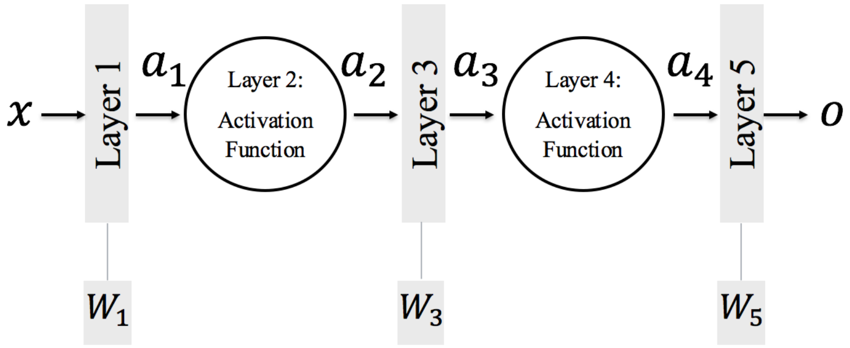
greatlearning
Learning for Life

Deep Learning (for Computer Vision)

Arjun Jain

Multiple Layers – Feed Forward - Composition of Functions



$$a_1 = F(x, W_1),\ x \in \mathbb{R}^n$$

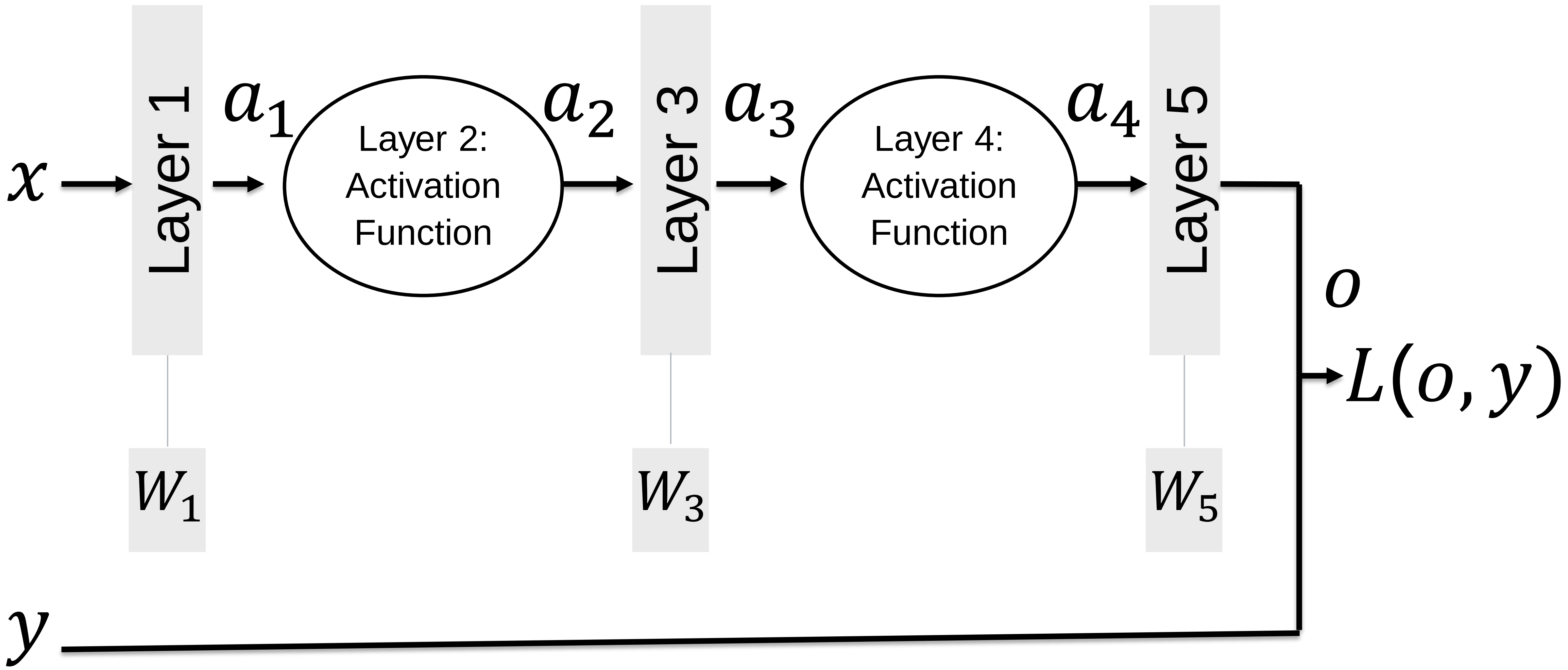
$$a_2 = G(a_1)$$

$$a_3 = H(a_2, W_3),$$

$$a_4 = J(a_3)$$

$$o = K(a_4, W_5) = K(J(H(G(F(x, W_1)), W_3)), W_5) \in \mathbb{R}^m$$

Multiple Layers – Feed Forward - Loss



Vector Calculus Refresher

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

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

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

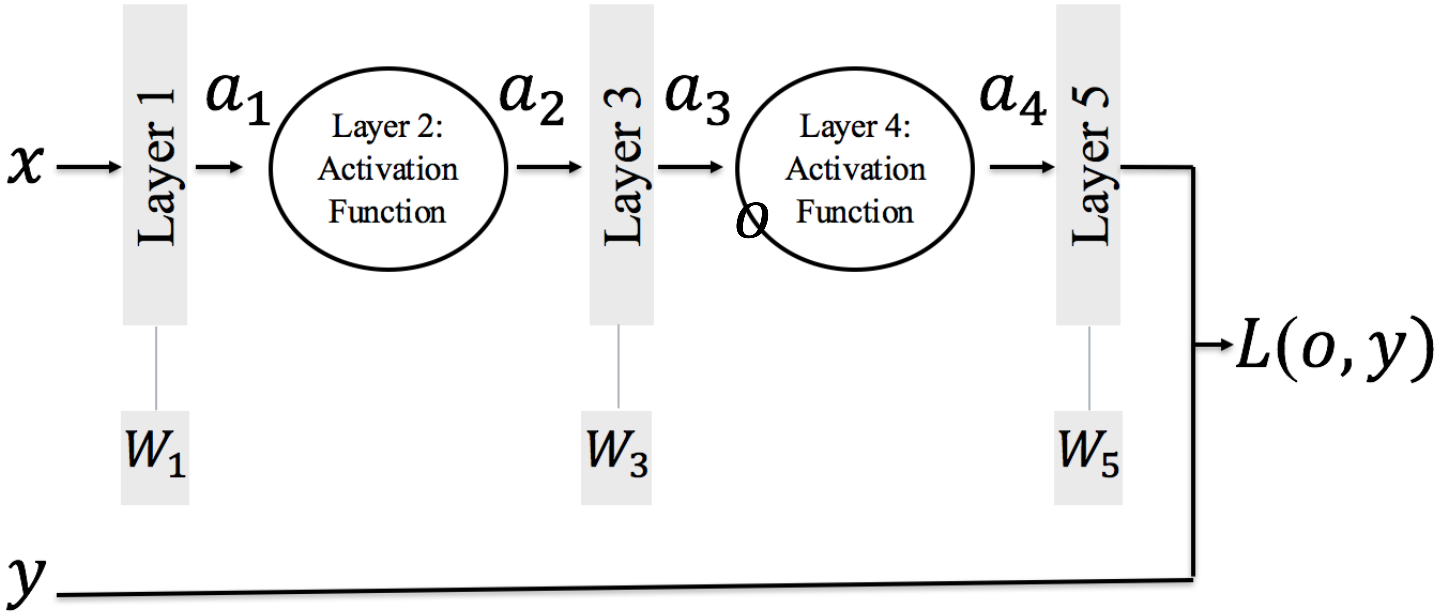
Let $x \in R^n$ (a column vector) and let $f : R^n \to R^m$. The derivative of f with respect to x is the $m \times n$ matrix:

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

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

Sourced from: http://www.cs.huji.ac.il/~csip/tirgul3_derivatives.pdf

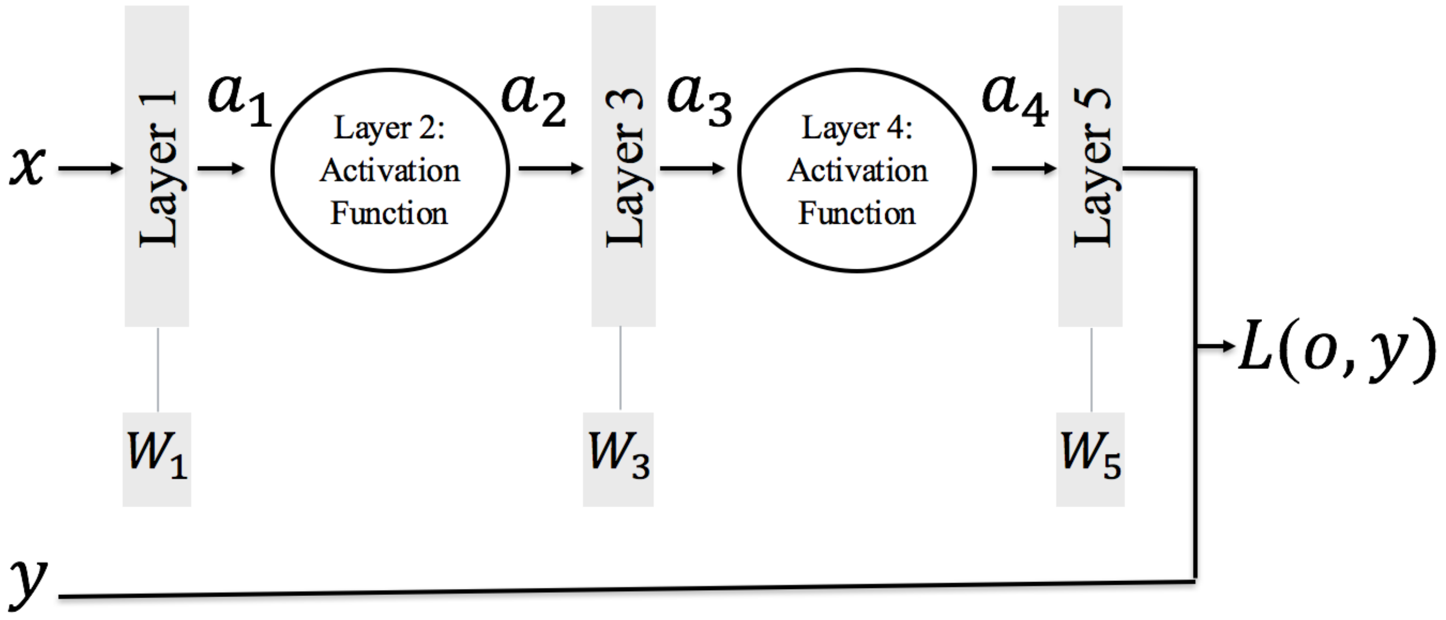
Multiple Layers – Back Prop: Chain Rule



We want:

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

Multiple Layers – Back Prop: Chain Rule



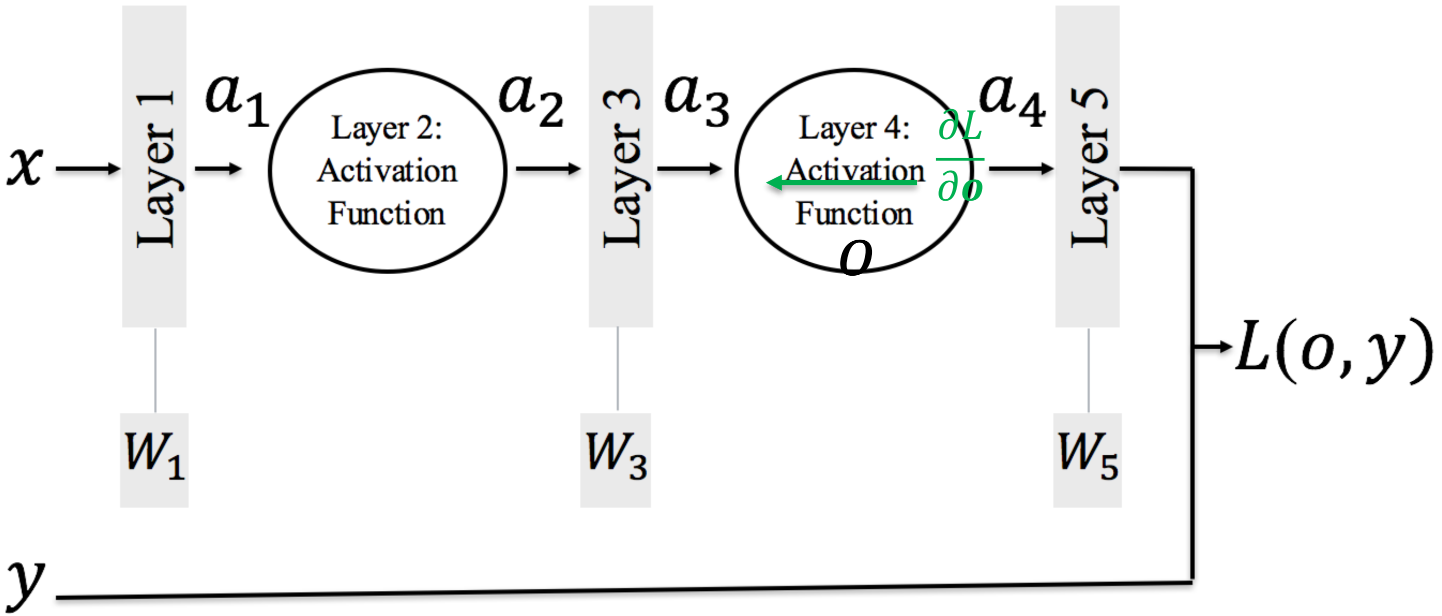
We want:

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

Compute:

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

Multiple Layers – Back Prop: Chain Rule



We want:

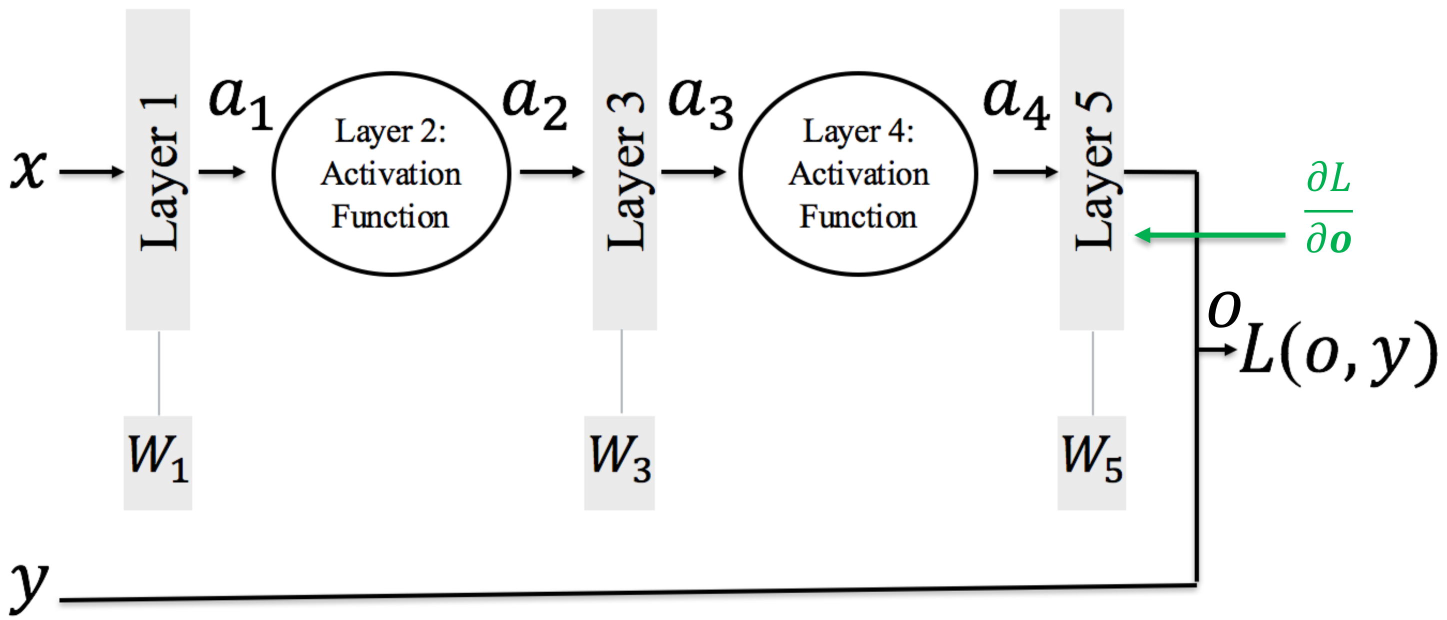
$$\frac{\partial L}{\partial \boldsymbol{W_1}}, \frac{\partial L}{\partial \boldsymbol{W_3}}, \frac{\partial L}{\partial \boldsymbol{W_5}}$$

Compute:

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

E.g: $L(\boldsymbol{o}, \boldsymbol{y}) = {}^1\!/_2 \parallel \boldsymbol{o} - \boldsymbol{y} \parallel^2$ then: $\frac{\partial L}{\partial \boldsymbol{o}}$

Multiple Layers – Back Prop: Chain Rule



We want:

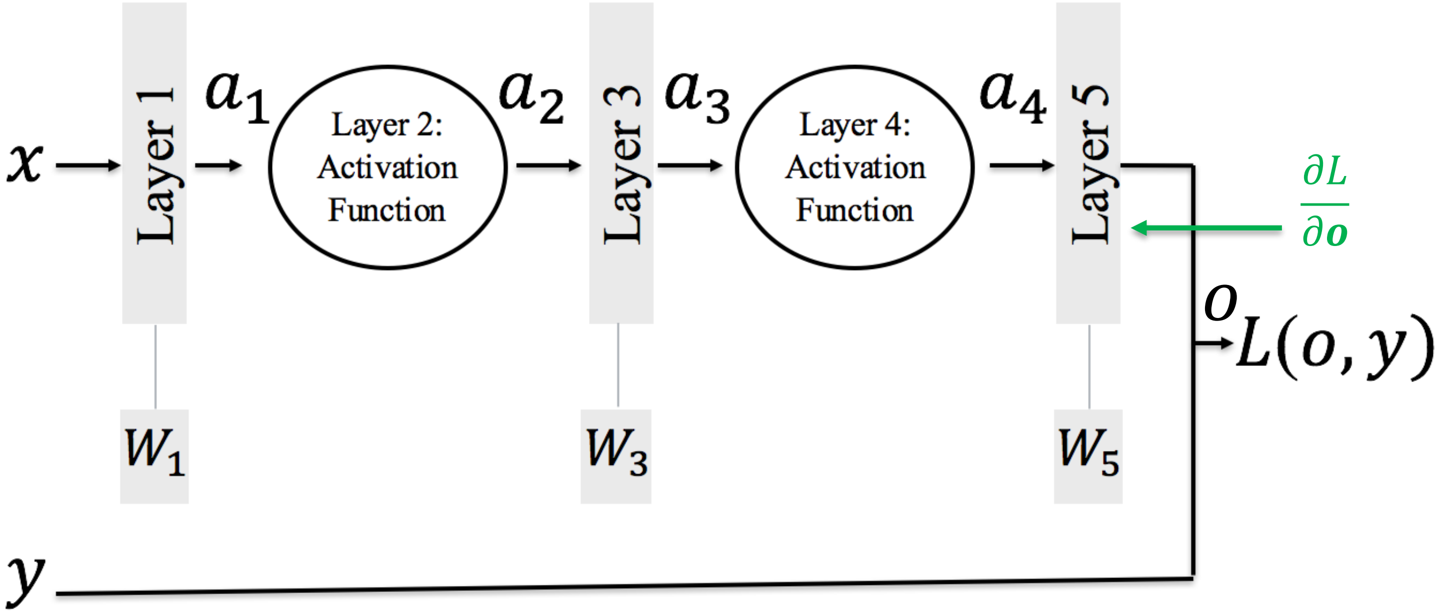
$$\frac{\partial L}{\partial \boldsymbol{W_1}}, \frac{\partial L}{\partial \boldsymbol{W_3}}, \frac{\partial L}{\partial \boldsymbol{W_5}}$$

Compute:

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

E.g: $L(\boldsymbol{o}, \boldsymbol{y}) = {}^1\!/_2 \parallel \boldsymbol{o} - \boldsymbol{y} \parallel^2$ then: $\frac{\partial L}{\partial \boldsymbol{o}} = (\boldsymbol{o} - \boldsymbol{y})$

Multiple Layers – Back Prop: Chain Rule



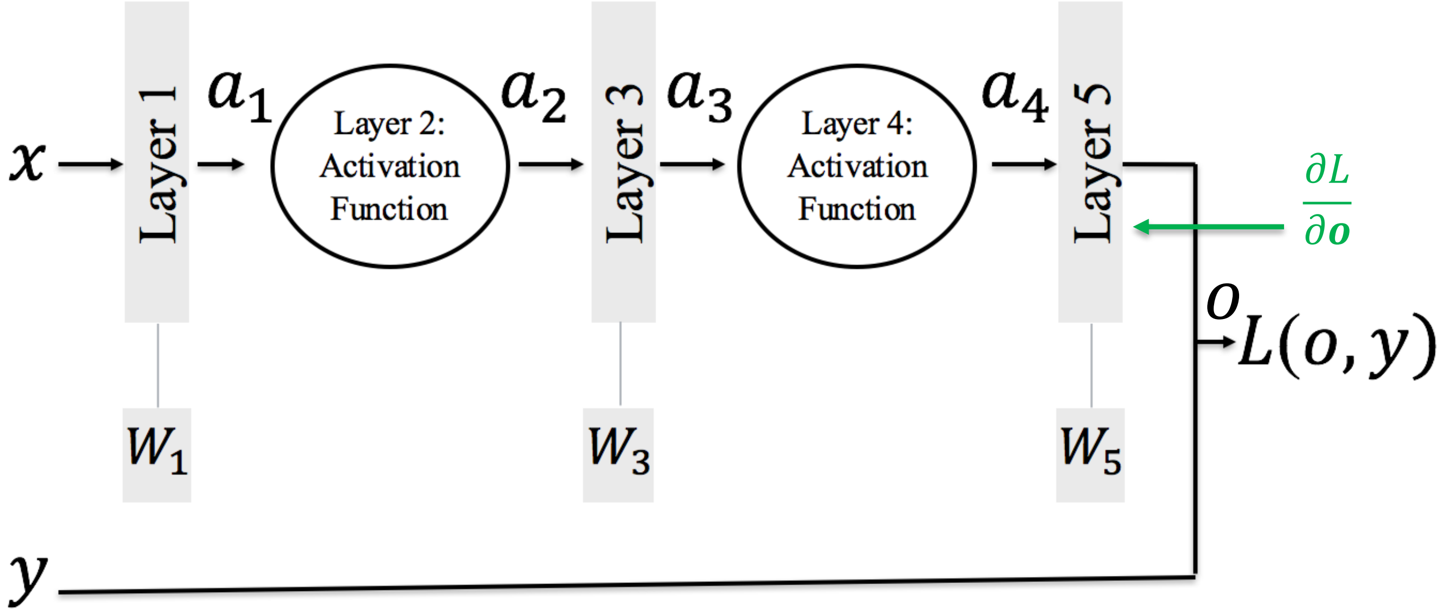
$$\frac{\partial L}{\partial \boldsymbol{o}} \epsilon\ \mathbb{R}^{1\times m}$$

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

Compute: $\frac{\partial L}{\partial \boldsymbol{o}}$

E.g: $L(\boldsymbol{o}, \boldsymbol{y}) = {}^1/_2 \parallel \boldsymbol{o} - \boldsymbol{y} \parallel^2$ then: $\frac{\partial L}{\partial \boldsymbol{o}} = (\boldsymbol{o} - \boldsymbol{y})$

Multiple Layers – Back Prop: Chain Rule



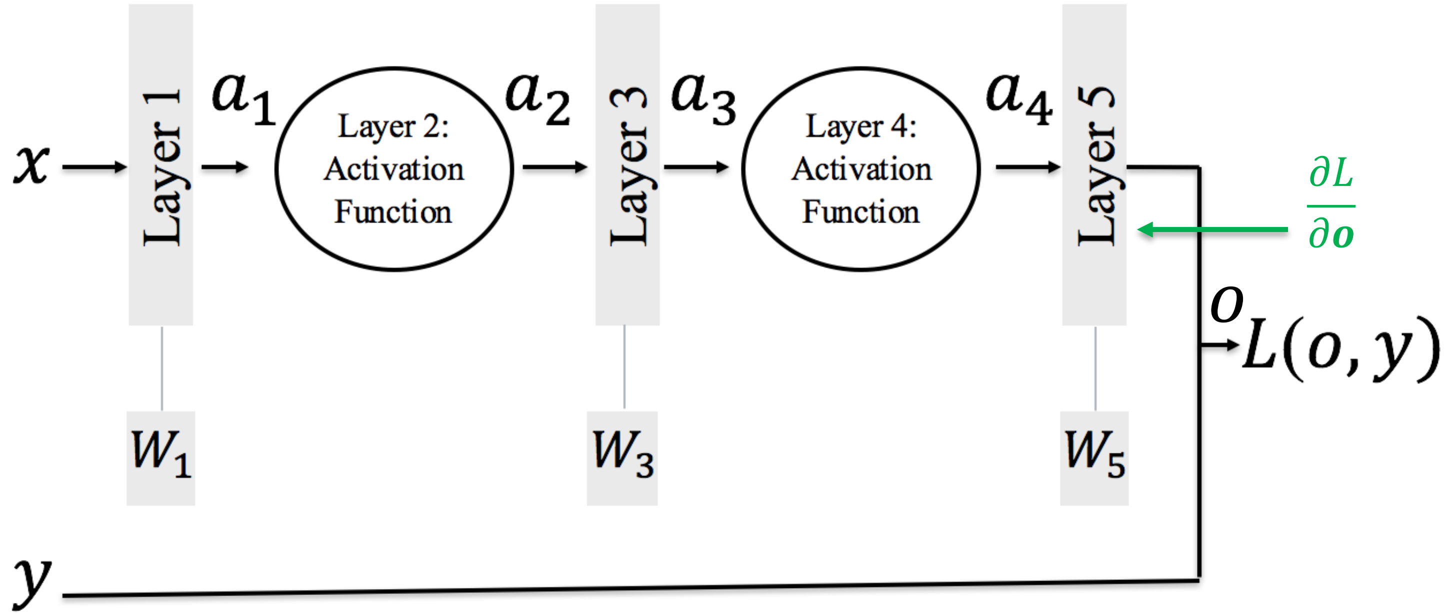
We want:

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

We can also compute:

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

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:

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

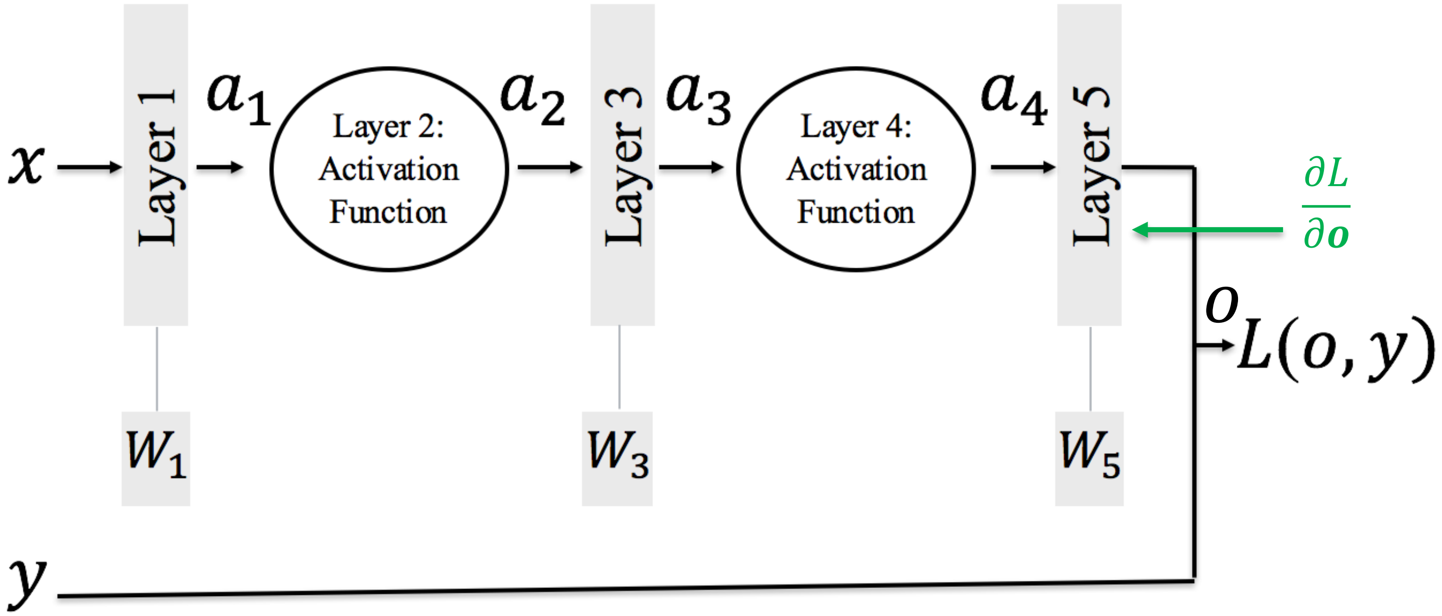
We can also compute:

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

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

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

Multiple Layers – Back Prop: Chain Rule



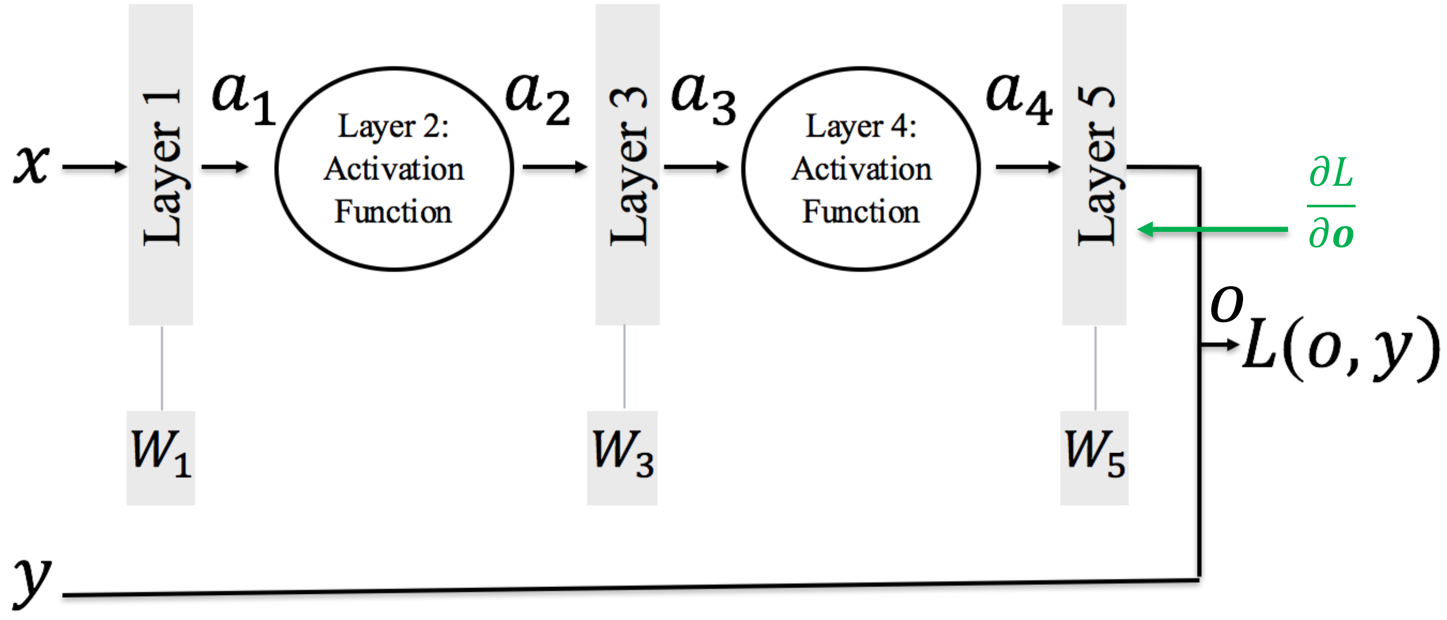
We want: $\frac{\partial L}{\partial \boldsymbol{W}_1}, \frac{\partial L}{\partial \boldsymbol{W}_3}, \frac{\partial L}{\partial \boldsymbol{W}_5}$

We can also compute: $\frac{\partial L}{\partial \boldsymbol{o}}, \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{W}_5}$

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

then: $\left[\frac{\partial \boldsymbol{o}}{\partial \boldsymbol{W}_5}\right]_{kl} = \frac{\partial [K(\boldsymbol{a}_4, \boldsymbol{W}_5)]_k}{\partial [\boldsymbol{W}_5]_l}$

Multiple Layers – Back Prop: Chain Rule



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

We want:

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

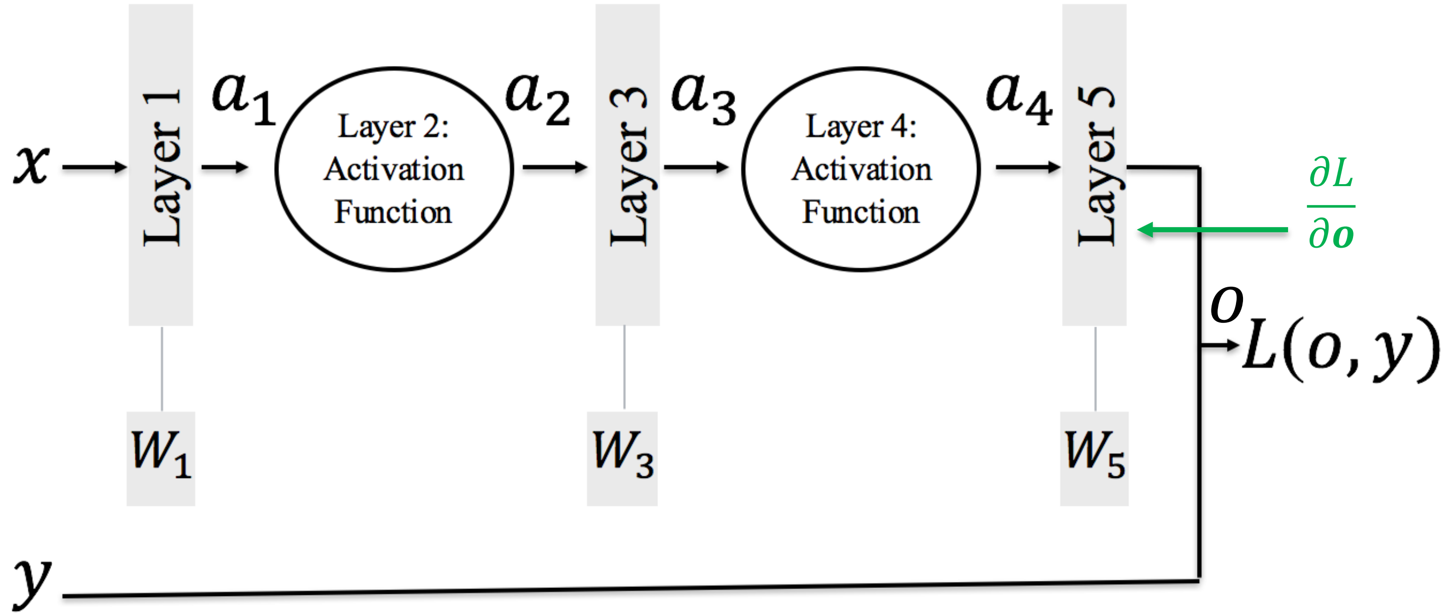
We can also compute:

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

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

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

Multiple Layers – Back Prop: Chain Rule



Remember:

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

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

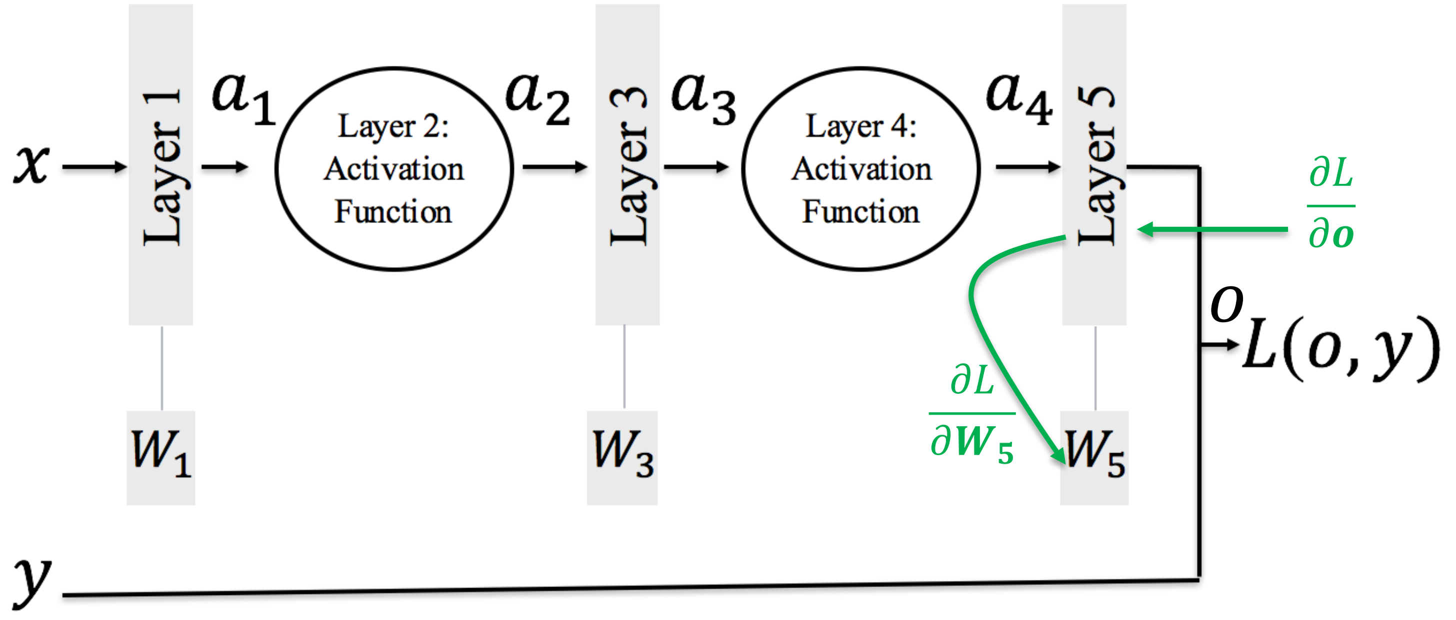
We want:

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

We can also compute:

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

Multiple Layers – Back Prop: Chain Rule



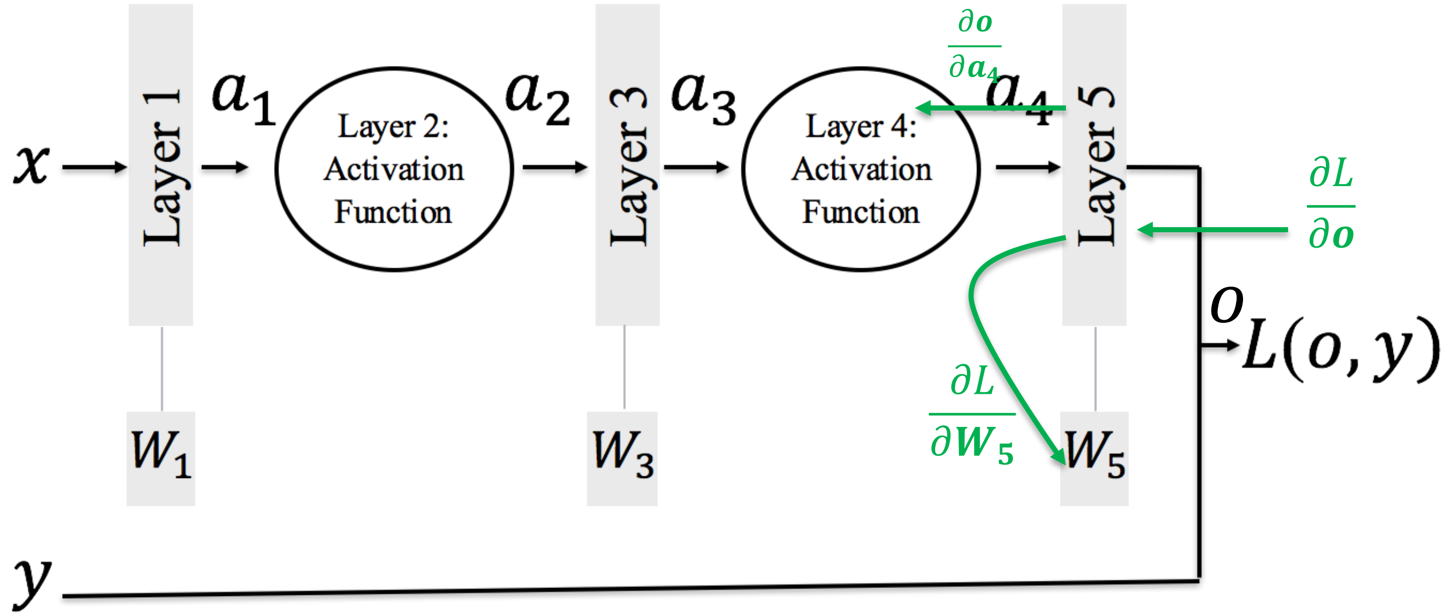
We want:

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

We know:

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

Multiple Layers – Back Prop: Chain Rule



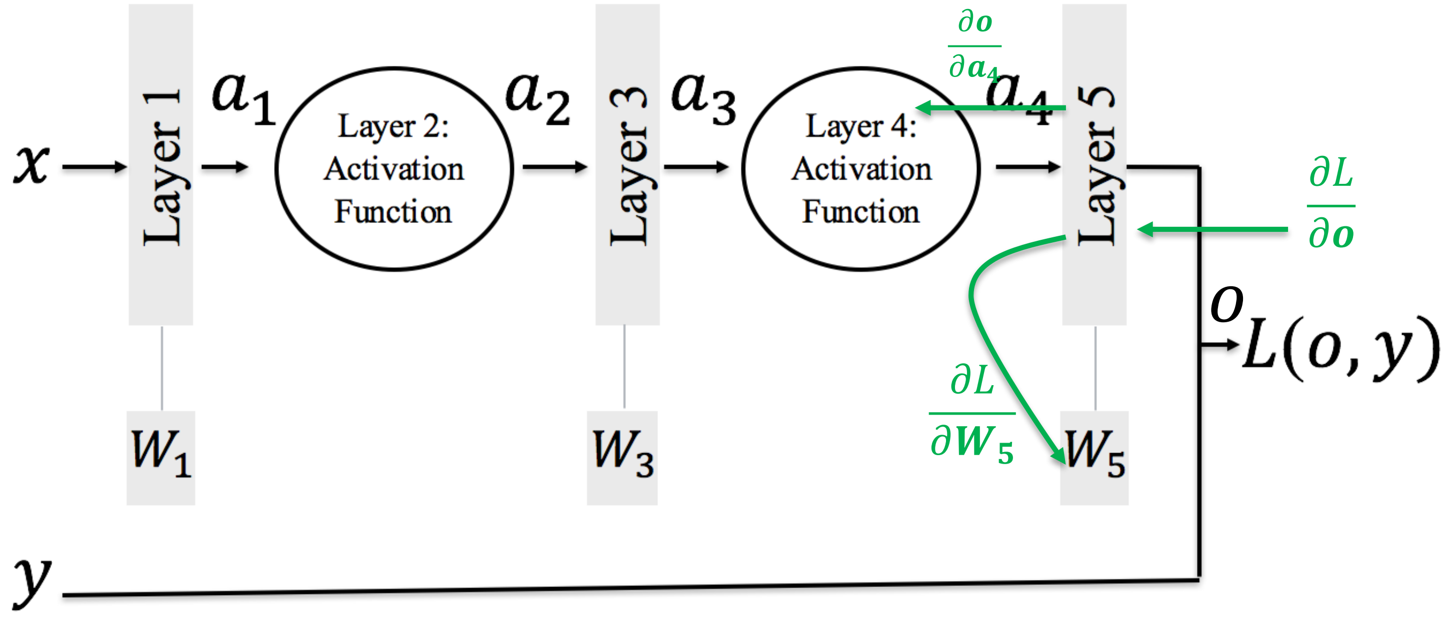
We want:

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

since: $\boldsymbol{o} = K(\boldsymbol{a_4}, \boldsymbol{W_5})$

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

Multiple Layers – Back Prop: Chain Rule



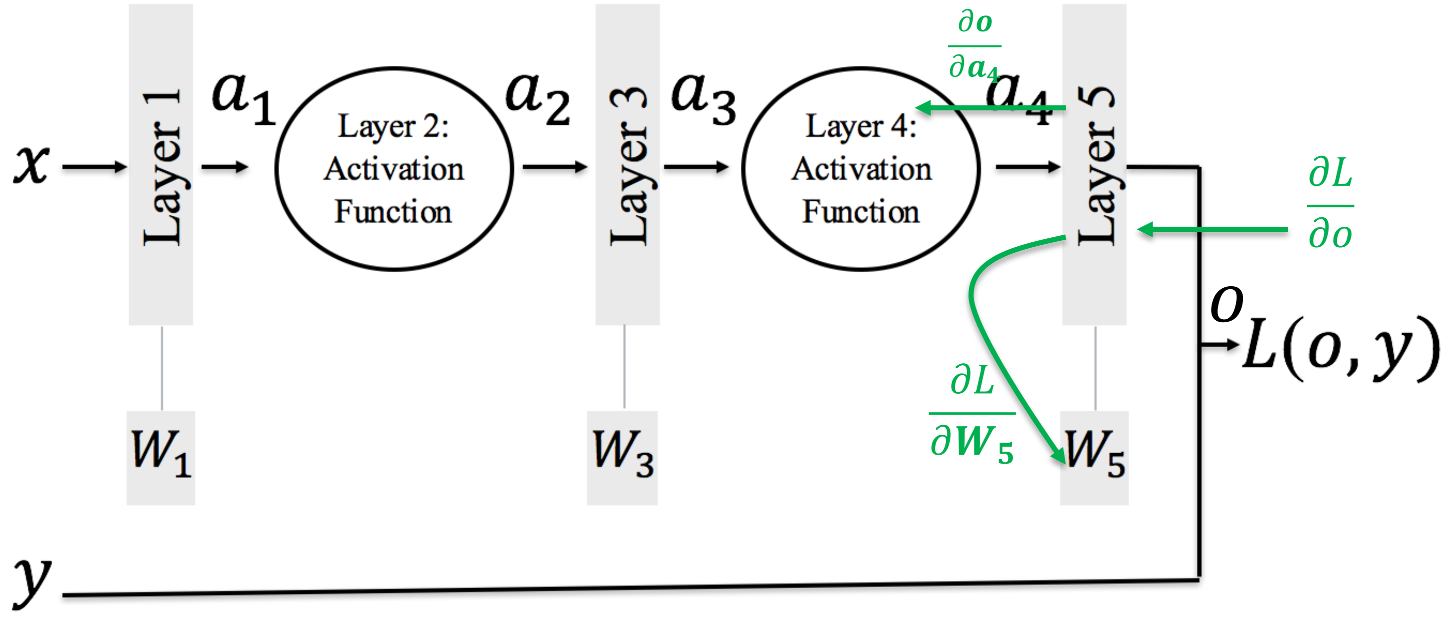
We want:

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

since: $\boldsymbol{o} = K(\boldsymbol{a_4}, \boldsymbol{W_5})$

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

Multiple Layers – Back Prop: Chain Rule



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

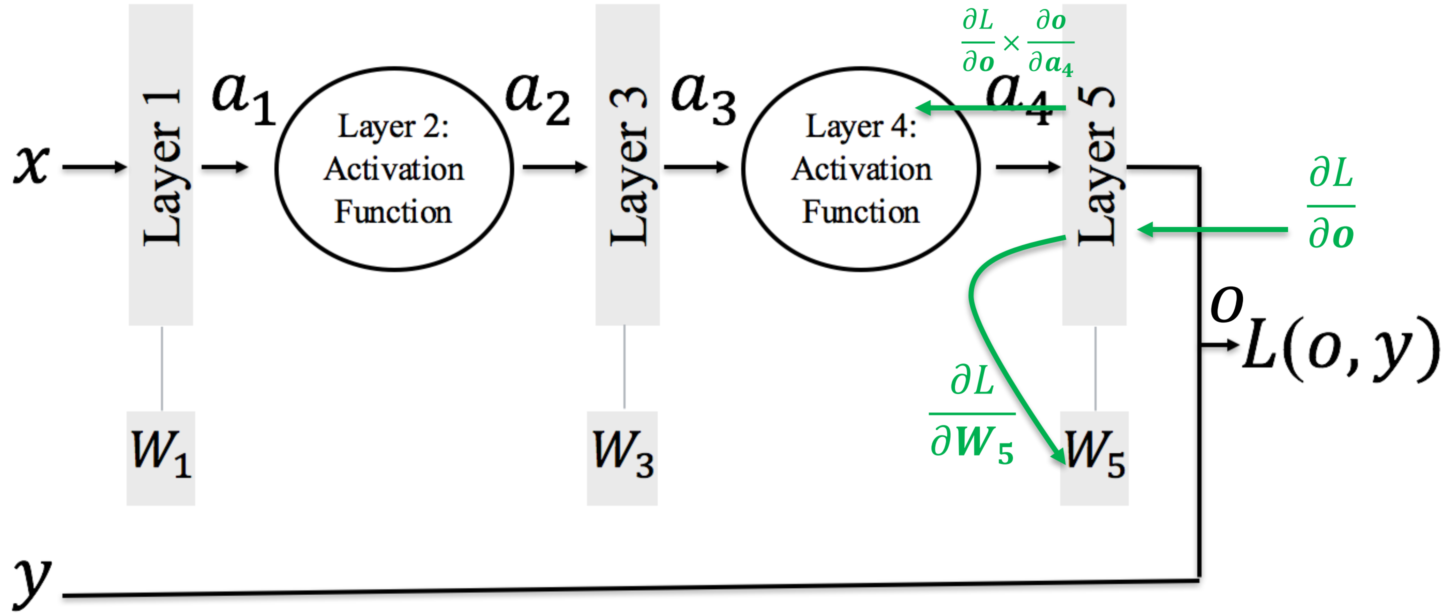
We want:

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

since: $\boldsymbol{o} = K(\boldsymbol{a_4}, \boldsymbol{W_5})$

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

Multiple Layers – Back Prop: Chain Rule



Remember:

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

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

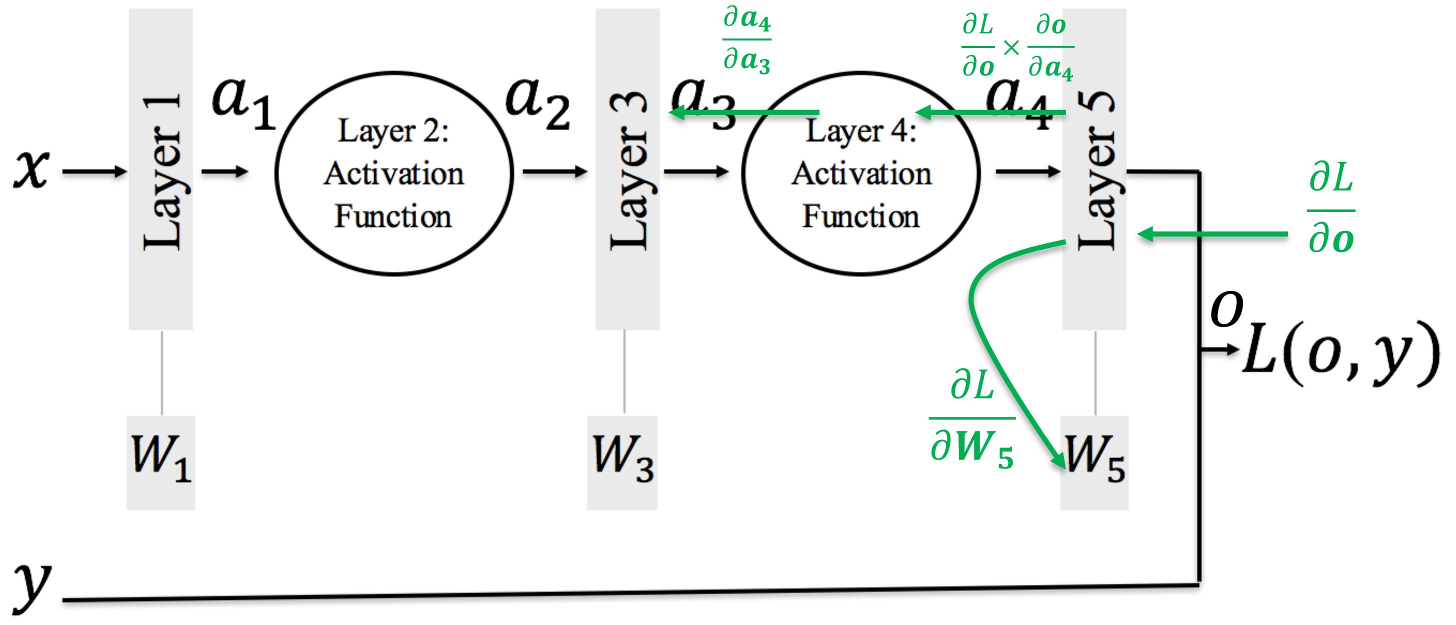
We want:

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

Backpropagate:

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

Multiple Layers – Back Prop: Chain Rule



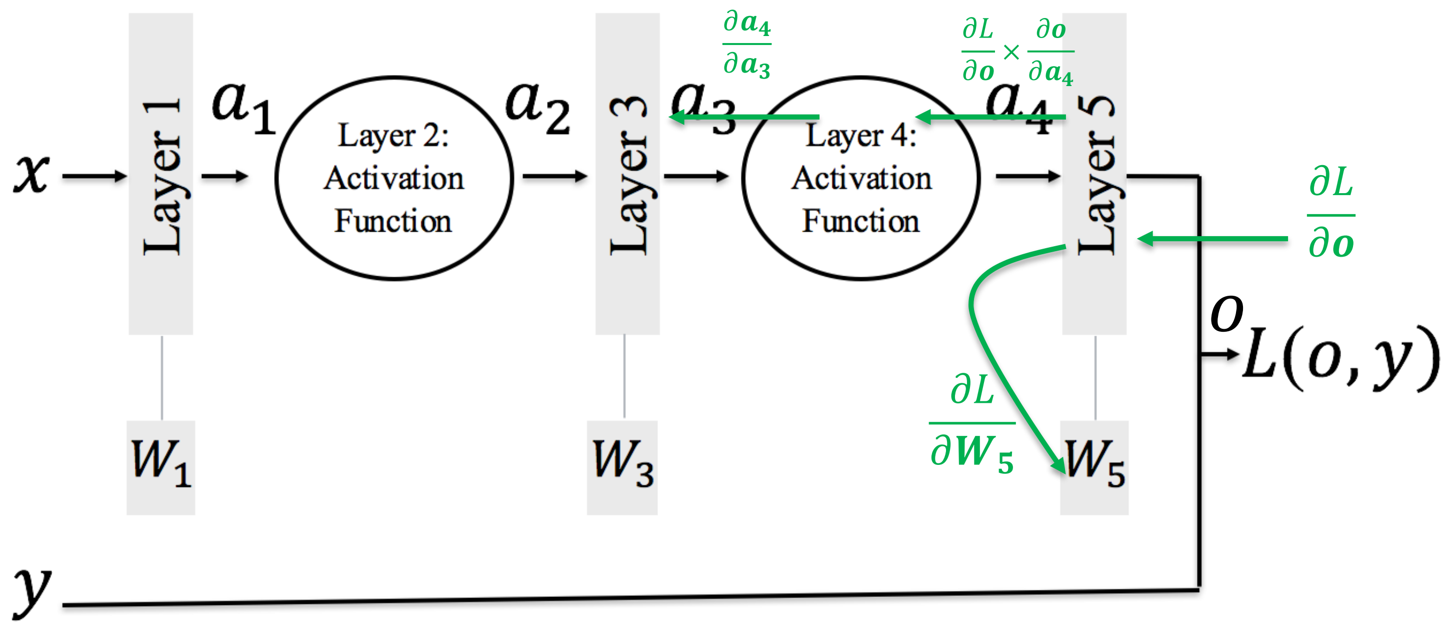
We want:

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

since: $a_4 = J(a_3)$

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

Multiple Layers – Back Prop: Chain Rule



Remember:

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

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

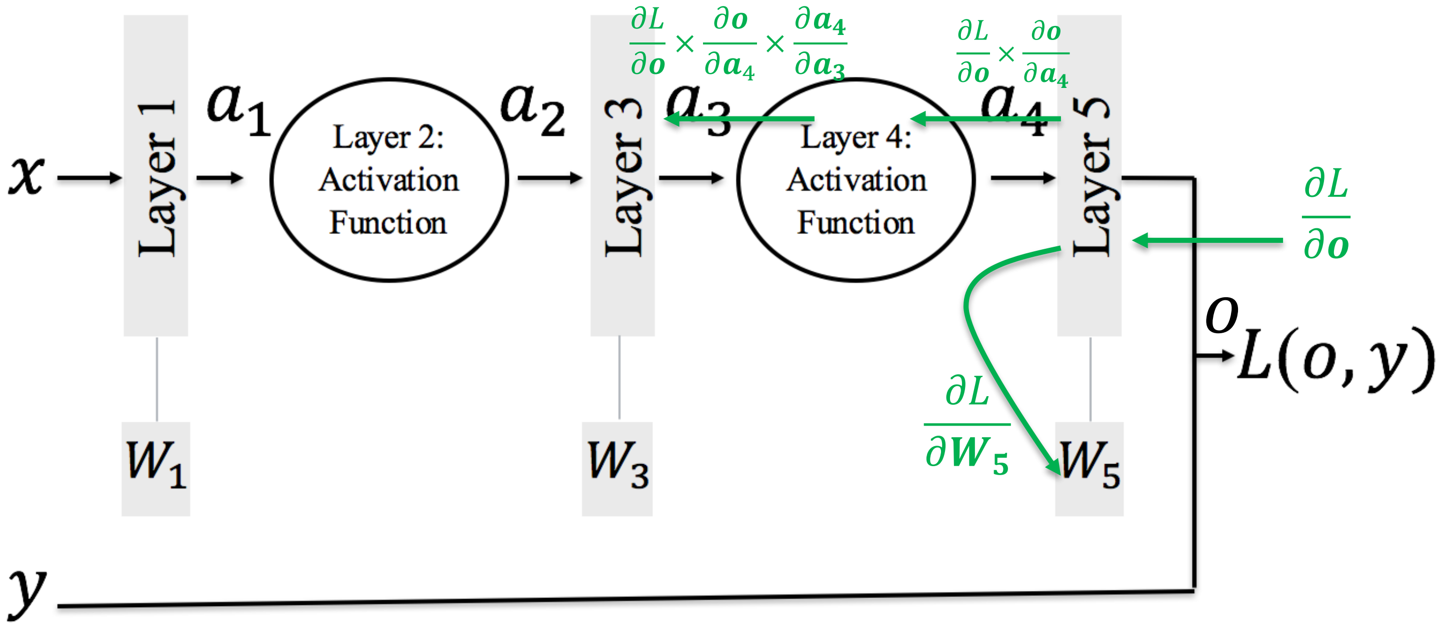
We want:

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

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:

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

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

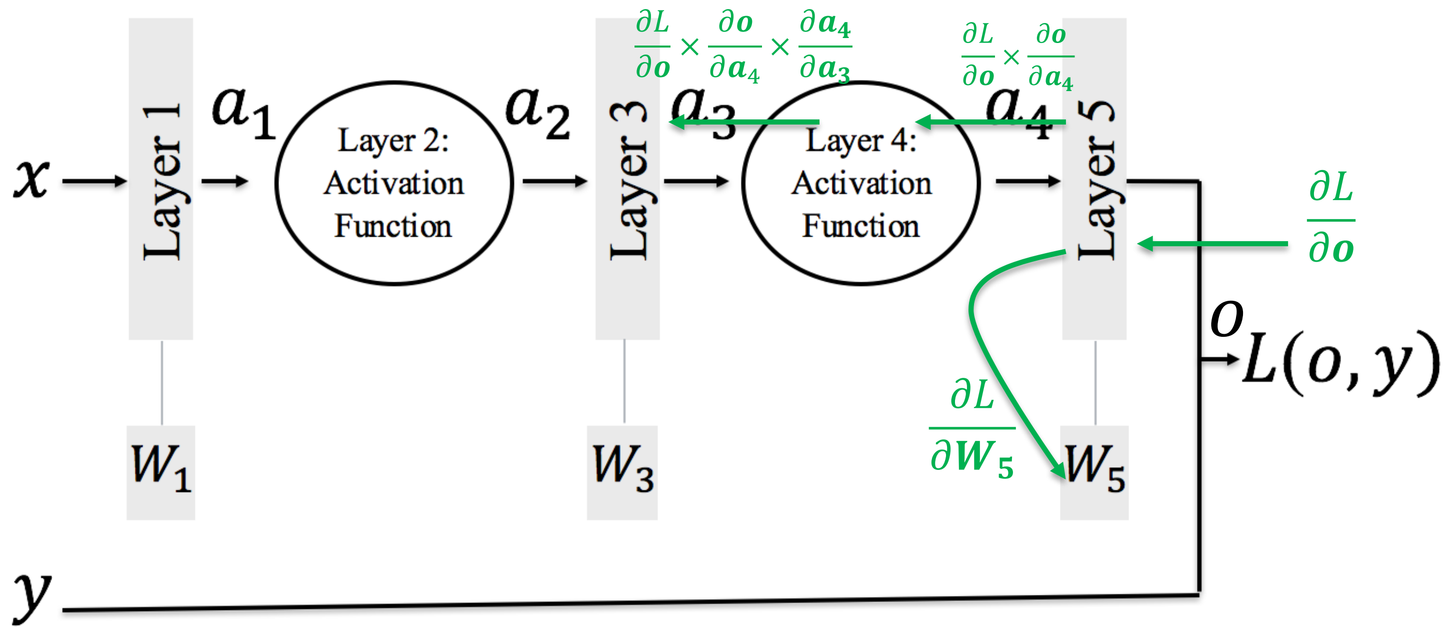
We want:

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

Backpropagate:

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

Multiple Layers – Back Prop: Chain Rule



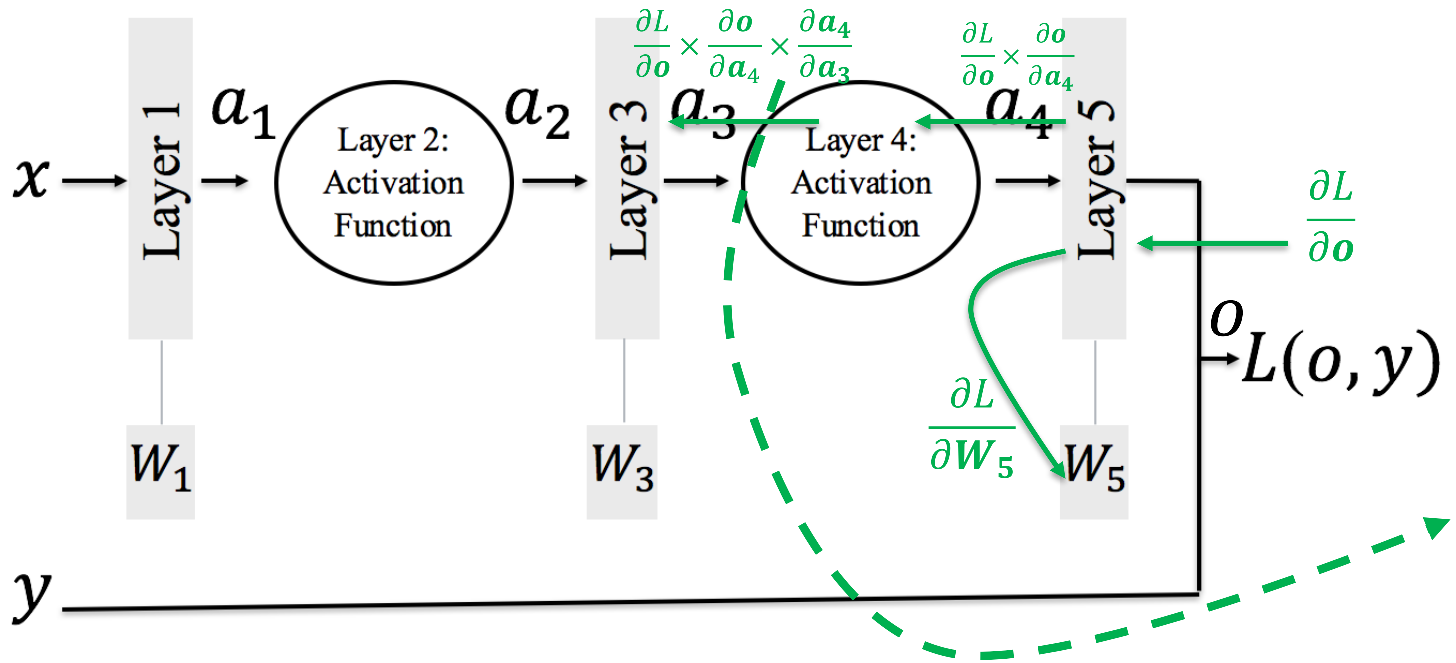
We want:

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

since: $\boldsymbol{a_3} = K(\boldsymbol{a_2}, \boldsymbol{W_3})$

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

Multiple Layers – Back Prop: Chain Rule



Remember
:

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

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

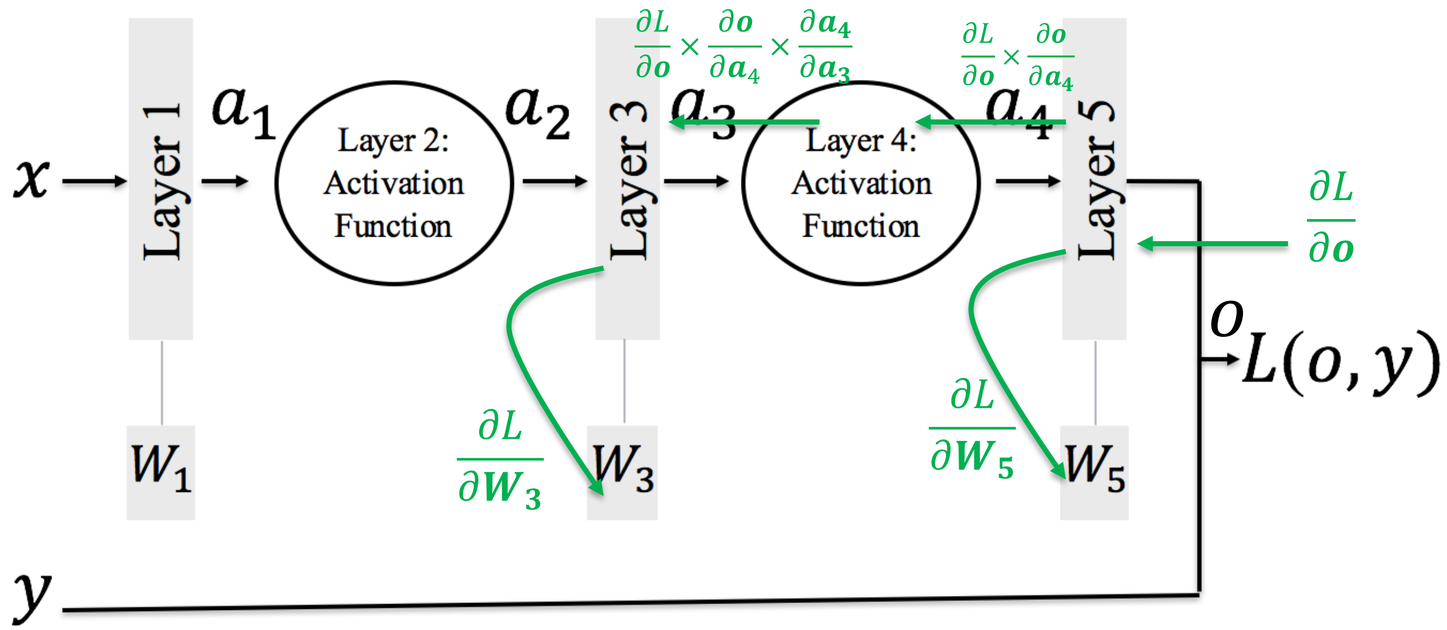
We want:

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

Now we can compute:

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

Multiple Layers – Back Prop: Chain Rule



Remember:

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

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

We want:

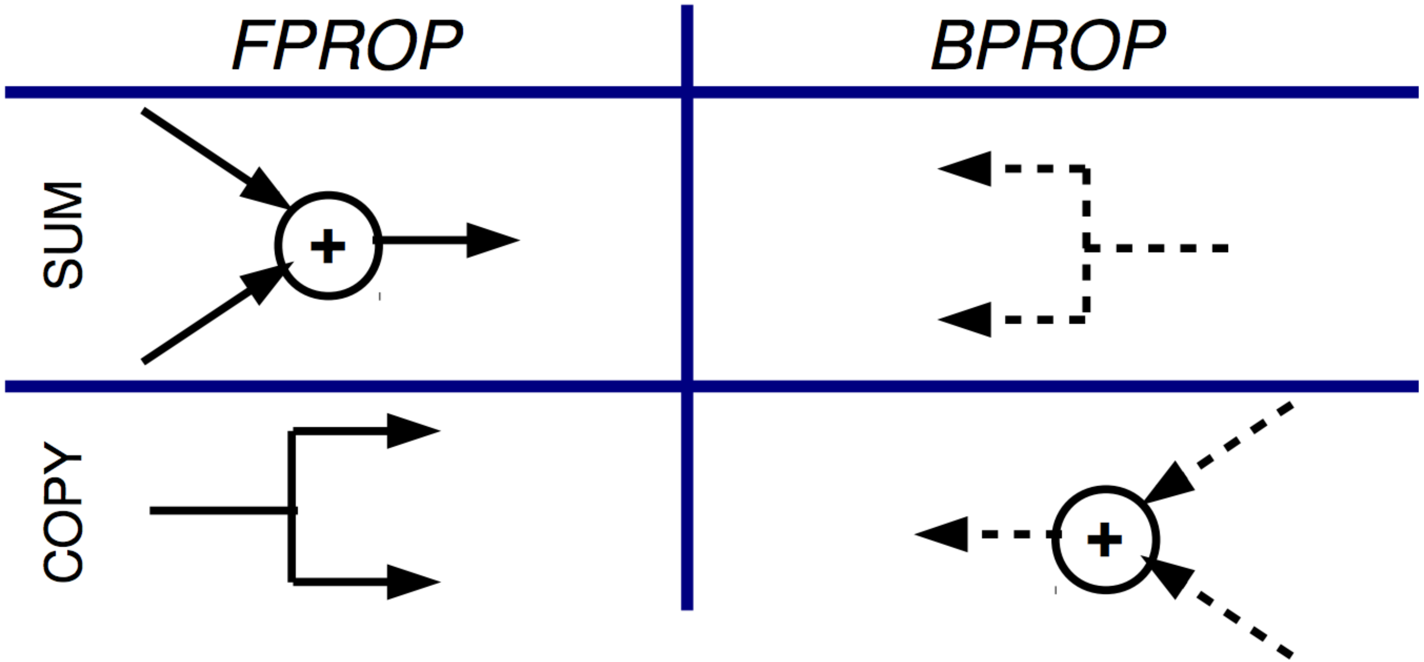
$$\frac{\partial L}{\partial \boldsymbol{W_1}}, \frac{\partial L}{\partial \boldsymbol{W_3}}, \frac{\partial L}{\partial \boldsymbol{W_5}}$$

Now we can compute:

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

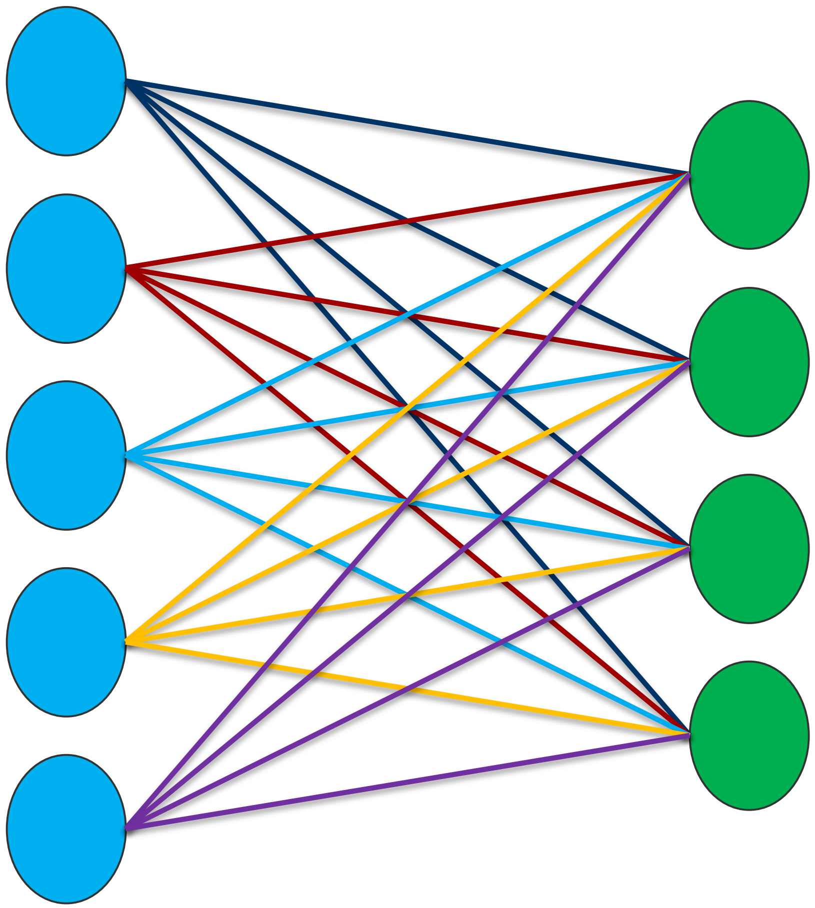
Multiple Layers – Back Prop: Chain Rule

FPROP and BPROP are duals of each other:



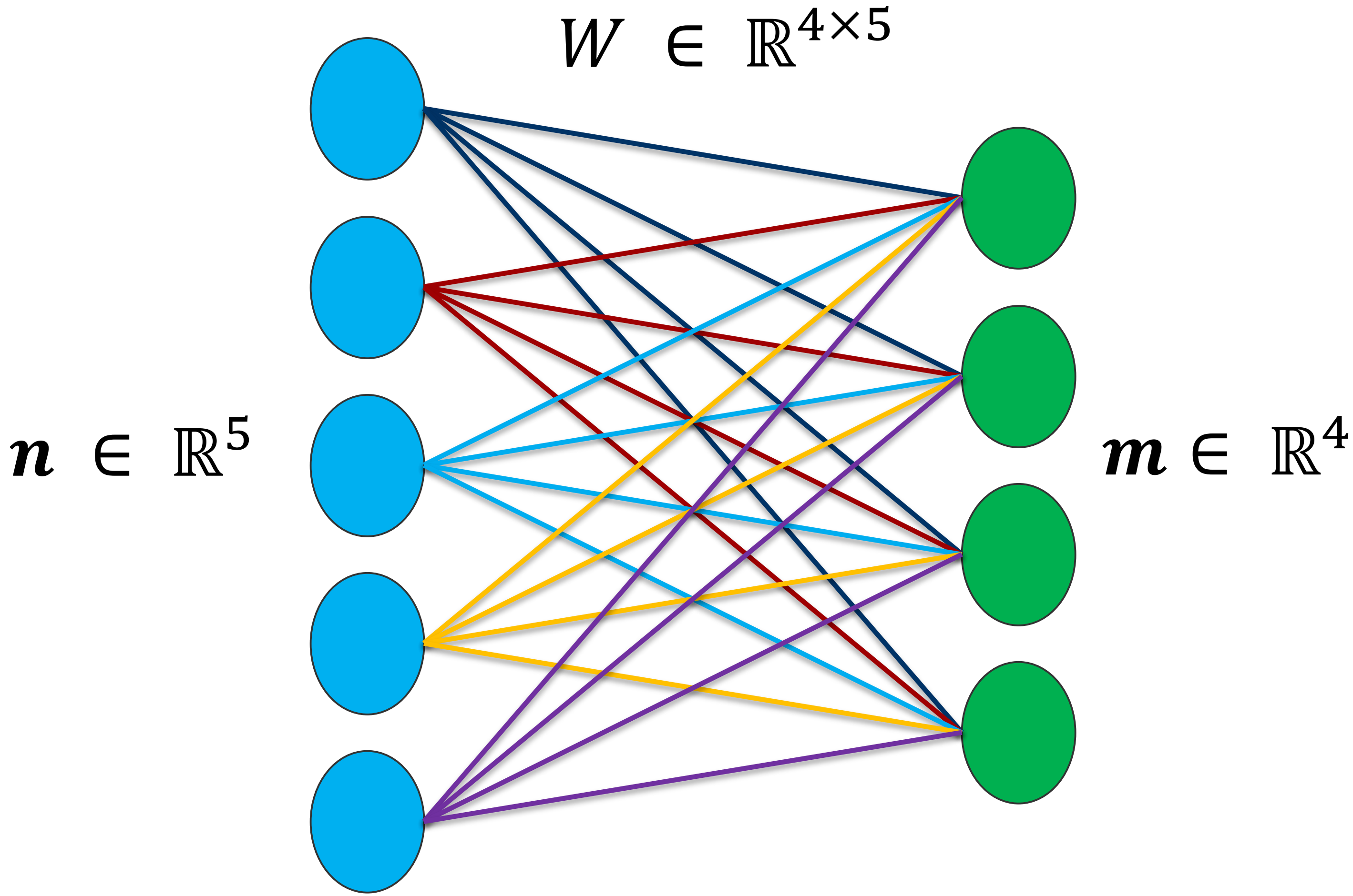
Building Blocks: **Fully Connected**

Building Blocks – Fully Connected

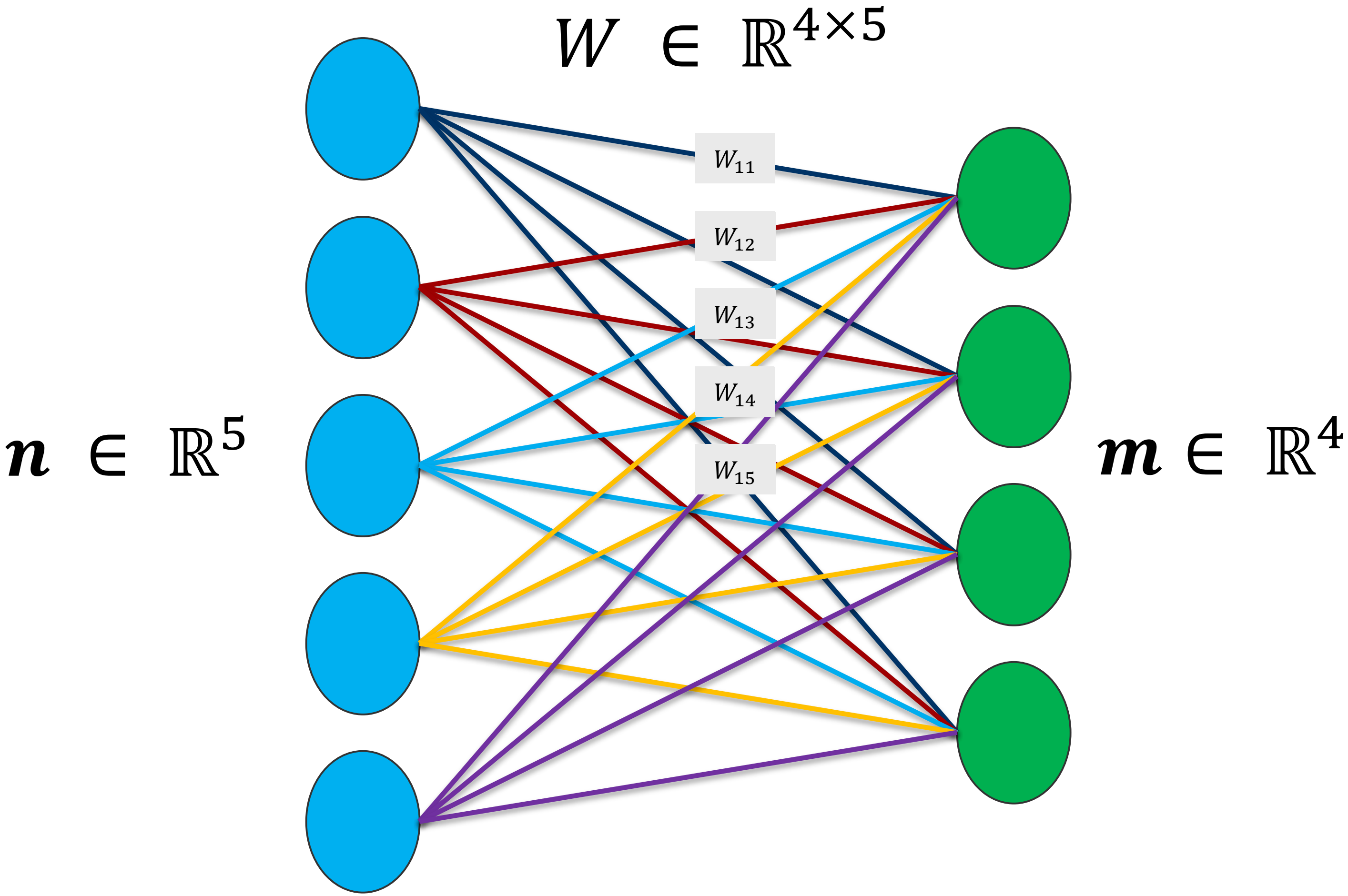


$\boldsymbol{n} \in \mathbb{R}^5$ $\boldsymbol{m} \in \mathbb{R}^4$

Building Blocks – Fully Connected



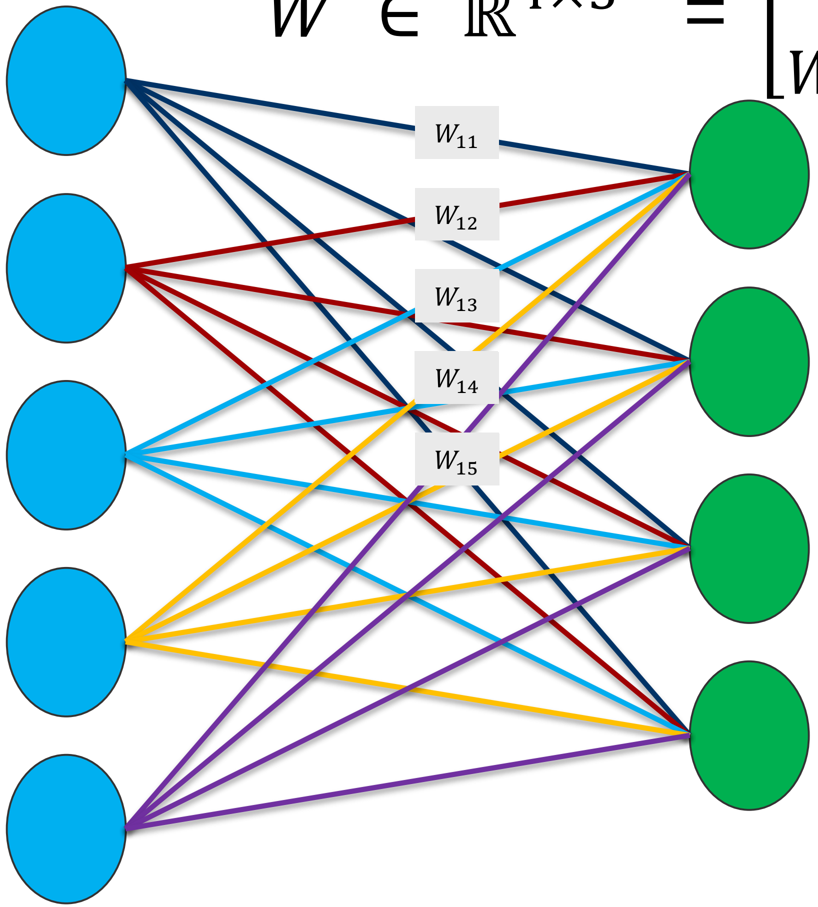
Building Blocks – Fully Connected



Building Blocks – Fully Connected

$$W \in \mathbb{R}^{4\times5} = \begin{bmatrix} W_{11} & \cdots & W_{15} \\ \vdots & \ddots & \vdots \\ W_{41} & \cdots & W_{45} \end{bmatrix}$$

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



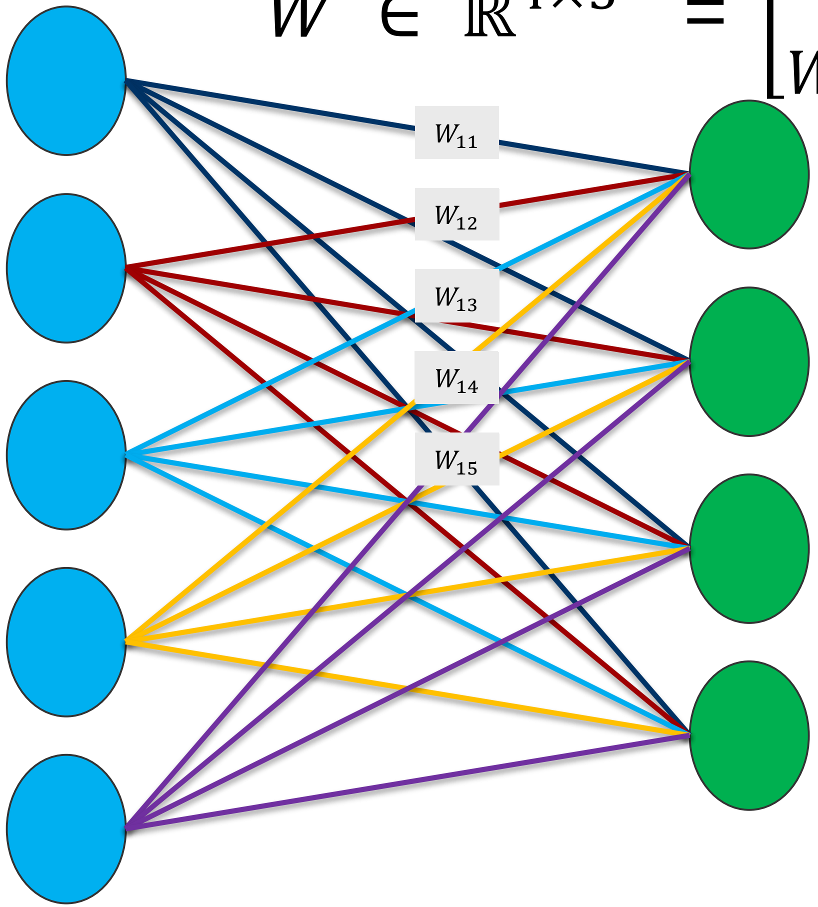
$\boldsymbol{m} \in \mathbb{R}^4$

Building Blocks – Fully Connected

$$W \in \mathbb{R}^{4\times 5} = \begin{bmatrix} W_{11} & \cdots & W_{15} \\ \vdots & \ddots & \vdots \\ W_{41} & \cdots & W_{45} \end{bmatrix}$$

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

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

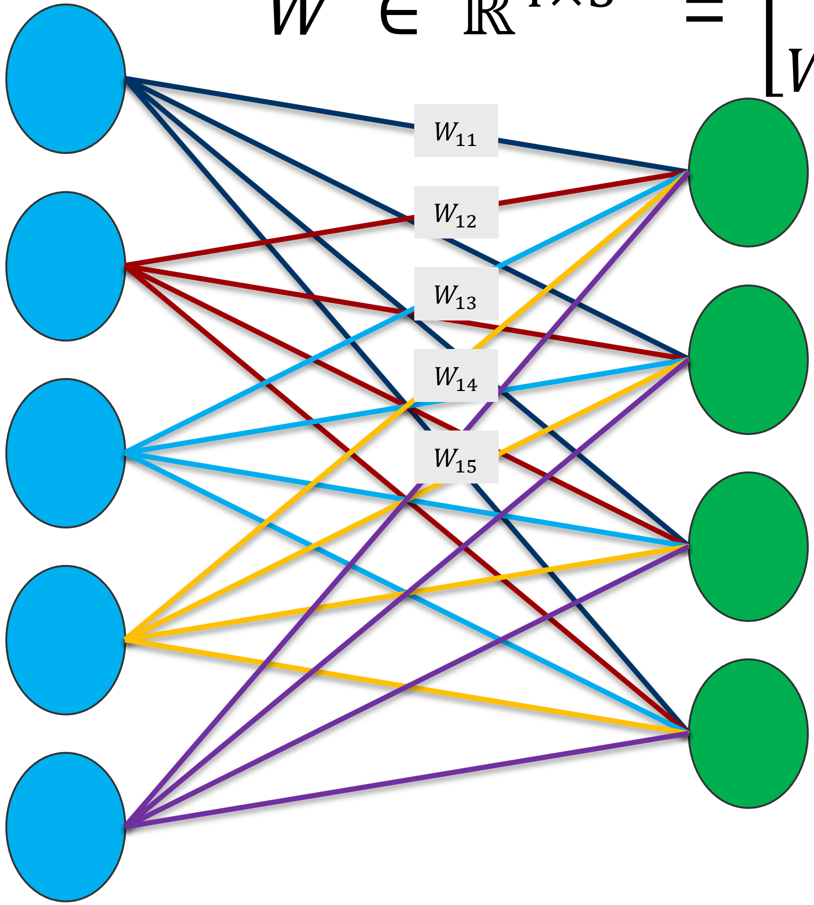


$\boldsymbol{m} \in \mathbb{R}^4$

Building Blocks – Fully Connected

$$W \in \mathbb{R}^{4\times5} = \begin{bmatrix} W_{11} & \cdots & W_{15} \\ \vdots & \ddots & \vdots \\ W_{41} & \cdots & W_{45} \end{bmatrix}$$

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



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

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

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

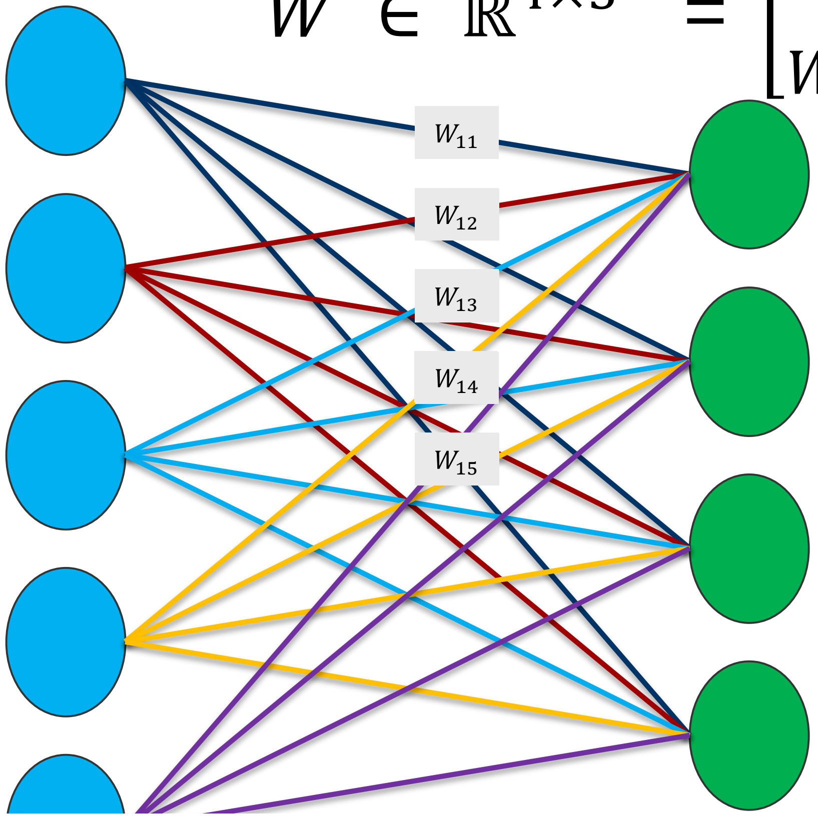
$m_3 = n_1 \times W_{31} + \ n_2 \times W_{32} + .. + \ n_5 \times W_{35}$

$m_4 = n_1 \times W_{41} + \ n_2 \times W_{24} + .. + \ n_5 \times W_{45}$

Building Blocks – Fully Connected

$$W \in \mathbb{R}^{4\times5} = \begin{bmatrix} W_{11} & \cdots & W_{15} \\ \vdots & \ddots & \vdots \\ W_{41} & \cdots & W_{45} \end{bmatrix}$$

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



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

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

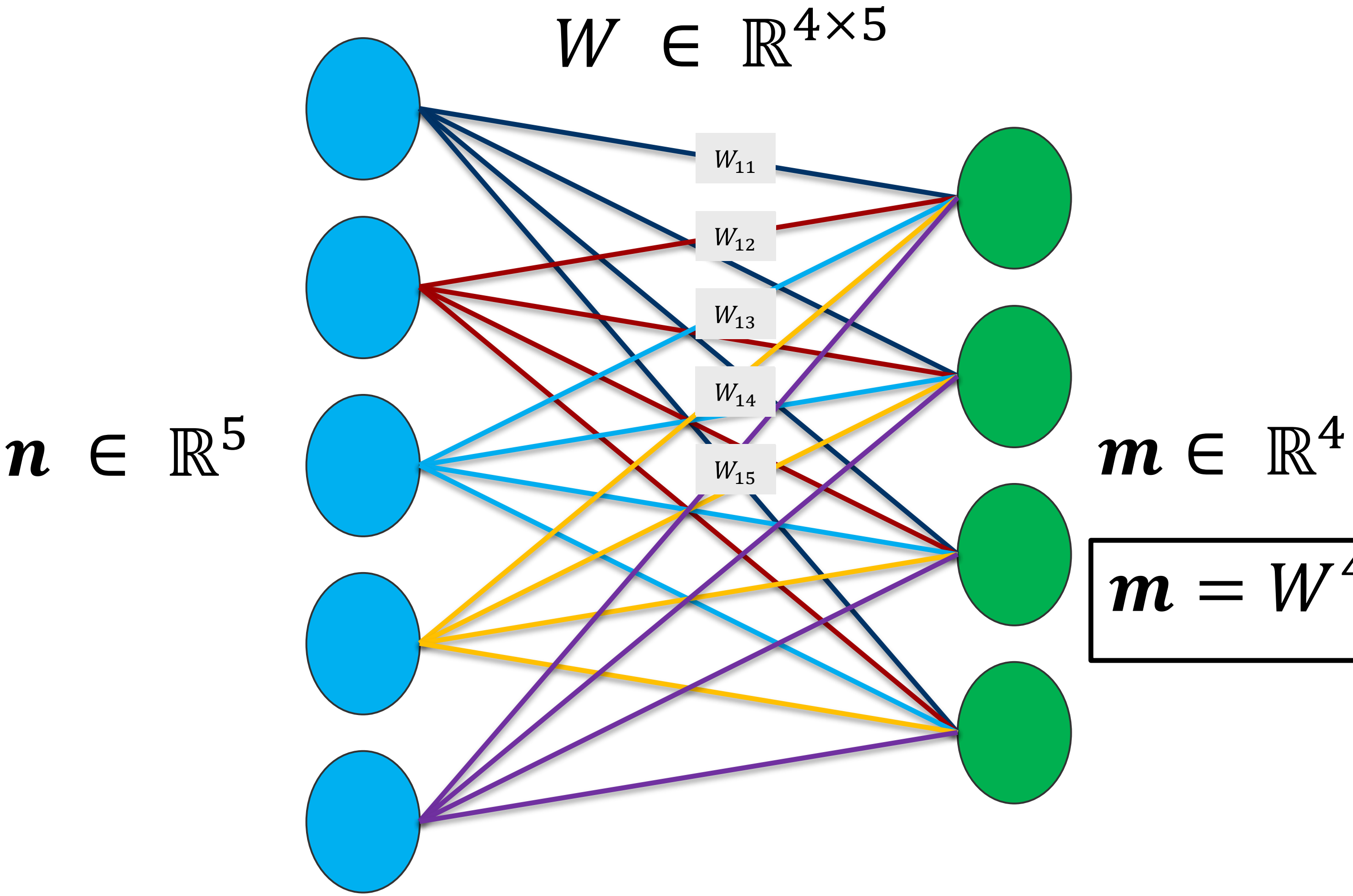
$$\boldsymbol{m} \in \mathbb{R}^4$$

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

$$m_4 = n_1 \times W_{41} + n_2 \times W_{24} + .. + n_5 \times W_{45}$$

Why is it called fully connected?

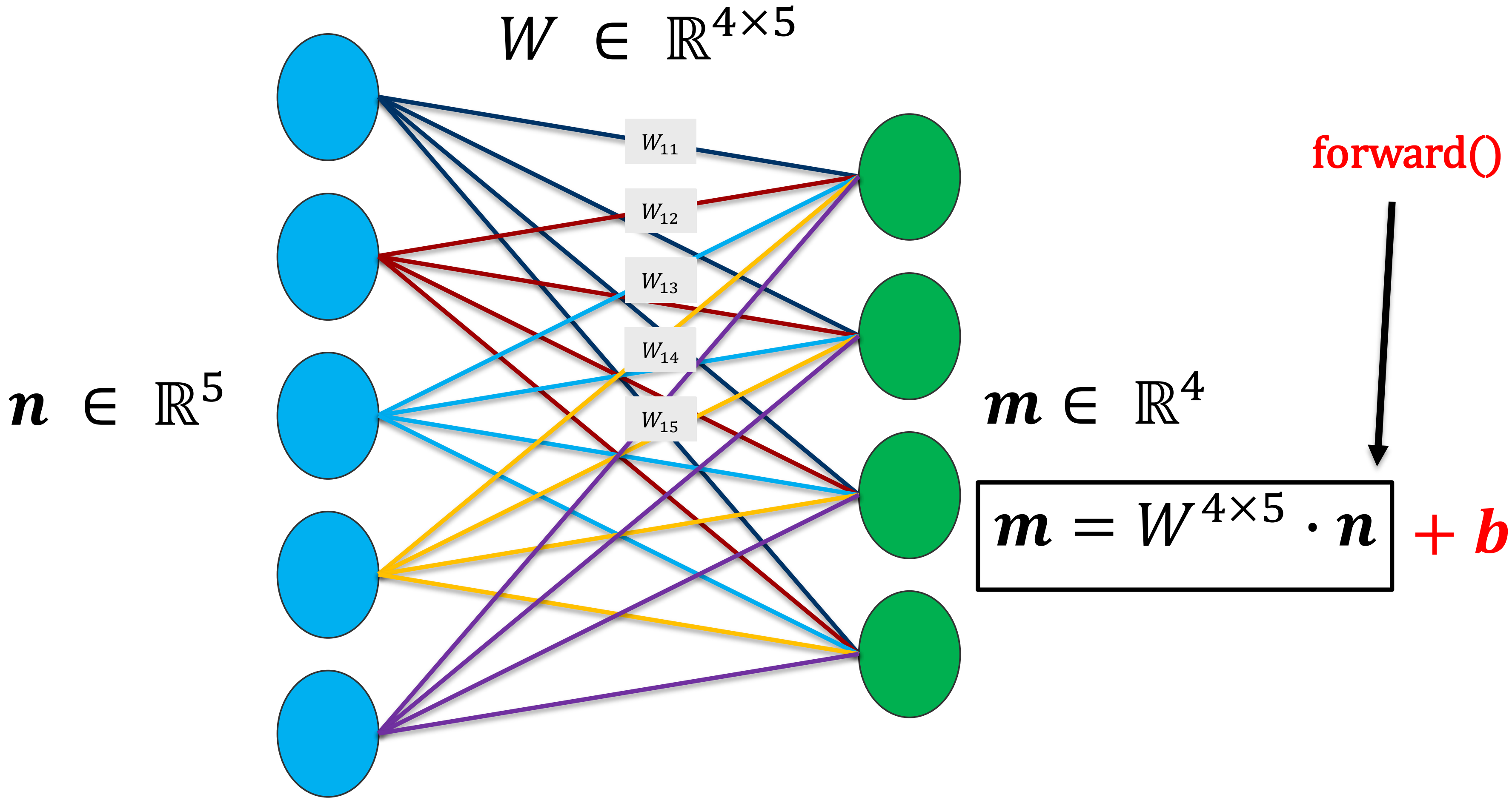
Building Blocks – Fully Connected



$$\boxed{\boldsymbol{m} = W^{4\times5} \cdot \boldsymbol{n}}$$

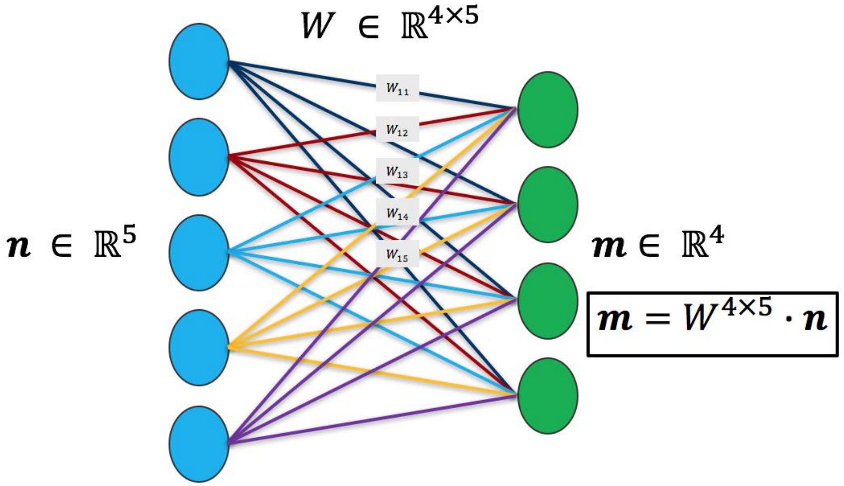
Building Blocks – Fully Connected

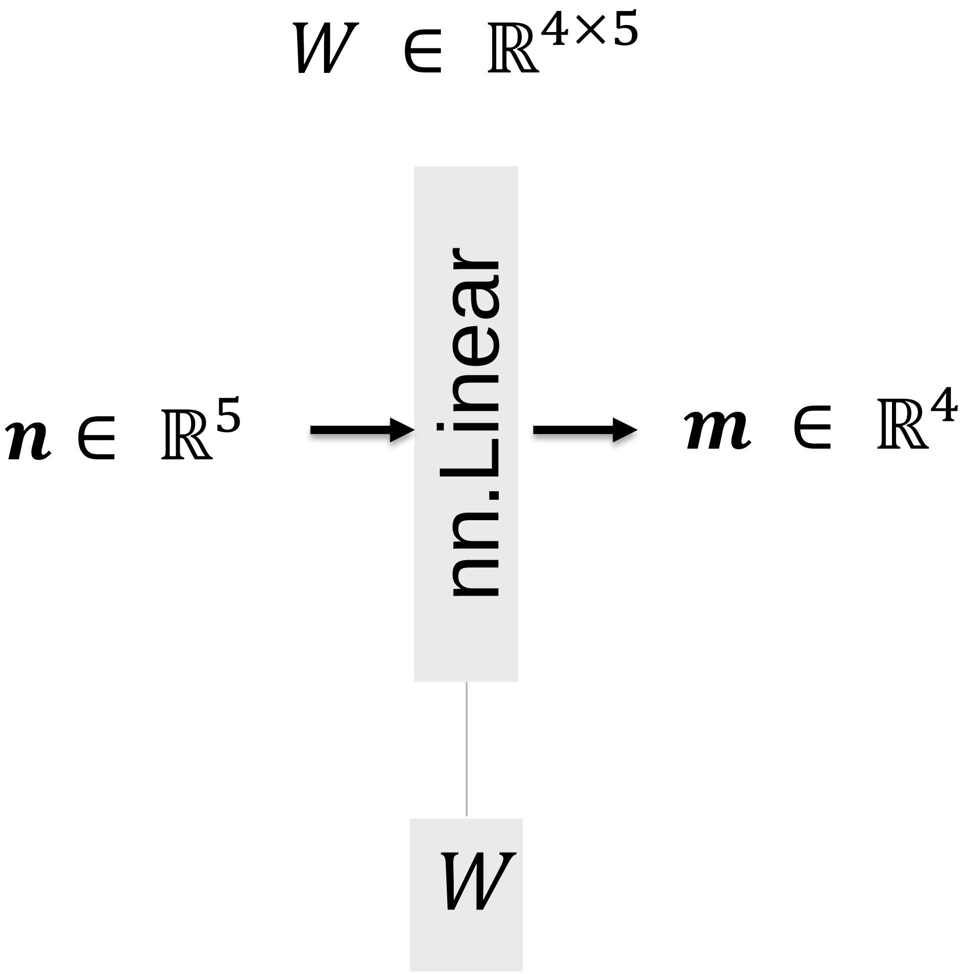
greatlearning
Learning for Life



$$\boxed{\boldsymbol{m} = W^{4\times5} \cdot \boldsymbol{n}} + \boldsymbol{b}^{4\times1}$$

Building Blocks – Fully Connected

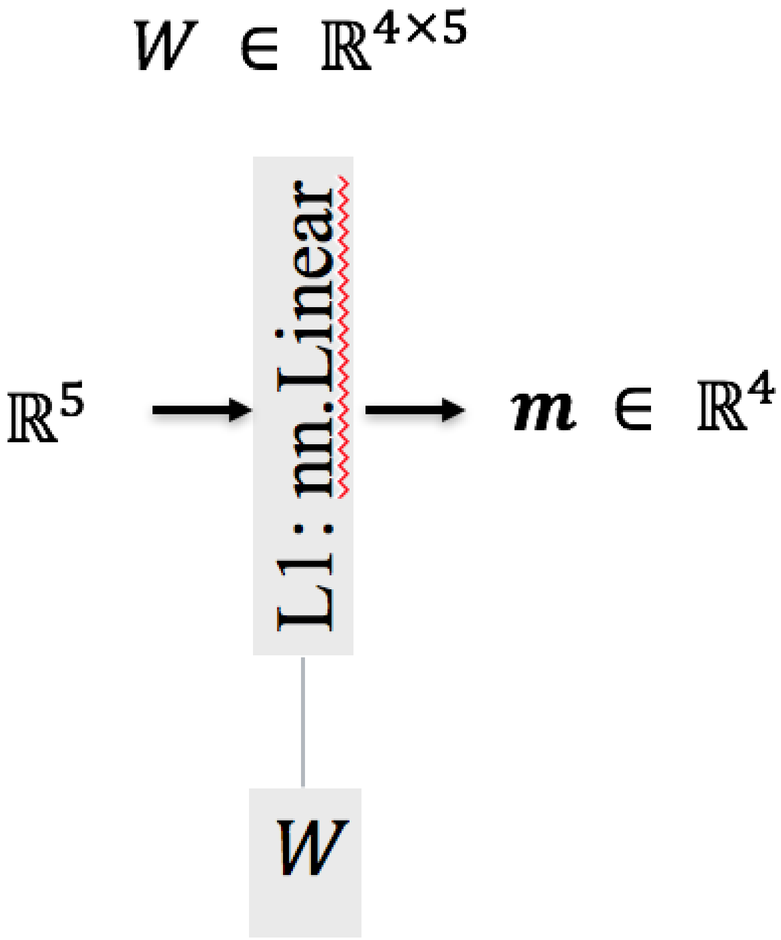




Building Blocks – Fully Connected – Forward

```
n
```

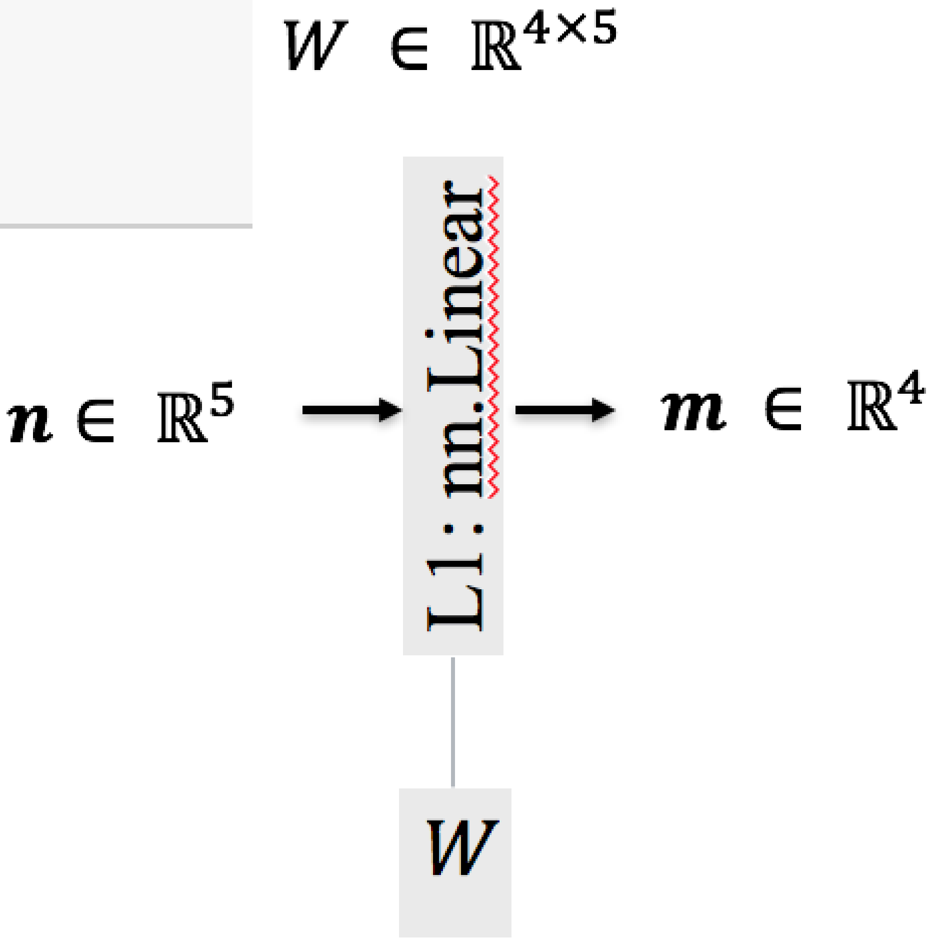
```
array([[0.91128205],
       [0.61846576],
       [0.61555748],
       [0.88395544],
       [0.64374925]])
```



```python
n = np.random.rand(5,1)
lin = Linear(5,4)
f = lin.forward(n)
```

```python
n
```

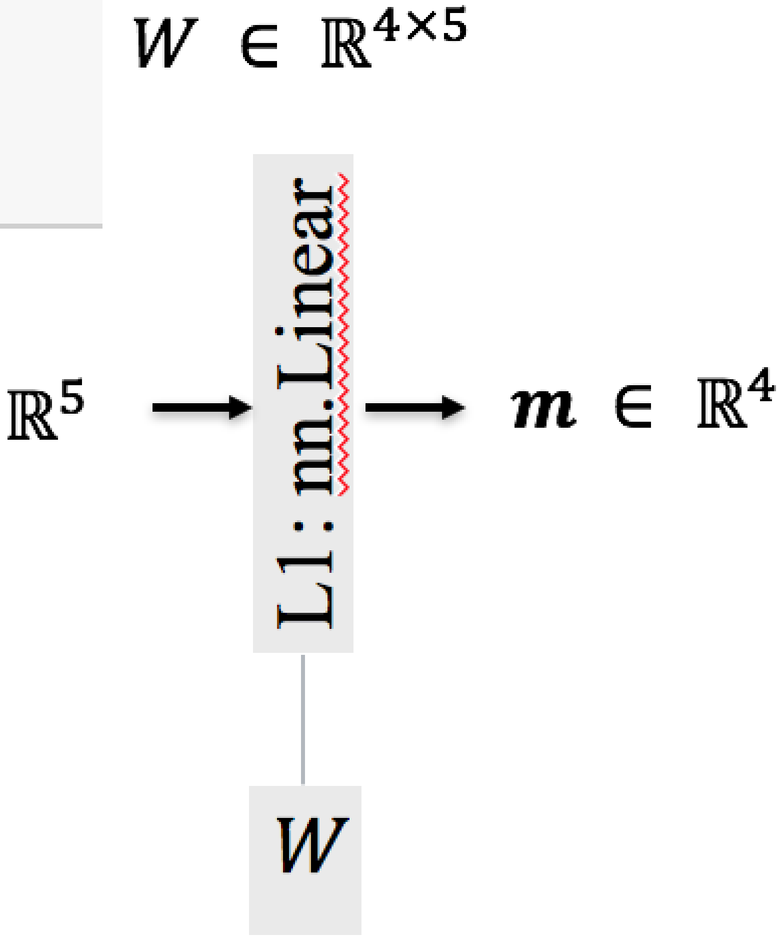
```
array([[0.91128205]
       [0.61846576]
       [0.61555748]
       [0.88395544]
       [0.64374925]
```



```python
n = np.random.rand(5,1)
lin = Linear(5,4)
f = lin.forward(n)
```

```python
n
```

```
array([[0.91128205],
       [0.61846576],
       [0.61555748],
       [0.88395544],
       [0.64374925]])
```




```python
m
```

```
array([[ 0.92942542],
       [-0.61528824],
       [ 1.84327823],
       [-3.16044288]])
```

```python
n = np.random.rand(5,1)
lin = Linear(5,4)
f = lin.forward(n)
```

```python
n
```

```
array([[0.91128205],
       [0.61846576],
       [0.61555748],
       [0.88395544],
       [0.64374925]])
```

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

$\mathbb{R}^5 \longrightarrow$ L1 : nn.Linear $\longrightarrow \boldsymbol{m} \in \mathbb{R}^4$

W

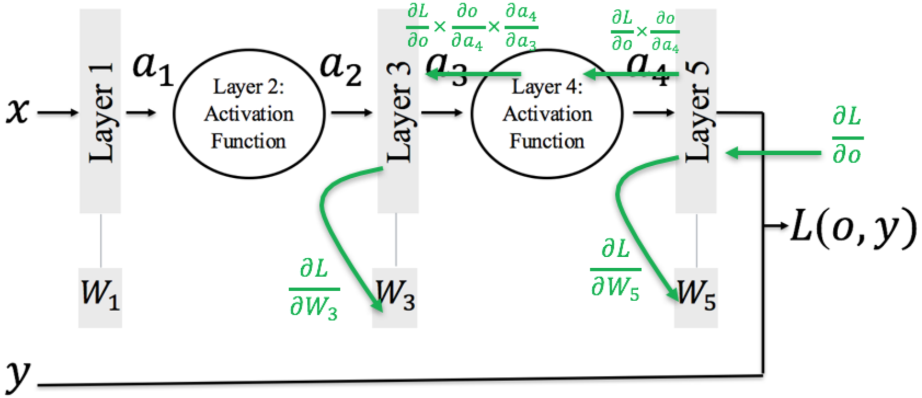
```python
m_ = np.matmul(self.w1.T, xi) + self.b
print(m_)
```

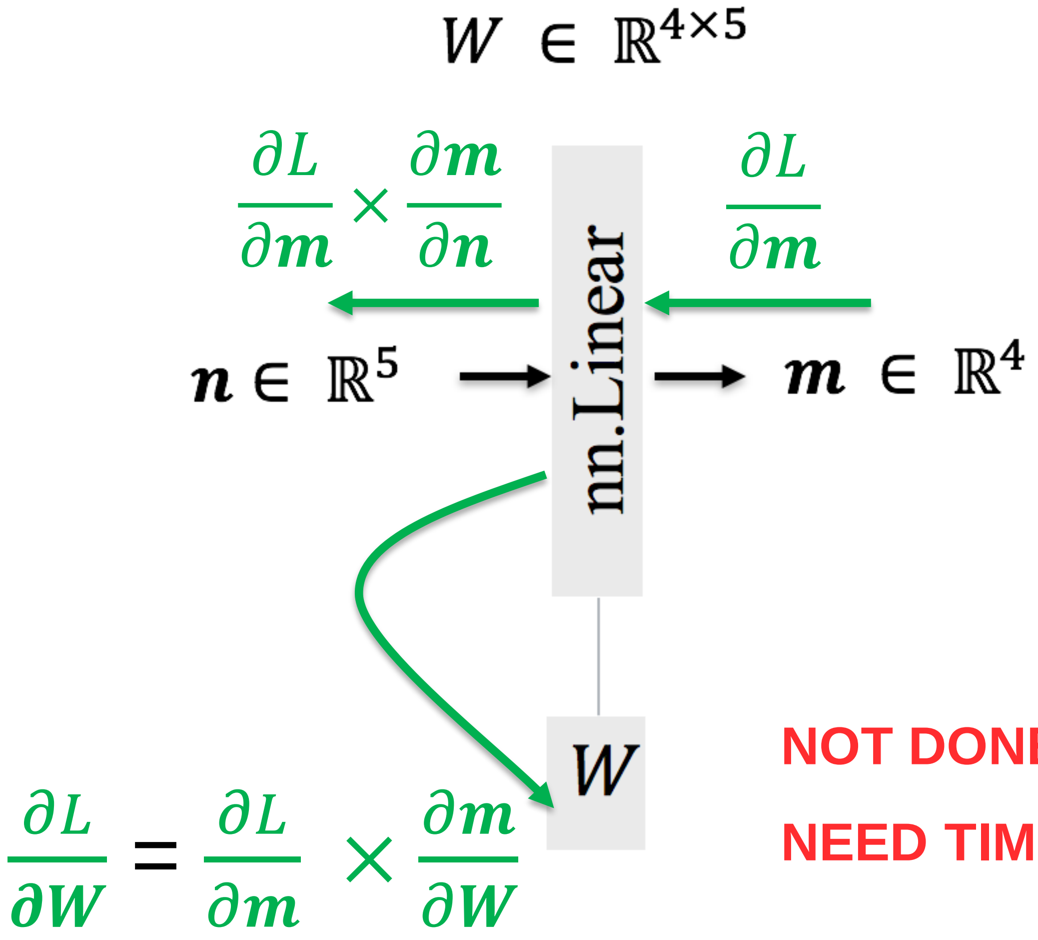
```
array([[ 0.92942542],
       [-0.61528824],
       [ 1.84327823],
       [-3.16044288]])
```

```python
m
```

```
array([[ 0.92942542],
       [-0.61528824],
       [ 1.84327823],
       [-3.16044288]])
```

Building Blocks – Fully Connected - Backward

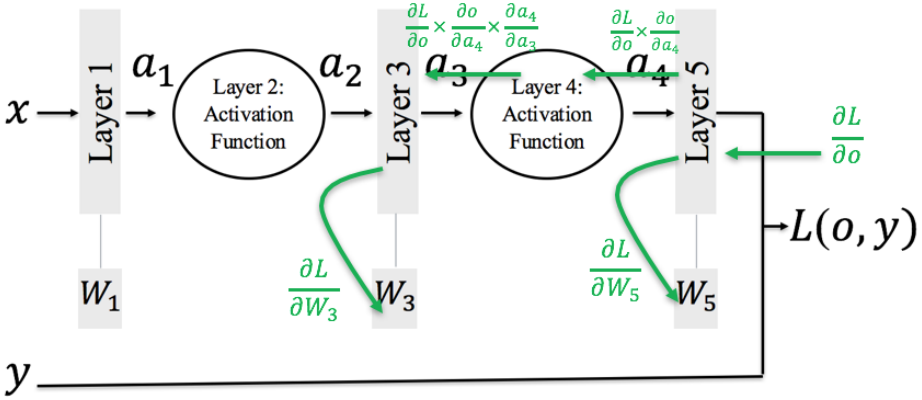




NOT DONE:

NEED TIME

Building Blocks – Fully Connected - Backward



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

Jacobian (Matrix) of size 4x5

Gradient (row vector 1x4)

$$\frac{\partial L}{\partial m} \times \frac{\partial m}{\partial n} \qquad \frac{\partial L}{\partial m}$$

$$\boldsymbol{n} \in \mathbb{R}^5 \longrightarrow \text{nn.Linear} \longrightarrow \boldsymbol{m} \in \mathbb{R}^4$$

W

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

Gradient (row vector 1x4) Jacobian (Matrix) of size 4x20

Fully Connected - Backward

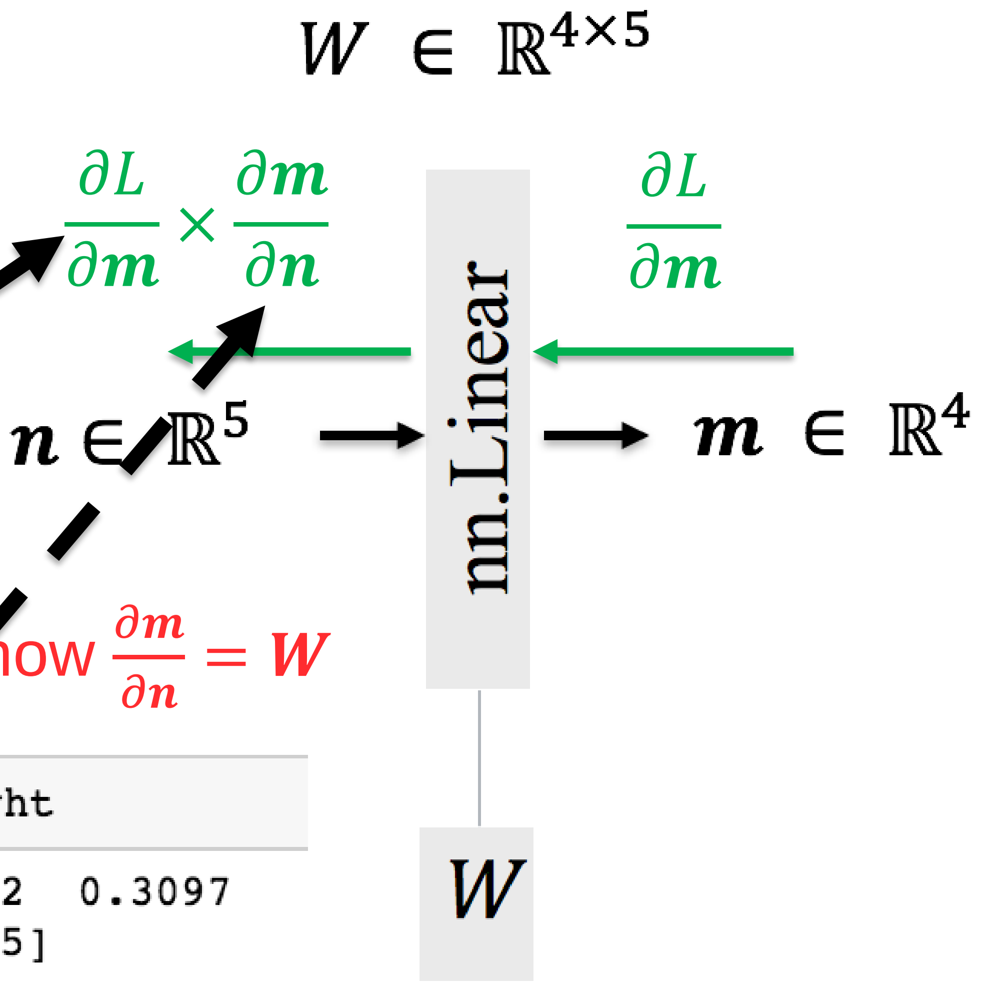

```
nextgrad = torch.rand(4)
lin:backward(n, nextgrad)
```

```
lin.gradInput
```

```
-0.1486
-0.3457
-0.0236
 0.2292
 0.3097
```

```
nextgrad:reshape(1,4)*lin.weight
```

```
-0.1486 -0.3457 -0.0236  0.2292  0.3097
[torch.DoubleTensor of size 1x5]
```



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

Fully Connected - Backward

```
nextgrad = torch.rand(4)
lin:backward(n, nextgrad)
```

```
lin.gradWeight
```

0.2135	0.6033	0.4343	0.1544	0.0673
0.0556	0.1572	0.1132	0.0402	0.0175
0.0850	0.2402	0.1729	0.0615	0.0268
0.1717	0.4850	0.3491	0.1242	0.0541

[torch.DoubleTensor of size 4x5]

```
(nextgrad:reshape(1,4) * dodw):reshape(4,5)
```

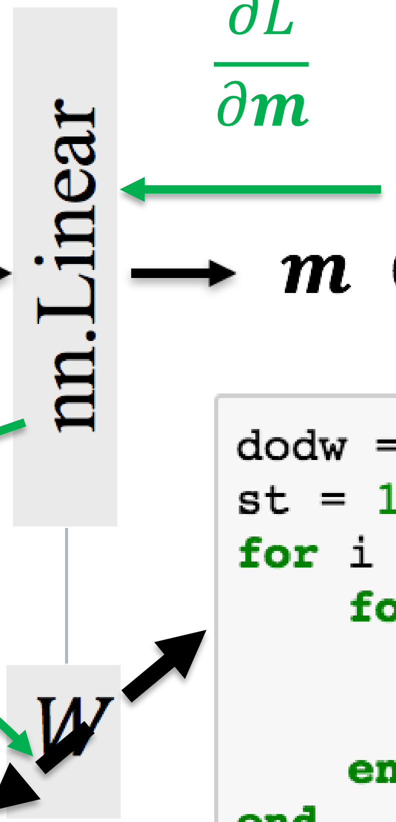
0.2135	0.6033	0.4343	0.1544	0.0673
0.0556	0.1572	0.1132	0.0402	0.0175
0.0850	0.2402	0.1729	0.0615	0.0268
0.1717	0.4850	0.3491	0.1242	0.0541

[torch.DoubleTensor of size 4x5]

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

$$\frac{\partial L}{\partial m}$$

$$\boldsymbol{n} \in \mathbb{R}^5 \rightarrow \text{nn.Linear} \rightarrow \boldsymbol{m} \in \mathbb{R}^4$$




```
dodw = torch.Tensor(4,20)
st = 1
for i = 1, 4 do
    for j = 1, 5 do
        dodw[i][st]=n[j]
        st = st + 1
    end
end
```

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

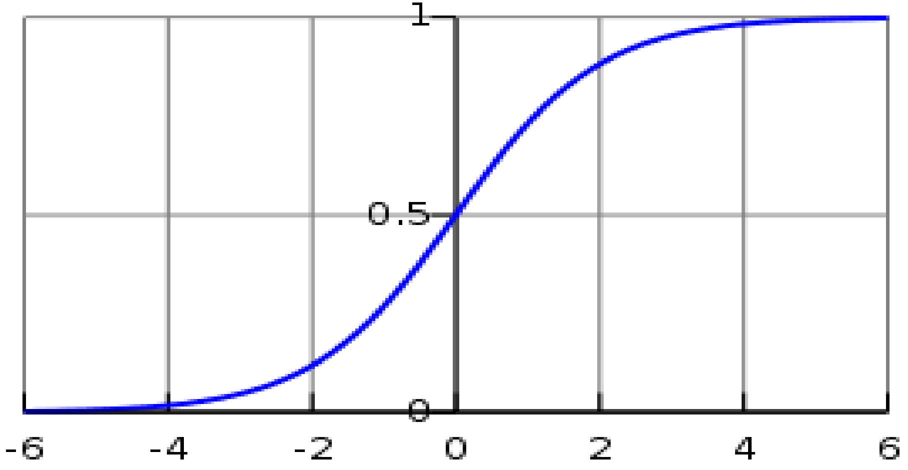
Building Blocks:
Activation Functions

Activation Functions: **Sigmoid**

- Activation function of 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

Drawbacks:

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

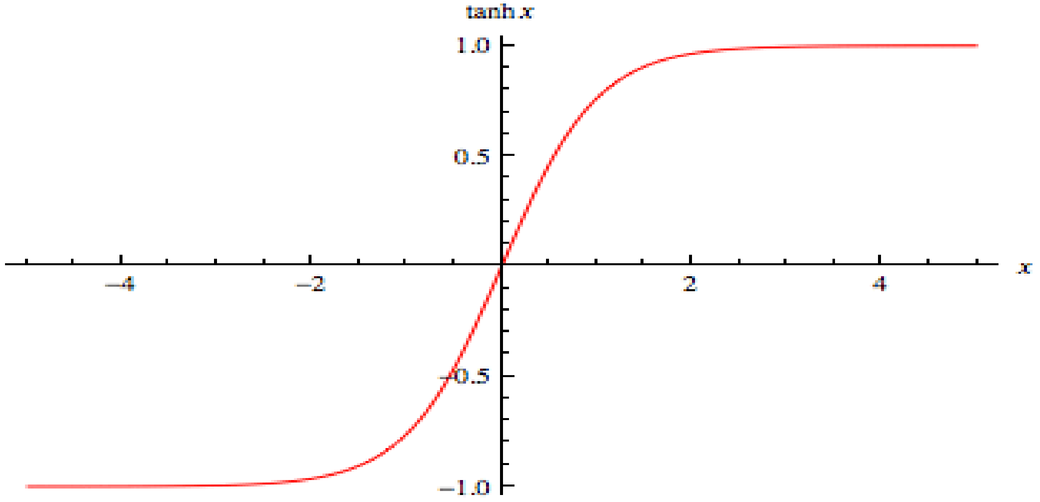
Image sourced from: DeepLearning.net

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

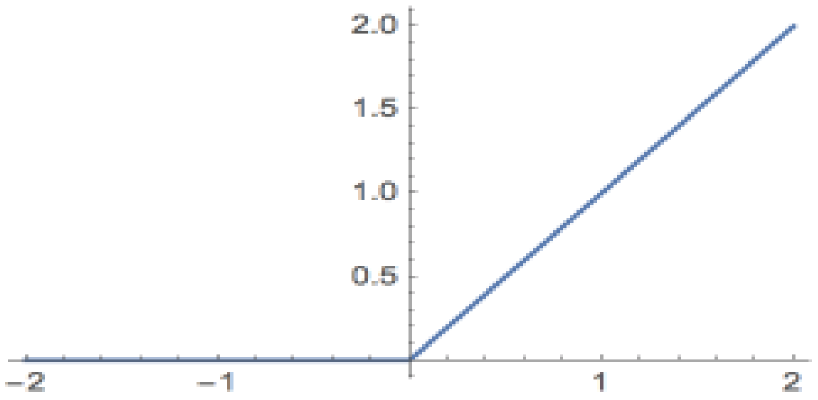
Image sourced from: Wolfram Alpha

Activation Functions: **ReLU**

- Very simple and efficient
- Have 6x times better convergence than tanh and sigmoid function.
- Very efficient in computation

Drawbacks:

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



ReLU

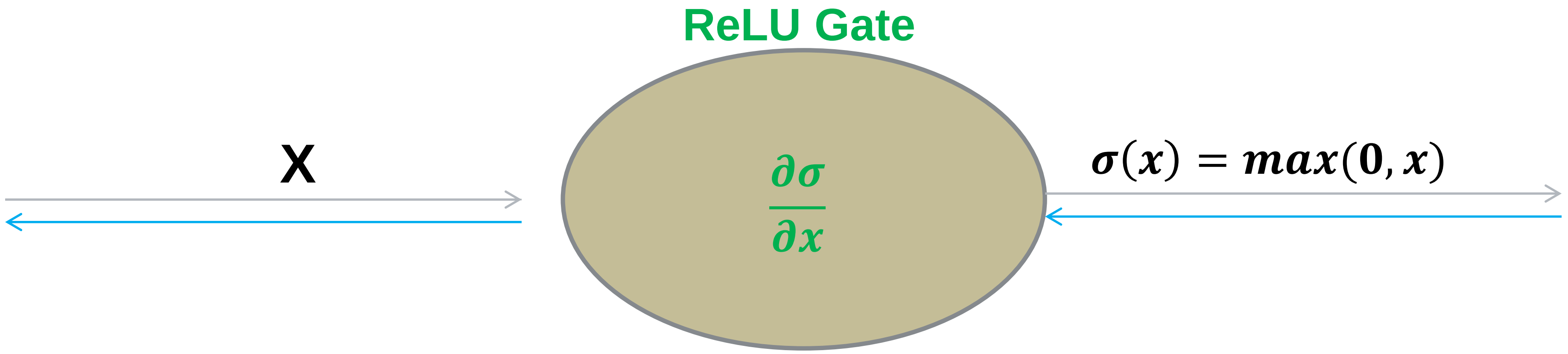
Image sourced from: Wolfram Alpha Documentation

Activation Functions: **ReLU Problems**

- Some of the gradients can be fragile during training and can die
- Results in weight update, and could result in never activating on any data point again
- ReLU could result in dead neurons

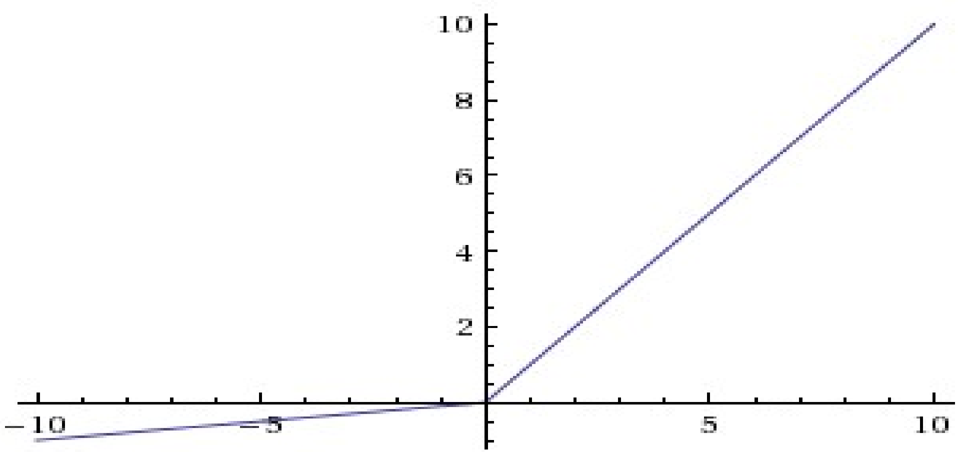
Scenario:

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



Activation Functions: **Leaky ReLU**

- Leaky ReLU was introduced to overcome the problem of dying -   neurons.
- Leaky ReLU introduces a small slope to keep the neurons alive
- Does not saturate (in +region)

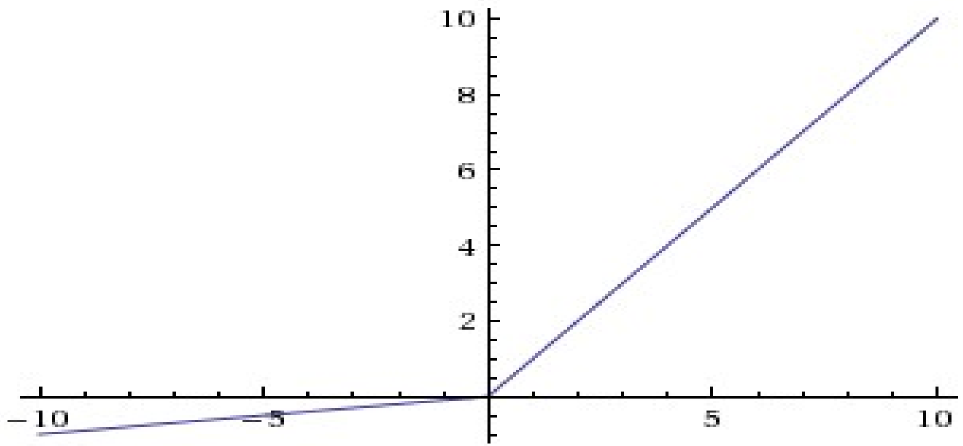


Leaky ReLU

Image sourced from: Wolfram Alpha Documentation

Activation Functions: **Leaky ReLU**

- Leaky ReLU was introduced to overcome the problem of dying - neurons.
- Leaky ReLU introduces a small slope to keep the neurons alive
- Does not saturate (in +region)



Leaky ReLU

Back Propagate into

Parametric Rectifier PReLU

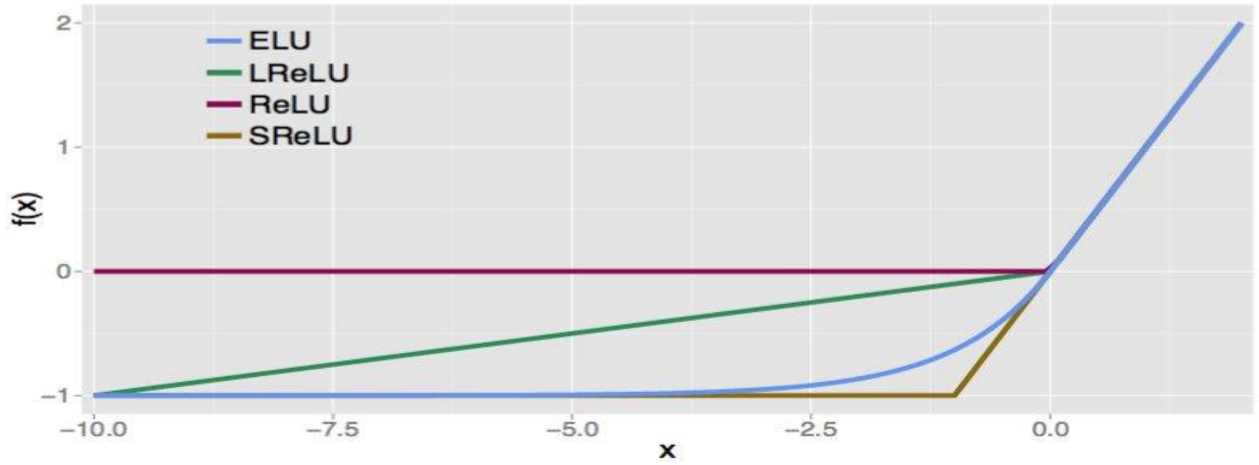
Image sourced from: Wolfram Alpha Documentation

Activation Functions: **ELU**

- ELU function tend to converge cost to zero faster and produce more accurate results
- Closer to zero mean outputs
- Has a extra alpha constant which should be positive number
- ELU is very similar to RELU except negative inputs
- Have all advantages of ReLU

Drawback:
Computation requires exp()

(identity function)



Exponential Linear Units (ELU)

Image sourced from: Fast and Accurate Deep Network Learning by Exponential Linear Units (ELU), Clevert et al., 2015

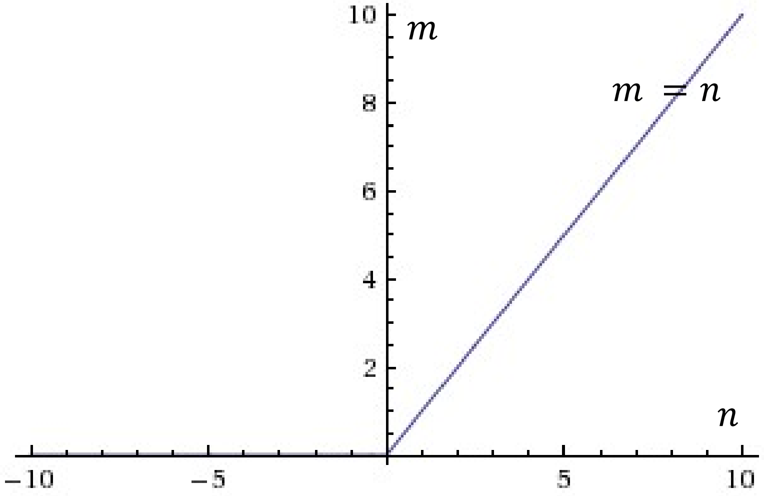
In Practice, what type of neuron should one use?

- Use **ReLU** non-linearity but be careful with the learning rates and don’t forget to monitor the fraction of “**dead**” units in your network
- Give Leaky ReLU or Maxout a try
- Never use sigmoid
- Try tanh, but expect worse performance than ReLU

Activation Function: ReLU (details)

$\boldsymbol{n} \in \mathbb{R}^d \longrightarrow$ nn.ReLU $\longrightarrow \boldsymbol{m} \in \mathbb{R}^d$

$$m_i = \max(0, n_i)$$



$$m_i = \begin{cases} 0 & if \;\; n_i < 0 \\ n_i & if \;\; n_i > 0 \end{cases}$$

Activation Function: ReLU (details)

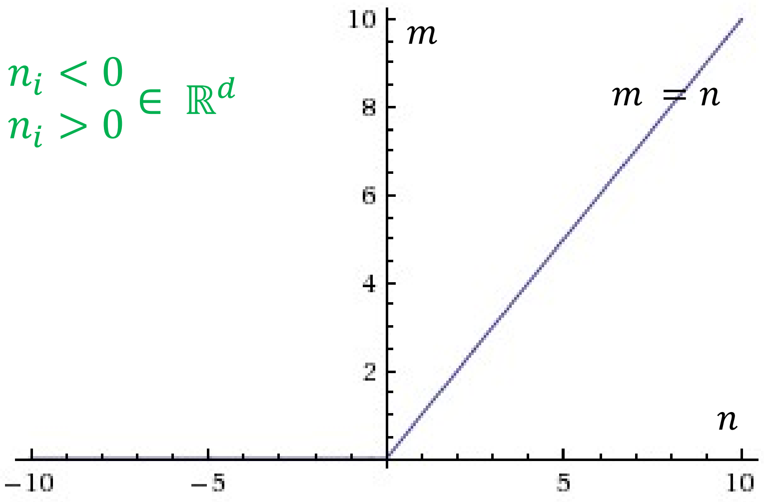
$$\frac{\partial L}{\partial \boldsymbol{m}} \cdot \frac{\partial \boldsymbol{m}}{\partial \boldsymbol{n}} \in \mathbb{R}^{1\times \dim(n)}$$

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

$$\boldsymbol{n} \in \mathbb{R}^d \rightarrow \text{nn.ReLU} \rightarrow \boldsymbol{m} \in \mathbb{R}^d$$

$$m_i = \max(0, n_i)$$

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



$$m_i = \begin{cases} 0 & if \;\; n_i < 0 \\ n_i & if \;\; n_i > 0 \end{cases}$$

Activation Function: ReLU (forward)

$\boldsymbol{n} \in \mathbb{R}^d \longrightarrow$ nn.ReLU $\longrightarrow \boldsymbol{m} \in \mathbb{R}^d$

```
n = np.random.rand(5,1) - 0.5
print(n)
```

```
[[ 0.05130641]
 [ 0.11932188]
 [-0.2101456 ]
 [-0.19987061]
 [ 0.09812002]]
```

```
m = tf.nn.relu(n)
print(m.eval())
```

```
[[0.05130641]
 [0.11932188]
 [0.        ]
 [0.        ]
 [0.09812002]]
```

Activation Function: ReLU (backward)

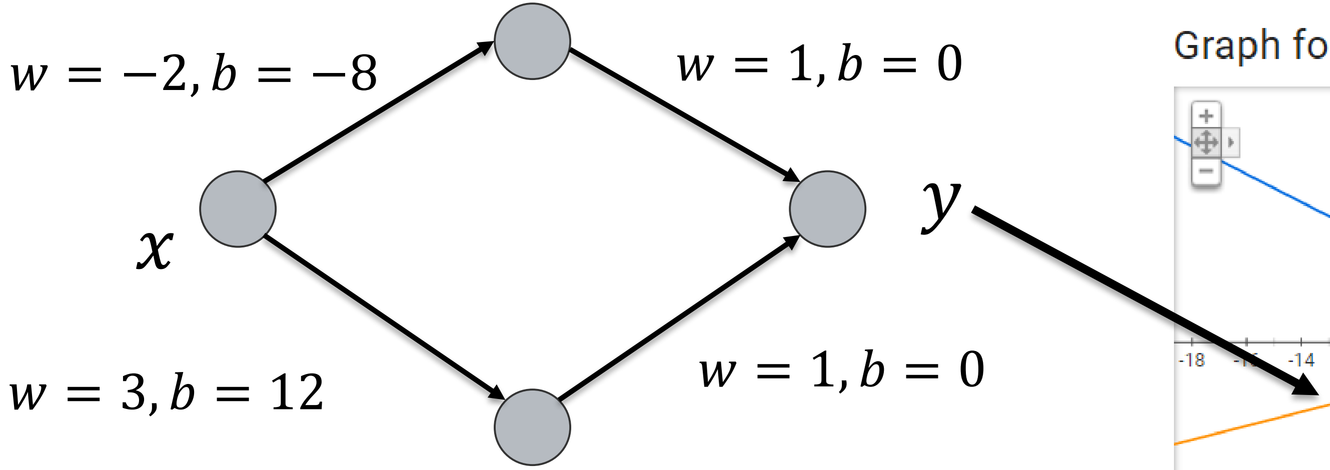
$$\frac{\partial L}{\partial \boldsymbol{m}} \cdot \frac{\partial \boldsymbol{m}}{\partial \boldsymbol{n}} \in \mathbb{R}^{1\times \dim(n)} \qquad \frac{\partial L}{\partial \boldsymbol{m}} \in \mathbb{R}^{1\times \dim(m)}$$

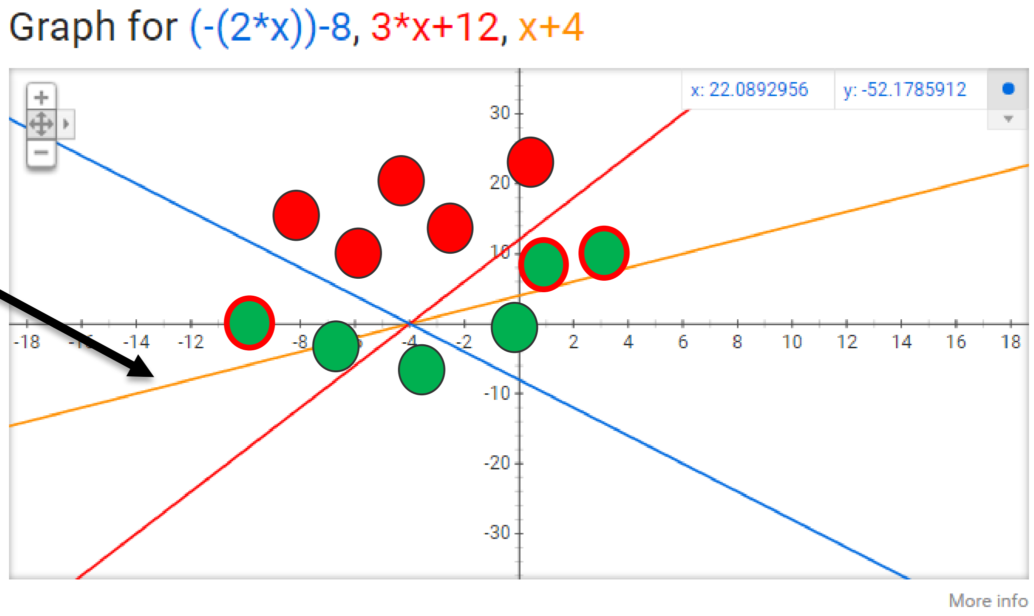
$$\boldsymbol{n} \in \mathbb{R}^d \longrightarrow \boxed{\text{nn.ReLU}} \longrightarrow \boldsymbol{m} \in \mathbb{R}^d$$

```
nextgrad=torch.ones(5)
relu:backward(n, nextgrad)
print(relu.gradInput)
```

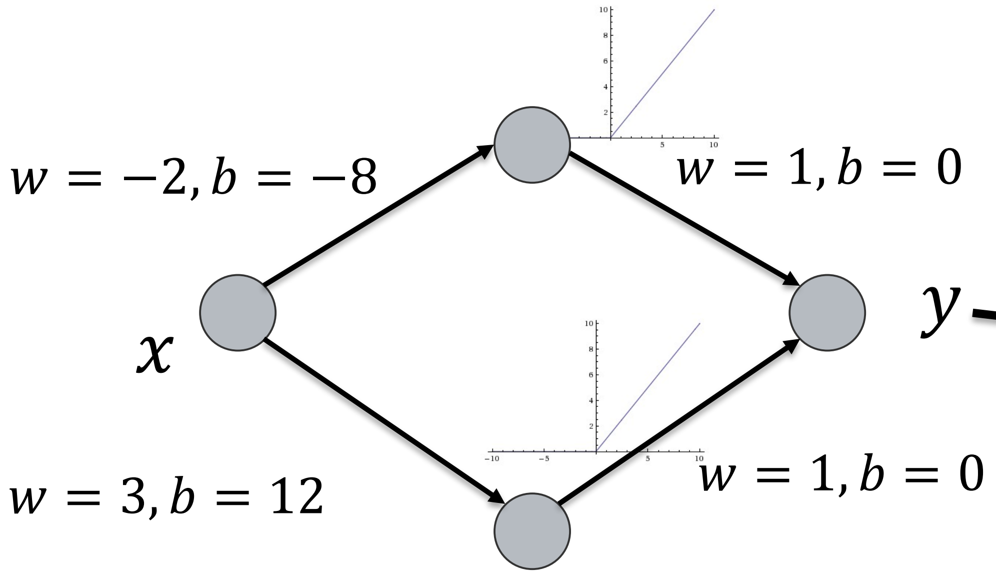
```
 0
 0
 1
 0
 1
[torch.DoubleTensor of size 5]
```

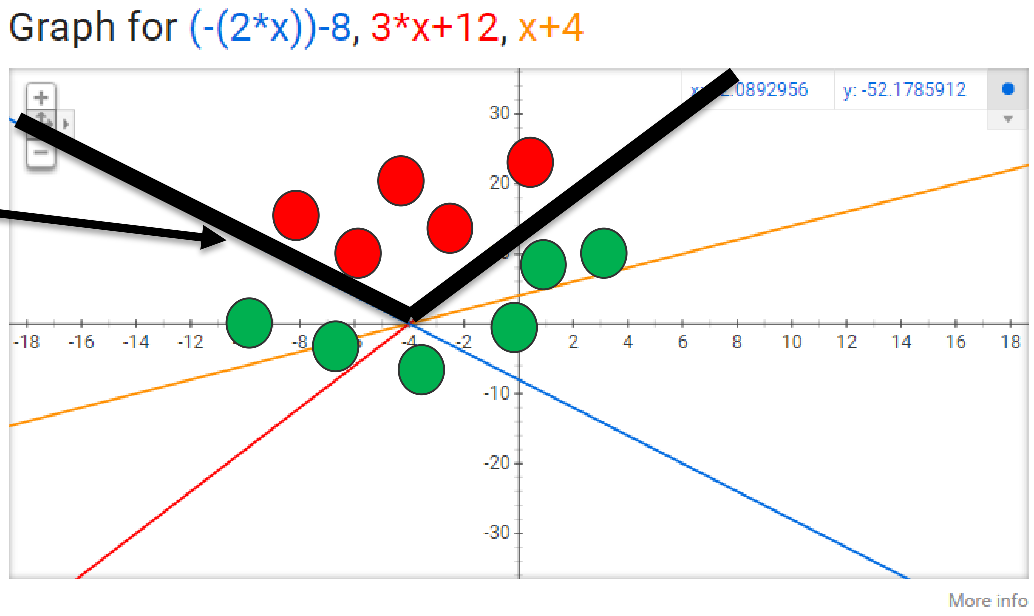
Building Blocks – ReLU



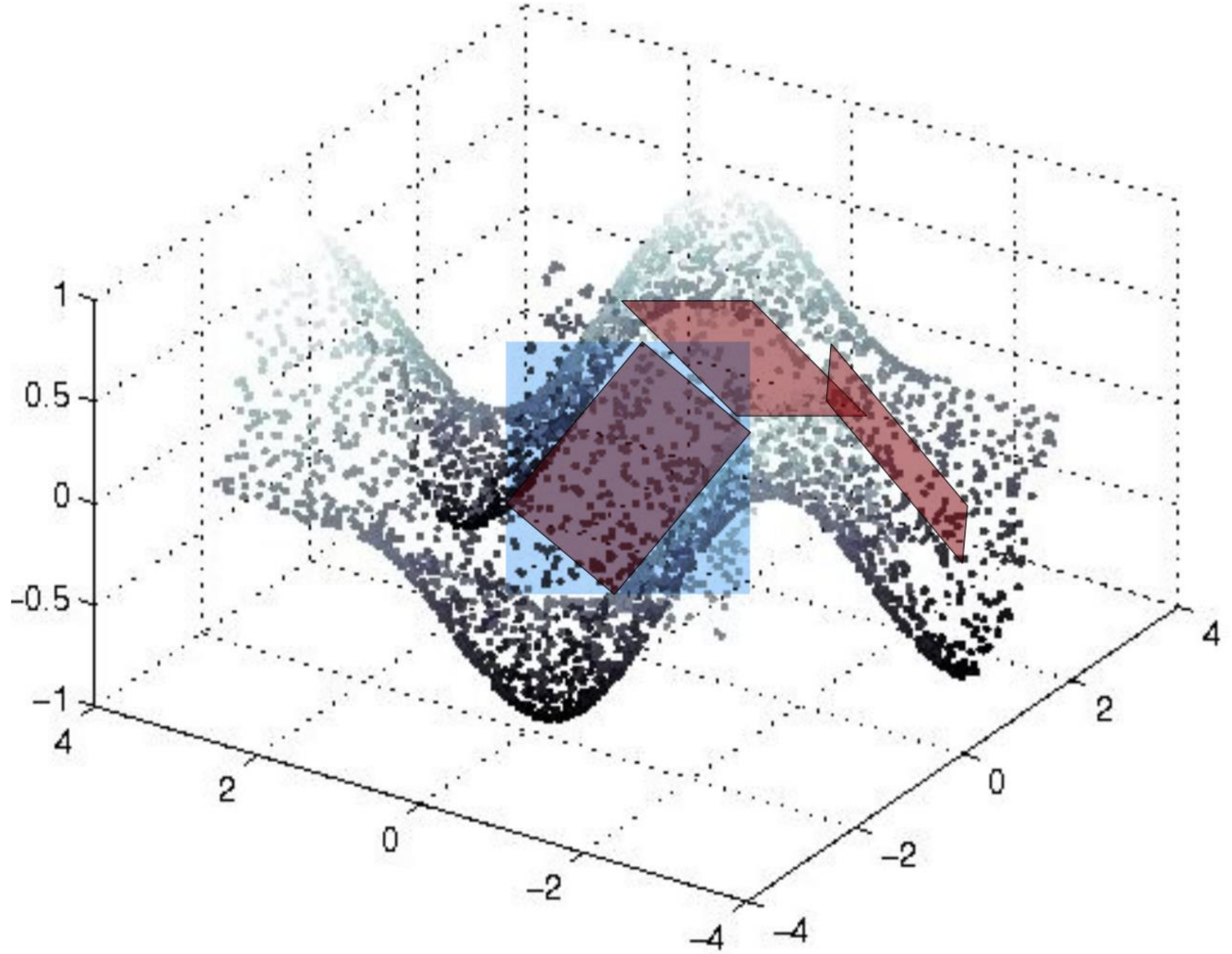


Building Blocks – ReLU

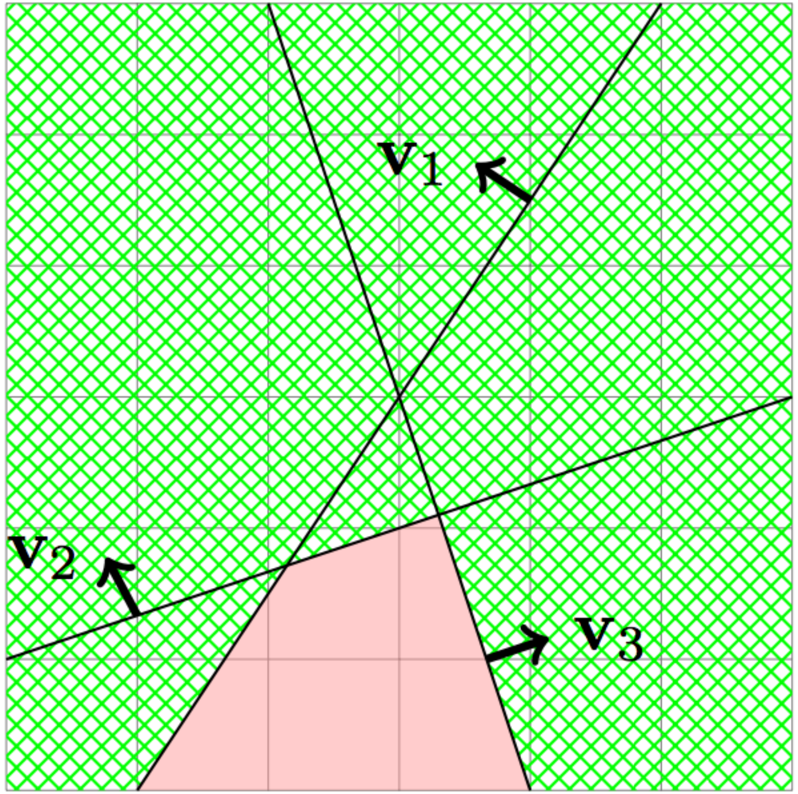




Building Blocks – ReLU



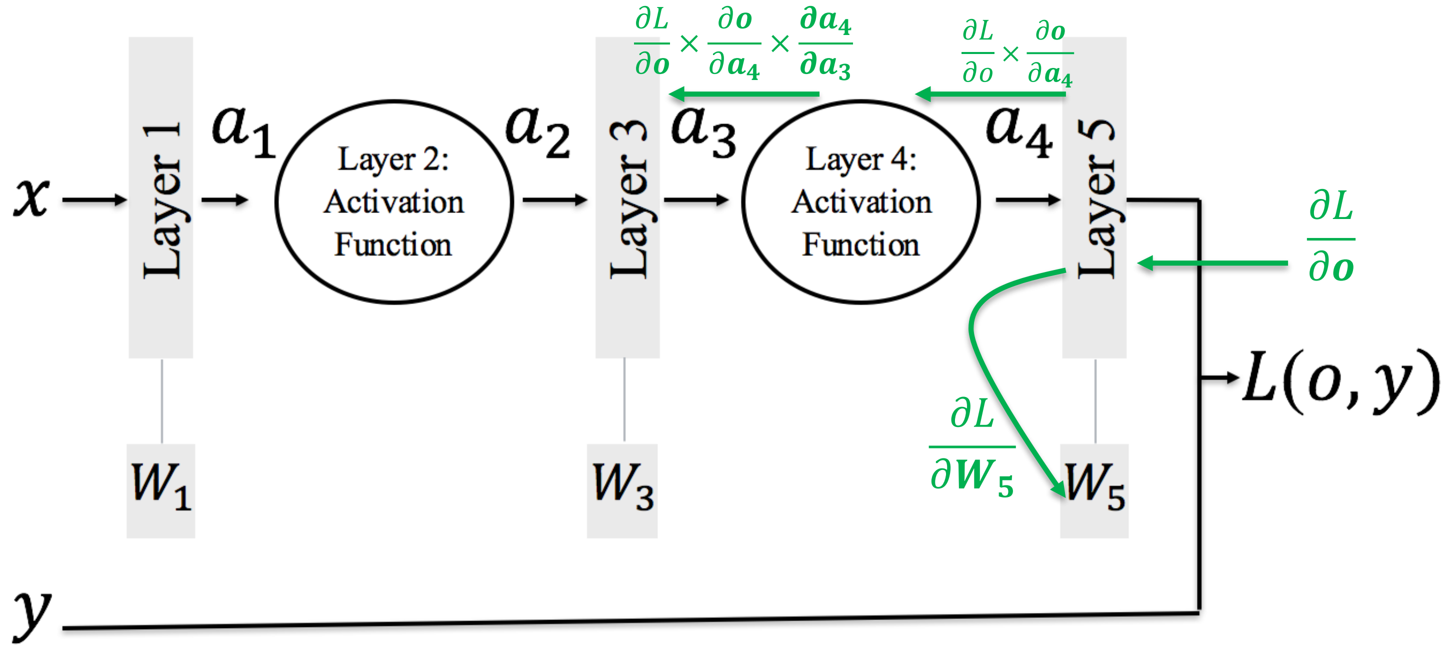
Building Blocks – ReLU



- Each hidden unit represents one hyperplane (parameterized by weight and bias) that bisects the input space into two half spaces.
- By choosing different weights in the hidden layer we can obtain arbitrary arrangement of n hyperplanes.
- The theory of hyperplane arrangement (Zaslavsky, 1975) tells us that for a general arrangement of n hyperplanes in d dimensions, the space is divided into $\sum_{s=0}^{d}\binom{n}{s}$ regions.

Sourced from: Expressiveness of Rectifier Networks Xingyuan Pan, Vivek Srikumar et al., 2016]

Vanishing/Exploding Gradients



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

Advice: Understand the engineering behind at least ONE framewor

Theano: A Python framework for fast computation of mathematical expressions

(The Theano Development Team)*

Orhan Firat,[1,23] Mathieu Germain,[1] Xavier Glorot,[1,18] Ian Goodfellow,[1,24] Matt Graham,[25] Caglar Gulcehre,[1]
Philippe Hamel,[1] Iban Harlouchet,[1] Jean-Philippe Heng,[1,26] Balázs Hidasi,[27] Sina Honari,[1] Arjun Jain,[28]
Sébastien Jean,[1,11] Kai Jia,[29] Mikhail Korobov,[30] Vivek Kulkarni,[6] Alex Lamb,[1] Pascal Lamblin,[1] Eric Larsen,[1,31]
César Laurent,[1] Sean Lee,[17] Simon Lefrancois,[1] Simon Lemieux,[1] Nicholas Léonard,[1] Zhouhan Lin,[1]
Jesse A. Livezey,[32] Cory Lorenz,[33] Jeremiah Lowin, Qianli Ma,[34] Pierre-Antoine Manzagol,[1] Olivier Mastropietro,[1]

[26] *Meiji University, Tokyo, Japan*
[27] *Gravity R&D*
[28] *Indian Institute of Technology, Bombay, India*
[29] *Megvii Technology Inc.*
[30] *ScrapingHub Inc.*
[31] *CIRRELT and Département d'informatique et recherche opérationnelle, Université de Montréal, QC, Canada*

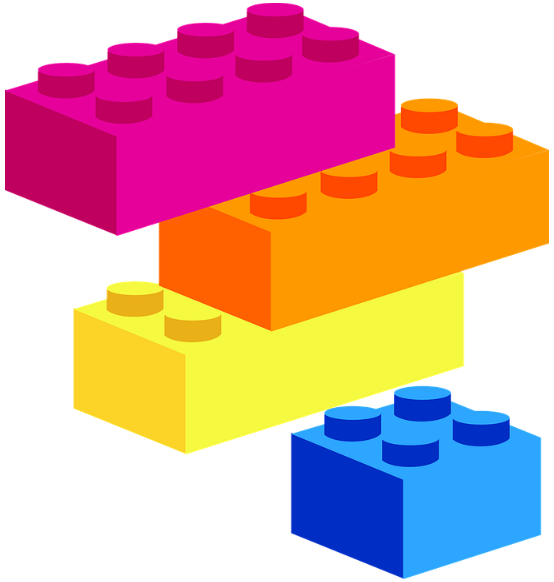
https://github.com/torch/cunn/blob/master/lib/THCUNN/VolumetricConvolution.cu

```
// Kernel for fast unfold+copy
// Borrowed from Theano
// Authors: Arjun Jain, Frédéric Bastien, Jan Schlüter, Nicolas Ballas
template <typename Dtype>
__global__ void im3d2col_kernel(const int n, const Dtype* data_im,
                                const int height, const int width, const int depth,
```

Do you want to use this when talking about DL frameworks?

In Summary: Deep Neural Networks - Building Blocks

- Forward Propagation
- Backward Propagation
- Activation Layers (ReLU, Sigmoid, tanh…)
- Fully Connected Layer
- Convolution Layer
- Max Pooling Layer
- … and so on



Thank you!