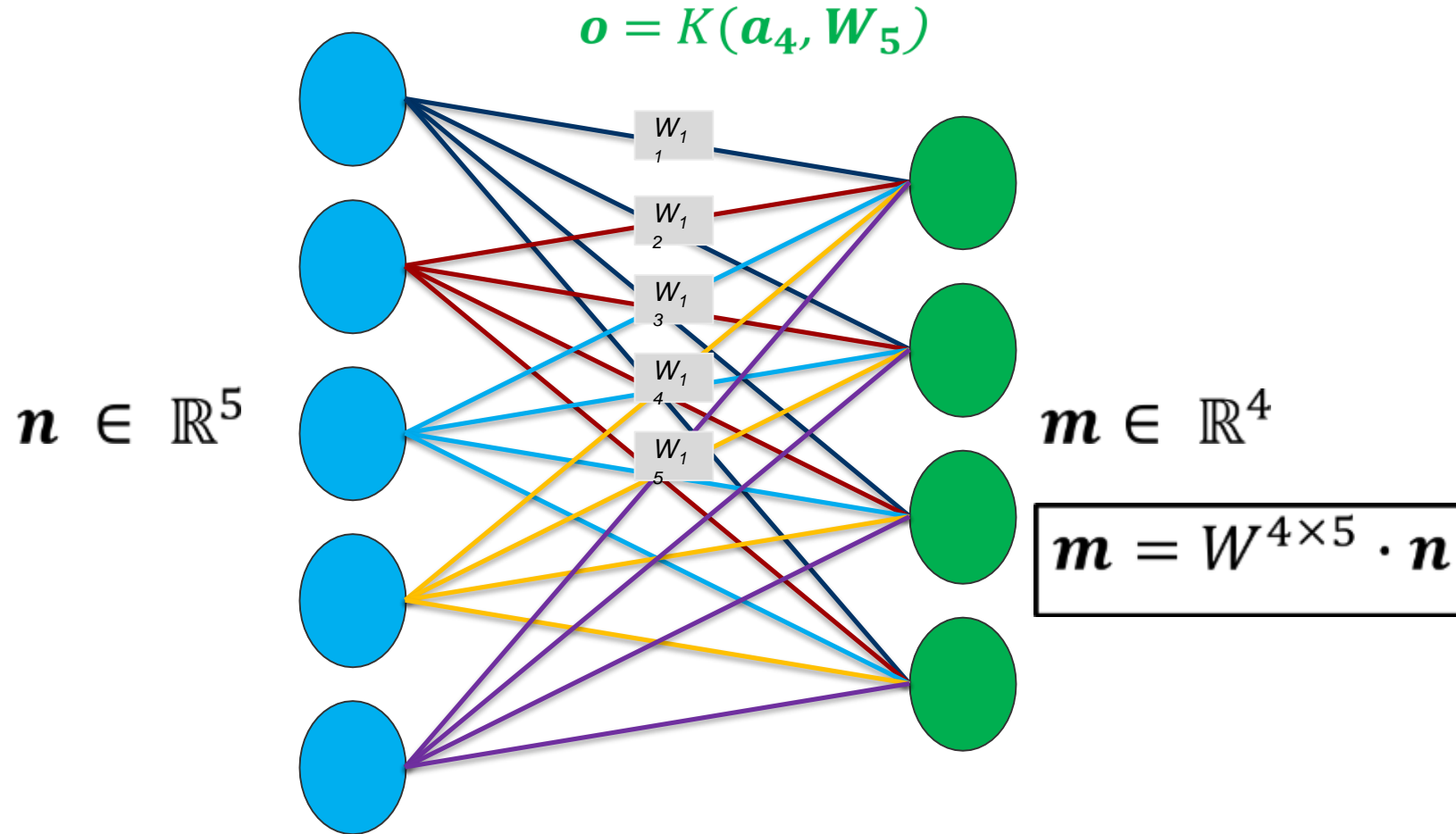
$$m_3 = n_1 \times W_{31} + n_2 \times W_{32} + \dots + n_5 \times W_{35}$$

$$m_4 = n_1 \times W_{41} + n_2 \times W_{42} + \dots + n_5 \times W_{45}$$

This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

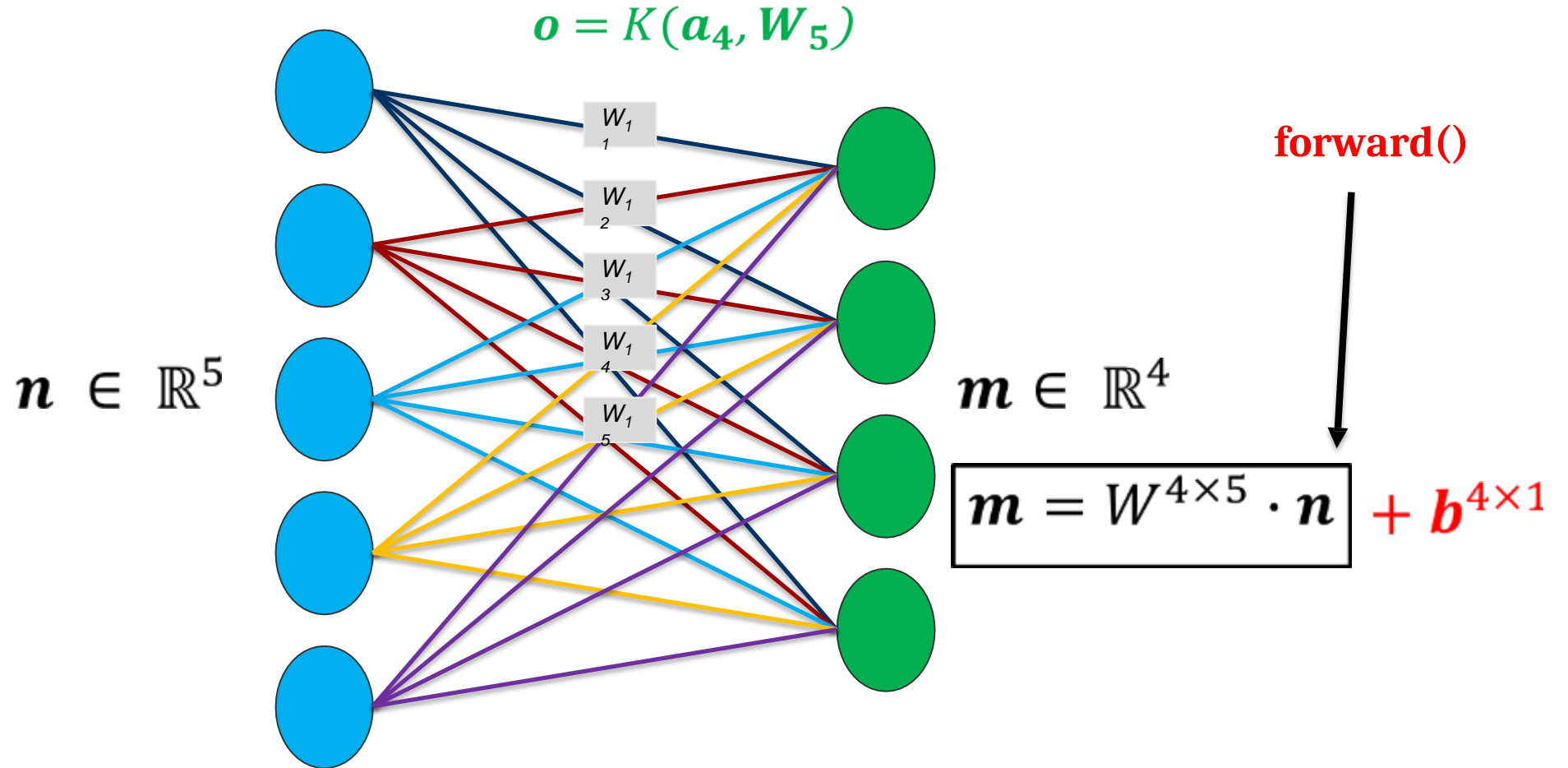
# Fully Connected



This file is meant for personal use by manju.crz@gmail.com only.



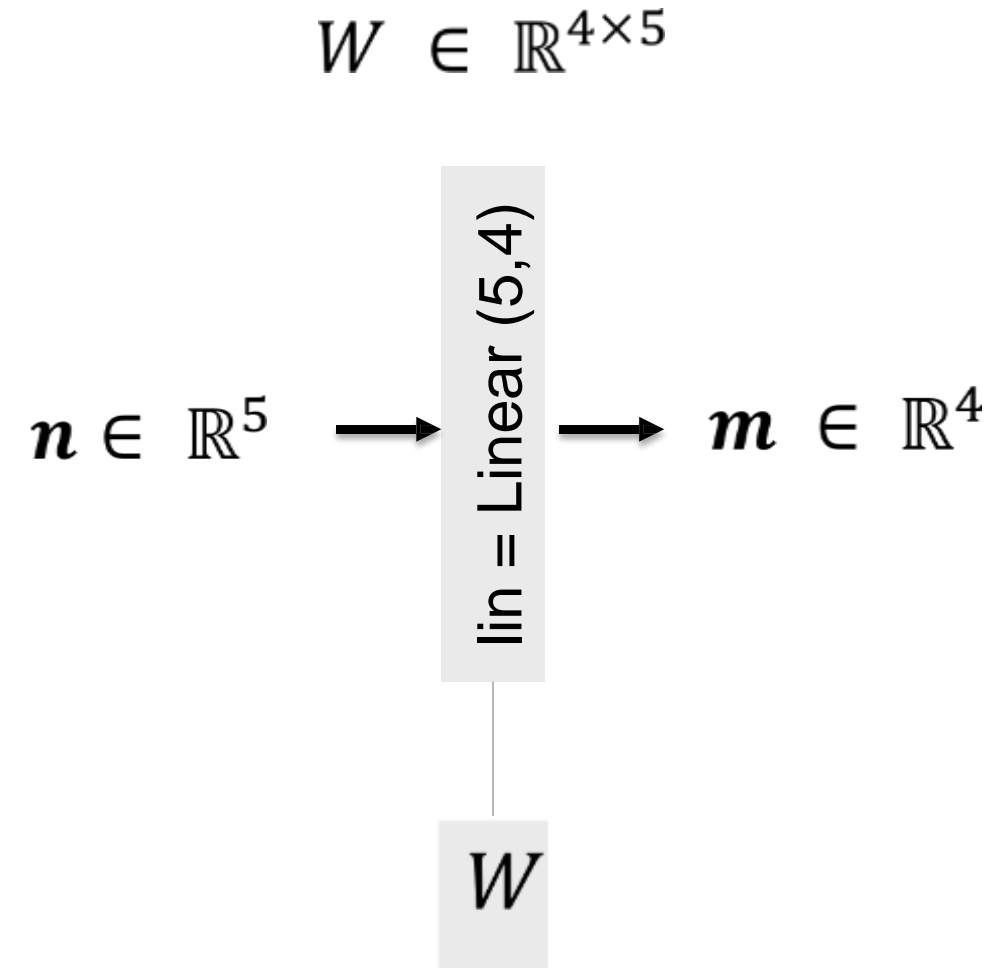
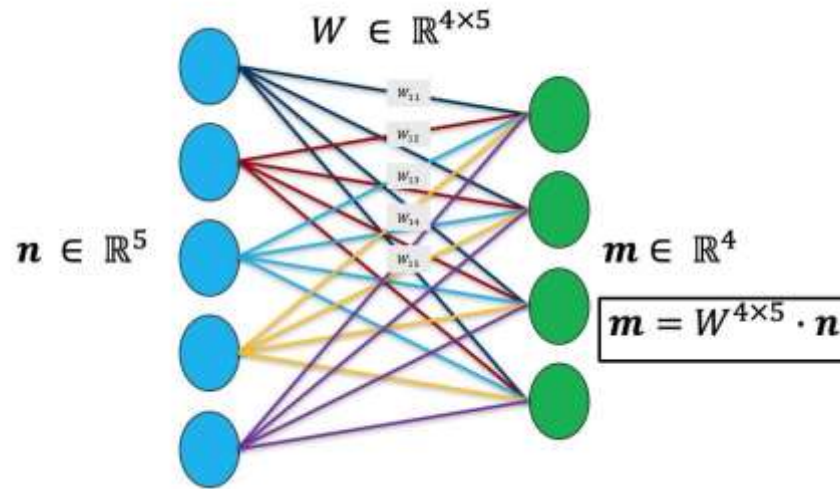
# Fully Connected



This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

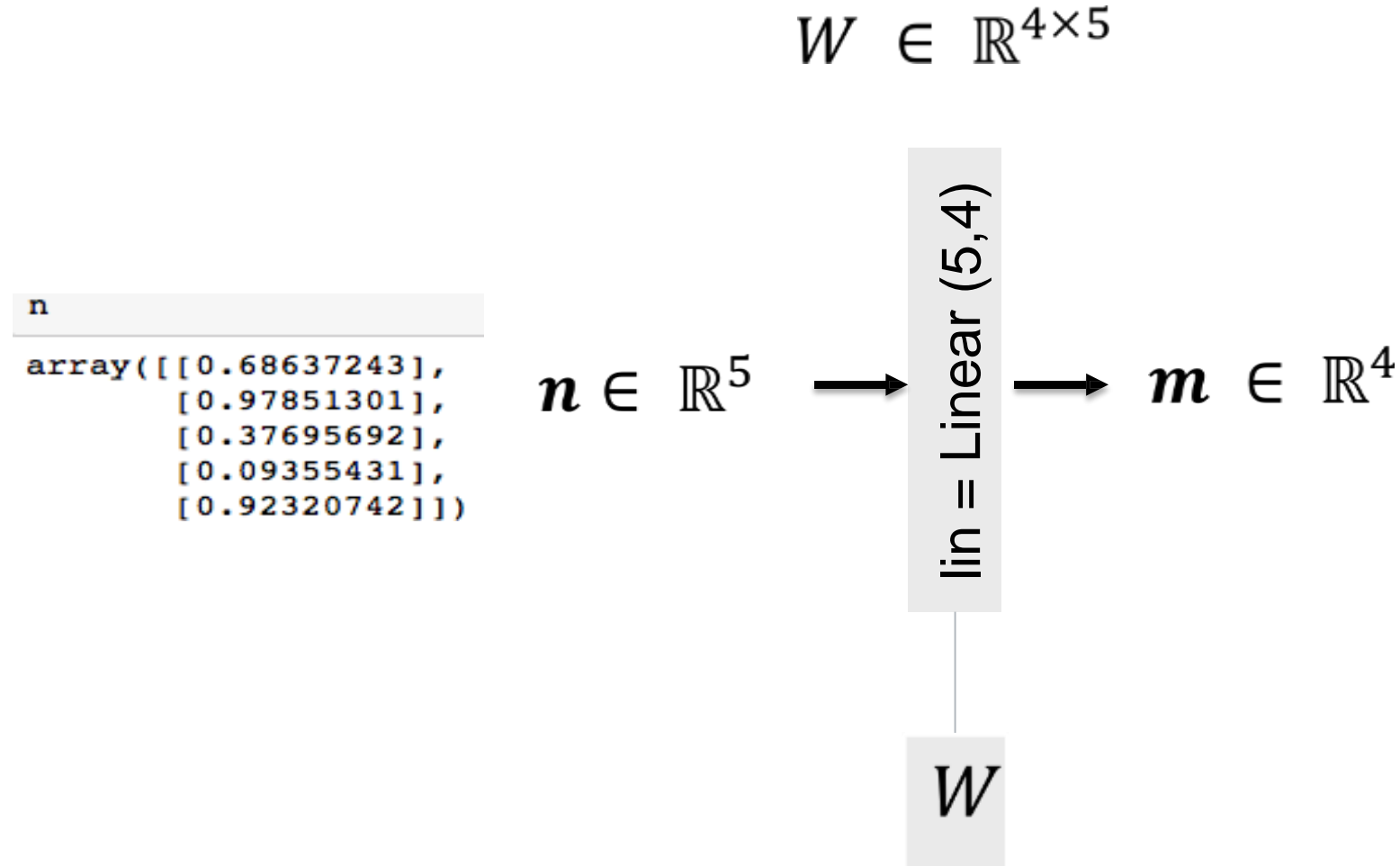
# Fully Connected



This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

# Fully Connected: Forward pass



This file is meant for personal use by manju.crz@gmail.com only.

# Fully Connected: Forward pass

```
n = np.random.rand(5,1)
n
```

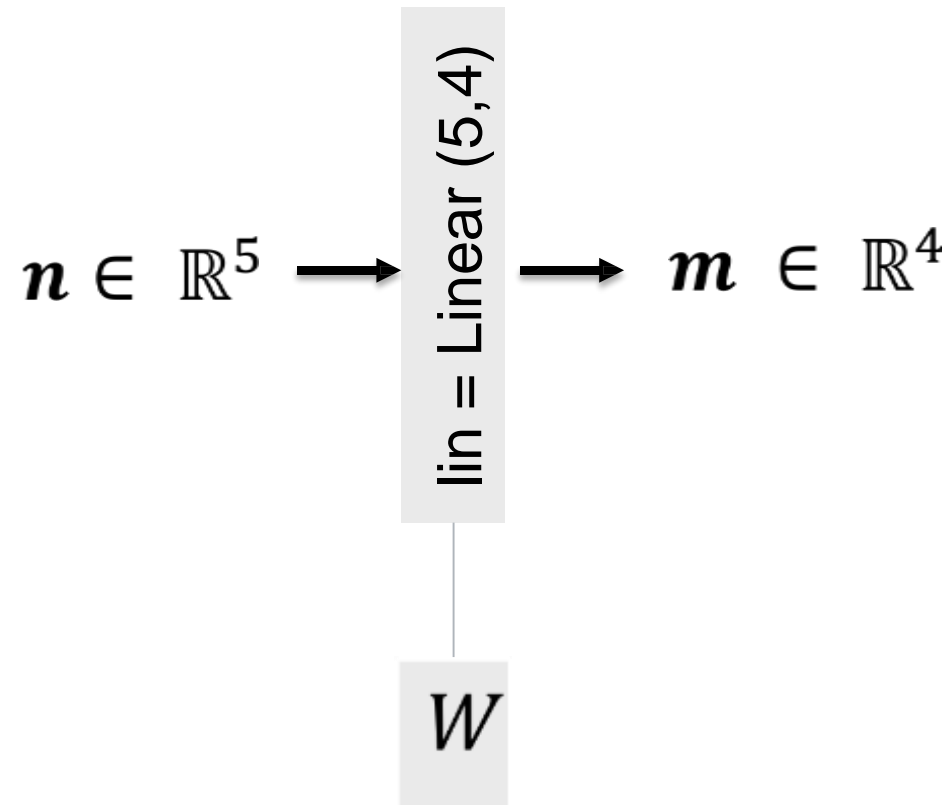
```
array([[0.68637243],
       [0.97851301],
       [0.37695692],
       [0.09355431],
       [0.92320742]])
```

```
class Linear():
    def __init__(self, in_size, out_size):
        self.W = np.random.randn(in_size, out_size) * 0.01
        self.b = np.zeros((1, out_size))
        self.params = [self.W, self.b]
        self.gradW = None
        self.gradB = None
        self.gradInput = None

    def forward(self, X):
        self.X = X
        output = np.dot(self.X, self.W) + self.b
        return output
```

```
lin = Linear(5,4)
m = lin.forward(n)
```

$$W \in \mathbb{R}^{4 \times 5}$$



This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

# Fully Connected: Forward pass

```
n = np.random.rand(5,1)
n
```

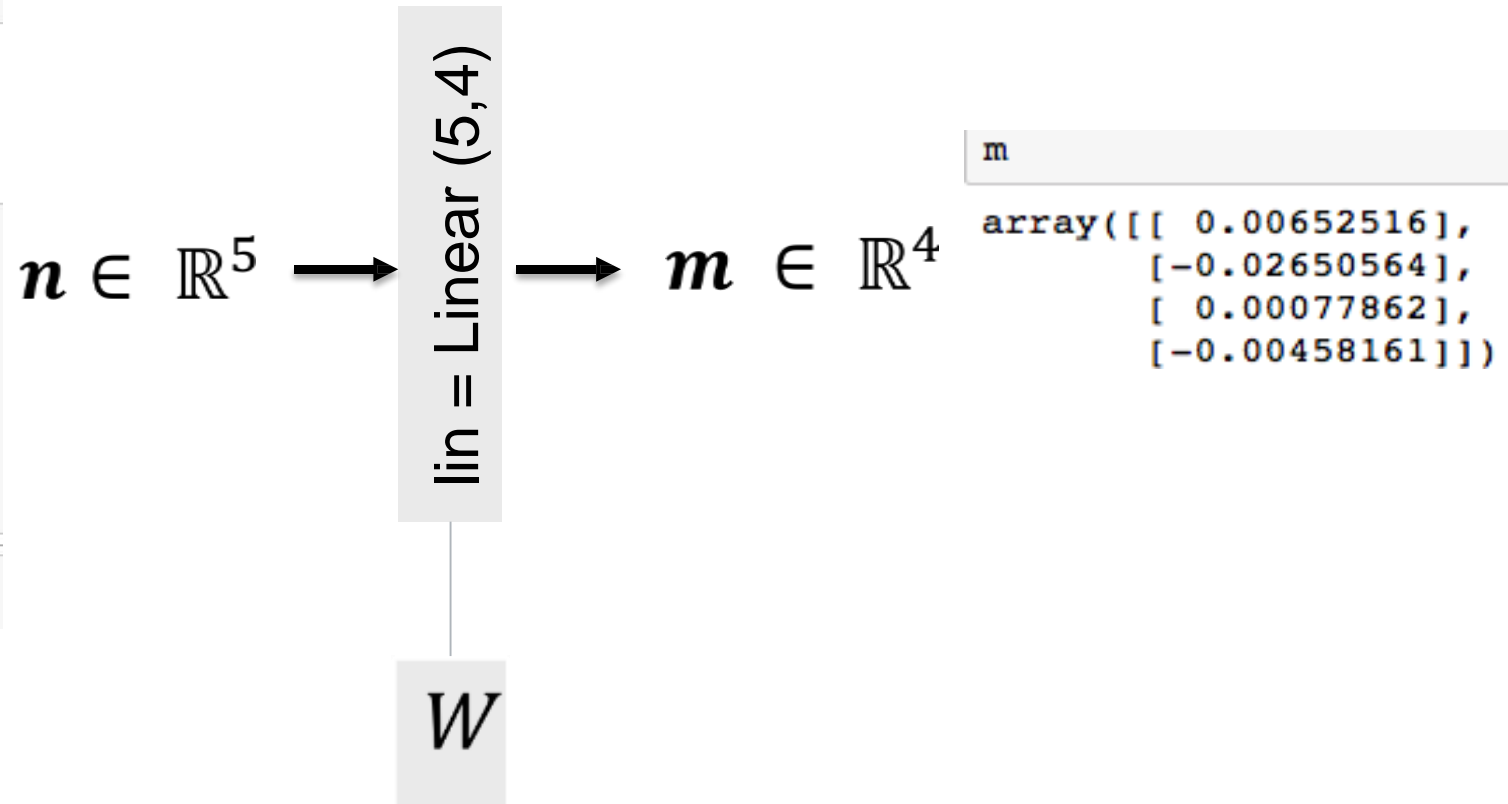
```
array([[0.68637243],
       [0.97851301],
       [0.37695692],
       [0.09355431],
       [0.92320742]])
```

```
class Linear():
    def __init__(self, in_size, out_size):
        self.W = np.random.randn(in_size, out_size) * 0.01
        self.b = np.zeros((1, out_size))
        self.params = [self.W, self.b]
        self.gradW = None
        self.gradB = None
        self.gradInput = None

    def forward(self, X):
        self.X = X
        output = np.dot(self.X, self.W) + self.b
        return output
```

```
lin = Linear(5,4)
m = lin.forward(n)
```

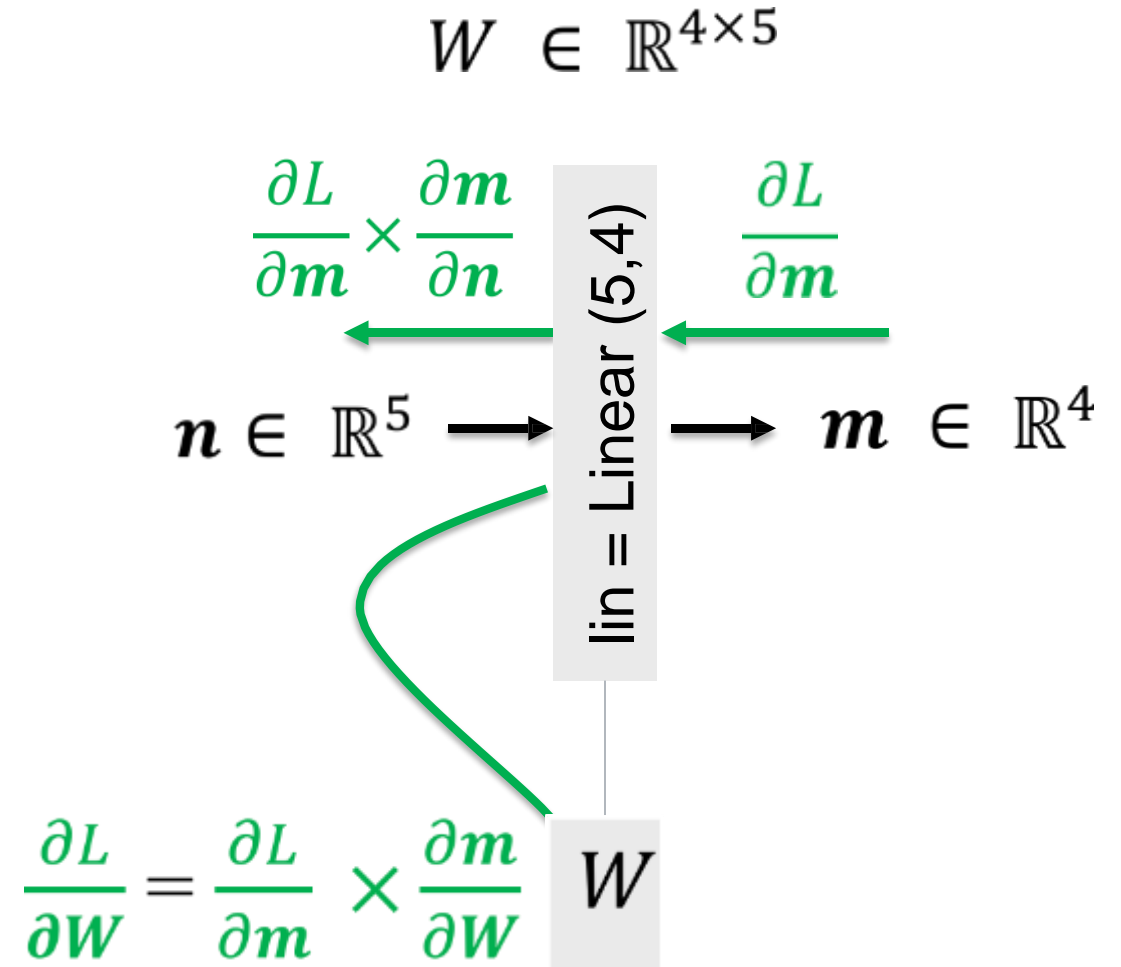
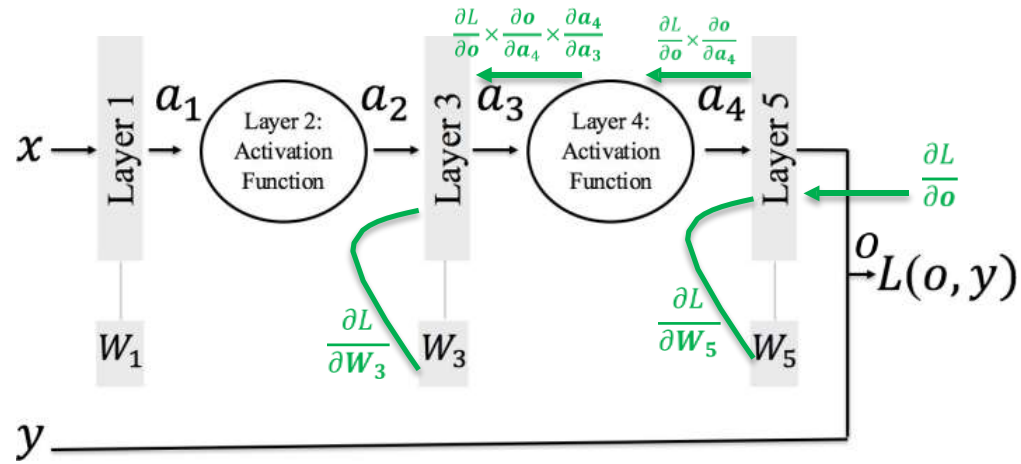
$$W \in \mathbb{R}^{4 \times 5}$$



This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

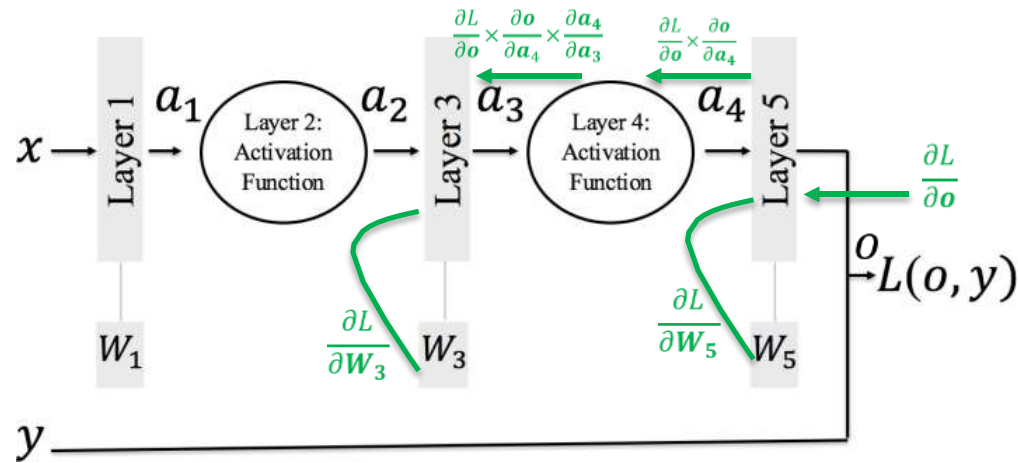
# Fully Connected: Backward pass



This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

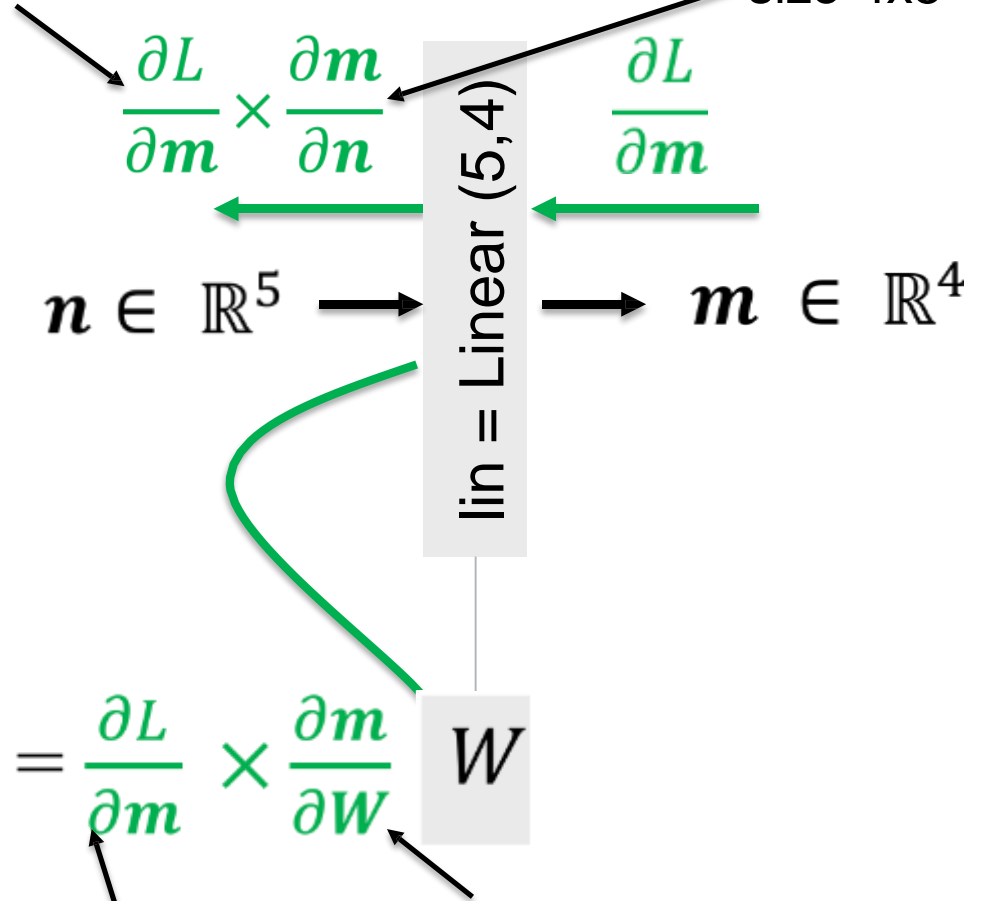
# Fully Connected: Backward pass



Gradient (row vector 1x4)

$$W \in \mathbb{R}^{4 \times 5}$$

Jacobian of size 4x5



Gradient

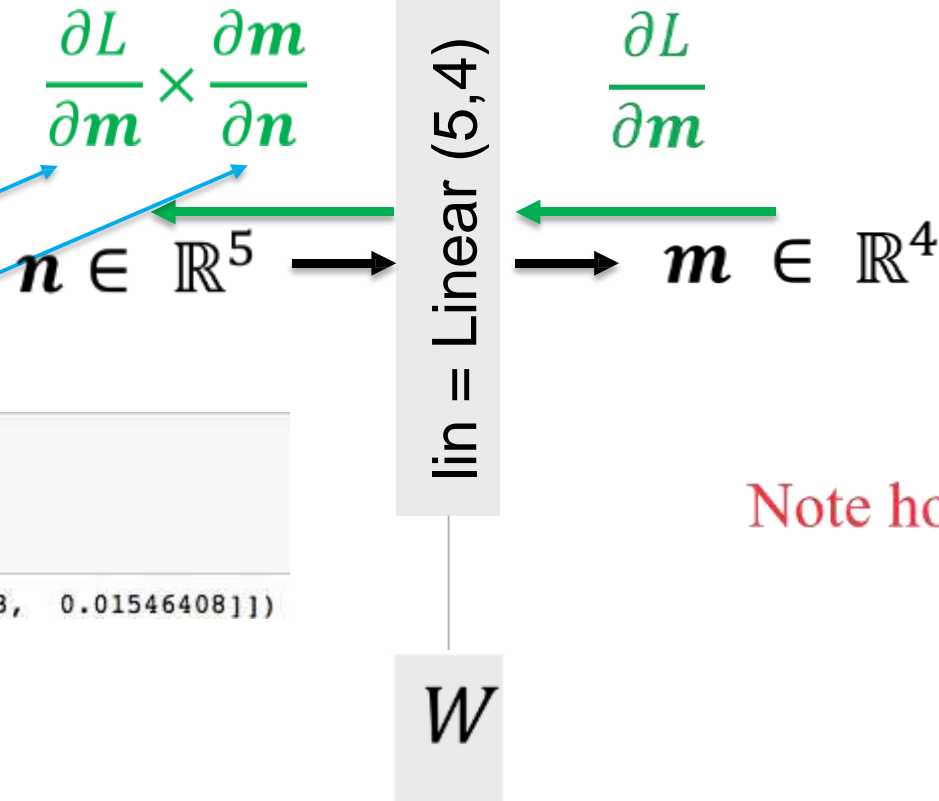
Jacobian of size 4x20

This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

# Fully Connected: Backward pass

$$W \in \mathbb{R}^{4 \times 5}$$



Note how  $\frac{\partial m}{\partial n} = W$

```
# upstream gradient * dm/dn
# upstream gradient = dL/dm (which is nextgrad)
# dm/dn = d(W*n)/dn = W
np.dot(nextgrad, W)

array([[ -0.01586083, -0.02361688, -0.00616945, -0.01881653,  0.01546408]])
```



# Fully Connected: Backward pass

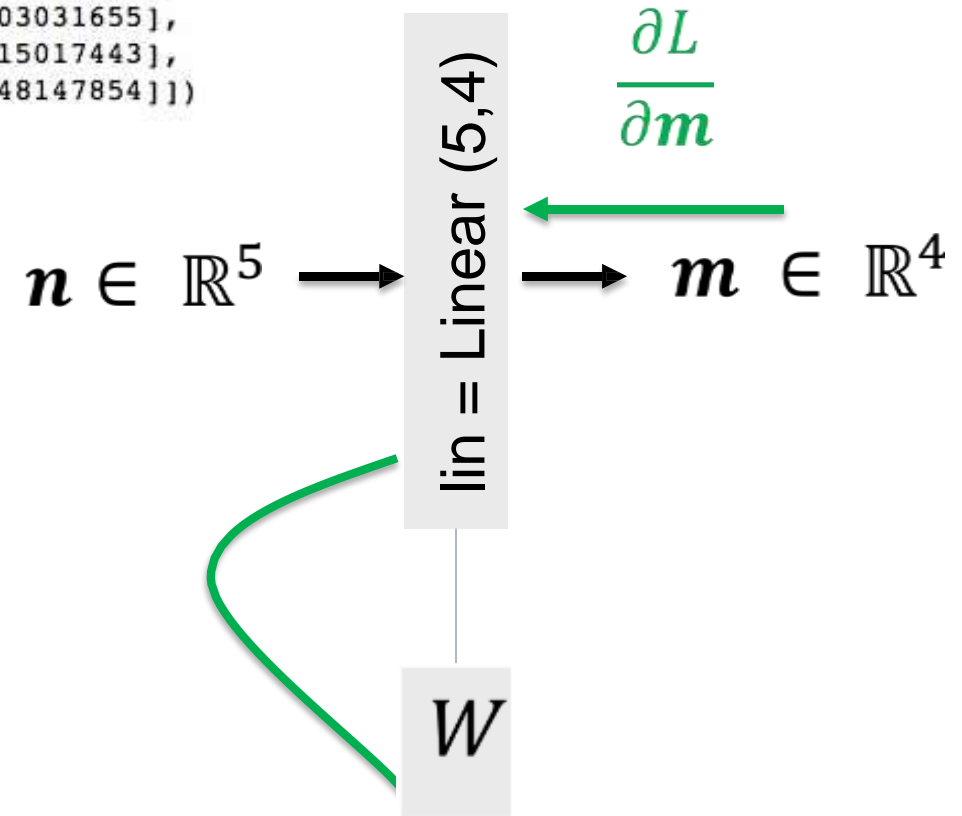
```
# dL/dW = dL/dm * dm/dW
np.dot(nextgrad, dodw).reshape(4,5)

array([[ -0.93909512, -1.33880201, -0.51575265, -0.12800105, -1.26313287],
       [  0.02253929,  0.03213269,  0.01237862,  0.00307216,  0.03031655],
       [-0.11164944, -0.15917077, -0.06131806, -0.0152181 , -0.15017443],
       [-0.35796246, -0.51032195, -0.19659359, -0.04879119, -0.48147854]])
```

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial m} \times \frac{\partial m}{\partial W}$$

```
# Jacobian of dm/dw (4x20)
dodw = np.zeros((4,20))
st = 0
for i in range(4):
    for j in range(5):
        dodw[i][st] = n[j]
        st = st + 1
```

$$W \in \mathbb{R}^{4 \times 5}$$



This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

# Activation Functions

This file is meant for personal use by manju.crz@gmail.com only.

Proprietary content: ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited

# Neural Network Constructed

Now, let's review the following in turn:

- Feed forward
- Back propagation
- Fully connected layer
- **Activation functions**
- Softmax function
- Cross-entropy loss

... and we'll have a fully functioning network

This file is meant for personal use by manju.crz@gmail.com only.

Proprietary content: ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited

# Activation Functions

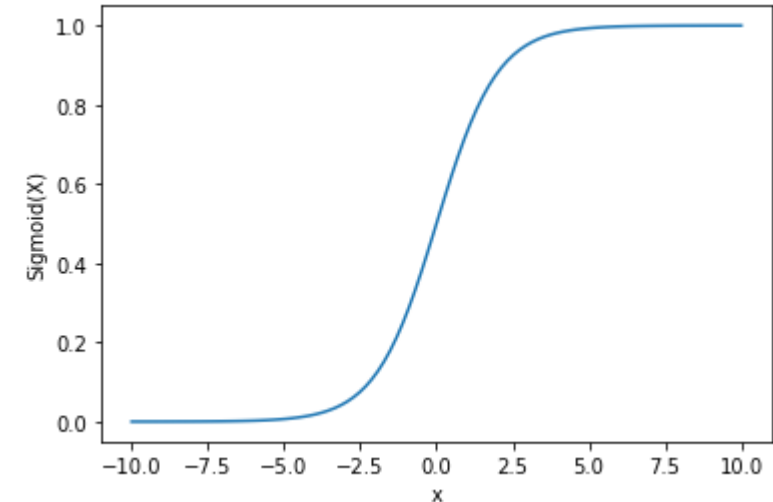
- An activation function takes a single input value and applies a function to it - typically a 'non-linearity'
- Why non-linearity: converts a linear function (sum of weights) into a polynomial of higher degree, this is what allows non-linear decision boundaries
- Activation functions decide whether a neuron is 'switched on', i.e. it acts as a gate, by applying a non-linear function on the input
- Many different types of activation functions exist

# Activation Functions: Sigmoid

- Activation function are in the form  $f(x) = 1 / 1 + \exp(-x)$
- Ranges from 0-1
- S-shaped curve
- Historically popular
  - Interpretation as a saturating “firing rate” of a neuron

## Limitations: :

1. Its output is not zero centered.
  - Hence, make the gradient go too far in different directions
2. Vanishing Gradient Problem
3. Slow convergence



## Sigmoid

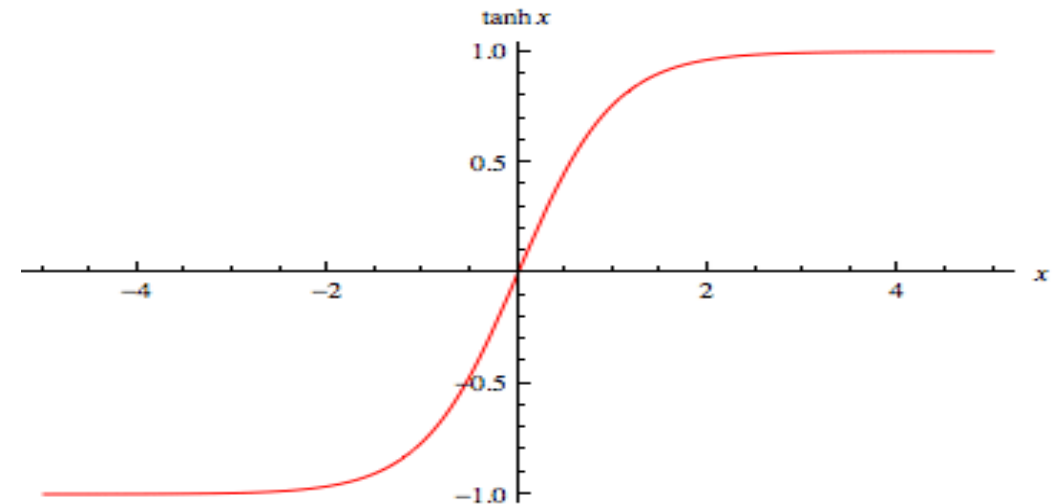
$$\sigma(x) = \frac{1}{(1 + e^{-x})}$$

# Activation Functions: **tanh(x)**

- Ranges between -1 to +1
- Output is zero centered
- Generally preferred over Sigmoid function

## Drawback:

Though optimisation is easier, it still suffers from the Vanishing Gradient Problem



**tanh**

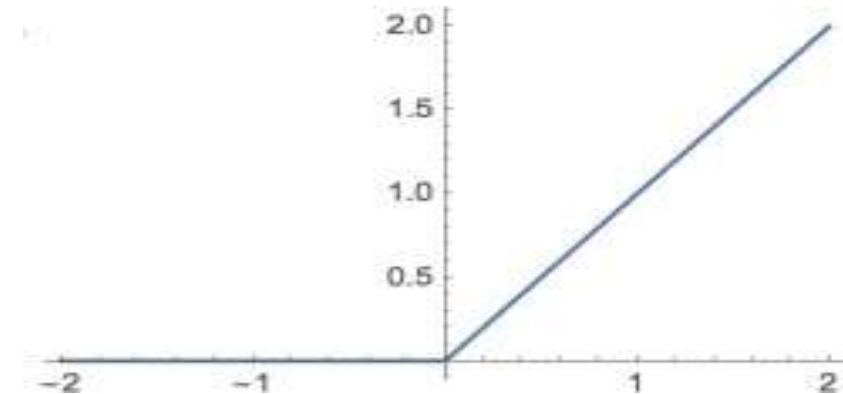
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

# Activation Functions: ReLU

- Simple
- Much better convergence than tanh and sigmoid function.
- Very efficient in computation

## Drawbacks:

- Output is not necessarily zero centered.
- Should only be used within hidden layers of a NN model



**ReLU**

$$f(x) = \max(0, x)$$

This file is meant for personal use by manju.crz@gmail.com only.

Image sourced from: Wolfram Alpha Documentation

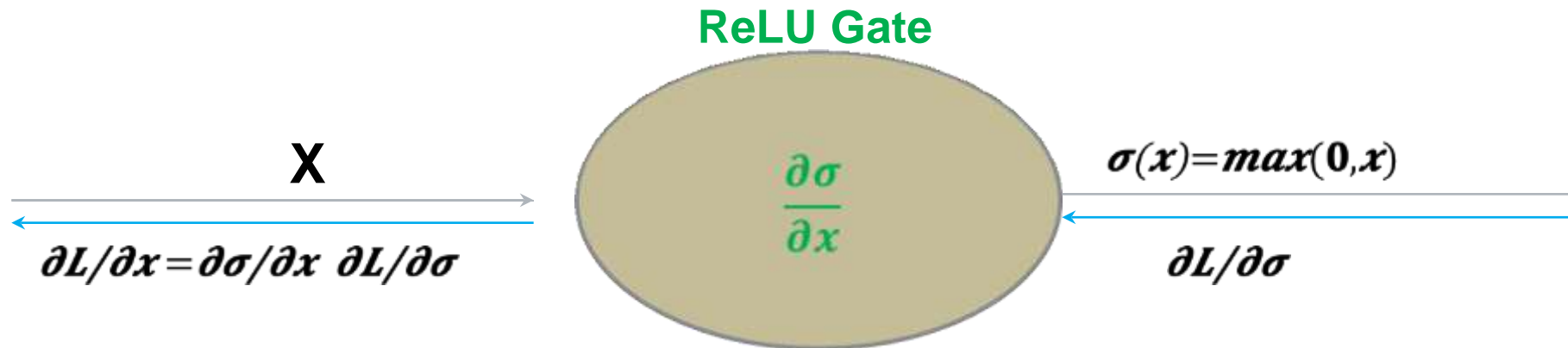
Proprietary content: ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited

# Activation Functions: Problems with ReLU

- Some gradients can be fragile during training and can 'die'
- Results in weight update, and possibly never activating again
- 'Dead neurons'

## Experiment:

- What happens when  $x = -10$ ?
- What happens when  $x = 0$ ?
- What happens when  $x = 10$ ?

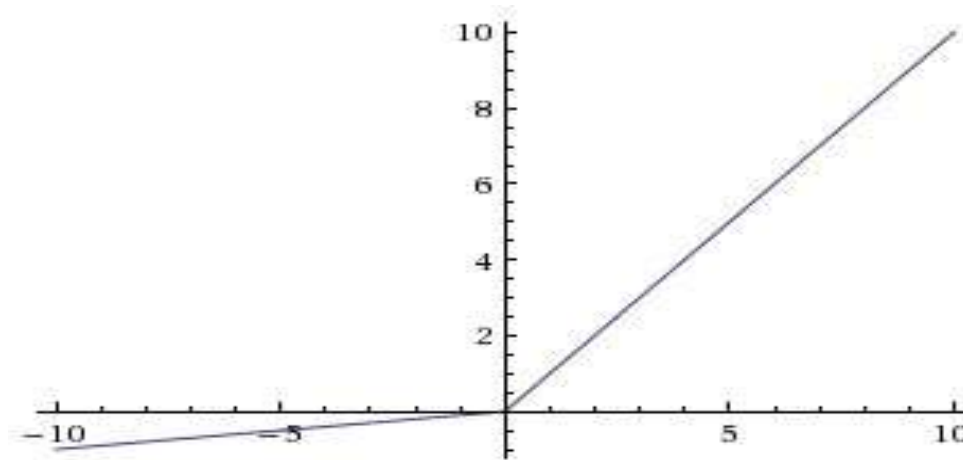


This file is meant for personal use by manju.crz@gmail.com only.



# Activation Functions: Leaky ReLU

- Introduced to overcome the problem of dying neurons.
- Introduces a small slope to keep the neurons alive
- Does not saturate in the positive region



## Leaky ReLU

$$f(x) = \max(0.01x, x)$$

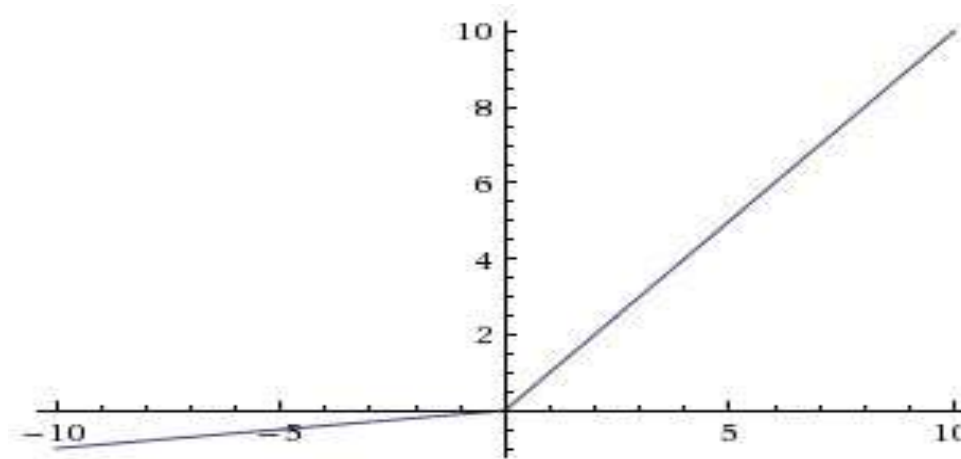
This file is meant for personal use by manju.crz@gmail.com only.

Image sourced from: Wolfram Alpha Documentation

Proprietary content: ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited

# Activation Functions: Leaky ReLU

- Introduced to overcome the problem of dying neurons.
- Introduces a small slope to keep the neurons alive
- Does not saturate in the positive region



## Leaky ReLU

$$f(x) = \max(0.01x, x)$$

Back Propagate into  $\alpha$   
(learnable parameter)

$$f(x) = \max(\alpha x, x)$$

## Parametric Rectifier PReLU

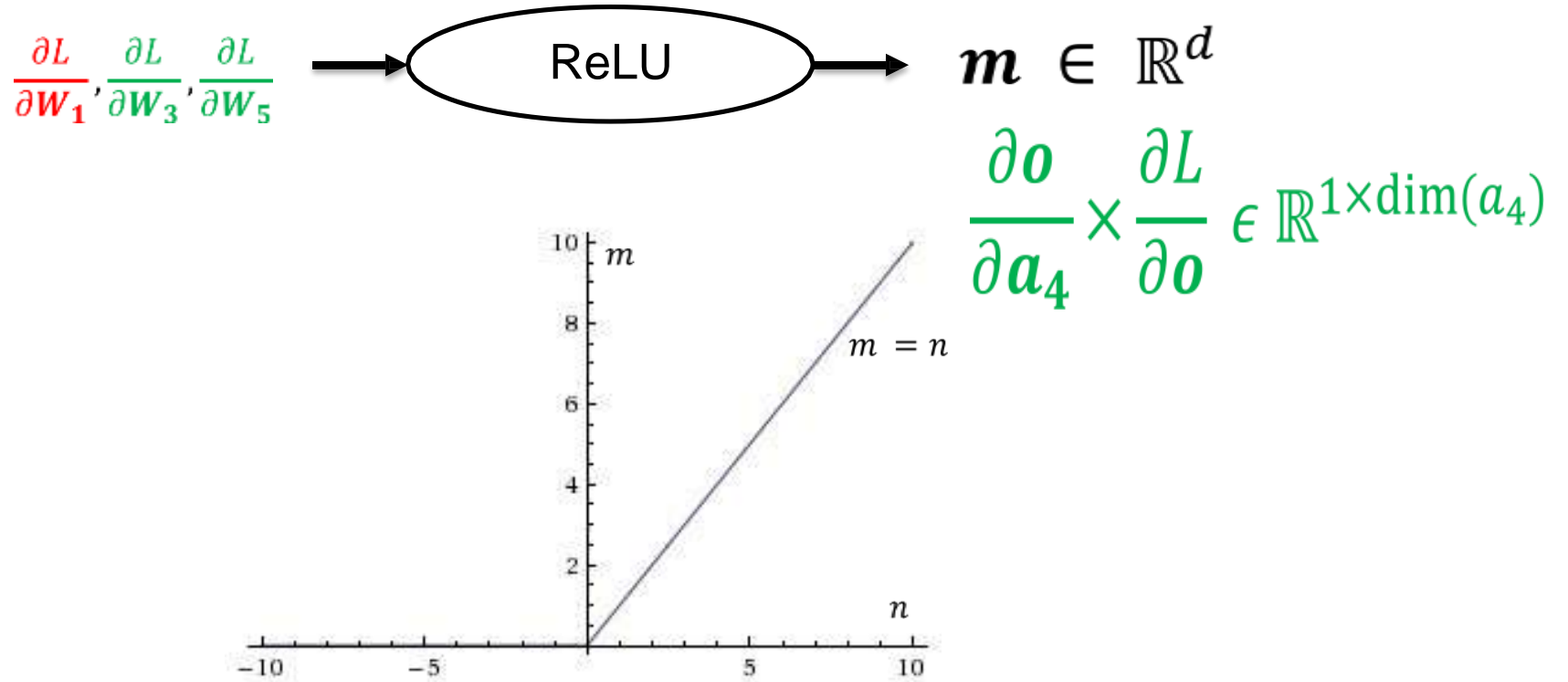
# Activation Functions: In practice...

- **ReLU** is preferred
  - *Note:* Be careful with learning rates + Monitor the fraction of “**dead**” units in network
- Possibly try Leaky ReLU or Maxout
- Never use Sigmoid
- Try tanh
  - Typically perform worse than ReLU though

This file is meant for personal use by manju.crz@gmail.com only.

Proprietary content: ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited

# Activation Functions: ReLU

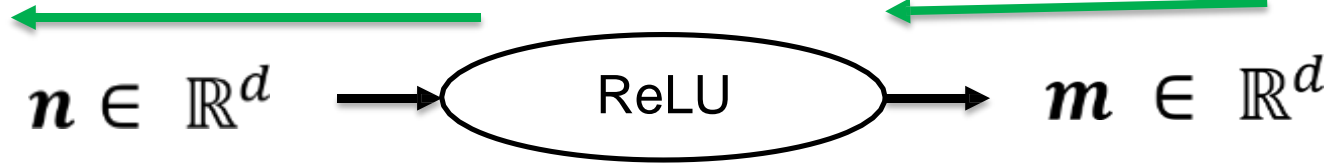


This file is meant for personal use by manju.crz@gmail.com only.

# Activation Functions: ReLU

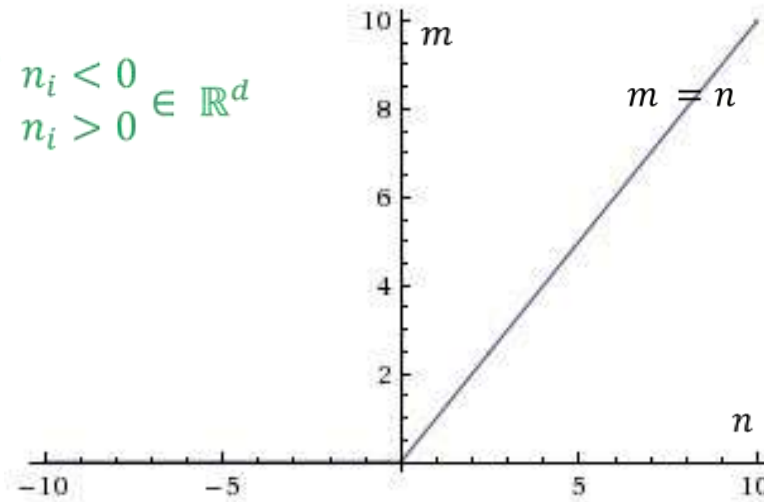
$$\frac{\partial L}{\partial \mathbf{m}} \cdot \frac{\partial \mathbf{m}}{\partial \mathbf{n}} \in \mathbb{R}^{1 \times \dim(\mathbf{n})}$$

$$\frac{\partial L}{\partial \mathbf{m}} \in \mathbb{R}^{1 \times \dim(\mathbf{m})}$$



$$\frac{\partial m_i}{\partial n_i} = \frac{\partial \max(0, n_i)}{\partial n_i} = \begin{cases} 0 & \text{if } n_i < 0 \\ 1 & \text{if } n_i > 0 \end{cases} \in \mathbb{R}^d$$

$$\frac{\partial o}{\partial \mathbf{a}_4} \times \frac{\partial L}{\partial o} \in \mathbb{R}^{1 \times \dim(\mathbf{a}_4)}$$



This file is meant for personal use by manju.crz@gmail.com only.

# Activation Functions: ReLU (forward)

```
class ReLU:
    def forward(self, X):
        self.output = np.maximum(X, 0)
    return self.output
```



n

```
array([[ 0.58439657],
       [ 0.6157001 ],
       [-1.24037245],
       [-0.93783622],
       [-1.41779859]])
```

```
relu = ReLU()
m = relu.forward(n)
```

m

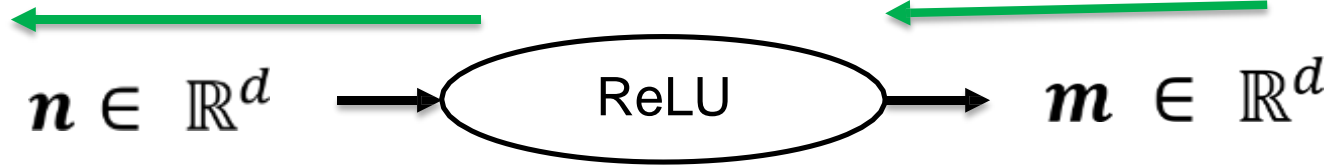
```
array([[0.58439657],
       [0.6157001 ],
       [0.          ],
       [0.          ],
       [0.          ]])
```

This file is meant for personal use by manju.crz@gmail.com only.

# Activation Functions: ReLU (backward)

$$\frac{\partial L}{\partial \mathbf{m}} \cdot \frac{\partial \mathbf{m}}{\partial \mathbf{n}} \in \mathbb{R}^{1 \times \dim(\mathbf{n})}$$

$$\frac{\partial L}{\partial \mathbf{m}} \in \mathbb{R}^{1 \times \dim(\mathbf{m})}$$



```
class ReLU:
    def forward(self, X):
        self.output = np.maximum(X, 0)
        return self.output

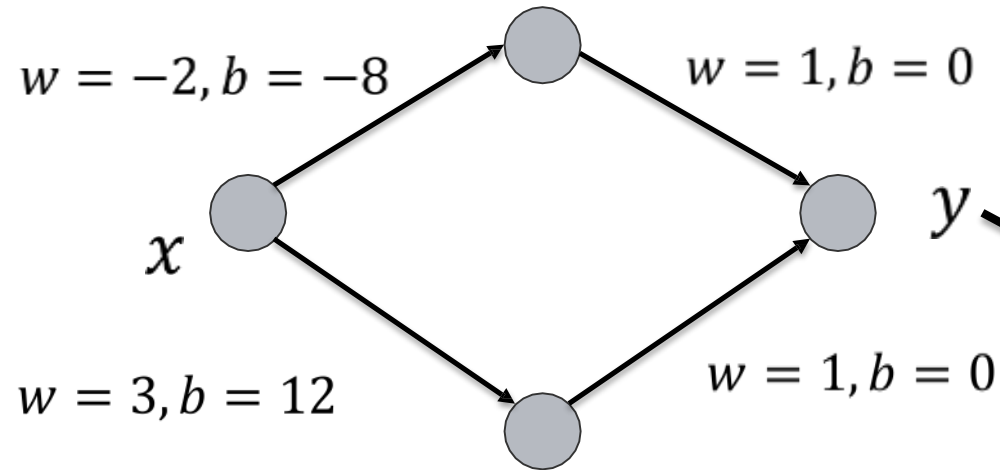
    def backward(self, nextgrad):
        self.gradInput = nextgrad.copy()
        self.gradInput[self.output <= 0] = 0
        return self.gradInput, []
```

```
relu.backward(m)
```

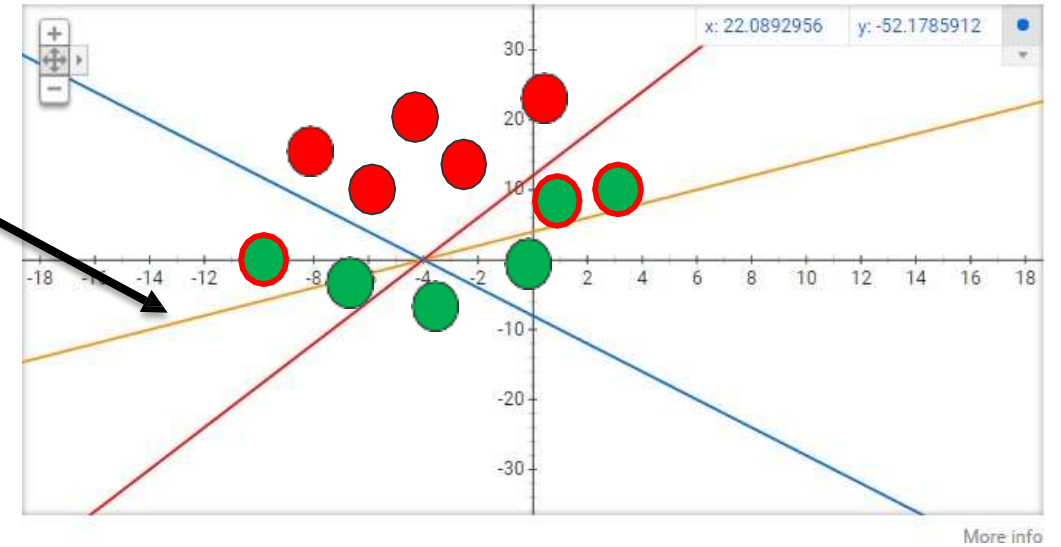
```
array([[1.],
       [1.],
       [0.],
       [0.],
       [0.]])
```

This file is meant for personal use by manju.crz@gmail.com only.

# ReLU



Graph for  $-(2*x)-8$ ,  $3*x+12$ ,  $x+4$

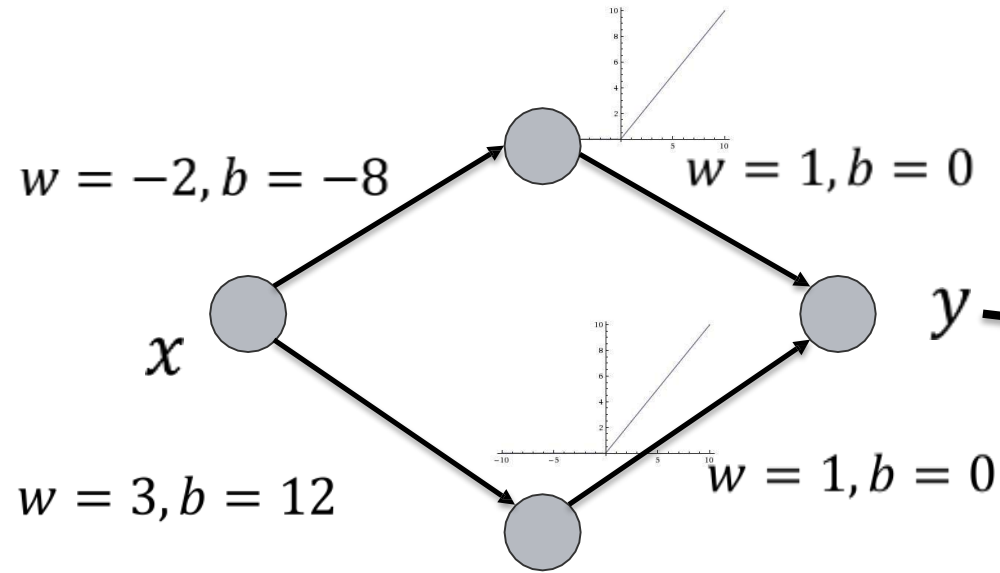


This file is meant for personal use by manju.crz@gmail.com only.

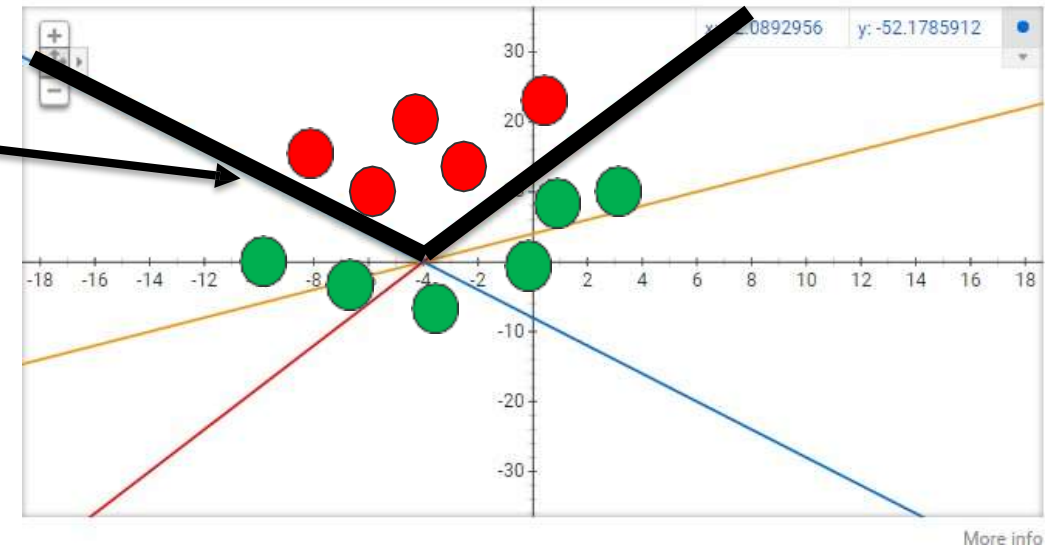
Sharing or publishing the contents in part or full is liable for legal action.



# ReLU



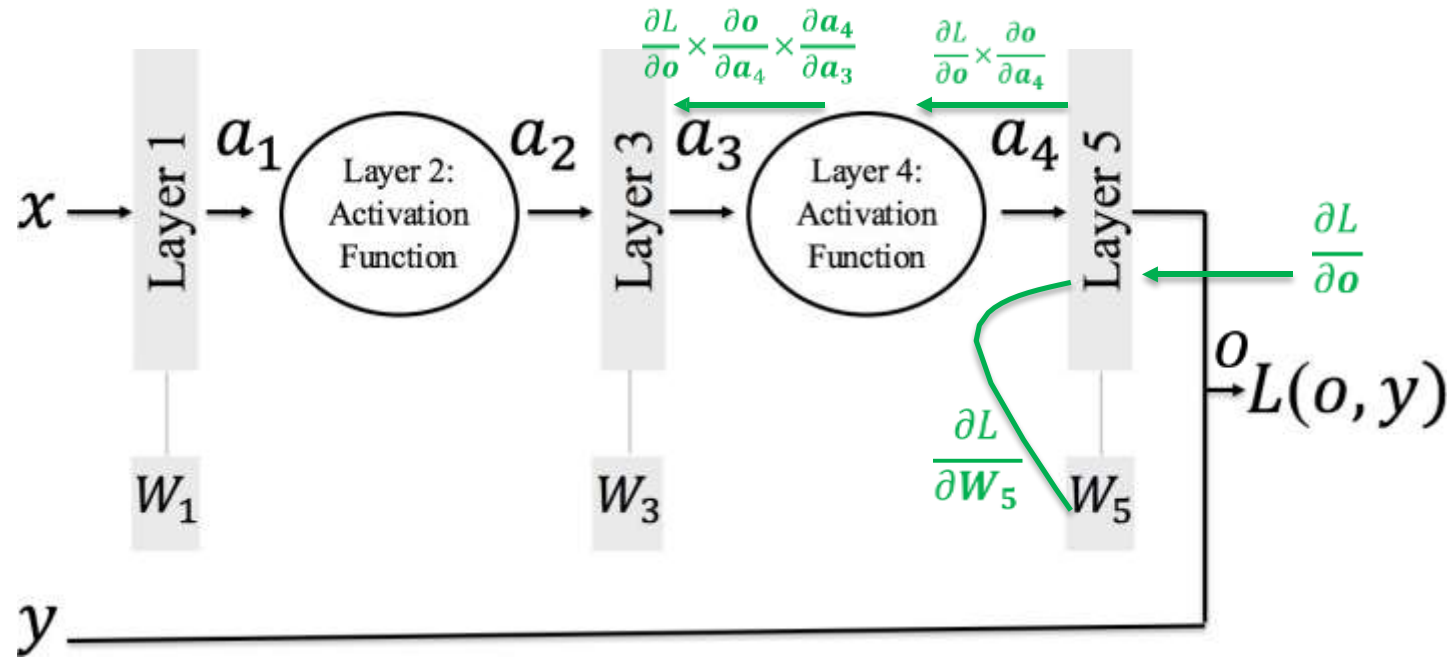
Graph for  $-(2*x))-8$ ,  $3*x+12$ ,  $x+4$



This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

# Vanishing/Exploding Gradients



$$\frac{\partial L}{\partial W_3} = \frac{\partial L}{\partial o} \times \frac{\partial o}{\partial a_4} \times \frac{\partial a_4}{\partial a_3} \times \frac{\partial a_3}{\partial a_2} \times \frac{\partial a_2}{\partial a_1} \times \frac{\partial a_1}{\partial W_1}$$

This file is meant for personal use by manju.crz@gmail.com only.

# Vanishing/Exploding Gradients

$$\frac{\partial L}{\partial W_3} = \frac{\partial L}{\partial o} \times \frac{\partial o}{\partial a_4} \times \frac{\partial a_4}{\partial a_3} \times \frac{\partial a_3}{\partial a_2} \times \frac{\partial a_2}{\partial a_1} \times \frac{\partial a_1}{\partial W_1}$$

Computing gradients involves many factors of W (and the repeated activation functions a):

- Largest singular value of the matrices > 1: Exploding gradients
- Largest singular value of the matrices < 1: Vanishing gradients

(Think of single neuron and a single weight. Then this becomes a geometric series and either goes to 0 or infinity.)

# SoftMax

This file is meant for personal use by manju.crz@gmail.com only.

Proprietary content: ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited

# Neural Network Constructed

Now, let's review the following in turn:

- Feed forward
- Back propagation
- Fully connected layer
- Activation functions
- **Softmax function**
- Cross-entropy loss

... and we'll have a fully functioning network

This file is meant for personal use by manju.crz@gmail.com only.

Proprietary content: ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited

# Softmax Layer

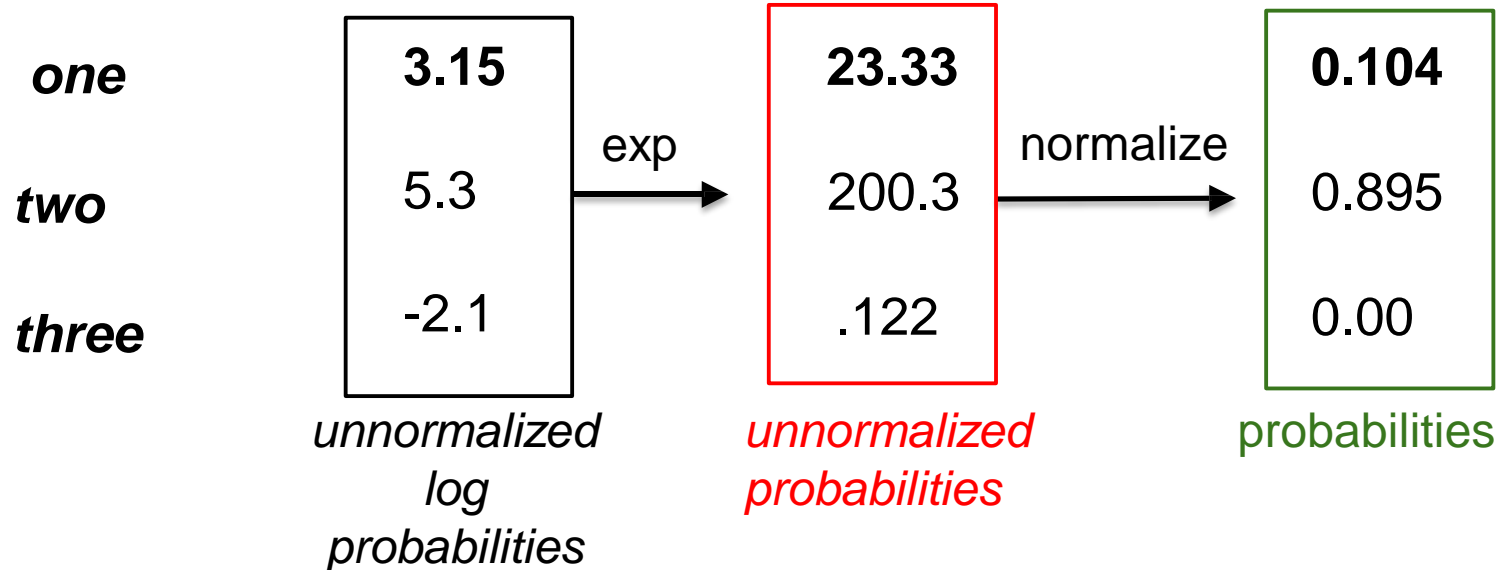
- Softmax function is a multinomial logistic classifier, i.e. it can handle multiple classes
- Softmax typically the last layer of a neural network based classifier
- Softmax function is itself an activation function, so doesn't need to be combined with an activation function

# Softmax

$$S \in \mathbb{R}^d \longrightarrow \text{SoftMax} \longrightarrow p \in \mathbb{R}^d \quad p_i = \frac{e^{s_i}}{\sum_{j=1}^d e^{s_j}}$$

# Softmax

$$S \in \mathbb{R}^d \longrightarrow \text{SoftMax} \longrightarrow p \in \mathbb{R}^d \quad p_i = \frac{e^{s_i}}{\sum_{j=1}^d e^{s_j}}$$



This file is meant for personal use by manju.crz@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.



# Softmax

$$S \in \mathbb{R}^d \xrightarrow{\text{SoftMax}} p \in \mathbb{R}^d \quad p_i = \frac{e^{s_i}}{\sum_{j=1}^d e^{s_j}}$$

```
n = np.array([3.15, 5.3, -2.1]).reshape(1, 3)
m = softmax(n)
```

```
n
array([[ 3.15,  5.3 , -2.1 ]])
```

```
m
array([[1.04274135e-01, 8.95178684e-01, 5.47180443e-04]])
```

```
def softmax(x):
    exp_x = np.exp(x - np.max(x, axis=1, keepdims=True))
    return exp_x / np.sum(exp_x, axis=1, keepdims=True)
```

This file is meant for personal use by manju.crz@gmail.com only.

# Cross Entropy

This file is meant for personal use by manju.crz@gmail.com only.

Proprietary content: ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited

# Neural Network Constructed

Now, let's review the following in turn:

- Feed forward
- Back propagation
- Fully connected layer
- Activation functions
- Softmax function
- Cross-entropy loss

... and we'll have a fully functioning network

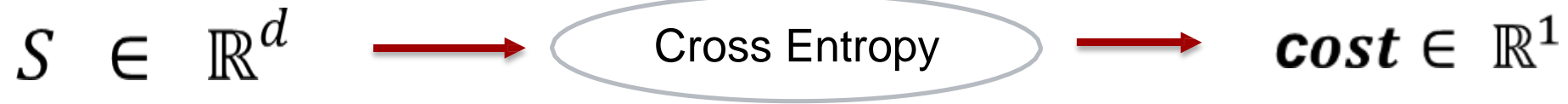
This file is meant for personal use by manju.crz@gmail.com only.

Proprietary content: ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited

# Cross-entropy loss

- Cross-entropy loss (often called Log loss) quantifies our unhappiness for the predicted output based on its deviation from the desired output
- Perfect prediction would have a loss of 0 (we will see how)
- With gradient descent, we try to reduce this (cross-entropy) loss for a classification problem

# Cross Entropy



---

$y_i$  is 1 (and 0 otherwise) if and only  
if sample belongs to class  $i$

$$s = f(x_i; W)$$

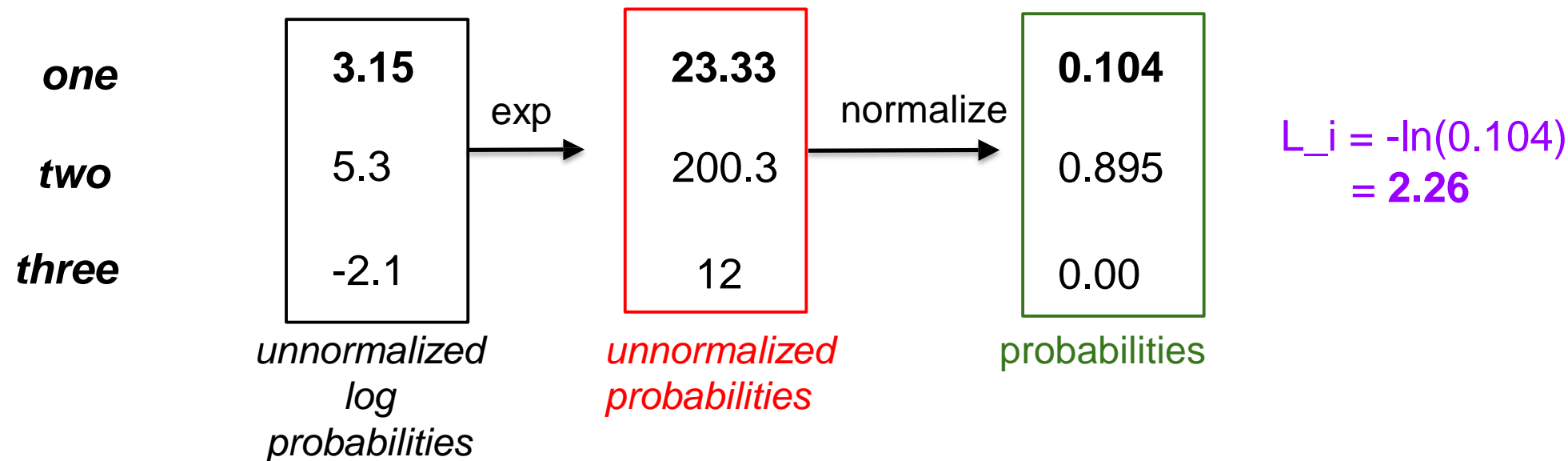
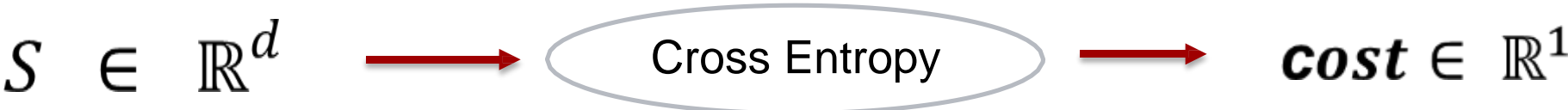
$$L_i = -y_i \cdot \log\left(\frac{e^{s_i}}{\sum_j e^{s_j}}\right)$$

$$L = \sum_i L_i$$

This file is meant for personal use by manju.crz@gmail.com only.

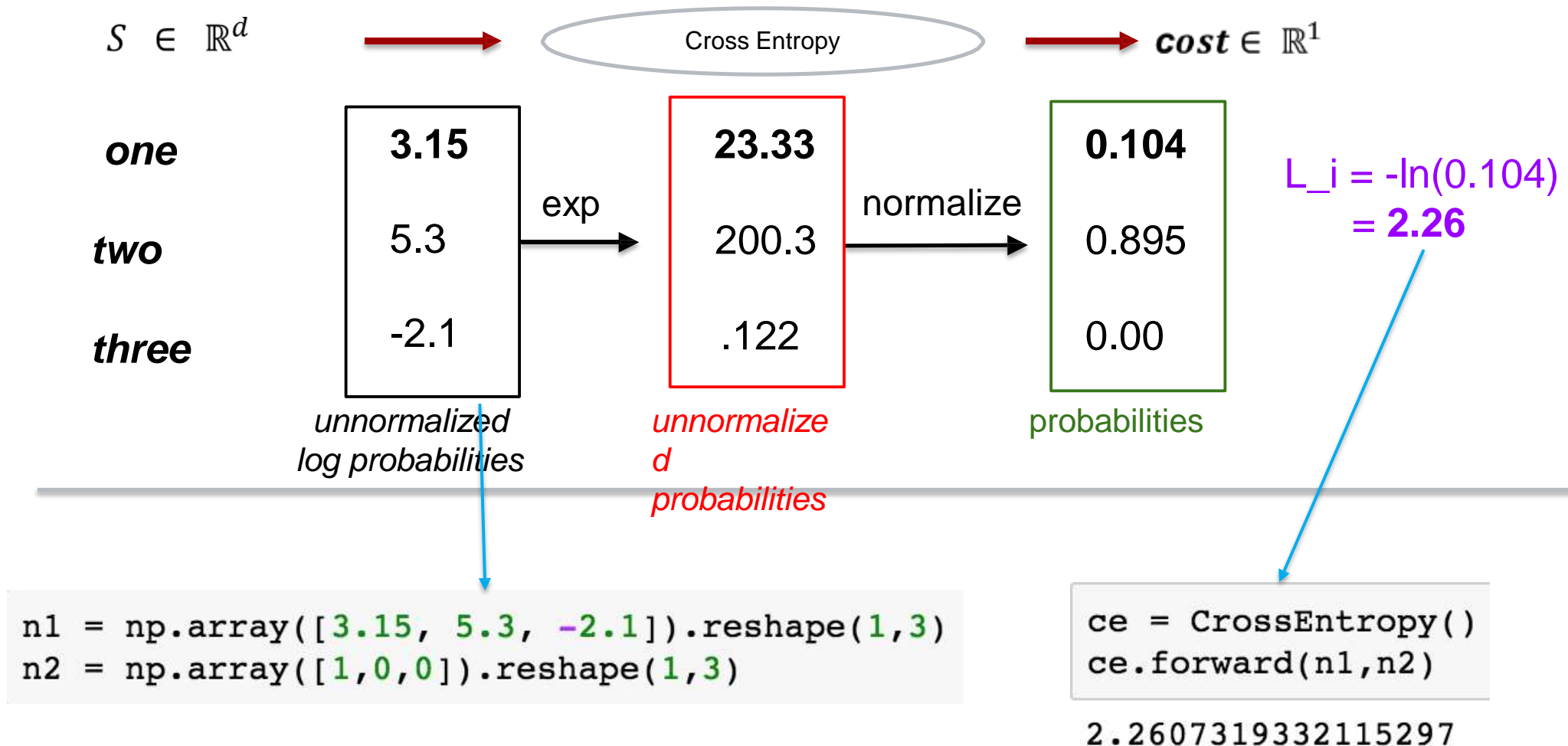
Proprietary content: ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited

# Cross Entropy



This file is meant for personal use by manju.crz@gmail.com only.

# Cross Entropy



This file is meant for personal use by manju.crz@gmail.com only.

# Cross Entropy

```
class CrossEntropy:
    def forward(self, X, y):
        y_idx = y.argmax()
        self.p = softmax(X)
        cross_entropy = -np.log(self.p[y_idx])
        loss = cross_entropy
        return loss

    def backward(self, X, y):
        y_idx = y.argmax()
        grad = softmax(X)
        grad[y_idx] -= 1
        return grad
```

```
n1 = np.array([3.15, 5.3, -2.1])
n2 = np.array([1, 0, 0])
```

```
ce = CrossEntropy()
ce.forward(n1, n2)
```

2.2607319332115297

This file is meant for personal use by manju.crz@gmail.com only.



# Cross Entropy



$y_i$  is 1 (and 0 otherwise) if and only if sample belongs to class  $i$

$$\frac{\partial p_j}{\partial s_k} = \begin{cases} p_j(1 - p_j) & \text{if } j = k \\ -p_j p_k & \text{if } j \neq k \end{cases}$$

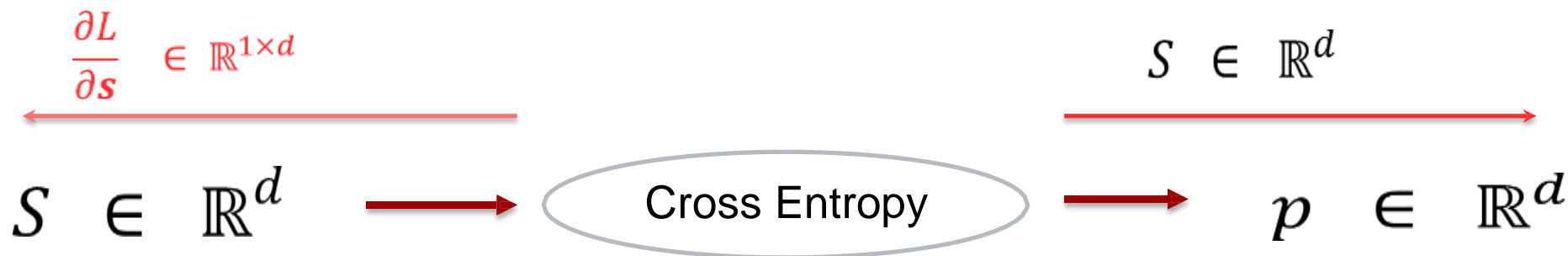
$$L_i = -y_i \cdot \log\left(\frac{e^{s_i}}{\sum_j e^{s_j}}\right)$$

$$L = \sum_i L_i$$

This file is meant for personal use by manju.crz@gmail.com only.

Proprietary content: ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited

# Cross Entropy



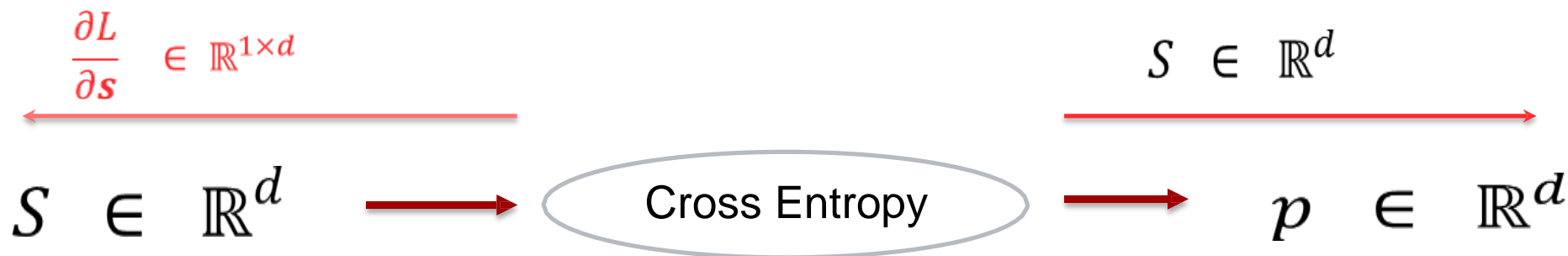
$$p_i = \frac{e^{s_i}}{\sum_j e^{s_j}}$$

$$S \in \mathbb{R}^d$$
$$L_i = -y_i \cdot \log\left(\frac{e^{s_i}}{\sum_j e^{s_j}}\right)$$

$$L = \sum_i L_i$$

This file is meant for personal use by manju.crz@gmail.com only.

# Cross Entropy



$$p_i = \frac{e^{s_i}}{\sum_j e^{s_j}}$$

$$s = f(x_i; W)$$

$$p_i = \frac{e^{s_i}}{\sum_j e^{s_j}}$$

$$\frac{\partial L}{\partial s_k} = p_k - y_k$$

$$L_i = -y_i \cdot \log(p_i)$$

$$L = \sum_i L_i$$

This file is meant for personal use by manju.crz@gmail.com only.

# Cross Entropy

```
class CrossEntropy:
    def forward(self, X, y):
        y_idx = y.argmax()
        self.p = softmax(X)
        cross_entropy = -np.log(self.p[y_idx])
        loss = cross_entropy
        return loss

    def backward(self, X, y):
        y_idx = y.argmax()
        grad = softmax(X)
        grad[y_idx] -= 1
        return grad
```

```
n1 = np.array([3.15, 5.3, -2.1])
n2 = np.array([1, 0, 0])
```

```
ohat = softmax(n1)
```

```
print ohat
```

```
[1.04274135e-01  8.95178684e-01  5.47180443e-04]
```

```
ce.backward(n1,n2)
```

```
array([-8.95725865e-01,  8.95178684e-01,  5.47180443e-04])
```

```
ohat - n2
```

```
array([-8.95725865e-01,  8.95178684e-01,  5.47180443e-04])
```

This file is meant for personal use by manju.crz@gmail.com only.

Proprietary content: ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited

# Summary

We saw how each of the components of a deep neural network contributes to its functioning.

- Feed forward
- Back propagation
- Fully connected layer
- Activation functions
- Softmax function
- Cross-entropy loss

Next, we will see how to make it all work.

# Thank you!