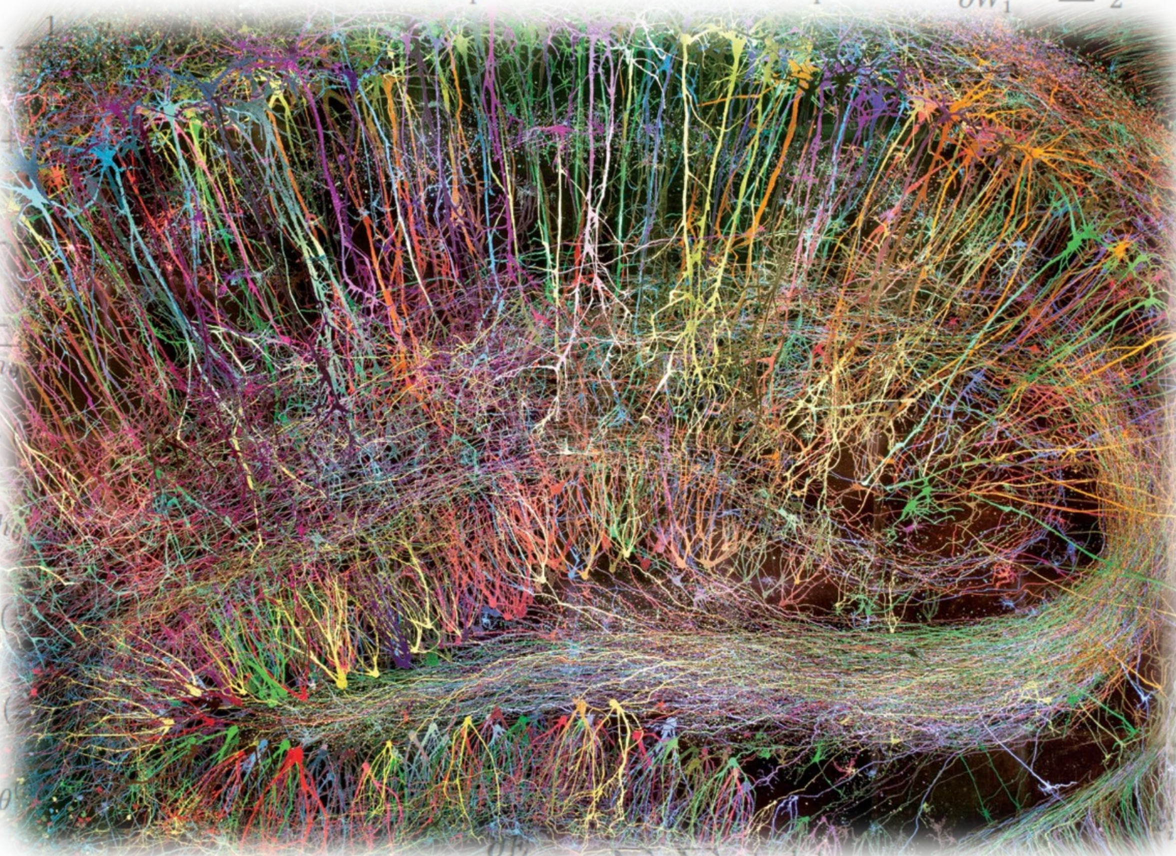


Backpropagation

Eren Erdal Aksoy

HALMSTAD UNIVERSITY

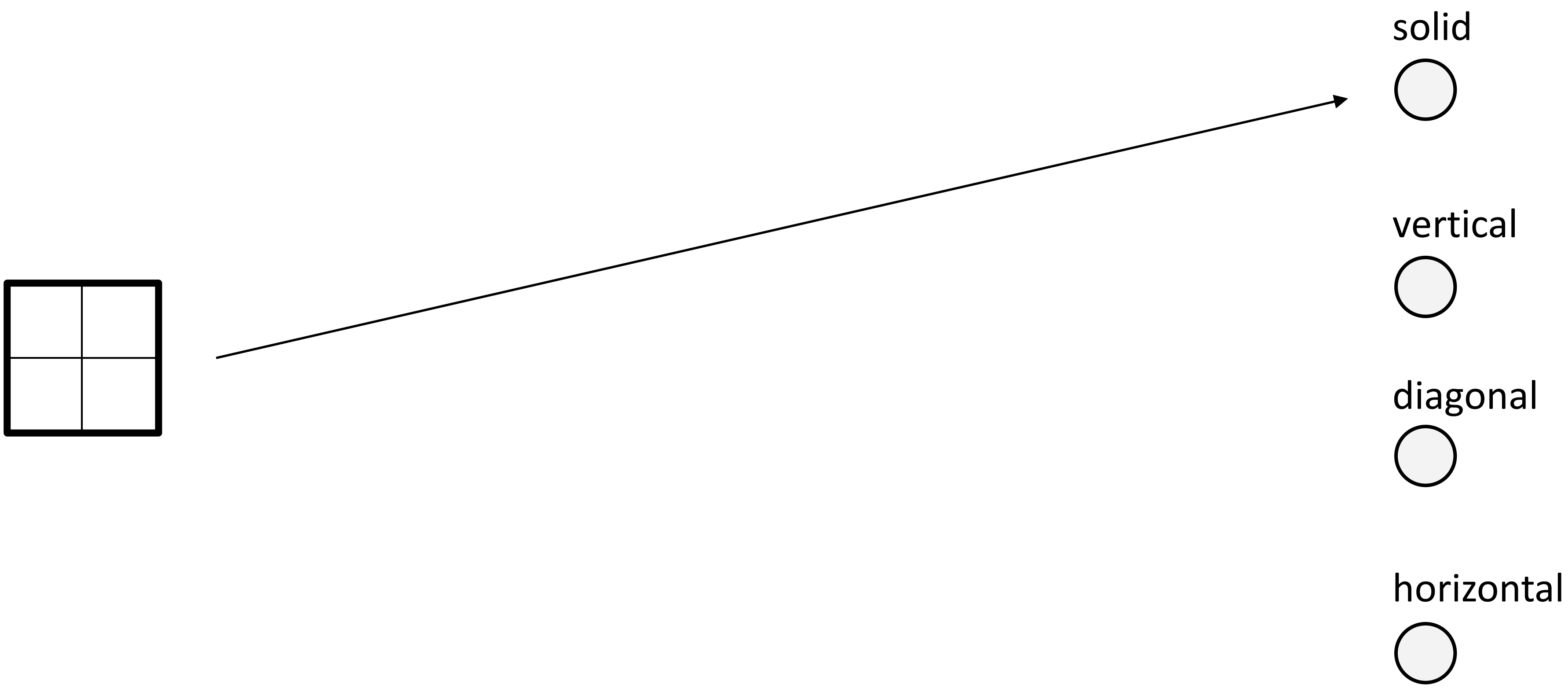
Backpropagation

- Let’s say we have ***four-pixel images to be categorized***

[Source: Brandon Rohrer](#)
[Click to go to the video in YouTube](#)

Backpropagation

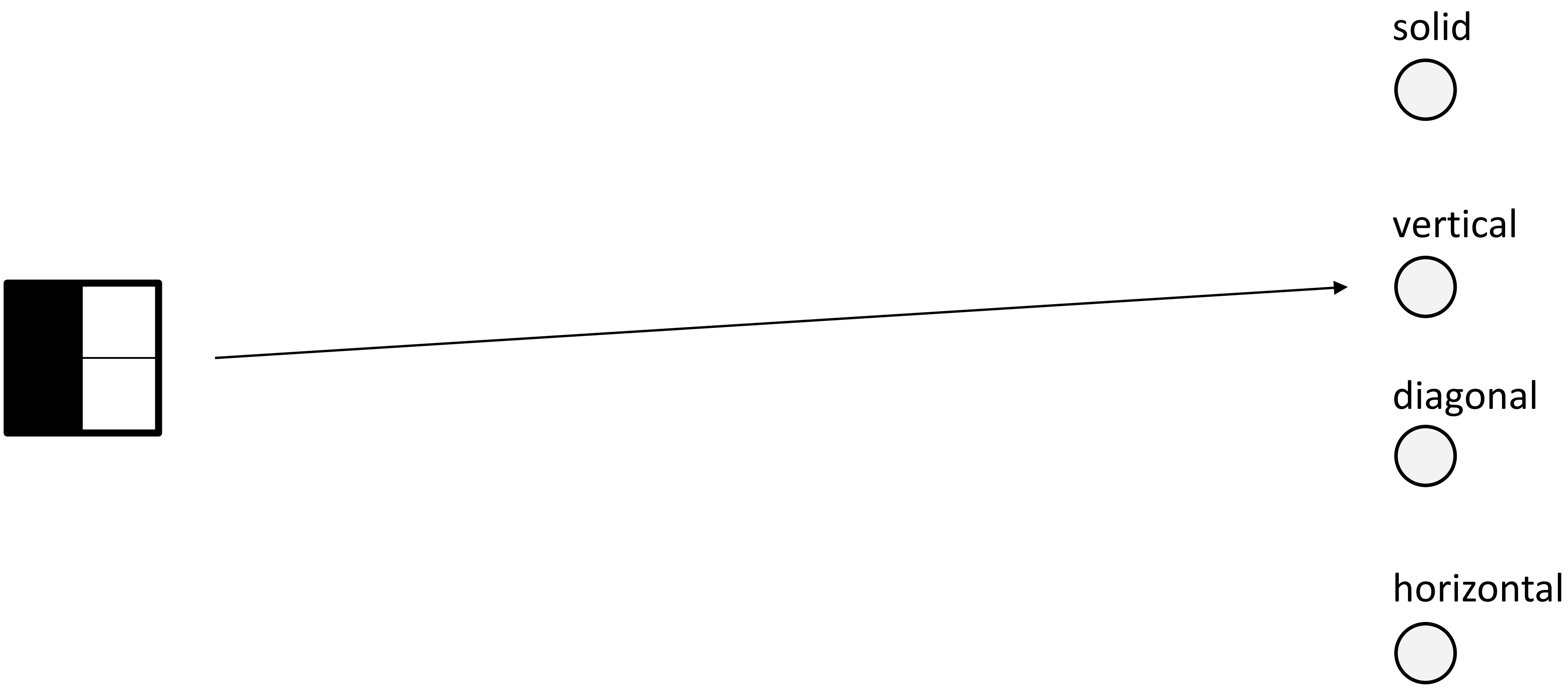
- Let's say we have ***four-pixel images to be categorized***



Source: Brandon Rohrer
Click to go to the video in YouTube

Backpropagation

- Let's say we have ***four-pixel images to be categorized***



[Source: Brandon Rohrer](#)
[Click to go to the video in YouTube](#)

Backpropagation

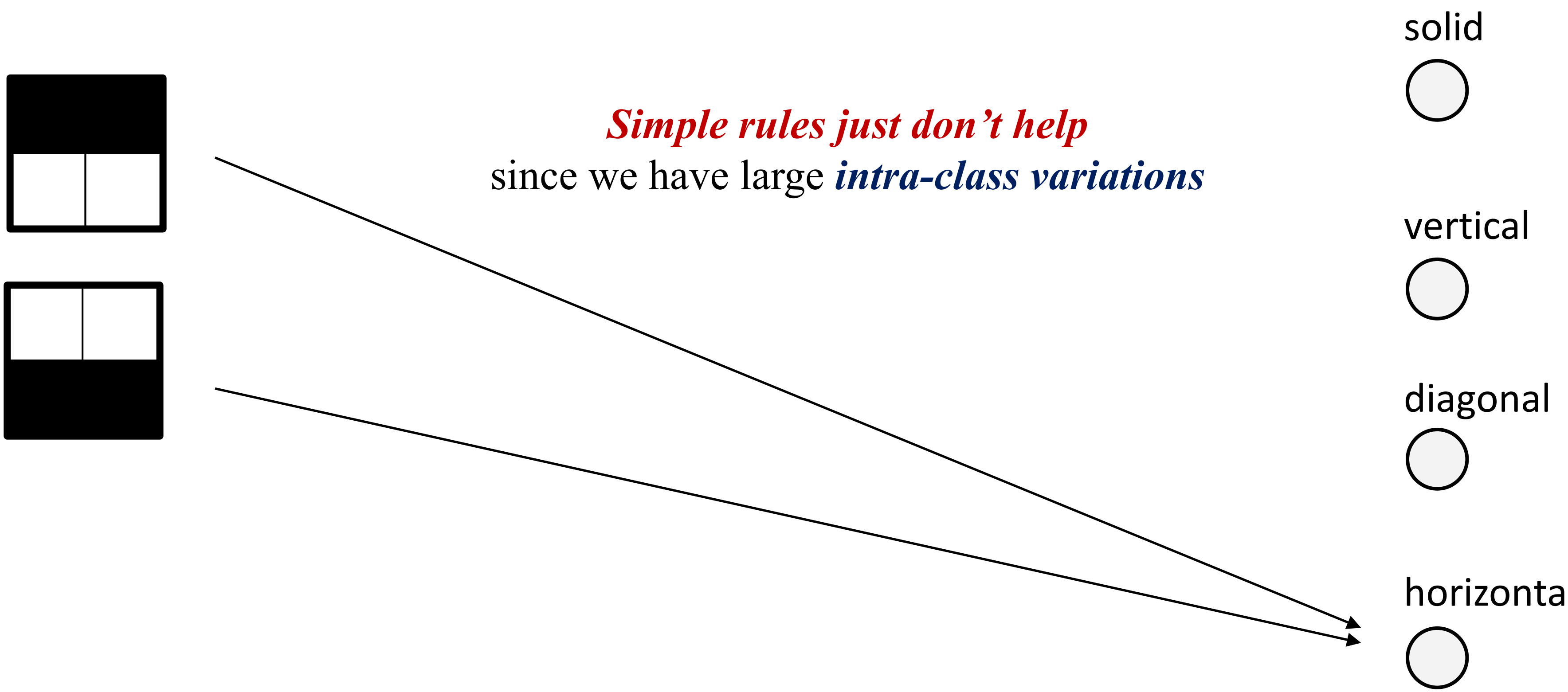
- Let's say we have ***four-pixel images to be categorized***

Source: Brandon Rohrer
Click to go to the video in YouTube

Backpropagation

- Let’s say we have ***four-pixel images to be categorized***

Simple rules just don’t help
since we have large ***intra-class variations***

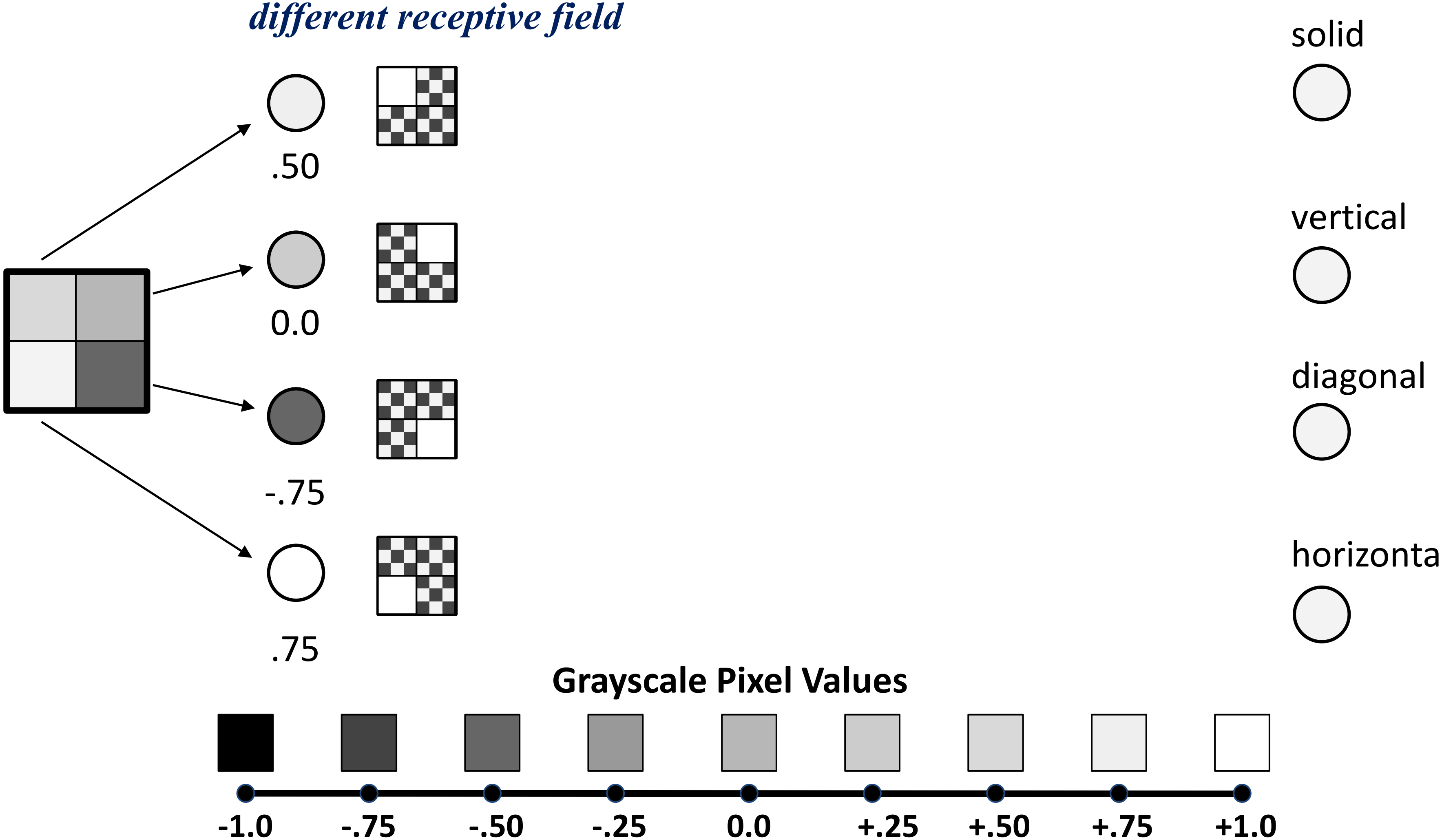


[Source: Brandon Rohrer](#)
[Click to go to the video in YouTube](#)

Backpropagation

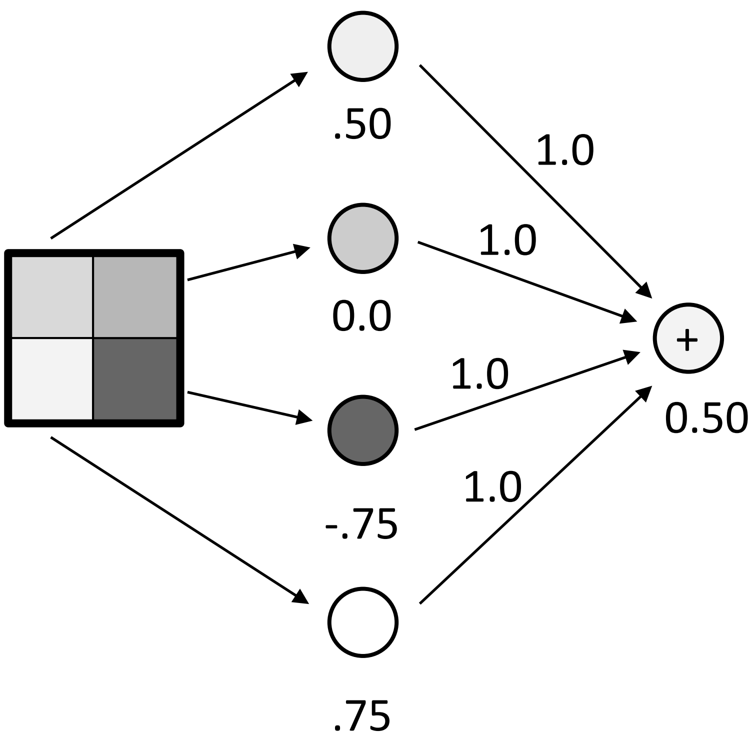
- Each input neuron corresponds to a different image pixel

Each input neuron has a ***different receptive field***

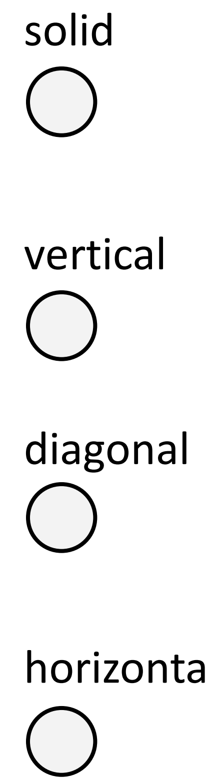


Backpropagation

- Each input neuron corresponds to a different image pixel
- In the next layer, neurons are connected



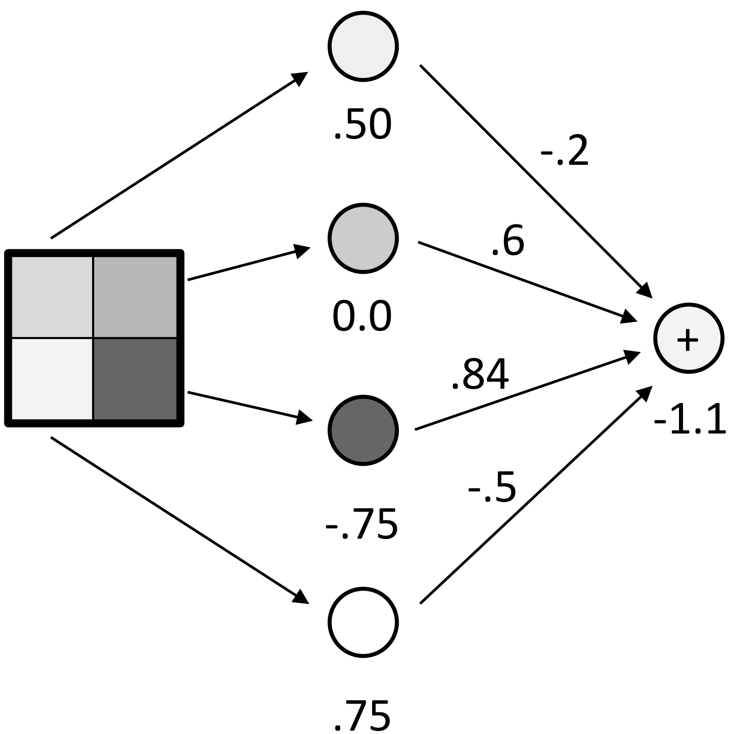
$$
\begin{array}{rr}
 & .50 \times 1.0 \\
 & 0.00 \times 1.0 \\
 & -.75 \times 1.0 \\
+ & .75 \times 1.0 \\
\hline
 & .50
\end{array}
$$



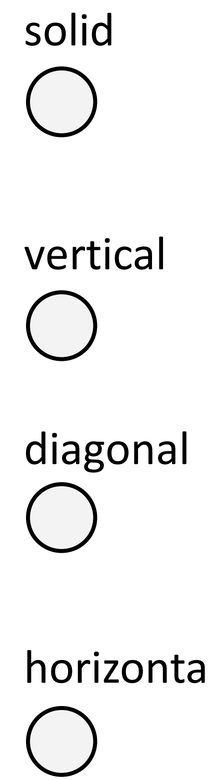
Source: Brandon Rohrer
Click to go to the video in YouTube

Backpropagation

- Each input neuron corresponds to a different image pixel
- In the next layer, neurons are connected
- Each connection has ***a randomly initialized weight value***



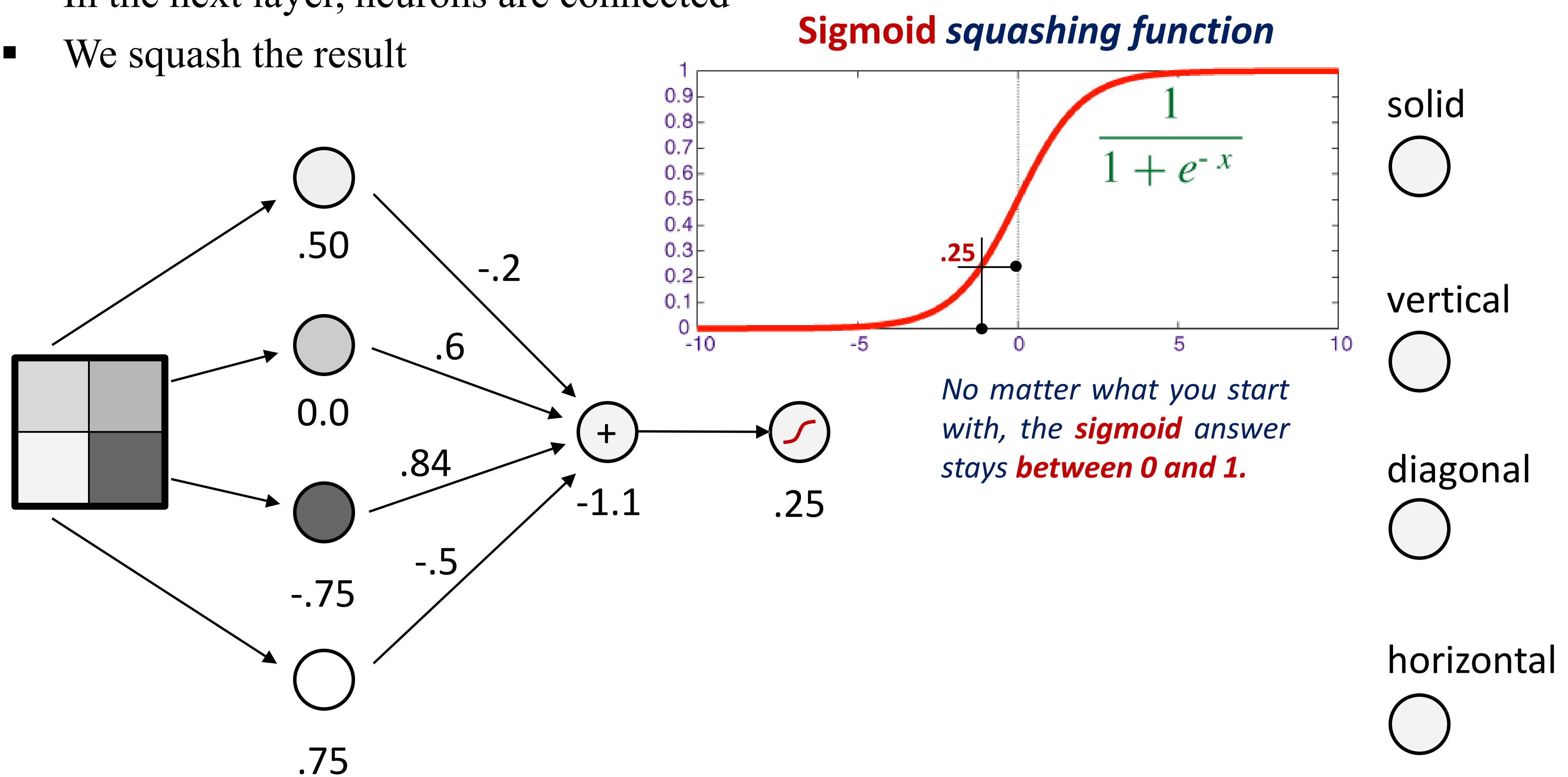
$$
\begin{array}{rcr}
.50 & \times & -.2 \\
0.00 & \times & .6 \\
-.75 & \times & .84 \\
+\ \ .75 & \times & -.5 \\
\hline
 & \approx & -1.1
\end{array}
$$



[Source: Brandon Rohrer](#)
[Click to go to the video in YouTube](#)

Backpropagation

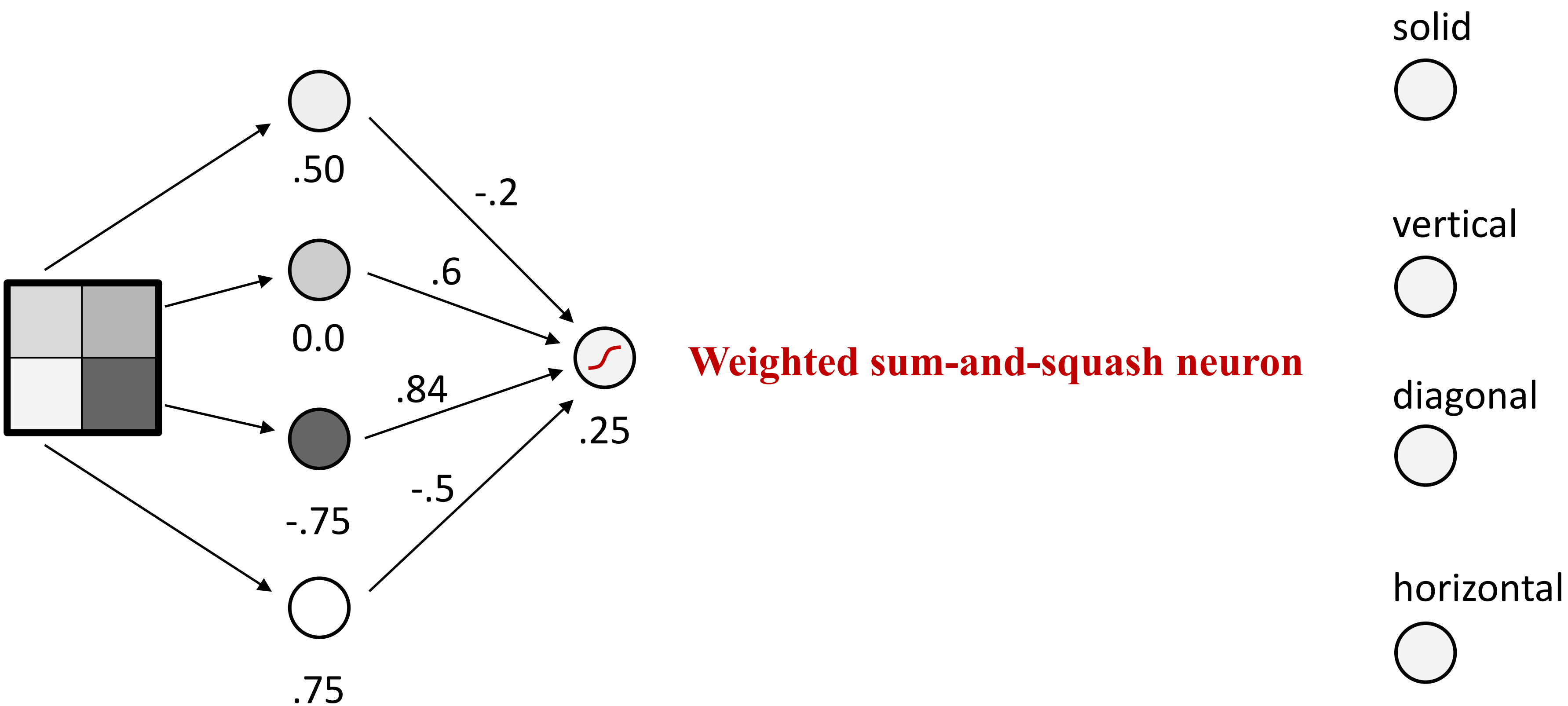
- Each input neuron corresponds to a different image pixel
- In the next layer, neurons are connected
- We squash the result



Source: Brandon Rohrer
Click to go to the video in YouTube

Backpropagation

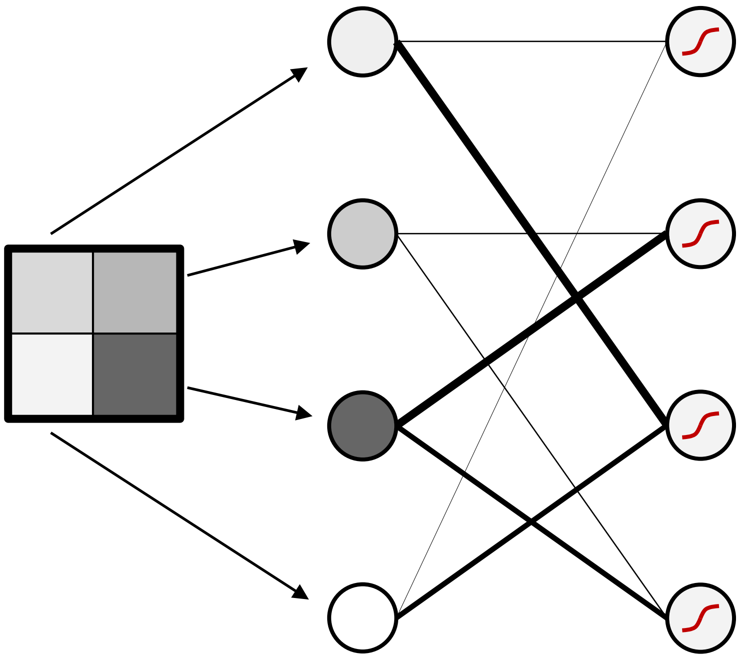
- Each input neuron corresponds to a different image pixel



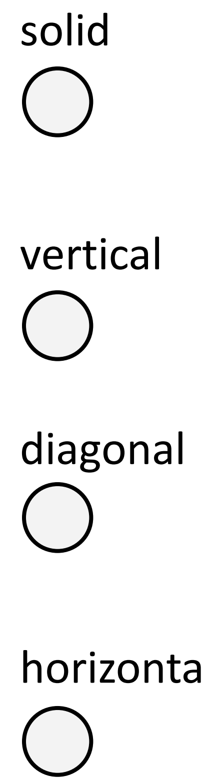
Source: Brandon Rohrer
Click to go to the video in YouTube

Backpropagation

- Each input neuron corresponds to a different image pixel
- Let's have lots of ***weighted sum-and-squash neurons*** with ***different weight values***



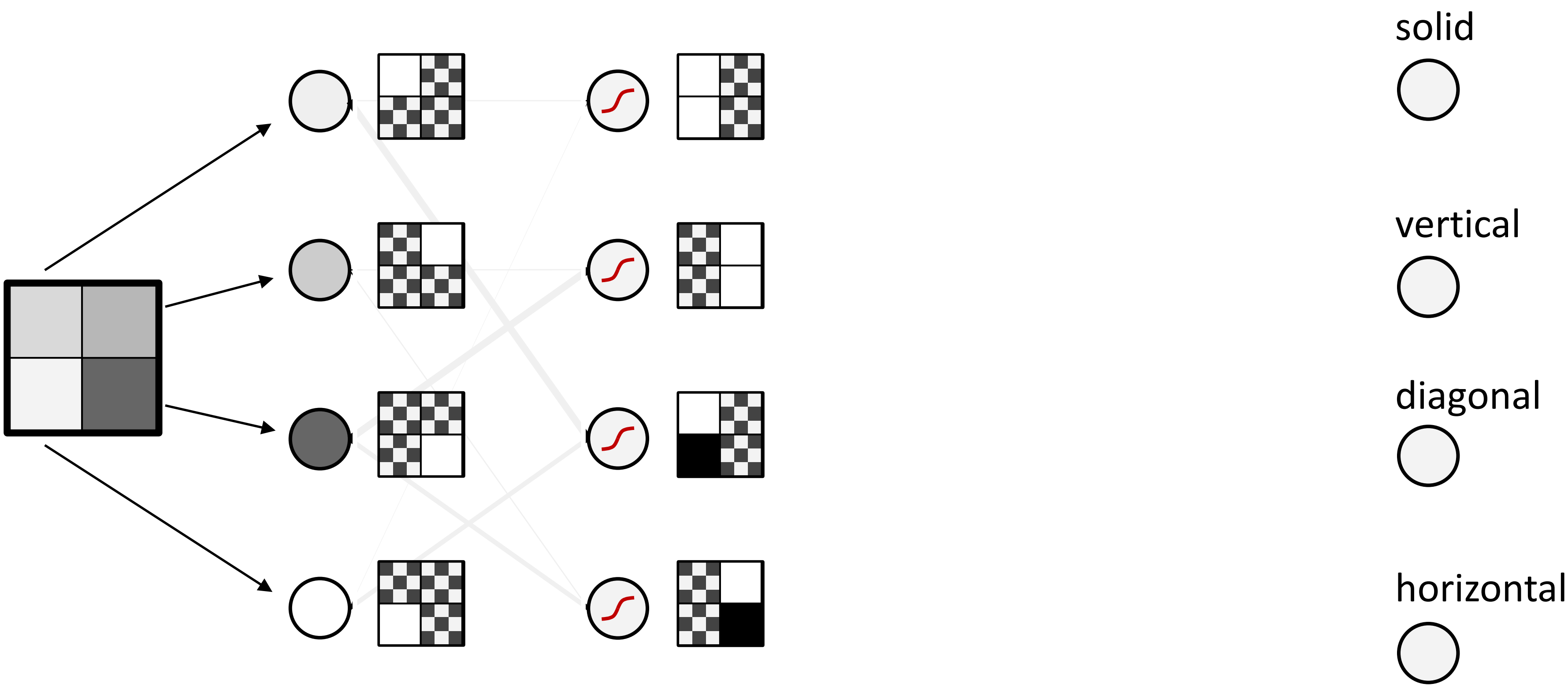
- Edges are created ***randomly***
- ***Thicker edges*** *represent* ***higher weights****.*
- *Note that some there exist* ***no edges*** *between some neurons since the corresending weight values are* ***zero****.*



[Source: Brandon Rohrer](#)
[Click to go to the video in YouTube](#)

Backpropagation

- Each input neuron corresponds to a different image pixel
- Let’s have lots of ***weighted sum-and-squash neurons*** with ***different weight values***

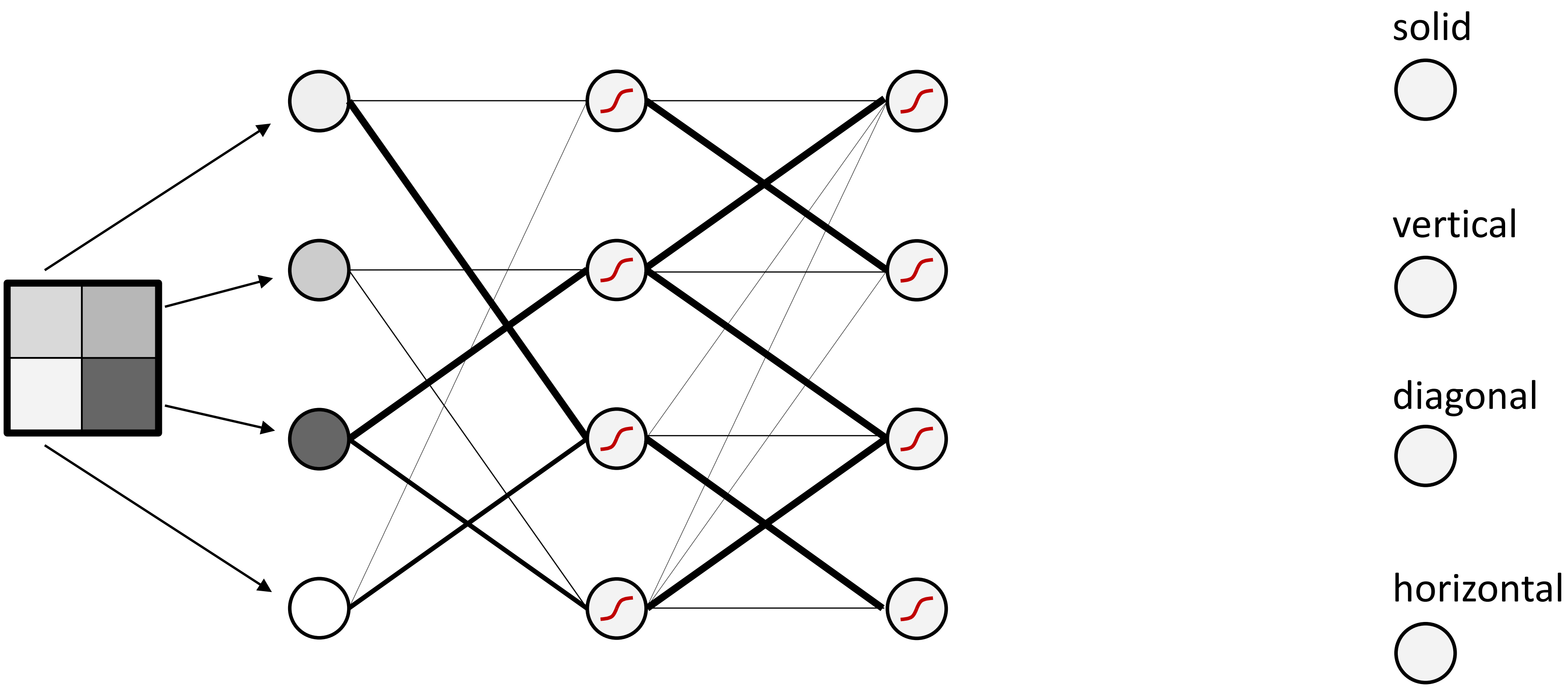


Receptive fields get more complex

[Source: Brandon Rohrer](#)
[Click to go to the video in YouTube](#)

Backpropagation

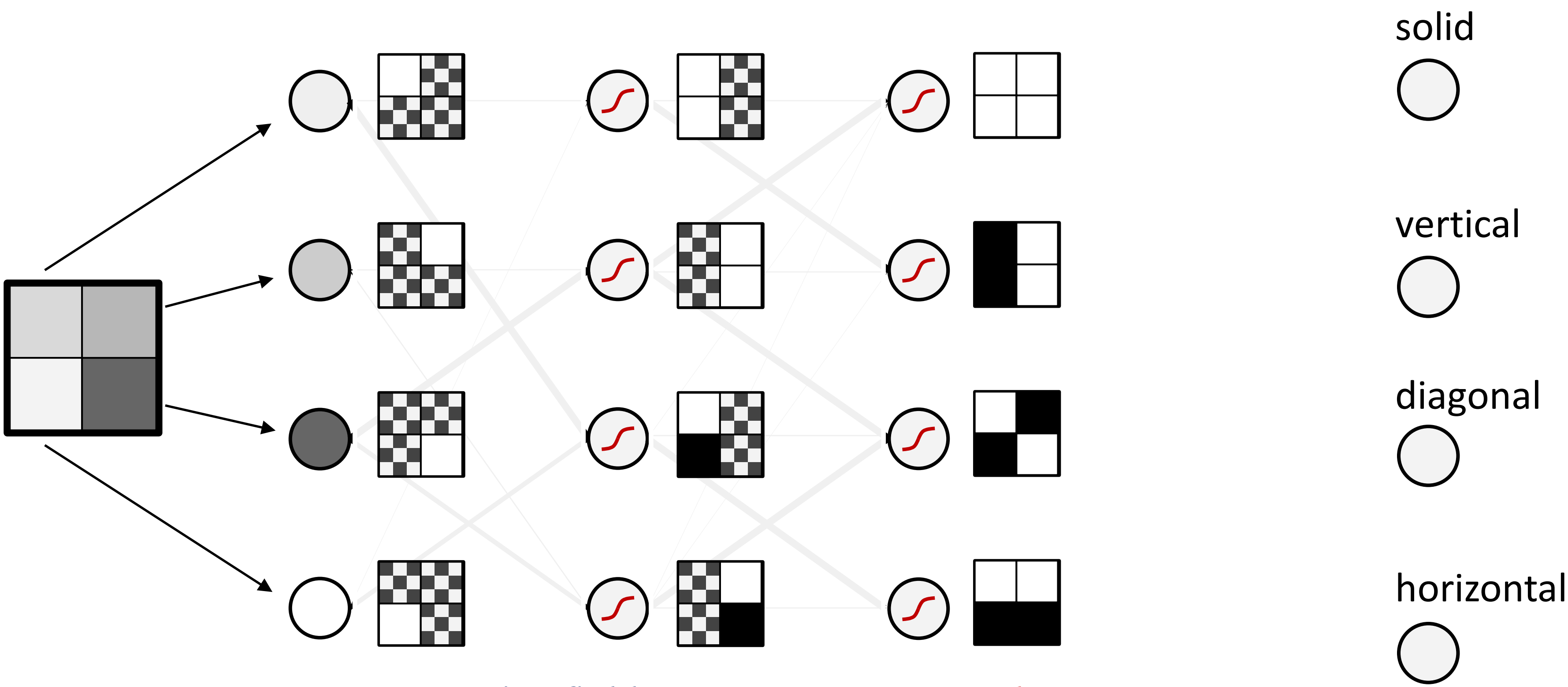
- Each input neuron corresponds to a different image pixel
- Let’s have lots of ***weighted sum-and-squash neurons*** with ***different weight values***
- We can add ***more layers***



[Source: Brandon Rohrer](#)
[Click to go to the video in YouTube](#)

Backpropagation

- Each input neuron corresponds to a different image pixel
- Let's have lots of ***weighted sum-and-squash neurons*** with ***different weight values***
- We can add ***more layers***

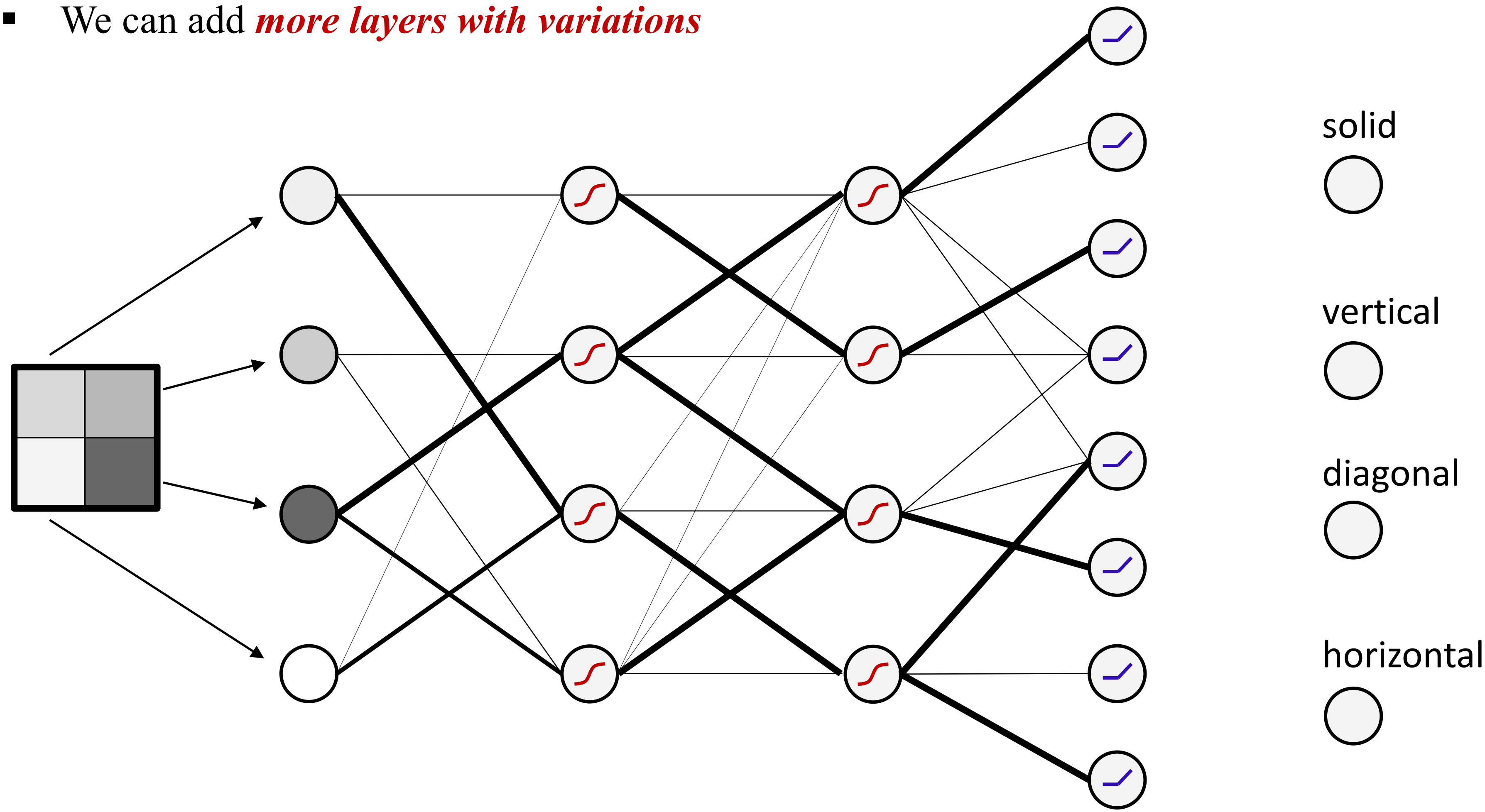


Receptive fields get even more complex

[Source: Brandon Rohrer](#)
[Click to go to the video in YouTube](#)

Backpropagation

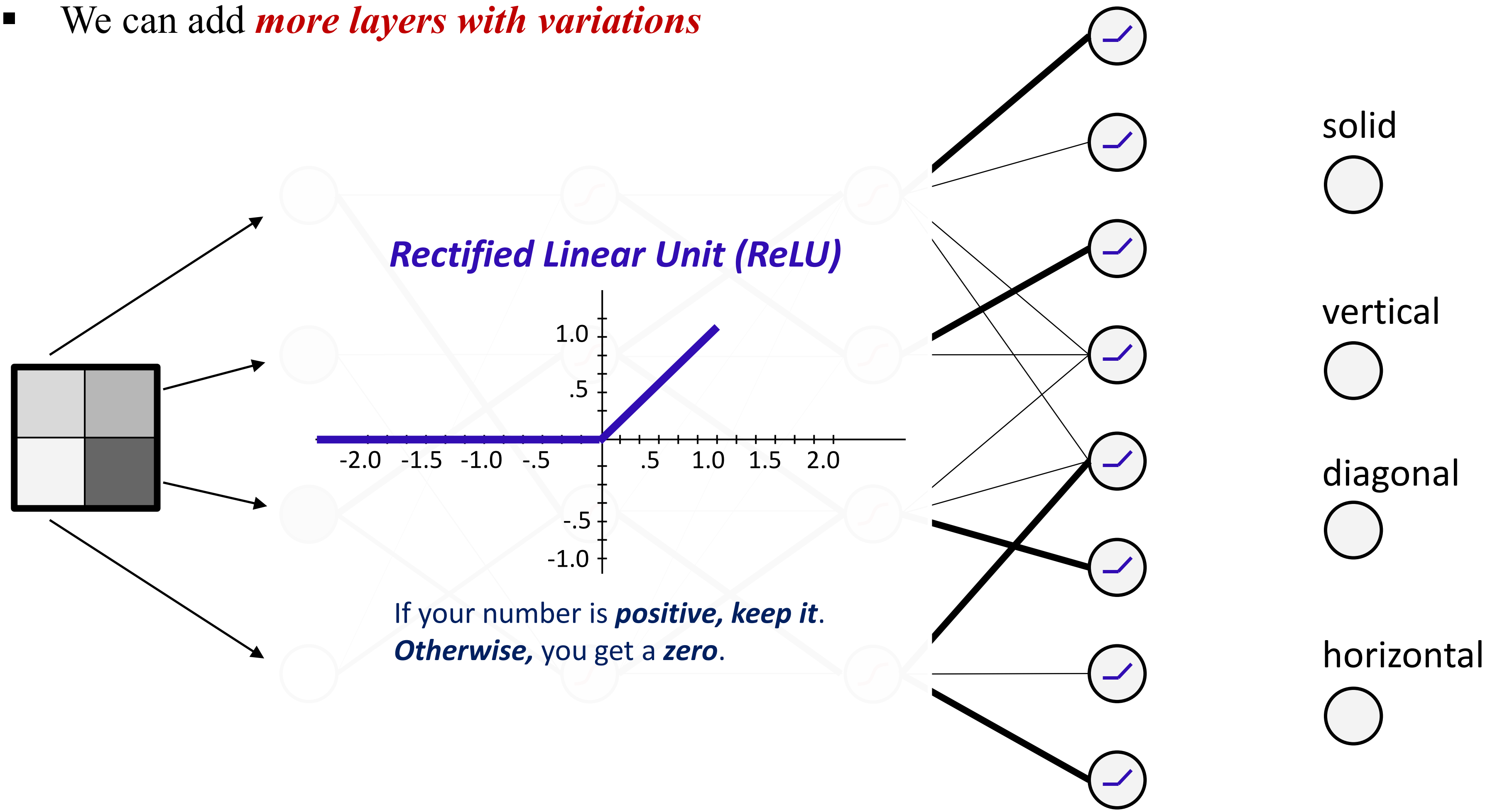
- Each input neuron corresponds to a different image pixel
- We can add ***more layers with variations***



[Source: Brandon Rohrer](#)
[Click to go to the video in YouTube](#)

Backpropagation

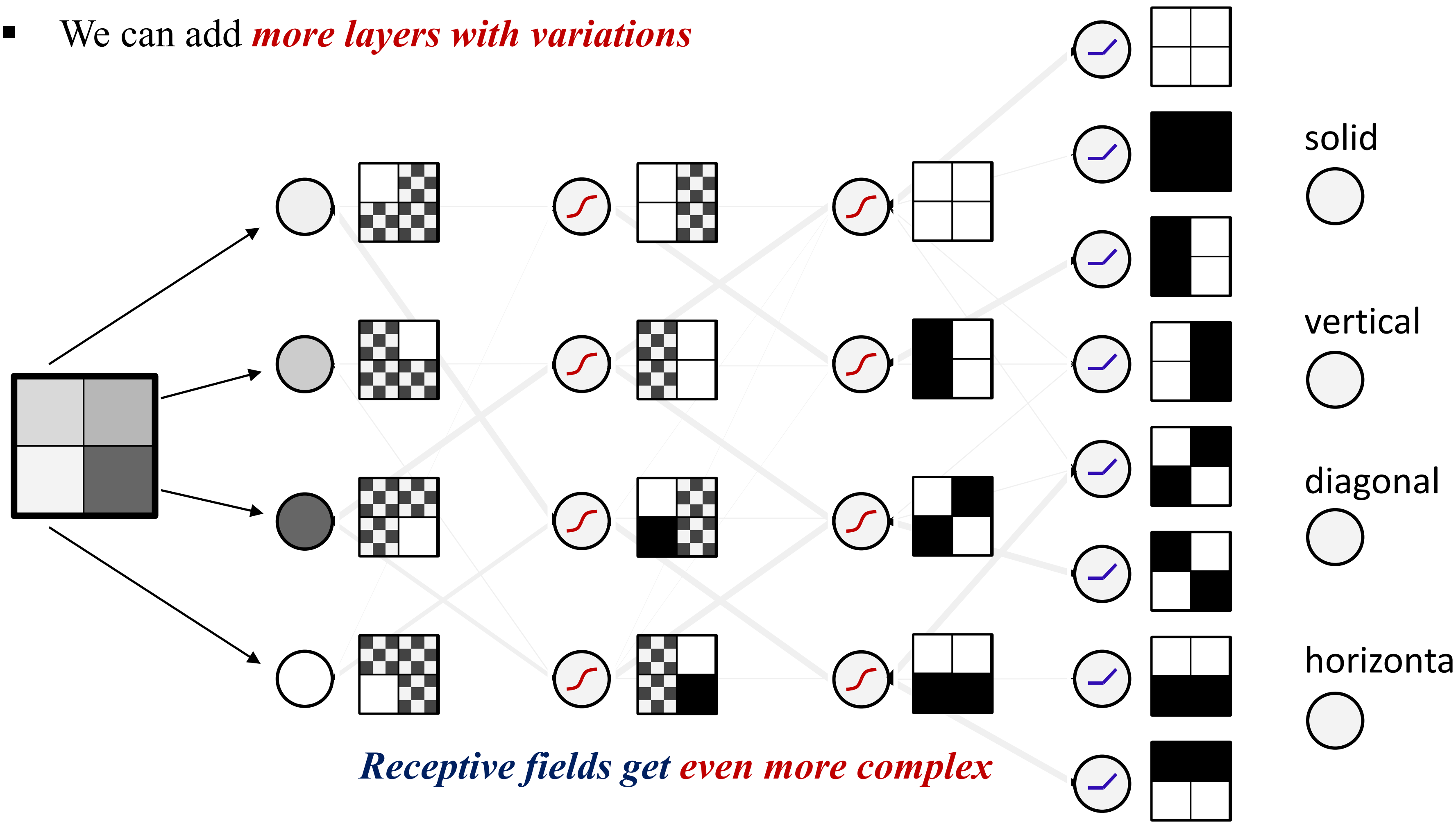
- Each input neuron corresponds to a different image pixel
- We can add ***more layers with variations***



[Source: Brandon Rohrer](#)
[Click to go to the video in YouTube](#)

Backpropagation

- Each input neuron corresponds to a different image pixel
- We can add ***more layers with variations***



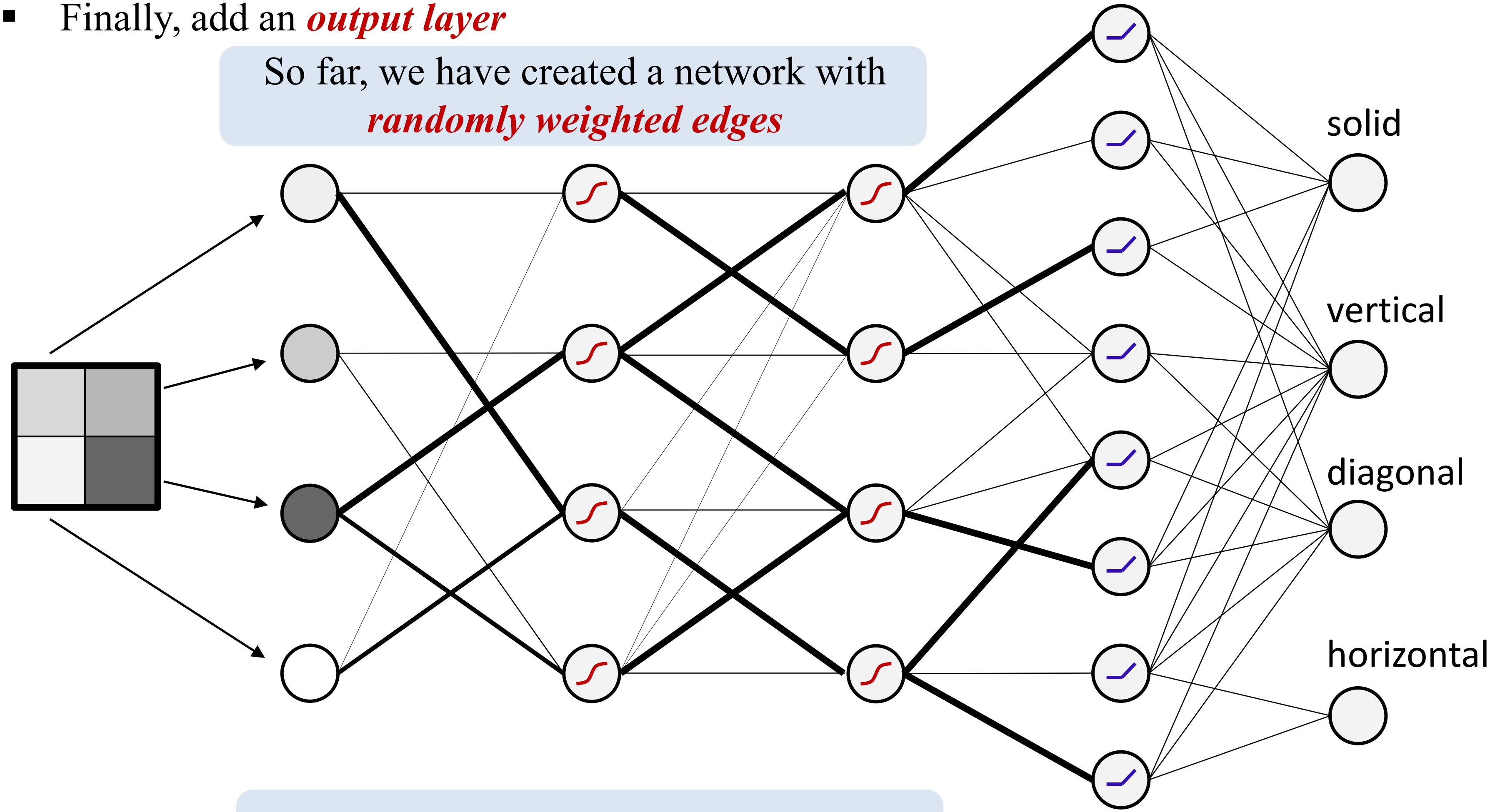
Receptive fields get even more complex

[Source: Brandon Rohrer](#)
[Click to go to the video in YouTube](#)

Backpropagation

- Each input neuron corresponds to a different image pixel
- Finally, add an ***output layer***

So far, we have created a network with ***randomly weighted edges***

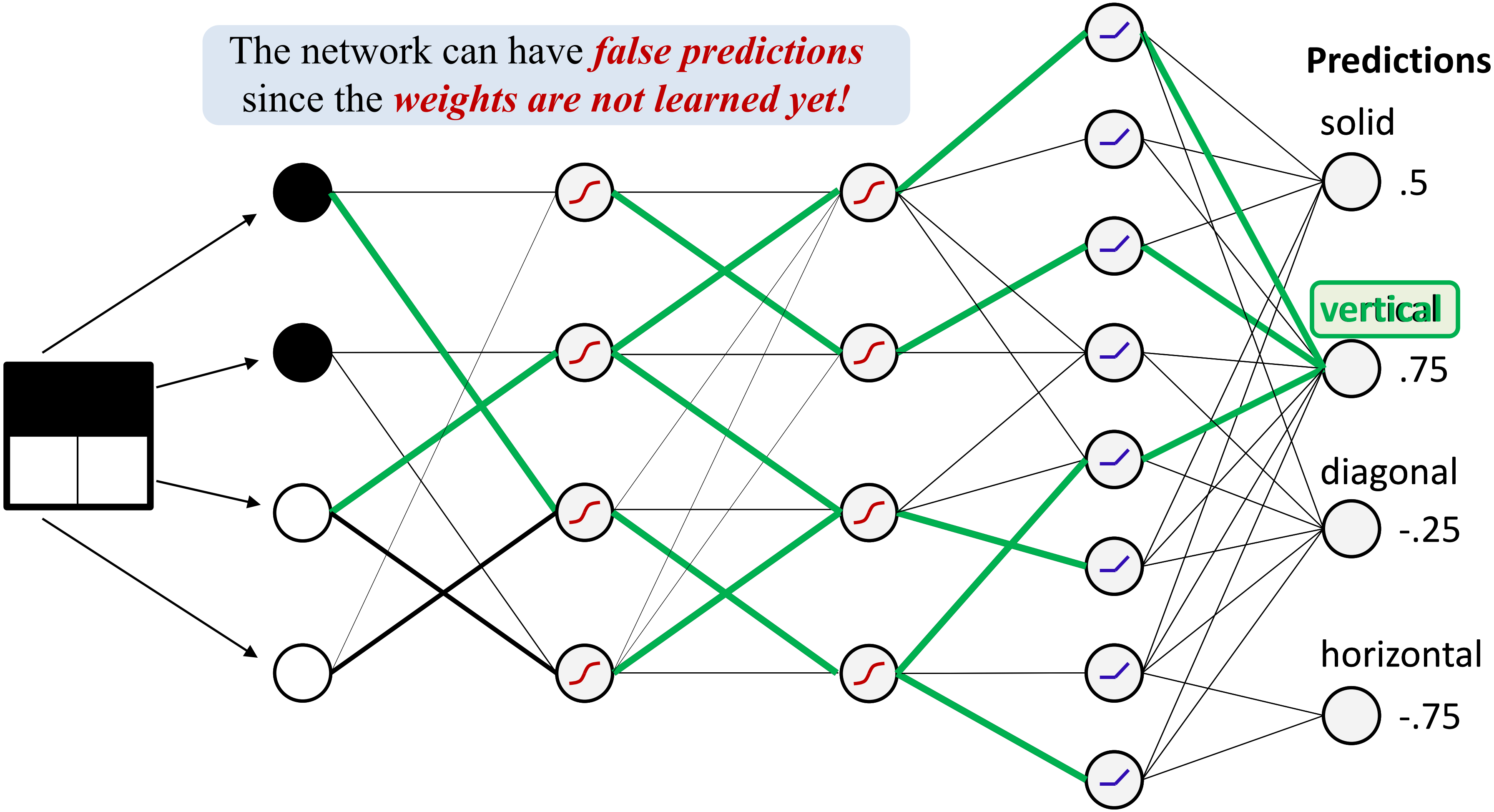


What is the response to an input image?

[Source: Brandon Rohrer](#)
[Click to go to the video in YouTube](#)

Backpropagation

- What if a ***new input image*** is provided?

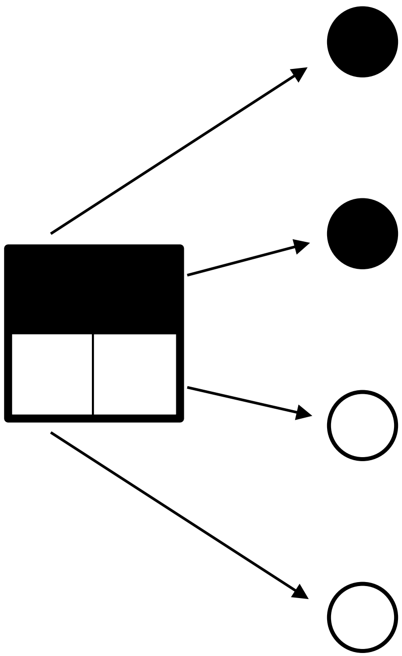


[Source: Brandon Rohrer](#)
[Click to go to the video in YouTube](#)

Backpropagation

- We have to ***compare*** the ***predictions*** with the ***ground truth***
- ***Error*** is the ***magnitude of the difference***

Known in advance



Our ultimate aim is to minimize the overall error in the network predictions!

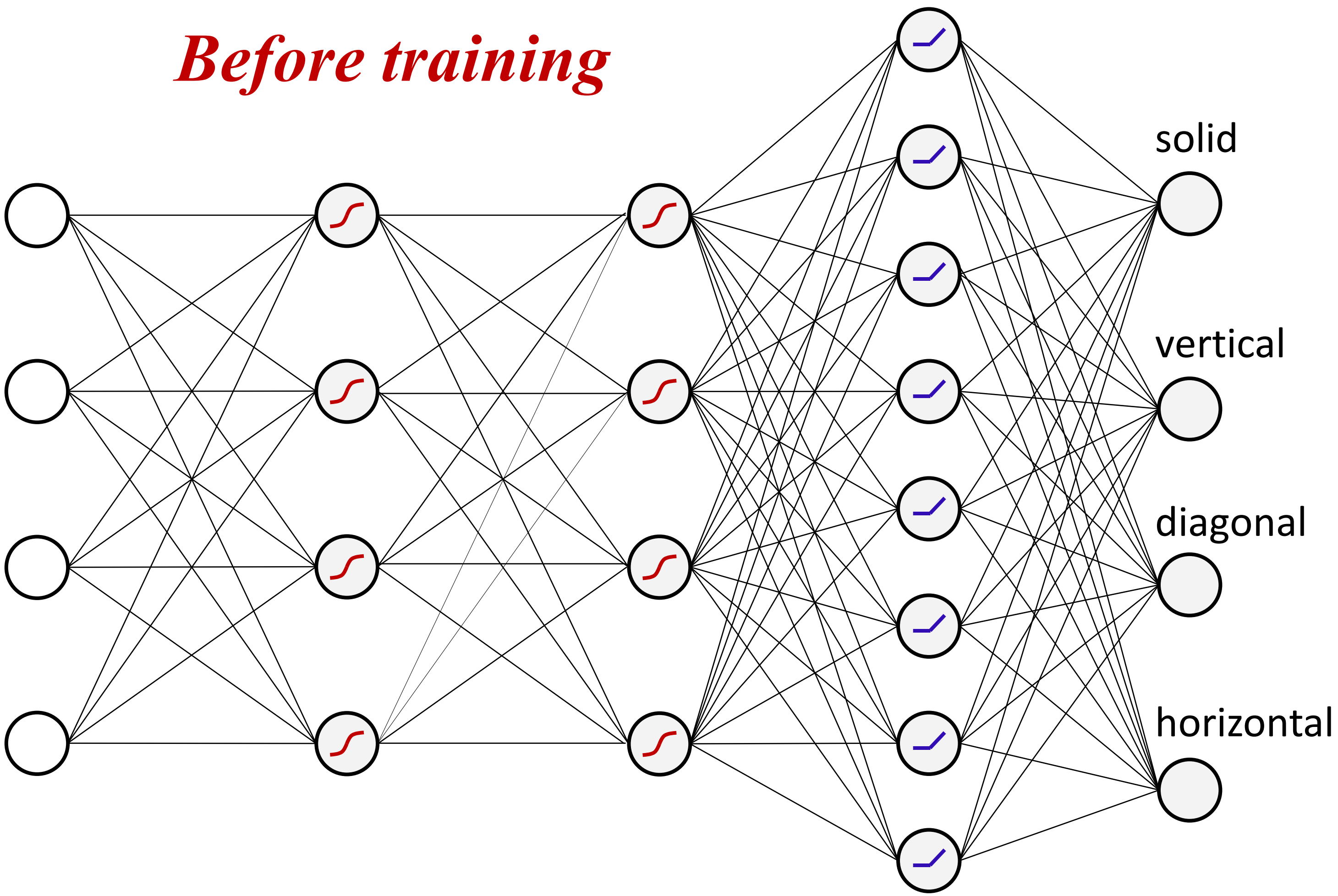
The error must be propagated back to the network to update the weights accordingly!

Error	Ground Truth	Predictions
.5	0	solid .5
.75	0	vertical .75
.25	0	diagonal -.25
+ 1.75	1	horizontal -.75

Total Error = 3.25

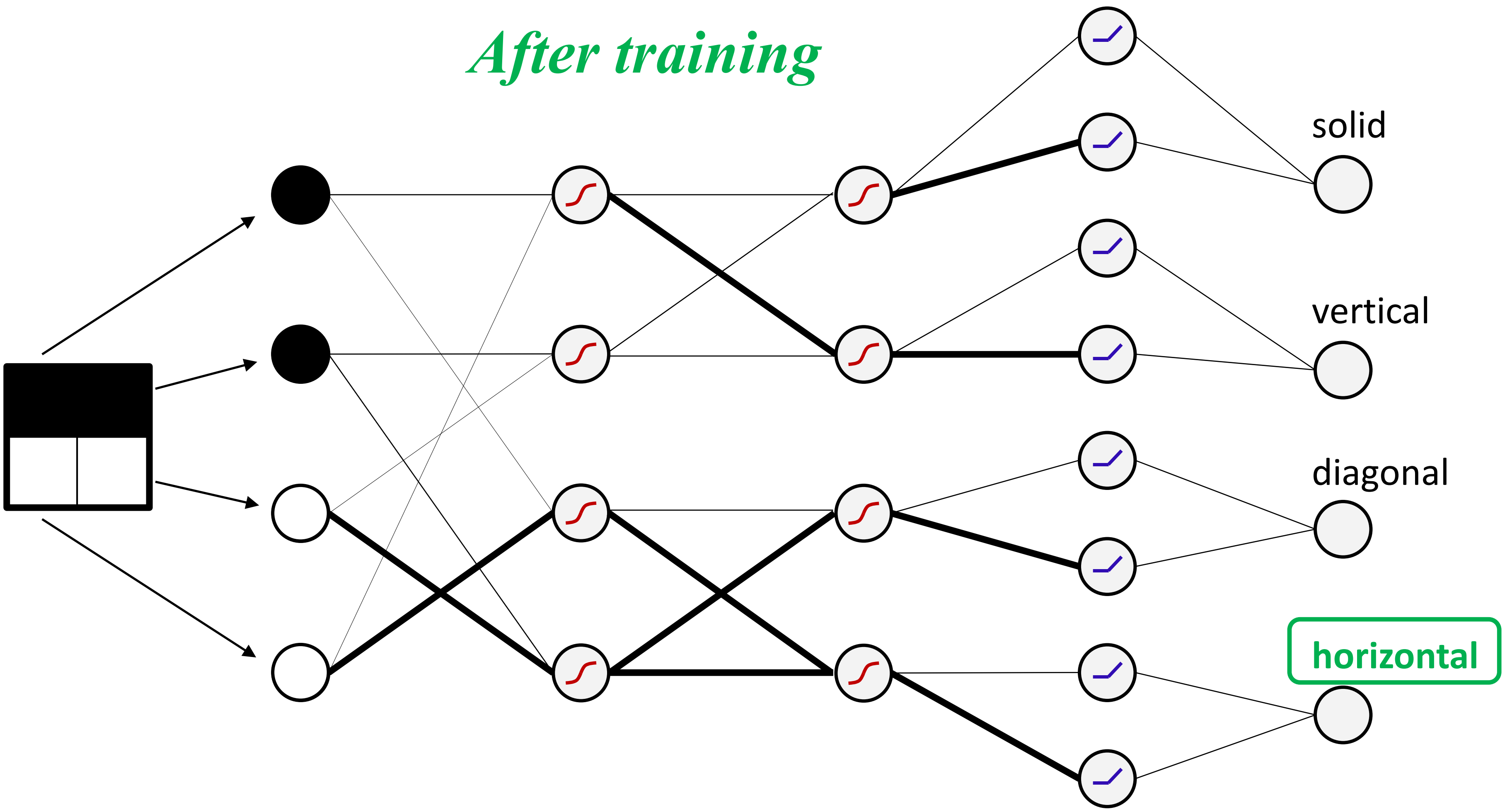
Source: Brandon Rohrer
Click to go to the video in YouTube

Backpropagation



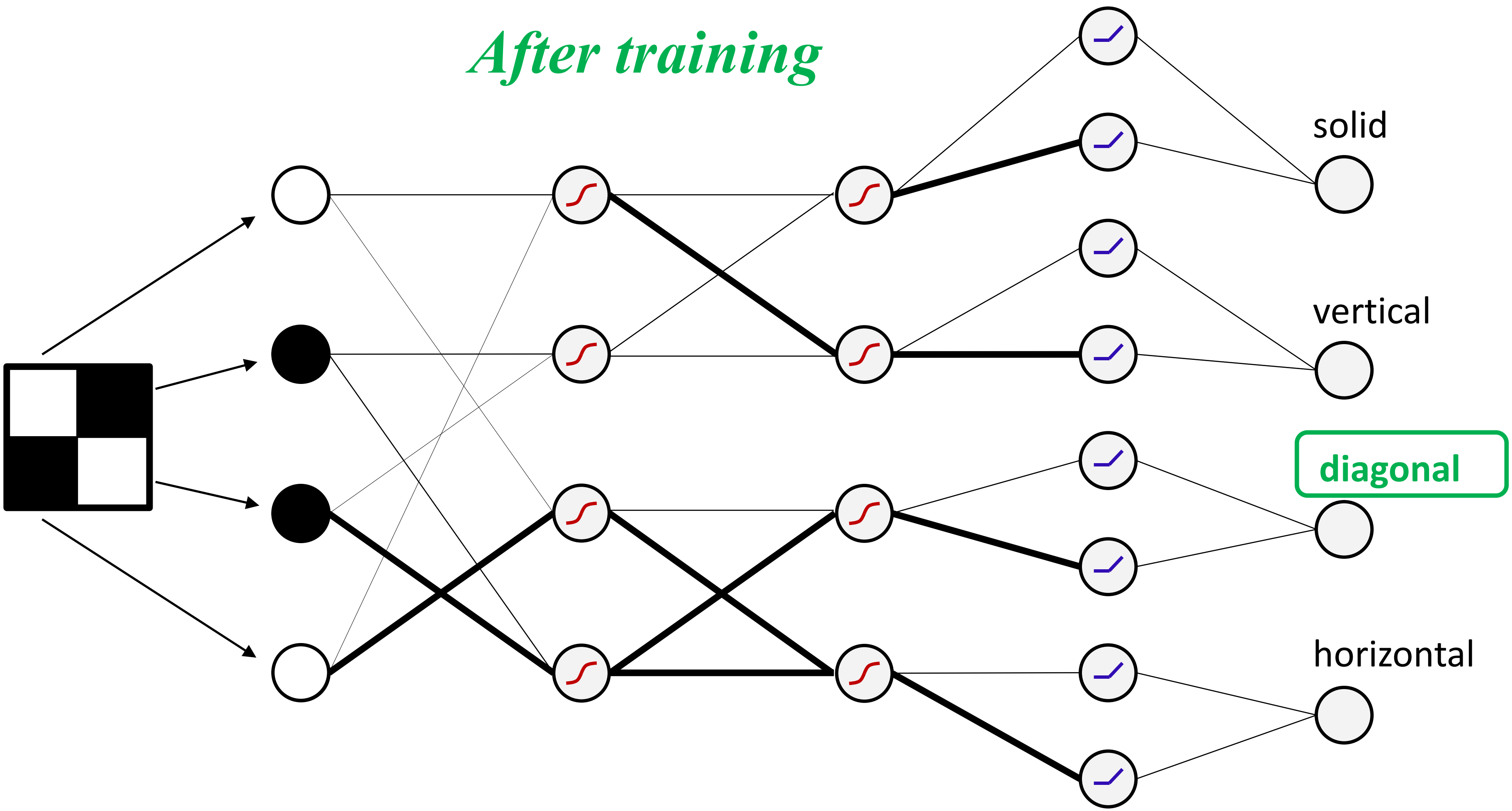
Source: Brandon Rohrer
Click to go to the video in YouTube

Backpropagation



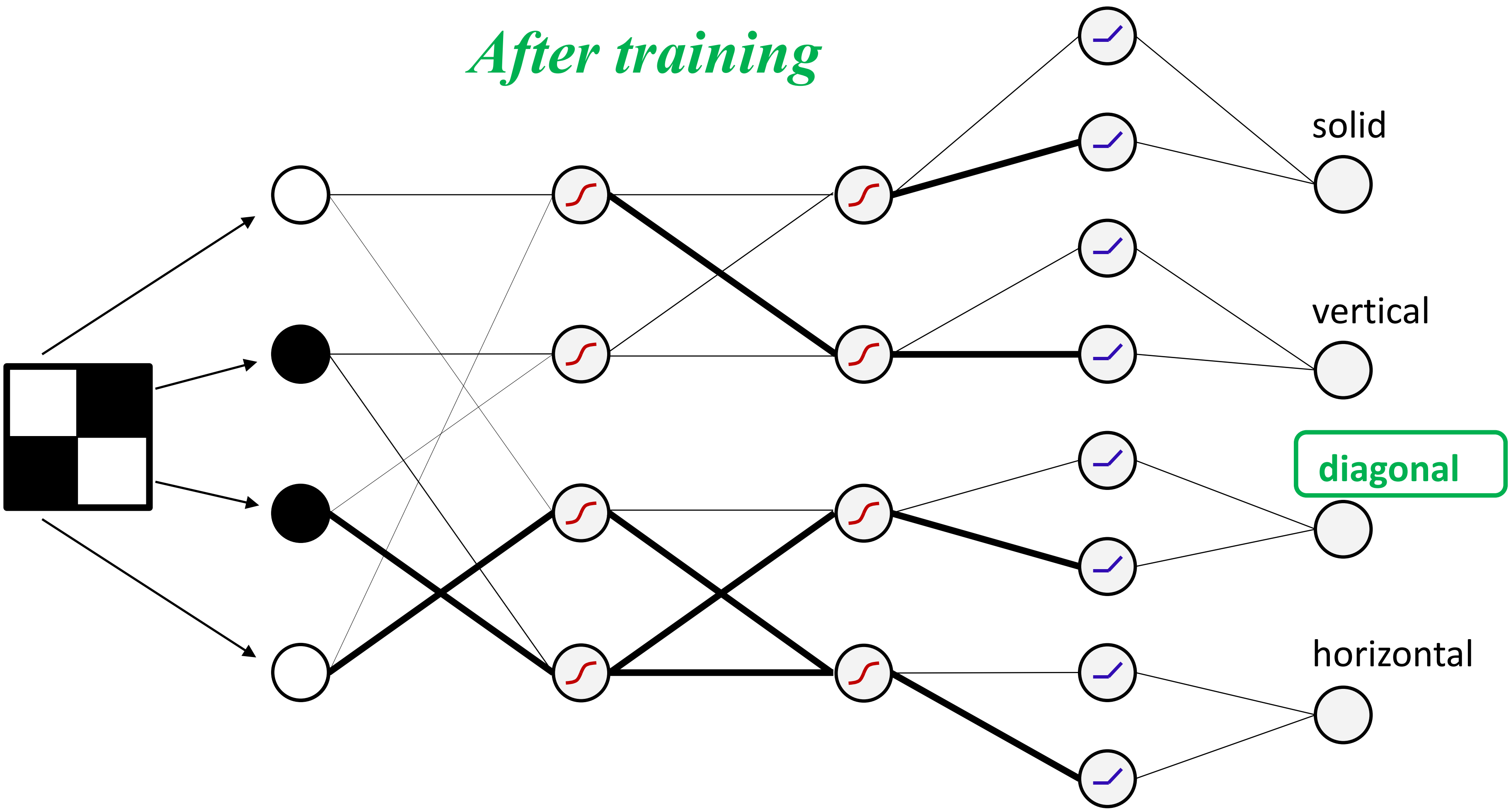
[Source: Brandon Rohrer](#)
[Click to go to the video in YouTube](#)

Backpropagation



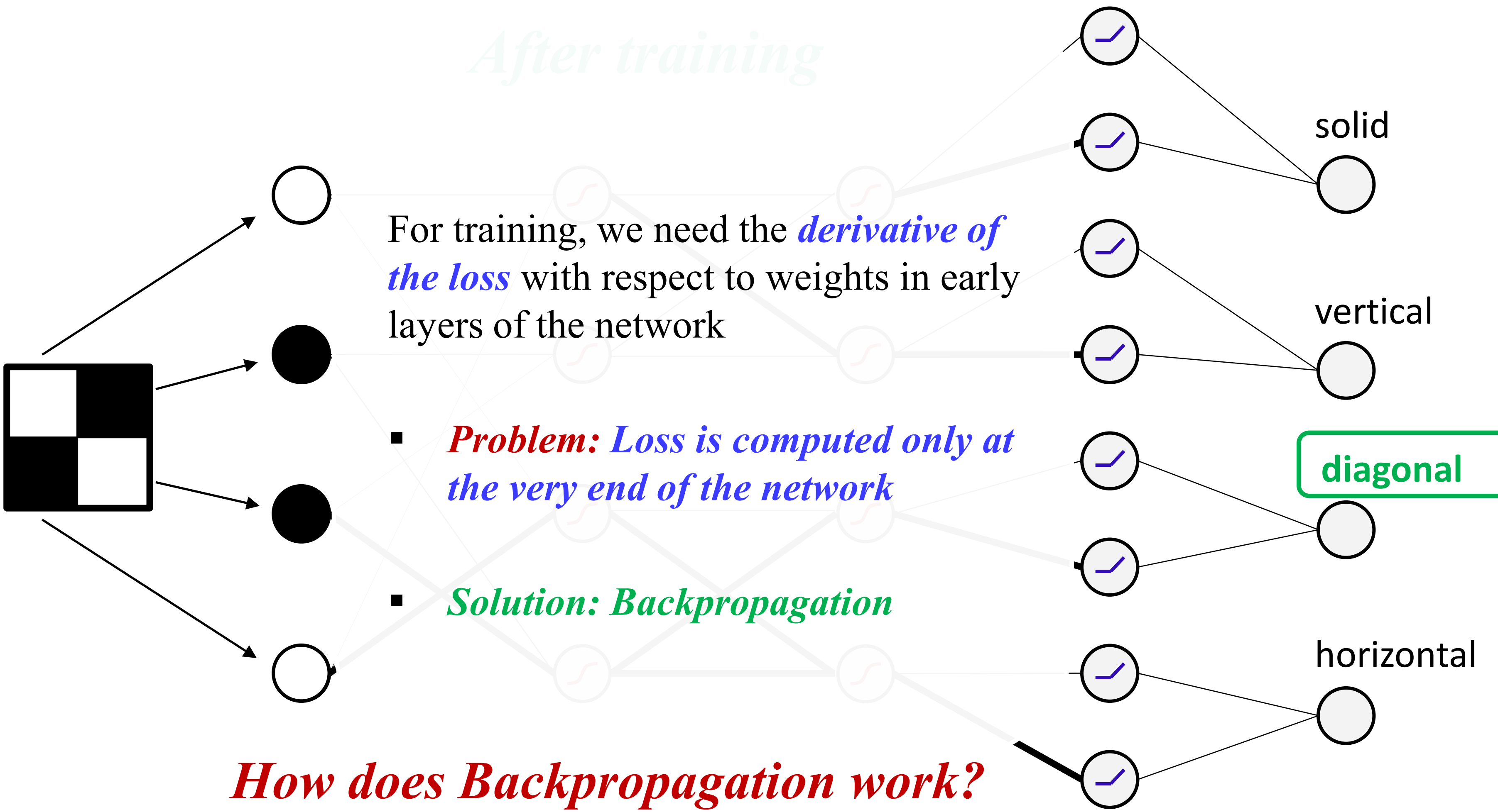
Source: Brandon Rohrer
Click to go to the video in YouTube

Backpropagation



[Source: Brandon Rohrer](#)
[Click to go to the video in YouTube](#)

Backpropagation



For training, we need the ***derivative of the loss*** with respect to weights in early layers of the network

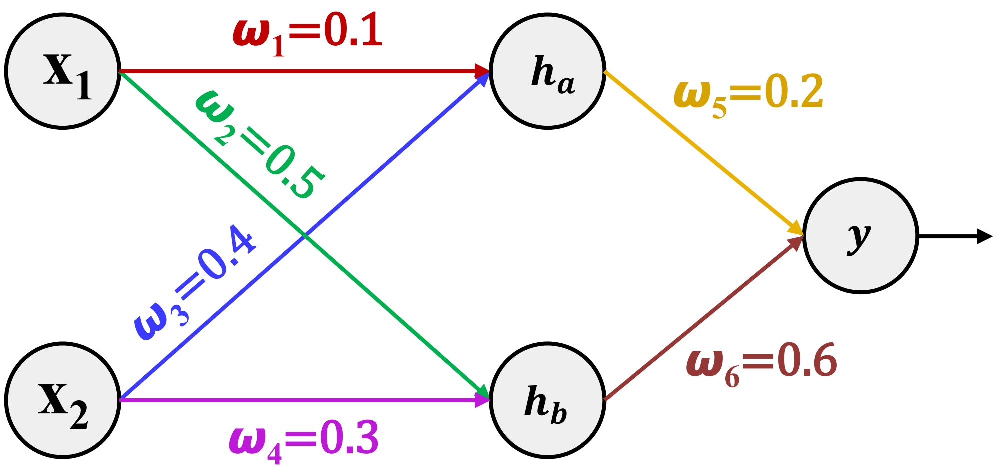
- ***Problem: Loss is computed only at the very end of the network***
- ***Solution: Backpropagation***

How does Backpropagation work?

[Source: Brandon Rohrer](#)
[Click to go to the video in YouTube](#)

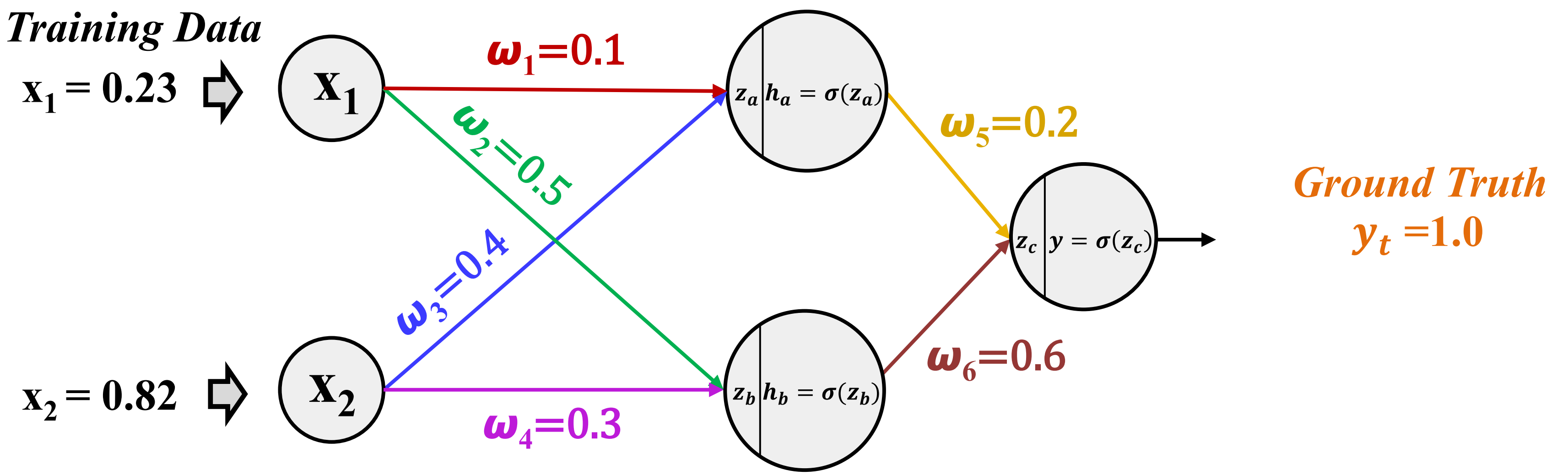
Backpropagation

- Let's have a simple and ***shallow feedforward neural network***.



Backpropagation

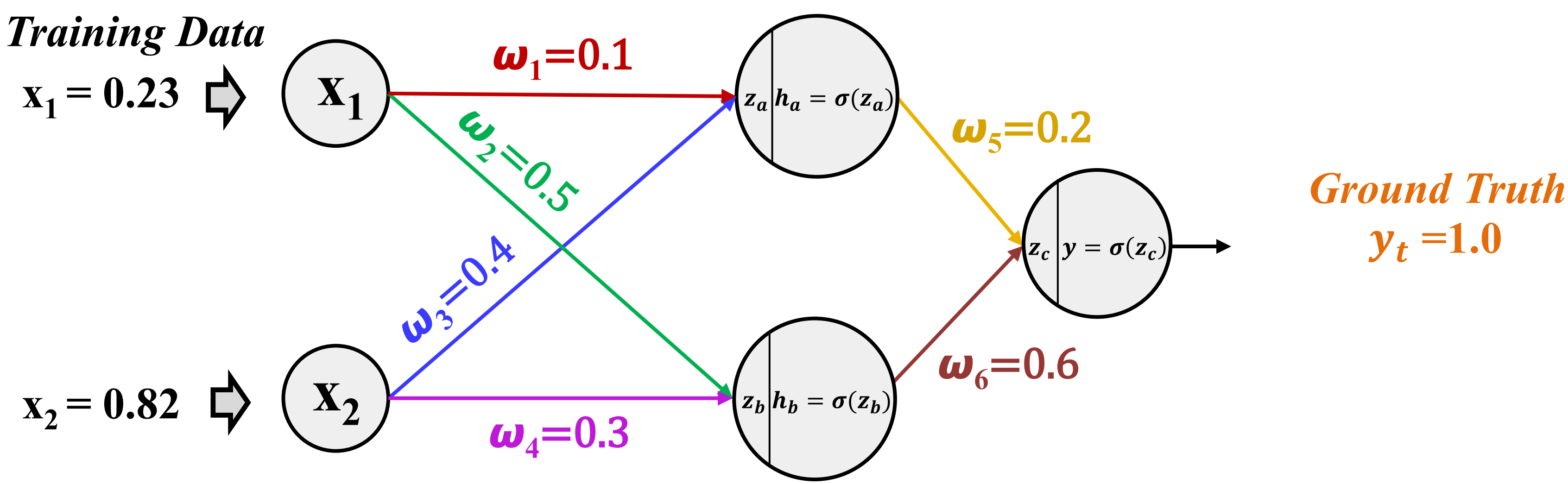
- Let's have a simple and ***shallow feedforward neural network***.



- To train the network we ***have three steps:***
  1. ***Forward pass***
  2. ***Error Backpropagation***
  3. ***Parameter updates***

Backpropagation

1. Forward Pass



$z_a = x_1\omega_1 + x_2\omega_3$ =0.23*0.1 + 0.82*0.4 = 0.351 $\quad h_a = \sigma(z_a)$ = σ(0.351) = 0.5868

$z_b = x_1\omega_2 + x_2\omega_4$=0.23*0.5 + 0.82*0.3 = 0.361 $\quad h_b = \sigma(z_b)$ = σ(0.361) = 0.5892

$z_c = h_a\omega_5 + h_b\omega_6 = \sigma(z_a)\omega_5 + \sigma(z_b)\omega_6$
= 0.5868*0.2 + 0.5892*0.6 = 0.4708

$y = \sigma(z_c)$ = σ(0.4708) = 0.6155

Mean Squared Error
(i.e., loss function)

$$E = \frac{1}{2}(y_t - y)^2 = \frac{1}{2}(1.0 - 0.6155)^2 = 0.0739$$

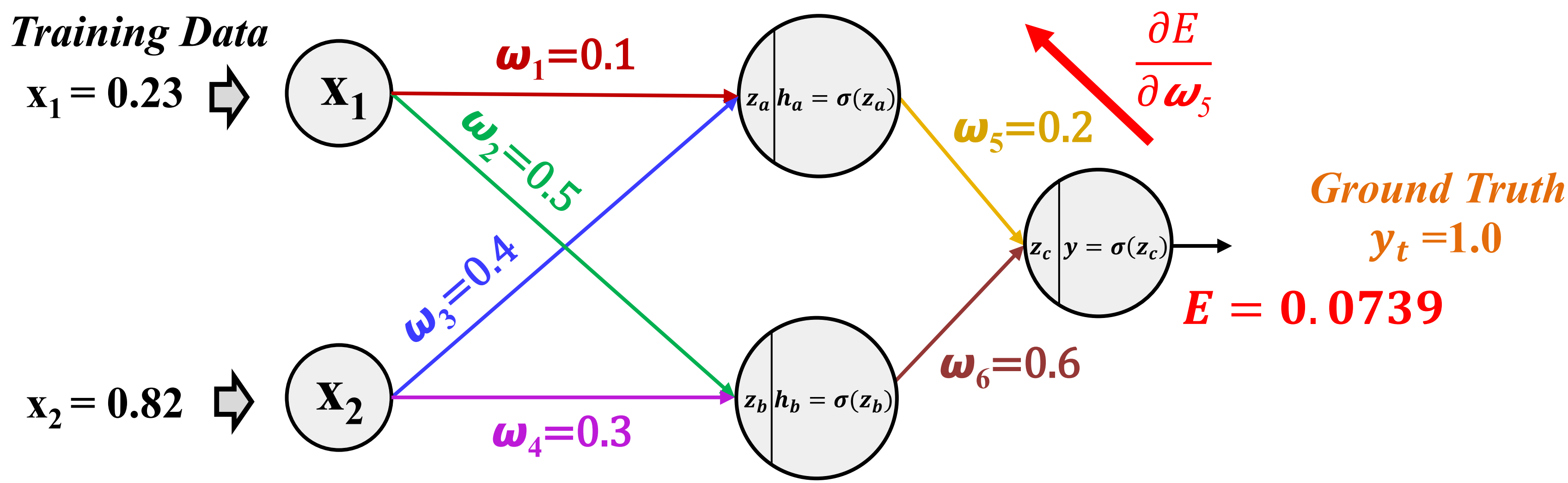
Backpropagation

2. Error Backpropagation

Sigmoid Function

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

$$\frac{\boldsymbol{d}(\sigma(\boldsymbol{z}))}{\boldsymbol{dz}} = (\boldsymbol{1} - \sigma(\boldsymbol{z}))\sigma(\boldsymbol{z})$$



$$\frac{\partial E}{\partial \boldsymbol{\omega}_5} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial z_c}\frac{\partial z_c}{\partial \boldsymbol{\omega}_5} = -0.3844 * 0.2365 * 0.5868 = -0.0533$$

Chain Rule

$$E = \frac{1}{2}(y_t - y)^2 \qquad \frac{\partial E}{\partial y} = -(y_t - y) = -(1.0 - 0.6155) = -0.3844$$

$$y = \sigma(z_c) \qquad \frac{\partial y}{\partial z_c} = \sigma(z_c)(1 - \sigma(z_c)) = 0.6155\,(1 - 0.6155) = 0.2365$$

$$z_c = h_a \boldsymbol{\omega}_5 + h_b \boldsymbol{\omega}_6 \qquad \frac{\partial z_c}{\partial \boldsymbol{\omega}_5} = h_a = 0.5868$$

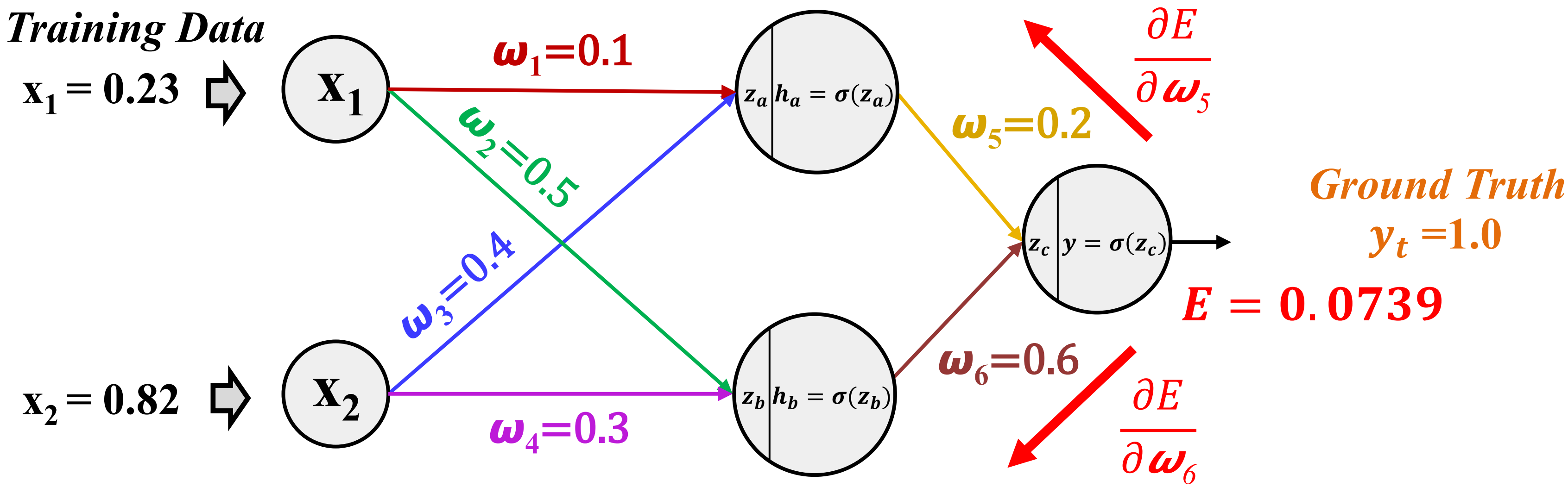
Backpropagation

2. Error Backpropagation

Sigmoid Function

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

$$\frac{d(\sigma(z))}{dz} = (1 - \sigma(z))\sigma(z)$$



$$\frac{\partial E}{\partial \omega_6} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial z_c}\frac{\partial z_c}{\partial \omega_6} = -0.3844 * 0.2365 * 0.5892 = -0.0535$$

Chain Rule

$$E = \frac{1}{2}(y_t - y)^2 \qquad \frac{\partial E}{\partial y} = -(y_t - y) = -(1.0 - 0.6155) = -0.3844$$

$$y = \sigma(z_c) \qquad \frac{\partial y}{\partial z_c} = \sigma(z_c)(1 - \sigma(z_c)) = 0.6155\,(1 - 0.6155) = 0.2365$$

$$z_c = h_a\omega_5 + h_b\omega_6 \qquad \frac{\partial z_c}{\partial \omega_6} = h_b = 0.5892$$

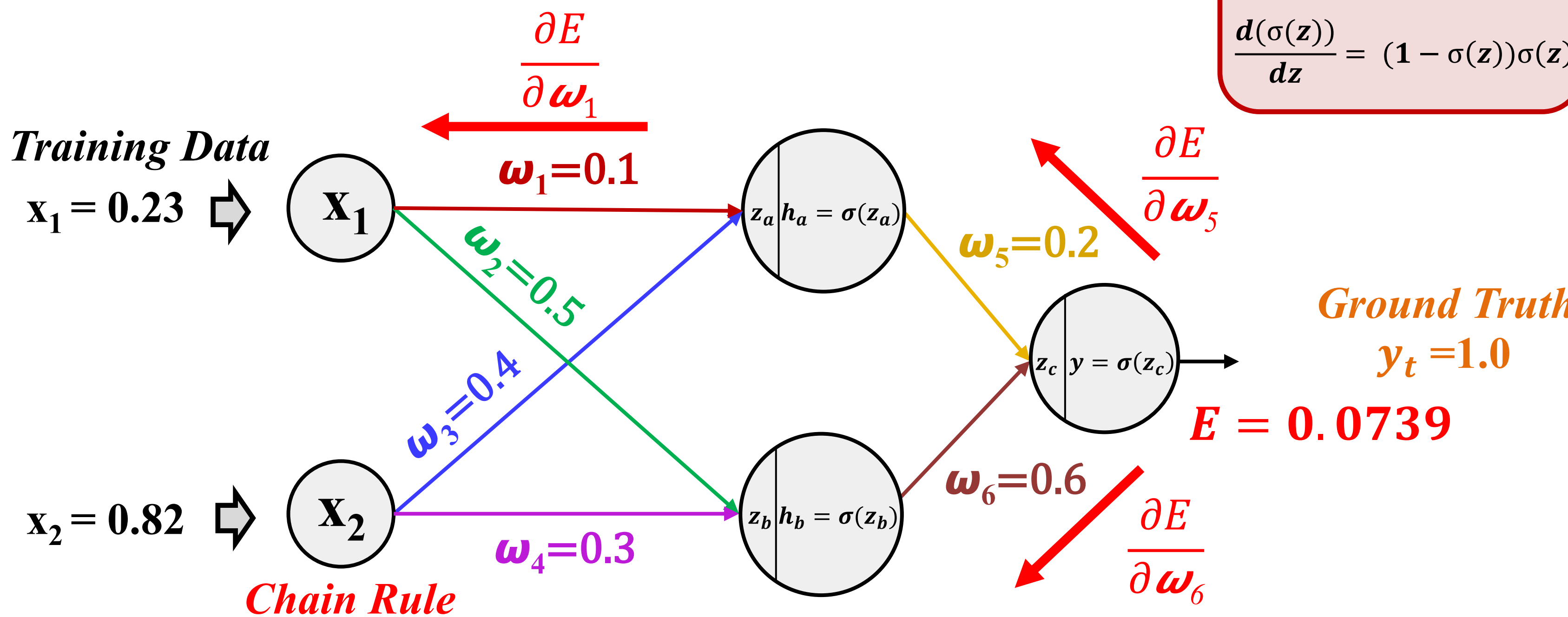
Backpropagation

2. Error Backpropagation

Sigmoid Function

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

$$\frac{d(\sigma(z))}{dz} = (1 - \sigma(z))\sigma(z)$$



Chain Rule

$$\frac{\partial E}{\partial \boldsymbol{\omega}_1} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial z_c}\frac{\partial z_c}{\partial h_a}\frac{\partial h_a}{\partial z_a}\frac{\partial z_a}{\partial \boldsymbol{\omega}_1} = -0.3844 * 0.2365 * 0.2 * 0.2424 * 0.23 = -0.0010$$

$E = \frac{1}{2}(y_t - y)^2$ $\qquad$ $\frac{\partial E}{\partial y} = -(y_t - y) = -(1.0 - 0.6155) = -0.3844$

$y = \sigma(z_c)$ $\qquad$ $\frac{\partial y}{\partial z_c} = \sigma(z_c)(1 - \sigma(z_c)) = 0.6155\,(1 - 0.6155) = 0.2365$

$z_c = h_a \boldsymbol{\omega}_5 + h_b \boldsymbol{\omega}_6$ $\qquad$ $\frac{\partial z_c}{\partial h_a} = \boldsymbol{\omega}_5 = 0.2$

$h_a = \sigma(z_a)$ $\qquad$ $\frac{\partial h_a}{\partial z_a} = \sigma(z_a)(1 - \sigma(z_a)) = 0.5868(1 - 0.5868) = 0.2424$

$z_a = x_1 \boldsymbol{\omega}_1 + x_2 \boldsymbol{\omega}_3$ $\qquad$ $\frac{\partial z_a}{\partial \boldsymbol{\omega}_1} = x_1 = 0.23$

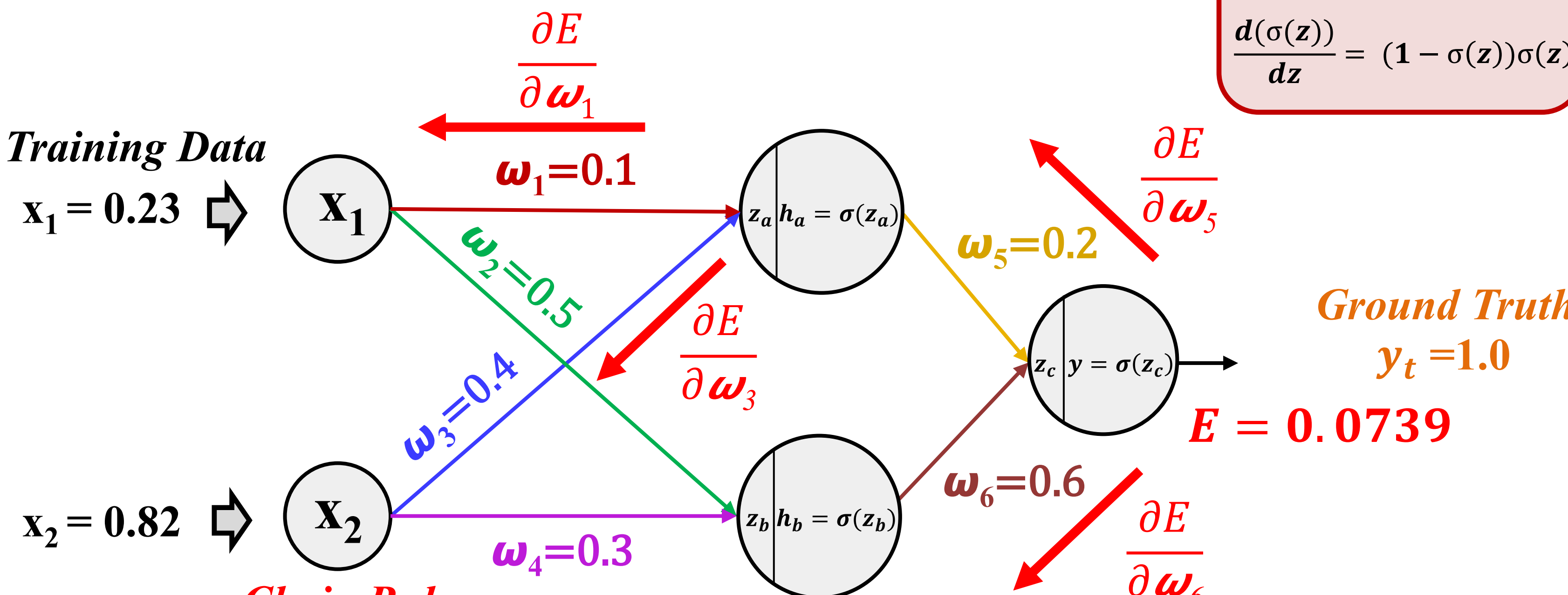
Backpropagation

2. Error Backpropagation

Sigmoid Function

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

$$\frac{d(\sigma(z))}{dz} = (1 - \sigma(z))\sigma(z)$$



Chain Rule

$$\frac{\partial E}{\partial \omega_3} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial z_c}\frac{\partial z_c}{\partial h_a}\frac{\partial h_a}{\partial z_a}\frac{\partial z_a}{\partial \omega_3} = -0.3844 * 0.2365 * 0.2 * 0.2424 * 0.82 = -0.0036$$

$E = \frac{1}{2}(y_t - y)^2$ $\qquad$ $\frac{\partial E}{\partial y} = -(y_t - y) = -(1.0 - 0.6155) = -0.3844$

$y = \sigma(z_c)$ $\qquad$ $\frac{\partial y}{\partial z_c} = \sigma(z_c)(1 - \sigma(z_c)) = 0.6155\,(1 - 0.6155) = 0.2365$

$z_c = h_a\omega_5 + h_b\omega_6$ $\qquad$ $\frac{\partial z_c}{\partial h_a} = \omega_5 = 0.2$

$h_a = \sigma(z_a)$ $\qquad$ $\frac{\partial h_a}{\partial z_a} = \sigma(z_a)(1 - \sigma(z_a)) = 0.5868(1 - 0.5868) = 0.2424$

$z_a = x_1\omega_1 + x_2\omega_3$ $\qquad$ $\frac{\partial z_a}{\partial \omega_3} = x_2 = 0.82$

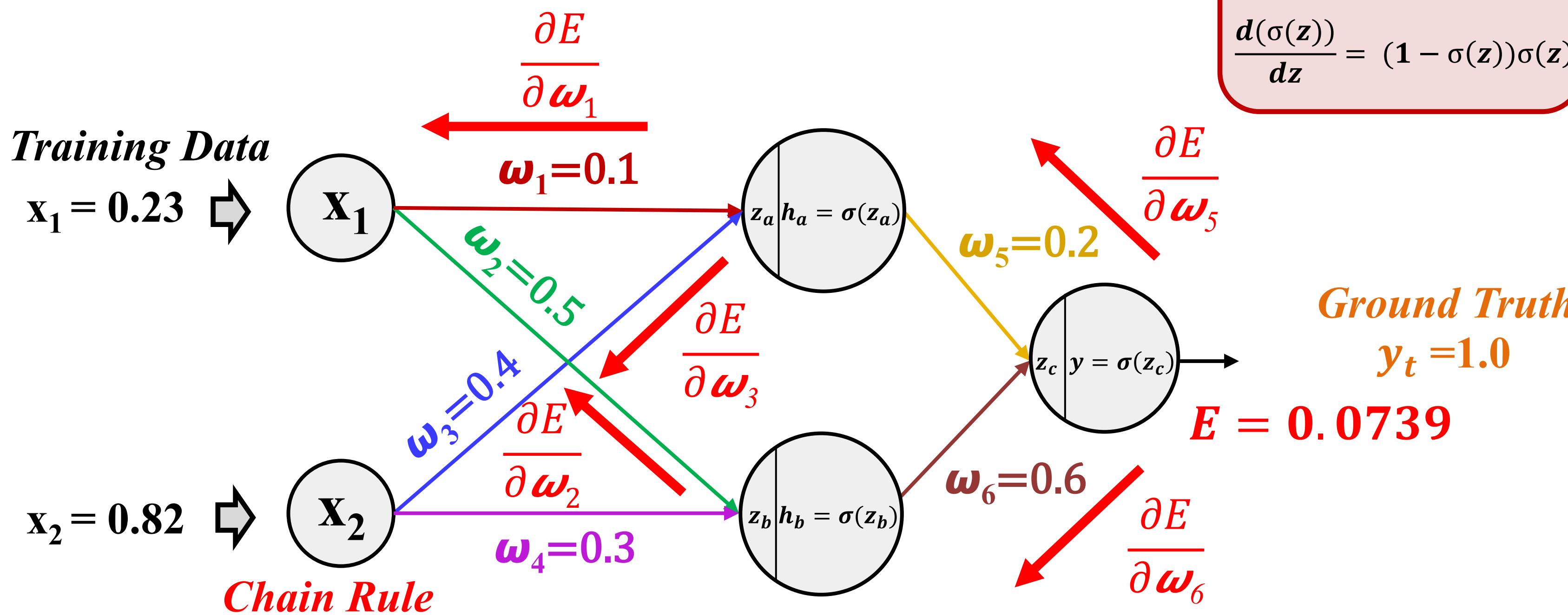
Backpropagation

2. Error Backpropagation

Sigmoid Function

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

$$\frac{\boldsymbol{d(\sigma(z))}}{\boldsymbol{dz}} = \boldsymbol{(1-\sigma(z))\sigma(z)}$$



Chain Rule

$$\frac{\partial E}{\partial \omega_2} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial z_c}\frac{\partial z_c}{\partial h_b}\frac{\partial h_b}{\partial z_b}\frac{\partial z_b}{\partial \omega_2} = -0.3844 * 0.2365 * 0.6 * 0.2420 * 0.23 = -0.0030$$

$E = \frac{1}{2}(y_t - y)^2$ $\qquad \frac{\partial E}{\partial y} = -(y_t - y) = -(1.0 - 0.6155) = -0.3844$

$y = \sigma(z_c)$ $\qquad \frac{\partial y}{\partial z_c} = \sigma(z_c)(1-\sigma(z_c)) = 0.6155\,(1-0.6155) = 0.2365$

$z_c = h_a\omega_5 + h_b\omega_6$ $\qquad \frac{\partial z_c}{\partial h_a} = \omega_6 = 0.6$

$h_b = \sigma(z_b)$ $\qquad \frac{\partial h_b}{\partial z_b} = \sigma(z_b)(1-\sigma(z_b)) = 0.5892(1-0.5892) = 0.2420$

$z_b = x_1\omega_2 + x_2\omega_4$ $\qquad \frac{\partial z_b}{\partial \omega_2} = x_1 = 0.23$

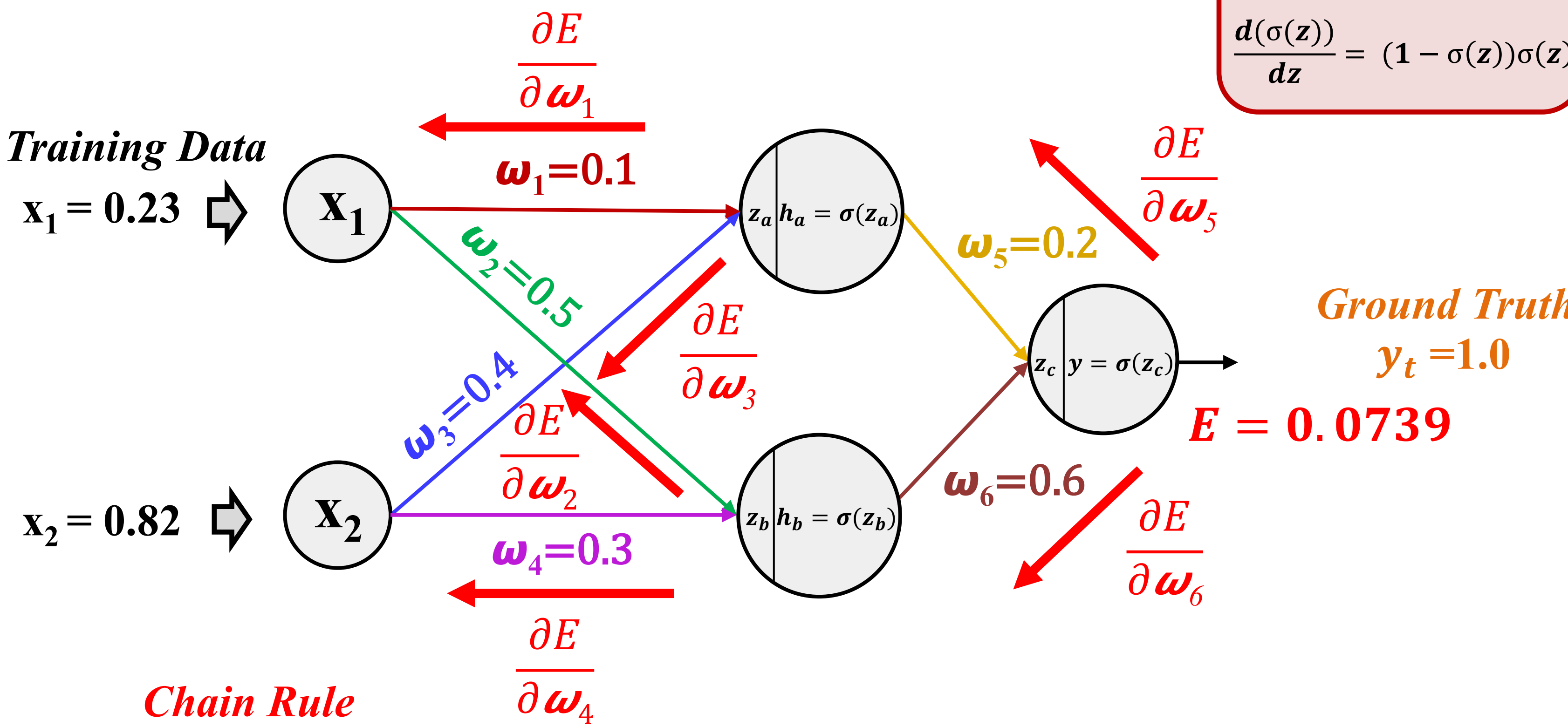
Backpropagation

2. Error Backpropagation

Sigmoid Function

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

$$\frac{d(\sigma(z))}{dz} = (1-\sigma(z))\sigma(z)$$



Chain Rule

$$\frac{\partial E}{\partial \omega_4} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial z_c}\frac{\partial z_c}{\partial h_b}\frac{\partial h_b}{\partial z_b}\frac{\partial z_b}{\partial \omega_4} = -0.3844 * 0.2365 * 0.6 * 0.2420 * 0.82 = -0.0108$$

$$\frac{\partial E}{\partial y} = -(y_t - y) = -(1.0 - 0.6155) = -0.3844$$

$$\frac{\partial y}{\partial z_c} = \sigma(z_c)(1-\sigma(z_c)) = 0.6155\,(1 - 0.6155) = 0.2365$$

$$\frac{\partial z_c}{\partial h_a} = \omega_6 = 0.6$$

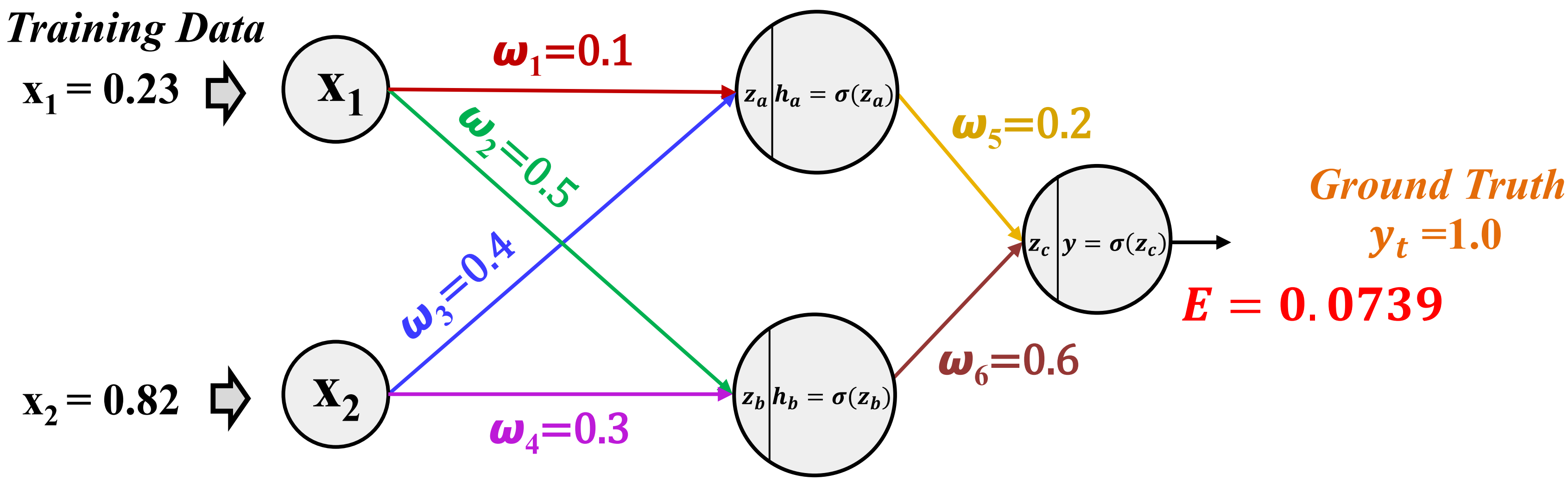
$$\frac{\partial h_b}{\partial z_b} = \sigma(z_b)(1-\sigma(z_b)) = 0.5892(1 - 0.5892) = 0.2420$$

$$\frac{\partial z_b}{\partial \omega_4} = x_2 = 0.82$$

Backpropagation

3. Parameter Updates

- Let's assume we use a ***learning rate***, $\boldsymbol{\eta}$=0.7 (Eta)



$$\omega_1^{\text{new}} = \omega_1 - \eta \frac{\partial E}{\partial \omega_1} = 0.1 - 0.7(-0.0010) = 0.1007$$

$$\omega_2^{\text{new}} = \omega_2 - \eta \frac{\partial E}{\partial \omega_2} = 0.5 - 0.7(-0.0030) = 0.5021$$

$$\omega_3^{\text{new}} = \omega_3 - \eta \frac{\partial E}{\partial \omega_3} = 0.4 - 0.7(-0.0036) = 0.4025$$

$$\omega_4^{\text{new}} = \omega_4 - \eta \frac{\partial E}{\partial \omega_4} = 0.3 - 0.7(-0.0108) = 0.3075$$

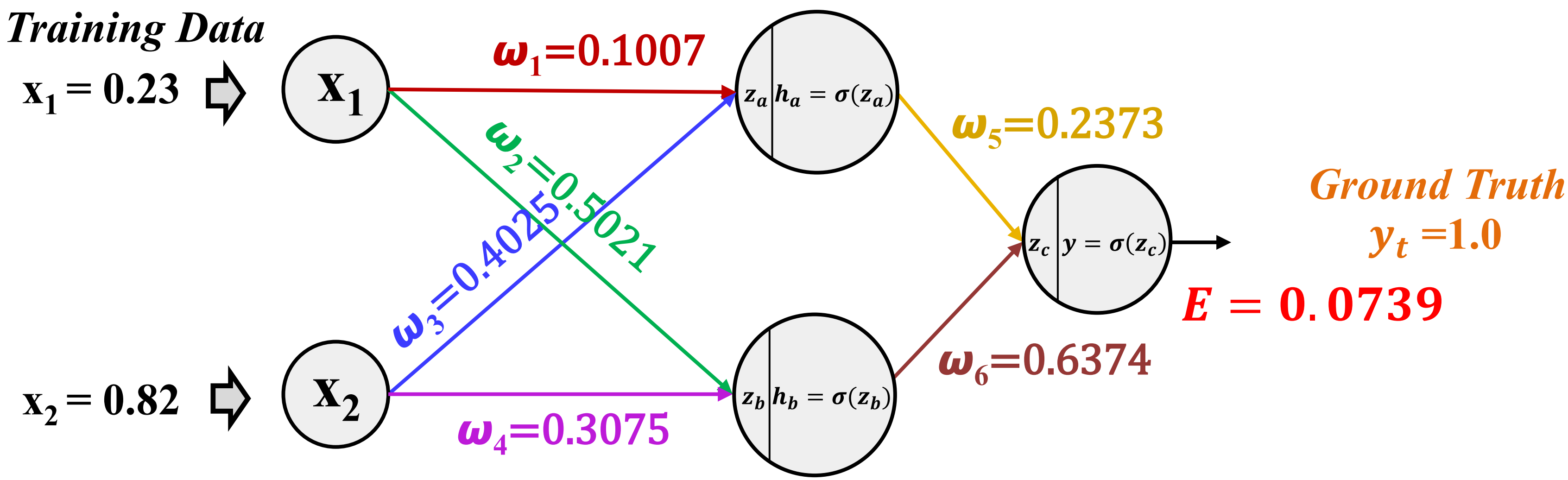
$$\omega_5^{\text{new}} = \omega_5 - \eta \frac{\partial E}{\partial \omega_5} = 0.2 - 0.7(-0.0533) = 0.2373$$

$$\omega_6^{\text{new}} = \omega_6 - \eta \frac{\partial E}{\partial \omega_6} = 0.6 - 0.7(-0.0535) = 0.6374$$

Backpropagation

3. Parameter Updates

- Let's assume we use a ***learning rate***, $\boldsymbol{\eta}$=0.7 (Eta)



$$\omega_1^{new} = \omega_1 - \eta \frac{\partial E}{\partial \omega_1} = 0.1 - 0.7(-0.0010) = 0.1007$$

$$\omega_2^{new} = \omega_2 - \eta \frac{\partial E}{\partial \omega_2} = 0.5 - 0.7(-0.0030) = 0.5021$$

$$\omega_3^{new} = \omega_3 - \eta \frac{\partial E}{\partial \omega_3} = 0.4 - 0.7(-0.0036) = 0.4025$$

$$\omega_4^{new} = \omega_4 - \eta \frac{\partial E}{\partial \omega_4} = 0.3 - 0.7(-0.0108) = 0.3075$$

$$\omega_5^{new} = \omega_5 - \eta \frac{\partial E}{\partial \omega_5} = 0.2 - 0.7(-0.0533) = 0.2373$$

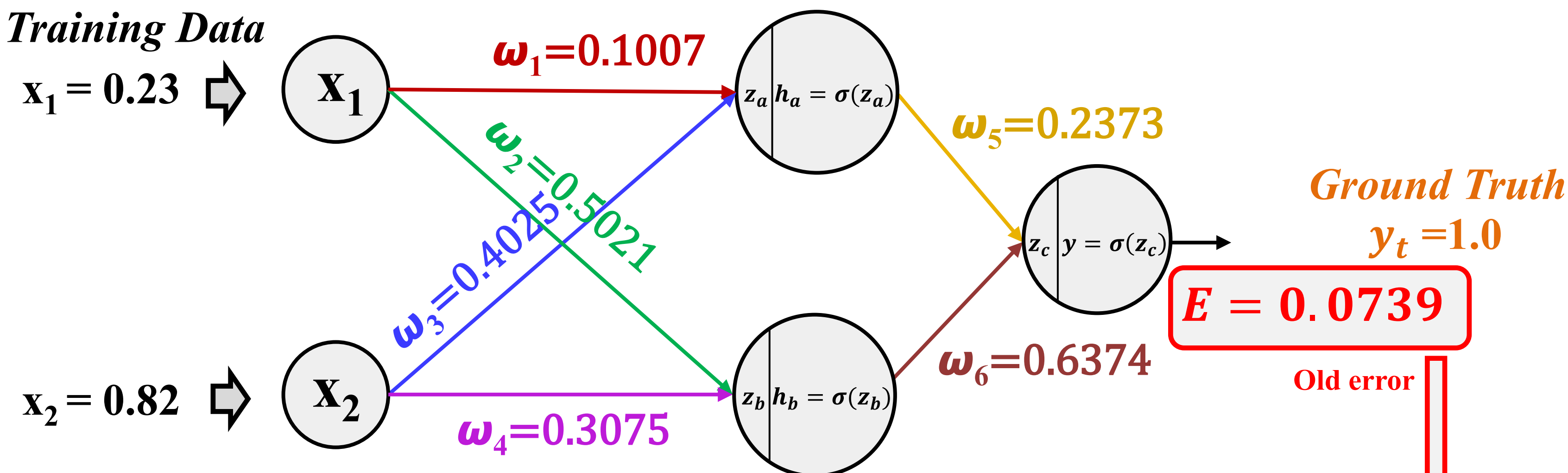
$$\omega_6^{new} = \omega_6 - \eta \frac{\partial E}{\partial \omega_6} = 0.6 - 0.7(-0.0535) = 0.6374$$

This is *stochastic gradient descent*: ***Updating the weights using backpropagation, making use of the respective gradient values.***

Backpropagation

1. Forward Pass

- Let's run ***another forward pass*** with the ***new weights*** to check that the ***error has decreased***



$z_a = x_1\omega_1 + x_2\omega_3$ =0.23*0.1007 + 0.82*0.4025 = 0.353

$h_a = \sigma(z_a)$ = σ(0.353) = 0.5873

$z_b = x_1\omega_2 + x_2\omega_4$=0.23*0.5021 + 0.82*0.3075 = 0.367

$h_b = \sigma(z_b)$ = σ(0.367) = 0.5907

$z_c = h_a\omega_5 + h_b\omega_6 = \sigma(z_a)\omega_5 + \sigma(z_b)\omega_6$
= 0.5873*0.2373 + 0.5907*0.6374 = 0.5158

$y = \sigma(z_c)$ = σ(0.5158) = 0.6261

Mean Squared Error (i.e., loss function)

$E = \frac{1}{2}(y_t - y)^2 = \frac{1}{2}(1.0 - 0.6261)^2 = \boxed{0.0699}$

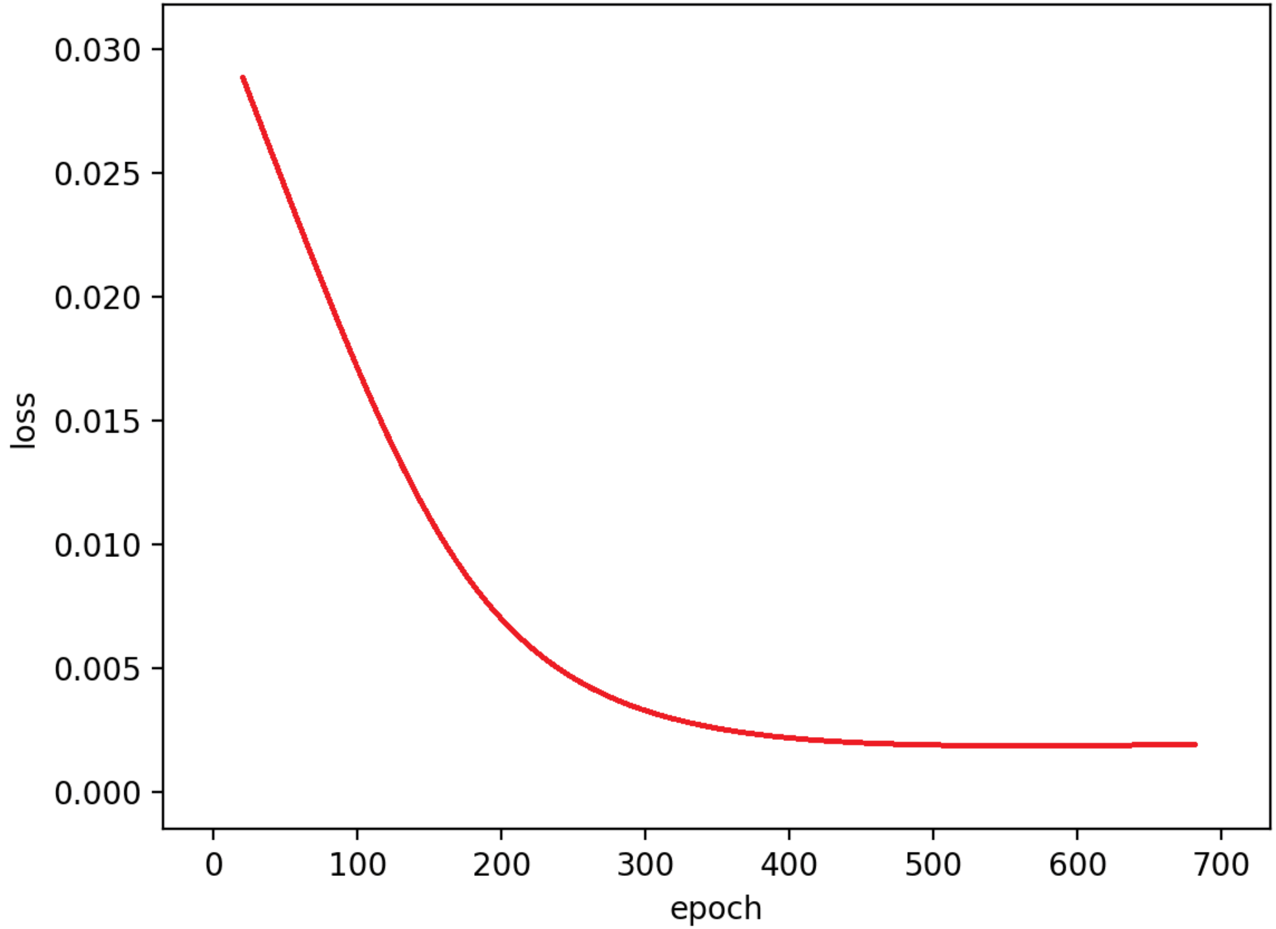
New error

This shows that the error has decreased!

Note that we have processed only *one training sample*. We repeat this process using the *entire training set over and over many times* until *the error goes down and the parameter estimates stabilize* or converge to some values.

Backpropagation

- You can visualize ***the loss plot*** over many iterations to check how it is converging.

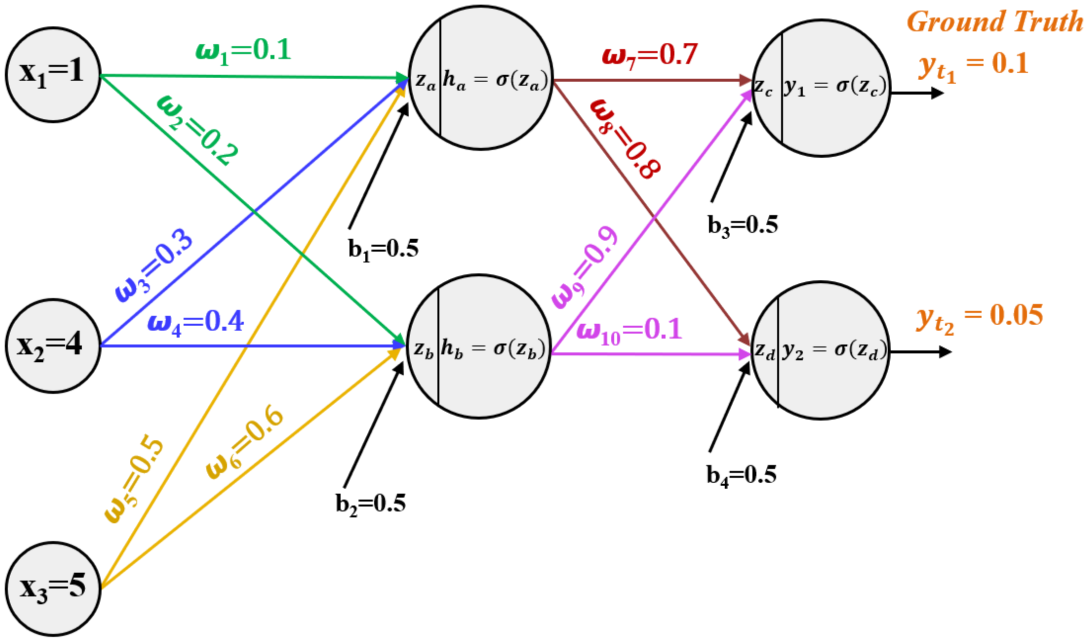


One epoch means that ***each sample in the training dataset has had an opportunity to update the internal model parameters***.

Note that we have processed only ***one training sample***. We repeat this process using the ***entire training set over and over many times*** until ***the error goes down and the parameter estimates stabilize*** or converge to some values.

Backpropagation

- Let's use a ***network with more inputs and outputs***.



$z_a = x_1\boldsymbol{\omega_1} + x_2\boldsymbol{\omega_3} + x_3\boldsymbol{\omega_5} + b_1$
=1***0.1** + 4***0.3** + 5***0.5** +0.5= 4.3

$h_a = \sigma(z_a) = \sigma(4.3) = 0.9866$

$z_b = x_1\boldsymbol{\omega_2} + x_2\boldsymbol{\omega_4} + x_3\boldsymbol{\omega_6} + b_2$
=1***0.2** + 4***0.4** + 5***0.6** +0.5= 5.3

$h_b = \sigma(z_b) = \sigma(5.3) = 0.9950$

$z_c = h_a\boldsymbol{\omega_7} + h_b\boldsymbol{\omega_9} + b_3$
= 0.9866 ***0.7** + 0.9950 ***0.9** +0.5= 2.086

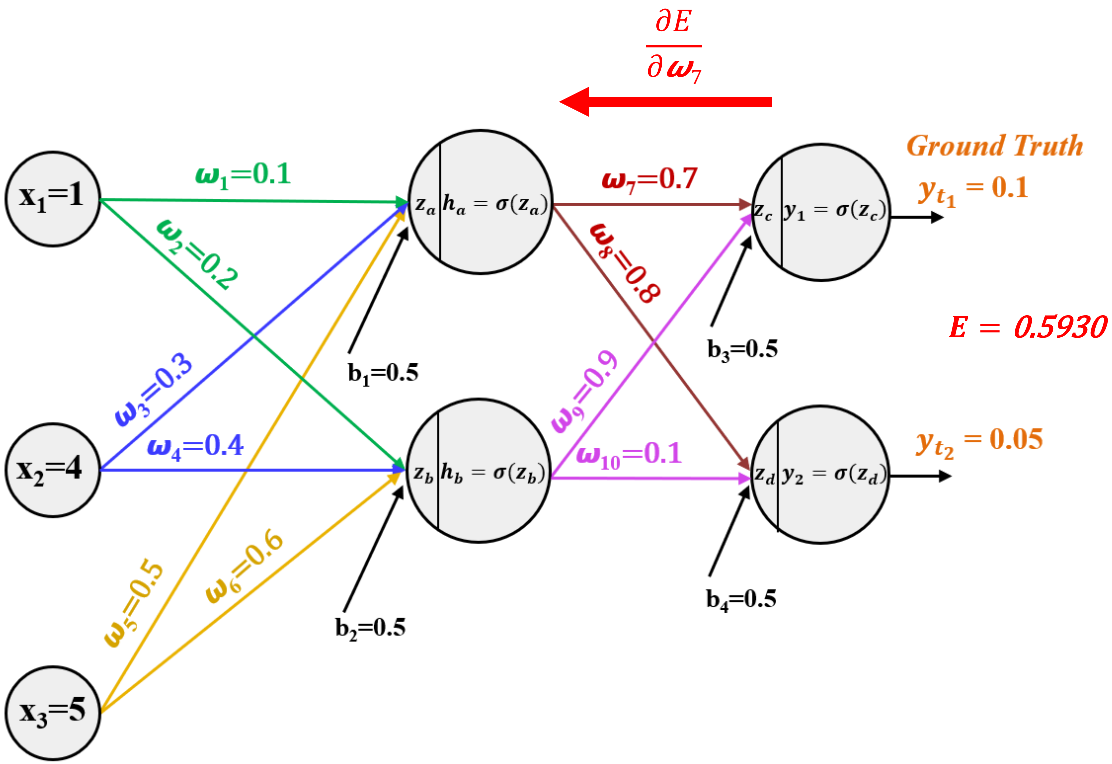
$y_1 = \sigma(z_c) = \sigma(2.086) = 0.8895$

$z_d = h_a\boldsymbol{\omega_8} + h_b\boldsymbol{\omega_{10}} + b_4$
= 0.9866 ***0.8** + 0.9950 ***0.1** +0.5= 1.388

$y_2 = \sigma(z_d) = \sigma(1.388) = 0.8002$

Mean Squared Error (i.e., loss function) ⇨ $E = \frac{1}{2}[(y_{t_1} - y_1)^2 + (y_{t_2} - y_2)^2] = \frac{1}{2}[(0.1 - 0.8895)^2 + (0.05 - 0.8002)^2] = 0.5930$

Backpropagation



$$\frac{\partial E}{\partial \boldsymbol{\omega}_7} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial z_c}\frac{\partial z_c}{\partial \boldsymbol{\omega}_7} = 0.7895 * 0.0982 * 0.9866 = 0.0764$$

$$\boldsymbol{E} = \frac{1}{2}[(\boldsymbol{y_{t_1}} - y_1)^2 + (\boldsymbol{y_{t_2}} - y_2)^2]$$

$$\frac{\partial E}{\partial y_1} = -(y_{t_1} - y_1) = -(0.1 - 0.8895) = 0.7895$$

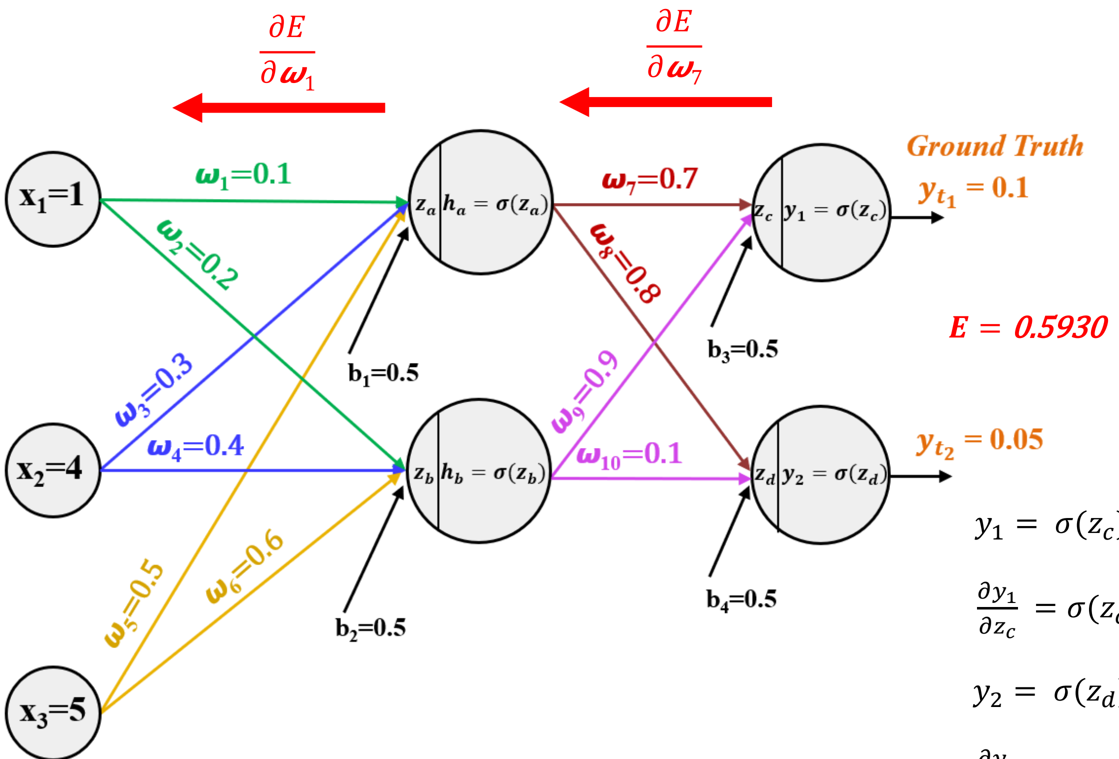
$$y_1 = \sigma(z_c)$$

$$\frac{\partial y_1}{\partial z_c} = \sigma(z_c)(1 - \sigma(z_c)) = 0.8895(1 - 0.8895) = 0.0982$$

$$z_c = h_a\boldsymbol{\omega}_7 + h_b\boldsymbol{\omega}_9 + b_3$$

$$\frac{\partial z_c}{\partial \boldsymbol{\omega}_7} = h_a = 0.9866$$

Backpropagation



$$E = \frac{1}{2}[(y_{t_1} - y_1)^2 + (y_{t_2} - y_2)^2]$$

$$\frac{\partial E}{\partial y_1} = -(y_{t_1} - y_1) = -(0.1 - 0.8895) = 0.7895$$

$$\frac{\partial E}{\partial y_2} = -(y_{t_2} - y_2) = -(0.05 - 0.8002) = 0.7502$$

$$y_1 = \sigma(z_c)$$

$$\frac{\partial y_1}{\partial z_c} = \sigma(z_c)(1 - \sigma(z_c)) = 0.8895(1 - 0.8895) = 0.0982$$

$$y_2 = \sigma(z_d)$$

$$\frac{\partial y_2}{\partial z_d} = \sigma(z_d)(1 - \sigma(z_d)) = 0.8002(1 - 0.8002) = 0.1598$$

$$z_c = h_a \boldsymbol{\omega}_7 + h_b \boldsymbol{\omega}_9 + b_3 \qquad \frac{\partial z_c}{\partial h_a} = \boldsymbol{\omega}_7 = 0.7$$

$$z_d = h_a \boldsymbol{\omega}_8 + h_b \boldsymbol{\omega}_{10} + b_4 \qquad \frac{\partial z_d}{\partial h_a} = \boldsymbol{\omega}_8 = 0.8$$

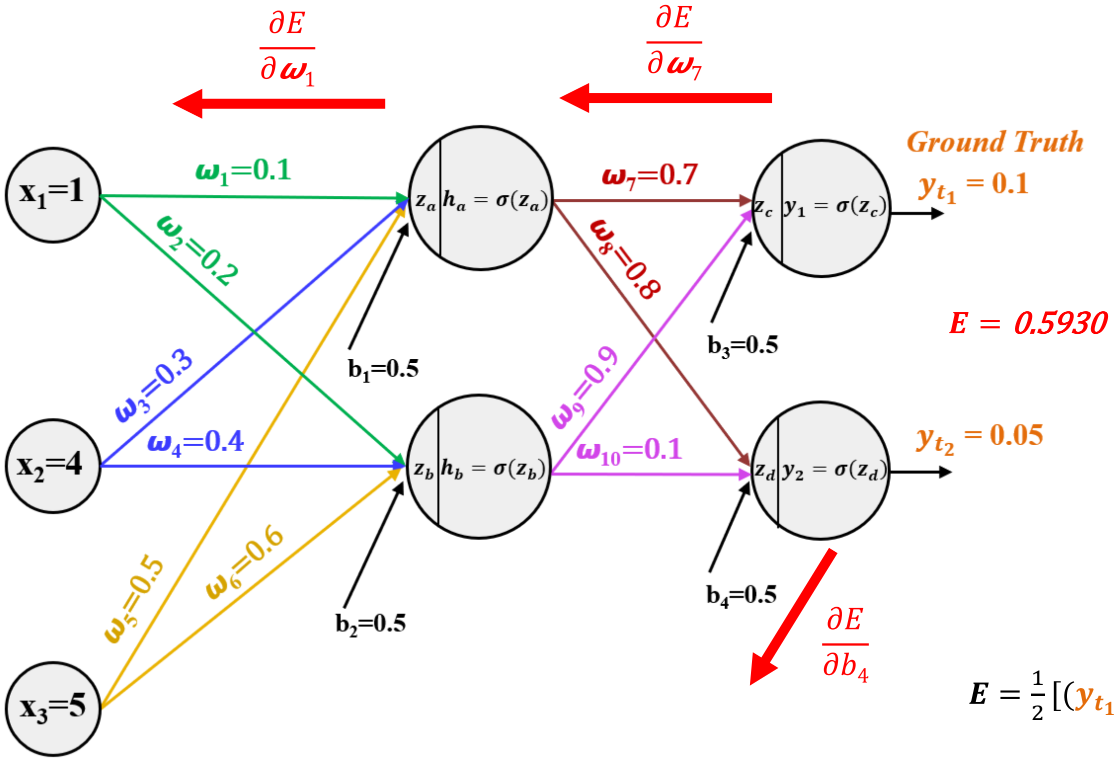
$$\frac{\partial E}{\partial \boldsymbol{\omega}_1} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial z_c}\frac{\partial z_c}{\partial h_a}\frac{\partial h_a}{\partial z_a}\frac{\partial z_a}{\partial \boldsymbol{\omega}_1} + \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial z_d}\frac{\partial z_d}{\partial h_a}\frac{\partial h_a}{\partial z_a}\frac{\partial z_a}{\partial \boldsymbol{\omega}_1}$$

$$\begin{aligned} &= 0.7895 * 0.0982 * 0.7 * 0.0132 * 1.0 \\ &+ 0.7502 * 0.1598 * 0.8 * 0.0132 * 1.0 \\ &= 0.0012 \end{aligned}$$

$$h_a = \sigma(z_a) \qquad \frac{\partial h_a}{\partial z_a} = \sigma(z_a)(1 - \sigma(z_a)) = 0.9866(1 - 0.9866) = 0.0132$$

$$z_a = x_1 \boldsymbol{\omega}_1 + x_2 \boldsymbol{\omega}_3 + x_3 \boldsymbol{\omega}_5 + b_1 \qquad \frac{\partial z_a}{\partial \boldsymbol{\omega}_1} = x_1 = 1$$

Backpropagation



$$\boldsymbol{E} = \frac{1}{2}[(\boldsymbol{y_{t_1}} - y_1)^2 + (\boldsymbol{y_{t_2}} - y_2)^2]$$

$$\frac{\partial E}{\partial y_2} = -(y_{t_2} - y_2) = -(0.05 - 0.8002) = 0.7502$$

$$y_2 = \sigma(z_d)$$

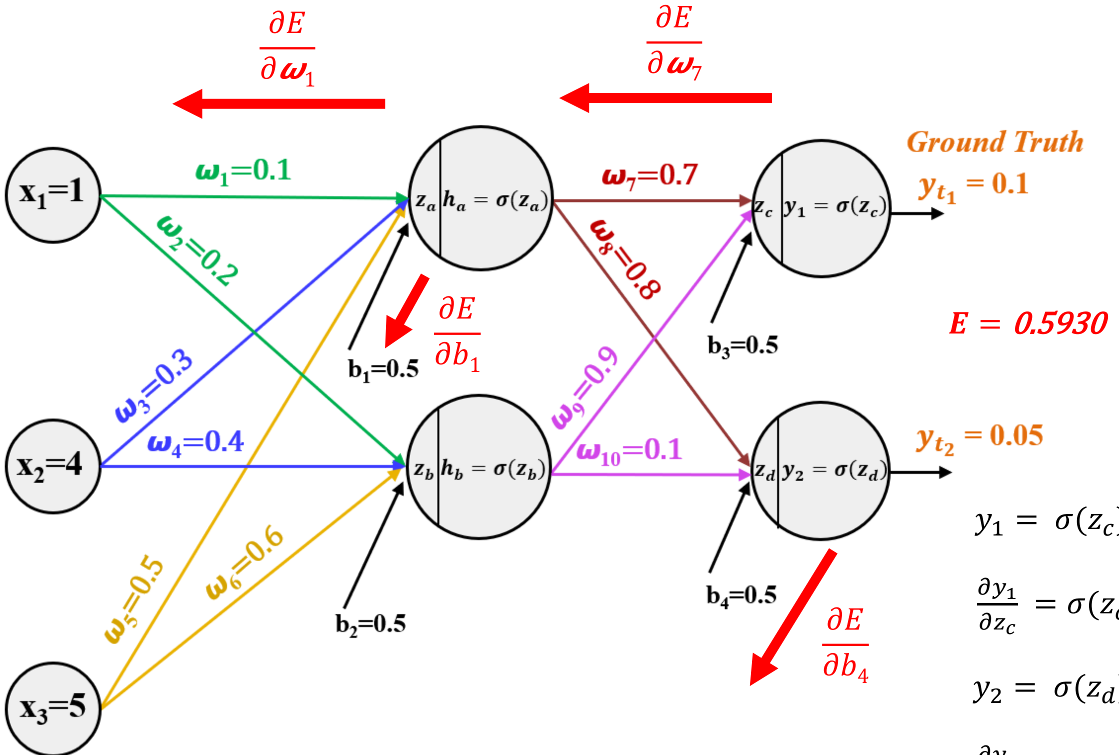
$$\frac{\partial y_2}{\partial z_d} = \sigma(z_d)(1 - \sigma(z_d)) = 0.8002(1 - 0.8002) = 0.1598$$

$$z_d = h_a \boldsymbol{\omega_8} + h_b \boldsymbol{\omega_{10}} + b_4 \qquad \frac{\partial z_d}{\partial b_4} = 1$$

$$\frac{\partial E}{\partial b_4} = \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial z_d}\frac{\partial z_d}{\partial b_4}$$

$$= 0.7502 * 0.1598 * 1 = 0.1198$$

Backpropagation



$$E = \frac{1}{2}[(y_{t_1} - y_1)^2 + (y_{t_2} - y_2)^2]$$

$$\frac{\partial E}{\partial y_1} = -(y_{t_1} - y_1) = -(0.1 - 0.8895) = 0.7895$$

$$\frac{\partial E}{\partial y_2} = -(y_{t_2} - y_2) = -(0.05 - 0.8002) = 0.7502$$

$$y_1 = \sigma(z_c)$$

$$\frac{\partial y_1}{\partial z_c} = \sigma(z_c)(1 - \sigma(z_c)) = 0.8895(1 - 0.8895) = 0.0982$$

$$y_2 = \sigma(z_d)$$

$$\frac{\partial y_2}{\partial z_d} = \sigma(z_d)(1 - \sigma(z_d)) = 0.8002(1 - 0.8002) = 0.1598$$

$$\frac{\partial E}{\partial b_l} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial z_c}\frac{\partial z_c}{\partial h_a}\frac{\partial h_a}{\partial z_a}\frac{\partial z_a}{\partial b_1} + \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial z_d}\frac{\partial z_d}{\partial h_a}\frac{\partial h_a}{\partial z_a}\frac{\partial z_a}{\partial b_1}$$

$$= 0.7895 * 0.0982 * 0.7 * 0.0132 * 1.0$$
$$+ 0.7502 * 0.1598 * 0.8 * 0.0132 * 1.0$$
$$= 0.0012$$

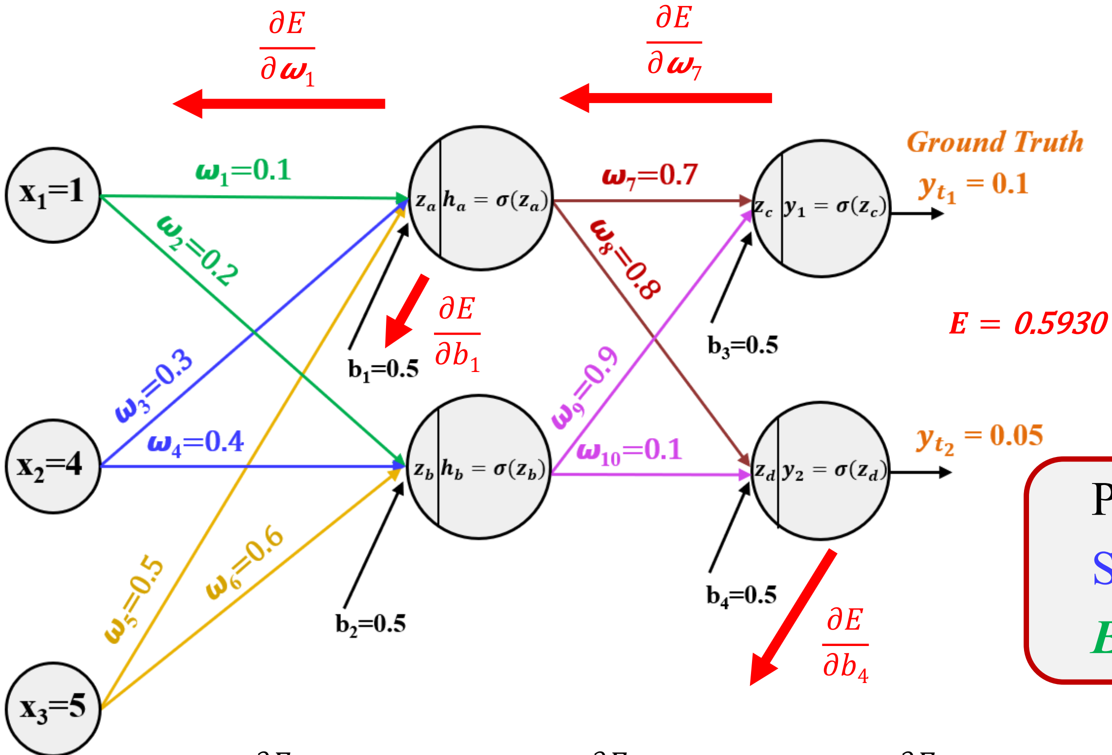
$$z_c = h_a \boldsymbol{\omega}_7 + h_b \boldsymbol{\omega}_9 + b_3 \qquad \frac{\partial z_c}{\partial h_a} = \boldsymbol{\omega}_7 = 0.7$$

$$z_d = h_a \boldsymbol{\omega}_8 + h_b \boldsymbol{\omega}_{10} + b_4 \qquad \frac{\partial z_d}{\partial h_a} = \boldsymbol{\omega}_8 = 0.8$$

$$h_a = \sigma(z_a) \qquad \frac{\partial h_a}{\partial z_a} = \sigma(z_a)(1 - \sigma(z_a)) = 0.9866(1 - 0.9866) = 0.0132$$

$$z_a = x_1 \boldsymbol{\omega}_1 + x_2 \boldsymbol{\omega}_3 + x_3 \boldsymbol{\omega}_5 + b_1 \qquad \frac{\partial z_a}{\partial b_1} = 1$$

Backpropagation



Please, *compute the rest yourself* !

See [this link](#) for additional help.

Expected final results:

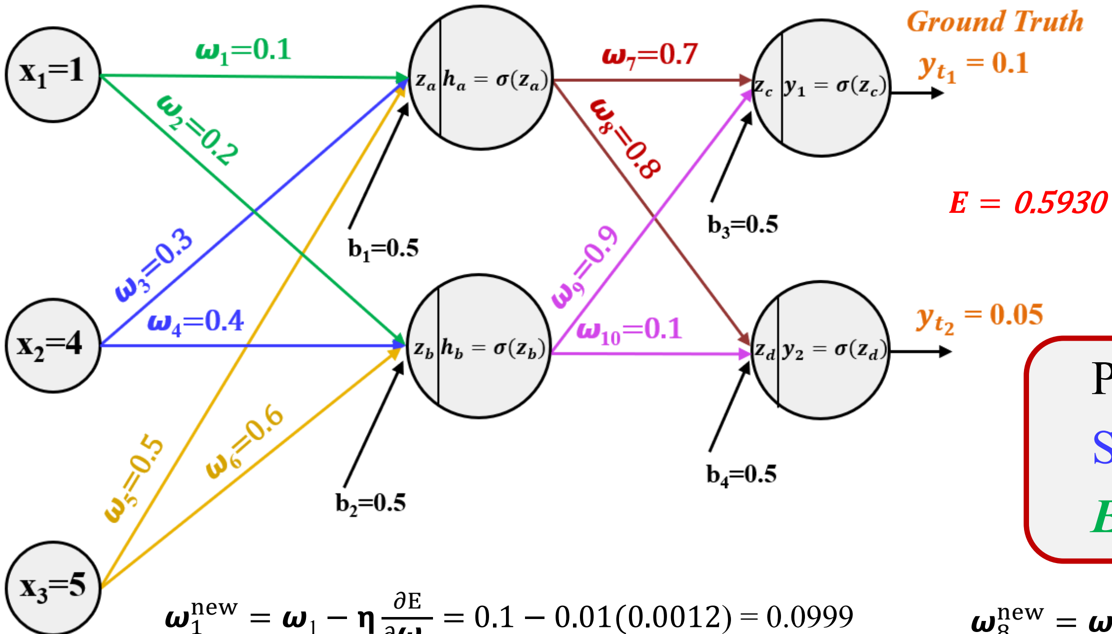
$$\frac{\partial E}{\partial \boldsymbol{\omega}_1} = 0.0012 \qquad \frac{\partial E}{\partial \boldsymbol{\omega}_4} = 0.0002 \qquad \frac{\partial E}{\partial \boldsymbol{\omega}_7} = 0.0764 \qquad \frac{\partial E}{\partial \boldsymbol{\omega}_{10}} = 0.1192 \qquad \frac{\partial E}{\partial b_3} = 0.0775$$

$$\frac{\partial E}{\partial \boldsymbol{\omega}_2} = 0.00005 \qquad \frac{\partial E}{\partial \boldsymbol{\omega}_5} = 0.0063 \qquad \frac{\partial E}{\partial \boldsymbol{\omega}_8} = 0.1182 \qquad \frac{\partial E}{\partial b_1} = 0.0012 \qquad \frac{\partial E}{\partial b_4} = 0.1198$$

$$\frac{\partial E}{\partial \boldsymbol{\omega}_3} = 0.0050 \qquad \frac{\partial E}{\partial \boldsymbol{\omega}_6} = 0.00029 \qquad \frac{\partial E}{\partial \boldsymbol{\omega}_9} = 0.0771 \qquad \frac{\partial E}{\partial b_2} = 0.00005$$

Backpropagation

- Let's assume we use a ***learning rate***, $\boldsymbol{\eta}$=0.01



Please, *compute the rest yourself*!
See this link for additional help.
Expected final results:

$$\boldsymbol{\omega}_1^{new} = \boldsymbol{\omega}_1 - \boldsymbol{\eta}\frac{\partial E}{\partial \boldsymbol{\omega}_1} = 0.1 - 0.01(0.0012) = 0.0999$$

$$\boldsymbol{\omega}_2^{new} = \boldsymbol{\omega}_2 - \boldsymbol{\eta}\frac{\partial E}{\partial \boldsymbol{\omega}_2} = 0.2 - 0.01(0.00005) = 0.1999$$

$$\boldsymbol{\omega}_3^{new} = \boldsymbol{\omega}_3 - \boldsymbol{\eta}\frac{\partial E}{\partial \boldsymbol{\omega}_3} = 0.3 - 0.01(0.0050) = 0.2999$$

$$\boldsymbol{\omega}_4^{new} = \boldsymbol{\omega}_4 - \boldsymbol{\eta}\frac{\partial E}{\partial \boldsymbol{\omega}_4} = 0.4 - 0.01(0.0002) = 0.3999$$

$$\boldsymbol{\omega}_5^{new} = \boldsymbol{\omega}_5 - \boldsymbol{\eta}\frac{\partial E}{\partial \boldsymbol{\omega}_5} = 0.5 - 0.01(0.0063) = 0.4999$$

$$\boldsymbol{\omega}_6^{new} = \boldsymbol{\omega}_6 - \boldsymbol{\eta}\frac{\partial E}{\partial \boldsymbol{\omega}_6} = 0.6 - 0.01(0.00029) = 0.5999$$

$$\boldsymbol{\omega}_7^{new} = \boldsymbol{\omega}_7 - \boldsymbol{\eta}\frac{\partial E}{\partial \boldsymbol{\omega}_7} = 0.7 - 0.01(0.0764) = 0.6992$$

$$\boldsymbol{\omega}_8^{new} = \boldsymbol{\omega}_8 - \boldsymbol{\eta}\frac{\partial E}{\partial \boldsymbol{\omega}_8} = 0.8 - 0.01(0.1182) = 0.7988$$

$$\boldsymbol{\omega}_9^{new} = \boldsymbol{\omega}_9 - \boldsymbol{\eta}\frac{\partial E}{\partial \boldsymbol{\omega}_9} = 0.9 - 0.01(0.0771) = 0.8992$$

$$\boldsymbol{\omega}_{10}^{new} = \boldsymbol{\omega}_{10} - \boldsymbol{\eta}\frac{\partial E}{\partial \boldsymbol{\omega}_{10}} = 0.1 - 0.01(0.1192) = 0.0988$$

$$b_1^{new} = b_1 - \boldsymbol{\eta}\frac{\partial E}{\partial b_1} = 0.5 - 0.01(0.0012) = 0.4999$$

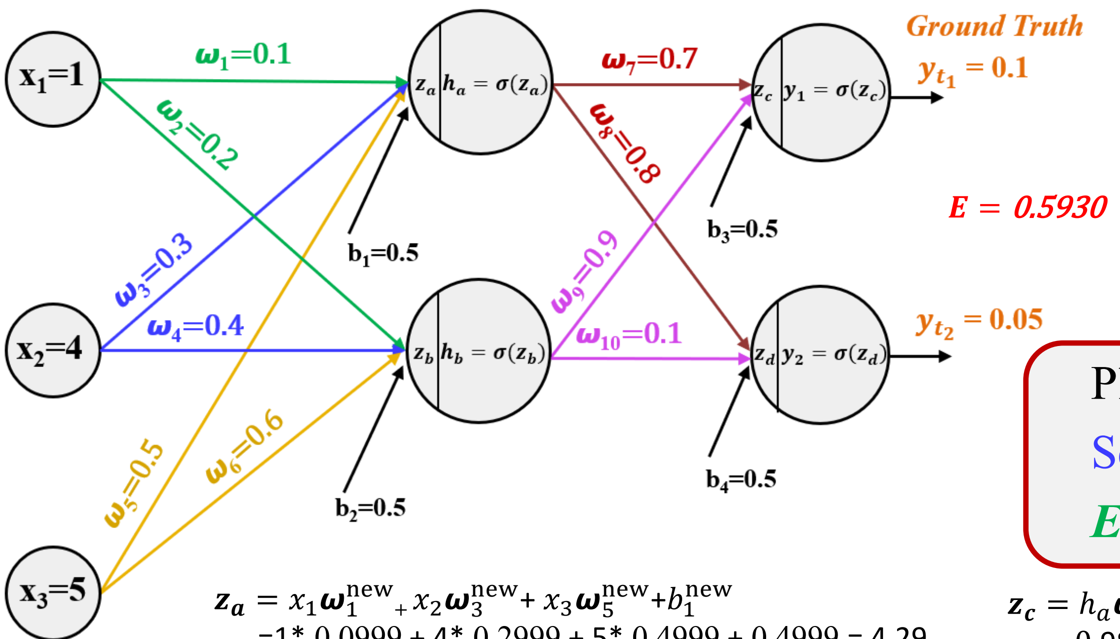
$$b_2^{new} = b_2 - \boldsymbol{\eta}\frac{\partial E}{\partial b_2} = 0.5 - 0.01(0.00005) = 0.4999$$

$$b_3^{new} = b_3 - \boldsymbol{\eta}\frac{\partial E}{\partial b_3} = 0.5 - 0.01(0.0775) = 0.4992$$

$$b_4^{new} = b_4 - \boldsymbol{\eta}\frac{\partial E}{\partial b_4} = 0.5 - 0.01(0.1198) = 0.4988$$

Backpropagation

- Let's run ***another forward pass*** with the ***new weights*** to check that the ***error has decreased***



Please, *compute the rest yourself*!

See this link for additional help.

Expected final results:

$\boldsymbol{z_a} = x_1\boldsymbol{\omega}_1^{\text{new}} + x_2\boldsymbol{\omega}_3^{\text{new}} + x_3\boldsymbol{\omega}_5^{\text{new}} + b_1^{\text{new}}$
=1* 0.0999 + 4* 0.2999 + 5* 0.4999 + 0.4999 = 4.29

$\boldsymbol{h_a} = \boldsymbol{\sigma}(\boldsymbol{z_a})$ = σ(4.29) = 0.9864

$\boldsymbol{z_b} = x_1\boldsymbol{\omega}_2^{\text{new}} + x_2\boldsymbol{\omega}_4^{\text{new}} + x_3\boldsymbol{\omega}_6^{\text{new}} + b_2^{\text{new}}$
=1*0.1999 + 4*0.3999 + 5*0.5999 + 0.4999 = 5.2989

$\boldsymbol{h_b} = \boldsymbol{\sigma}(\boldsymbol{z_b})$ = σ(5.2989) = 0.9950

$\boldsymbol{z_c} = h_a\boldsymbol{\omega}_7^{\text{new}} + h_b\boldsymbol{\omega}_9^{\text{new}} + b_3^{\text{new}}$
= 0.9864 * 0.6992 + 0.9950 * 0.8992 + 0.4992 = 2.083

$\boldsymbol{y_1} = \boldsymbol{\sigma}(\boldsymbol{z_c})$ = σ(2.083) = 0.8892

$\boldsymbol{z_d} = h_a\boldsymbol{\omega}_8^{\text{new}} + h_b\boldsymbol{\omega}_{10}^{\text{new}} + b_4^{\text{new}}$
= 0.9864 * 0.7988 + 0.9950 * 0.0988 + 0.4988 = 1.385

$\boldsymbol{y_2} = \boldsymbol{\sigma}(\boldsymbol{z_d})$ = σ(1.385) = 0.7997

Mean Squared Error (i.e., loss function)

$$\boldsymbol{E} = \frac{1}{2}[(y_{t_1} - y_1)^2 + (y_{t_2} - y_2)^2] = \frac{1}{2}[(0.1 - 0.8892)^2 + (0.05 - 0.7997)^2] = \mathbf{0.5924}$$

Backpropagation is not Learning

- ***Backpropagation often misunderstood*** as the ***whole learning algorithm*** for multilayer networks

- ***Backpropagation*** only refers to method of ***computing gradient*** for intermediate layers

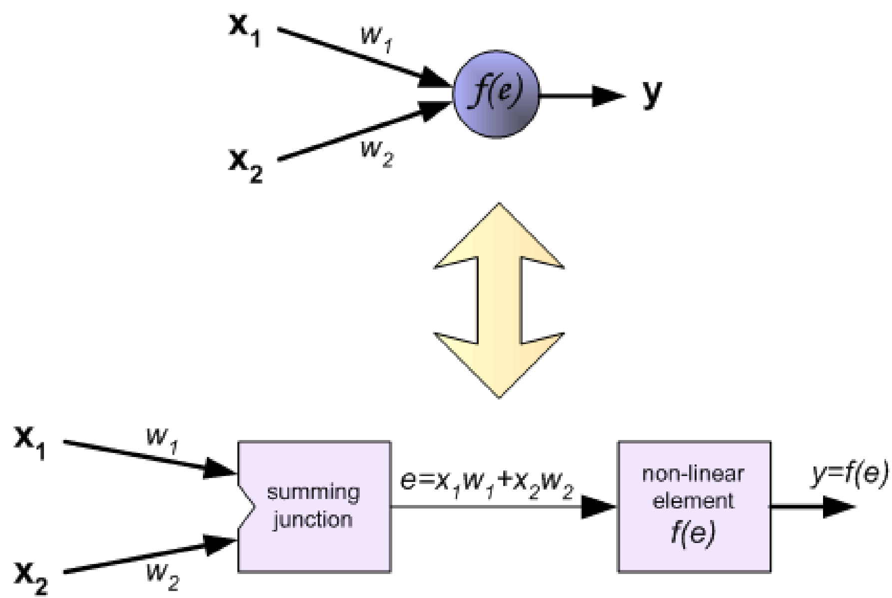
- Another algorithm, e.g., ***SGD***, is used to perform learning using this gradient
  - Learning is updating weights using gradient:

$$\boldsymbol{\omega}_i^{new} = \boldsymbol{\omega}_i - \boldsymbol{\eta} \frac{\partial E}{\partial \boldsymbol{\omega}_i}$$

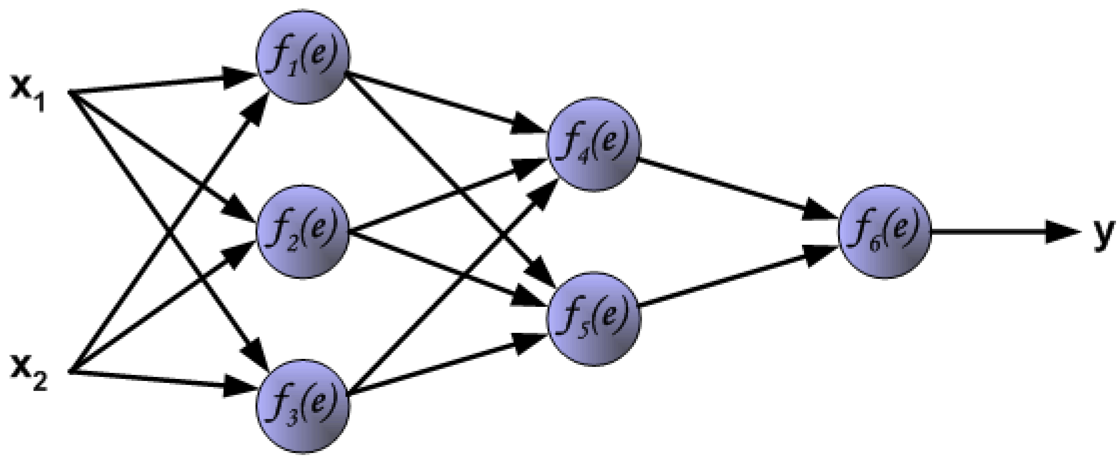
- ***Backpropagation*** is also misunderstood to being specific to multilayer neural networks
  - It can be used to compute derivatives for any function

Network Training

- Let's have a ***three-layer*** neural network with ***two inputs*** and ***one output***



- Each neuron has ***two units***:
  1. ***Adding products*** of weights coefficients and input signals.
  2. ***A nonlinear activation function***.

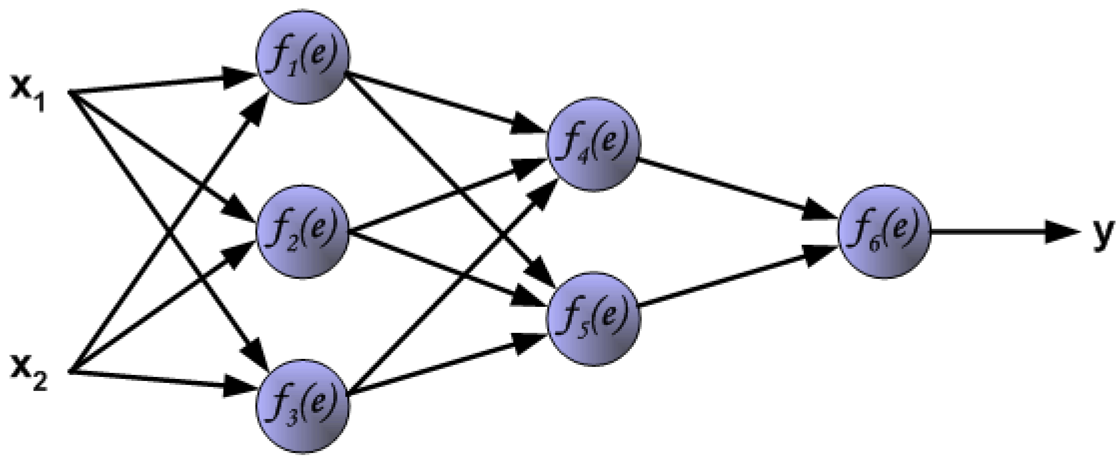


[Click to go to the source!]

[Mariusz Bernacki]
[Przemysław Włodarczyk]
mgr inż. Adam Gołda (2005)
Katedra Elektroniki AGH

Network Training

- Symbols $w_{(xm)n}$ represent ***weights of connections*** between ***network input*** x_m and ***neuron n in the input layer***. Symbols y_n represents the output signal of ***neuron n.***

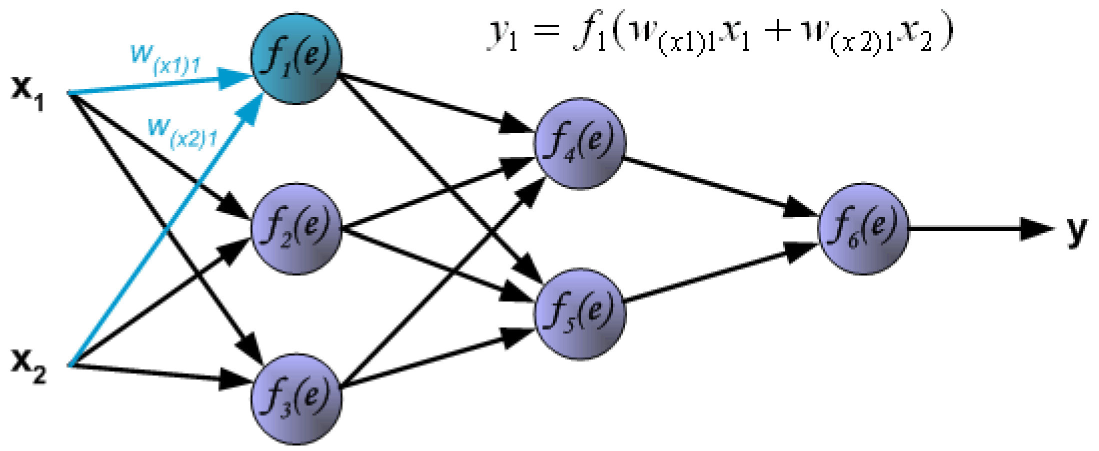


[Click to go to the source!]

[Mariusz Bernacki]
[Przemysław Włodarczyk]
mgr inż. Adam Gołda (2005)
Katedra Elektroniki AGH

Network Training – Forward Pass

- Symbols $w_{(xm)n}$ represent ***weights of connections*** between ***network input*** x_m and ***neuron n in the input layer***. Symbols y_n represents the output signal of ***neuron n.***
- ***Propagation*** of signals ***through the hidden layer.***

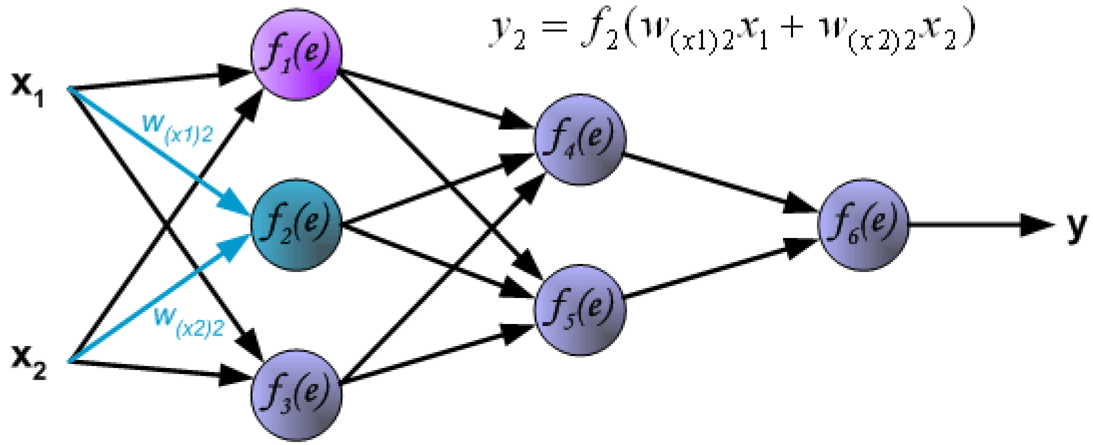


Click to go to the source!

Mariusz Bernacki
Przemysław Włodarczyk
mgr inż. Adam Gołda (2005)
Katedra Elektroniki AGH

Network Training – Forward Pass

- Symbols $w_{(xm)n}$ represent ***weights of connections*** between ***network input*** x_m and ***neuron n in the input layer***. Symbols y_n represents the output signal of ***neuron n.***
- ***Propagation*** of signals ***through the hidden layer.***

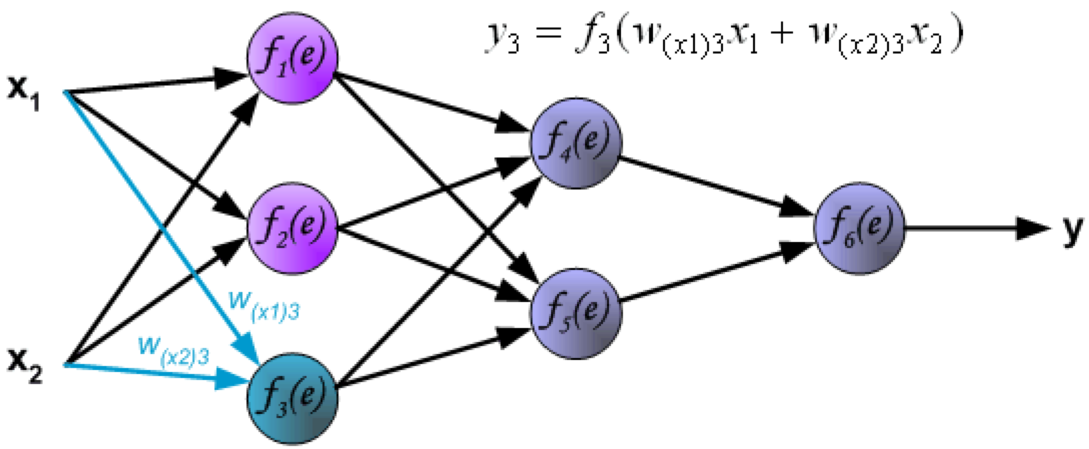


[Click to go to the source!]

Mariusz Bernacki
Przemysław Włodarczyk
mgr inż. Adam Gołda (2005)
Katedra Elektroniki AGH

Network Training – Forward Pass

- Symbols $w_{(xm)n}$ represent ***weights of connections*** between ***network input*** x_m and ***neuron n in the input layer***. Symbols y_n represents the output signal of ***neuron n.***
- ***Propagation*** of signals ***through the hidden layer.***

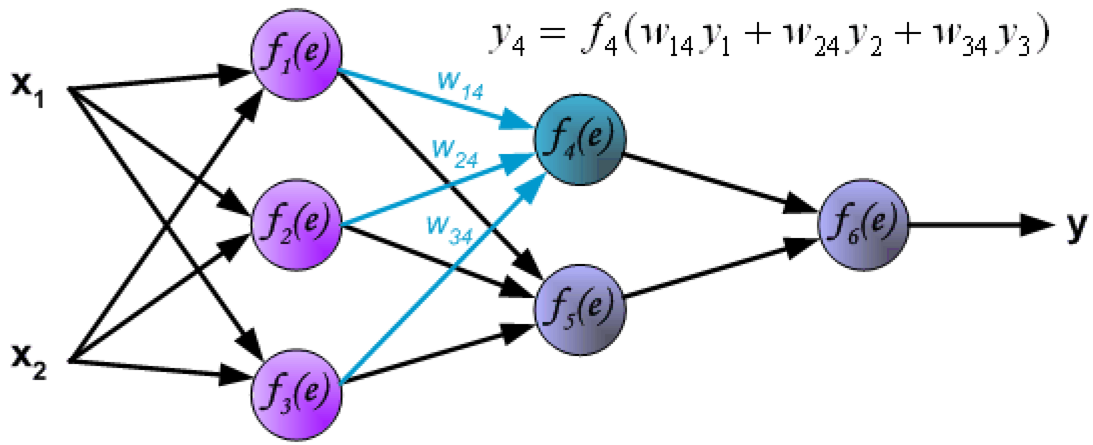


[Click to go to the source!]

[Mariusz Bernacki]
[Przemysław Włodarczyk]
mgr inż. Adam Gołda (2005)
Katedra Elektroniki AGH

Network Training – Forward Pass

- Symbols $w_{(xm)n}$ represent ***weights of connections*** between ***network input*** x_m and ***neuron n in the input layer***. Symbols y_n represents the output signal of ***neuron n.***
- ***Propagation*** of signals ***through the hidden layer.***

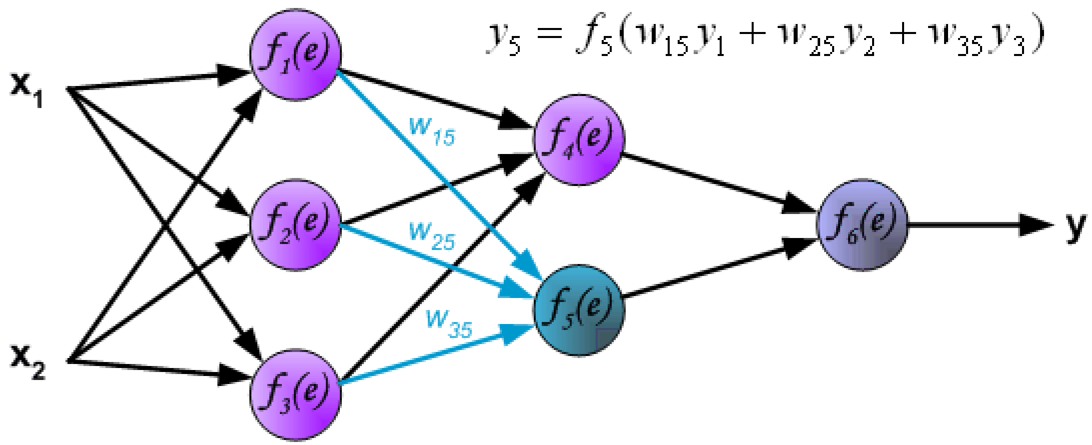


Click to go to the source!

Mariusz Bernacki
Przemysław Włodarczyk
mgr inż. Adam Gołda (2005)
Katedra Elektroniki AGH

Network Training – Forward Pass

- Symbols $w_{(xm)n}$ represent ***weights of connections*** between ***network input*** x_m and ***neuron n in the input layer***. Symbols y_n represents the output signal of ***neuron n.***
- ***Propagation*** of signals ***through the hidden layer.***

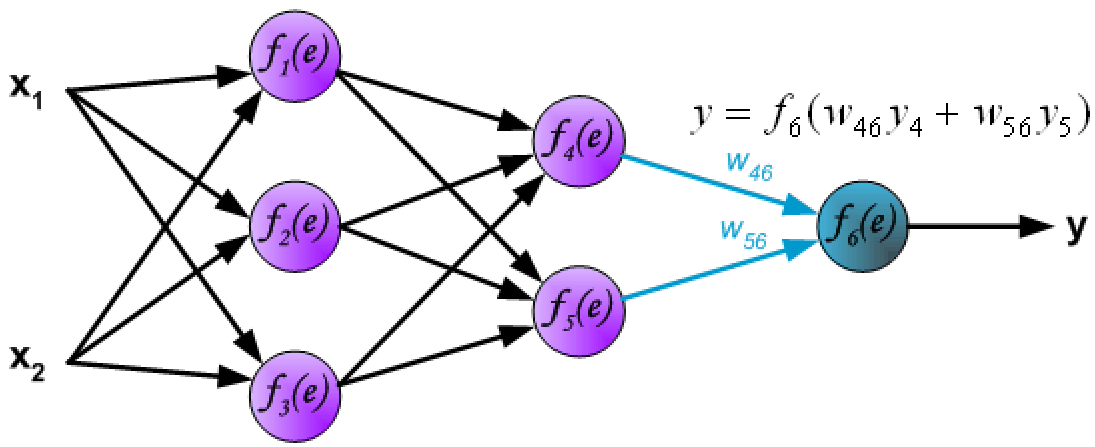


Click to go to the source!

Mariusz Bernacki
Przemysław Włodarczyk
mgr inż. Adam Gołda (2005)
Katedra Elektroniki AGH

Network Training – Forward Pass

- Symbols $w_{(xm)n}$ represent ***weights of connections*** between ***network input*** x_m and ***neuron n in the input layer***. Symbols y_n represents the output signal of ***neuron n.***
- ***Propagation*** of signals ***through the hidden layer.***



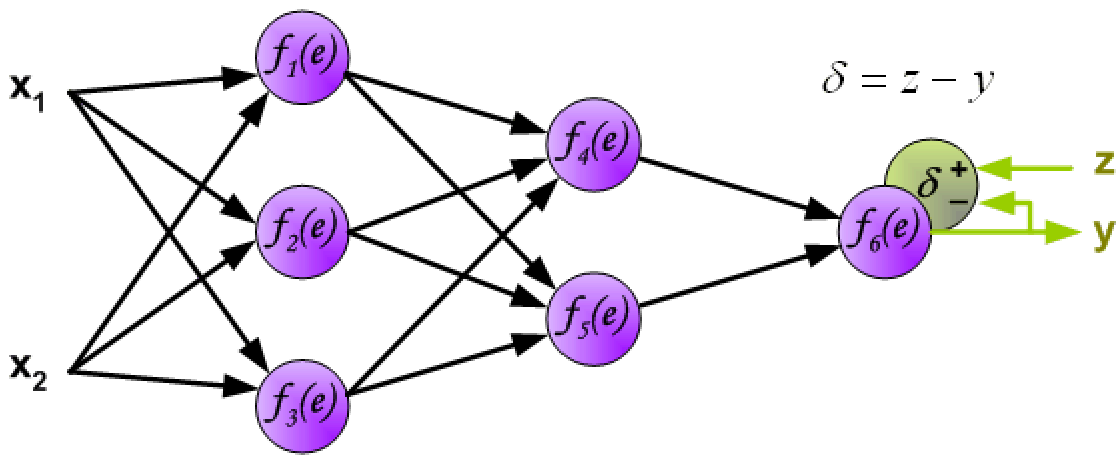
[Click to go to the source!]

Mariusz Bernacki
Przemysław Włodarczyk
mgr inż. Adam Gołda (2005)
Katedra Elektroniki AGH

Network Training – Forward Pass

- Symbols $w_{(xm)n}$ represent ***weights of connections*** between ***network input*** x_m and ***neuron n in the input layer***. Symbols y_n represents the output signal of ***neuron n.***
- ***Propagation*** of signals ***through the hidden layer.***

Computing Error: Next, the ***output signal of the network y*** is compared with the ***desired output value*** (the label), which is found in the training data set. The difference is called ***error signal δ.***

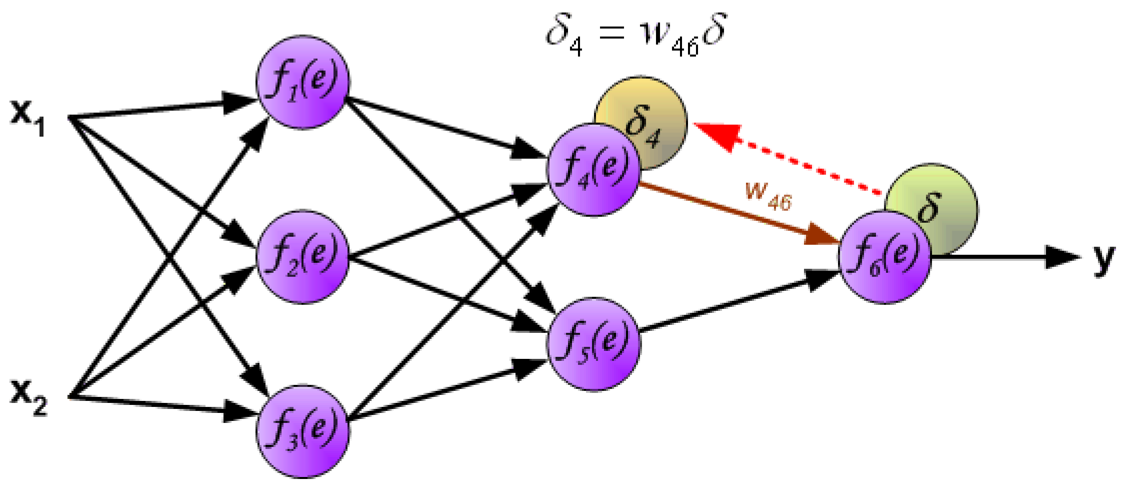


Click to go to the source!

Mariusz Bernacki
Przemysław Włodarczyk
mgr inż. Adam Gołda (2005)
Katedra Elektroniki AGH

Network Training – Backward Pass

- It is ***impossible*** to compute the ***error signal for internal neurons directly*** because the output values of these neurons are ***unknown (i.e., no labels for hidden layers)***.
- ***Error signal δ*** (computed in single teaching step) is ***propagated back*** to all neurons

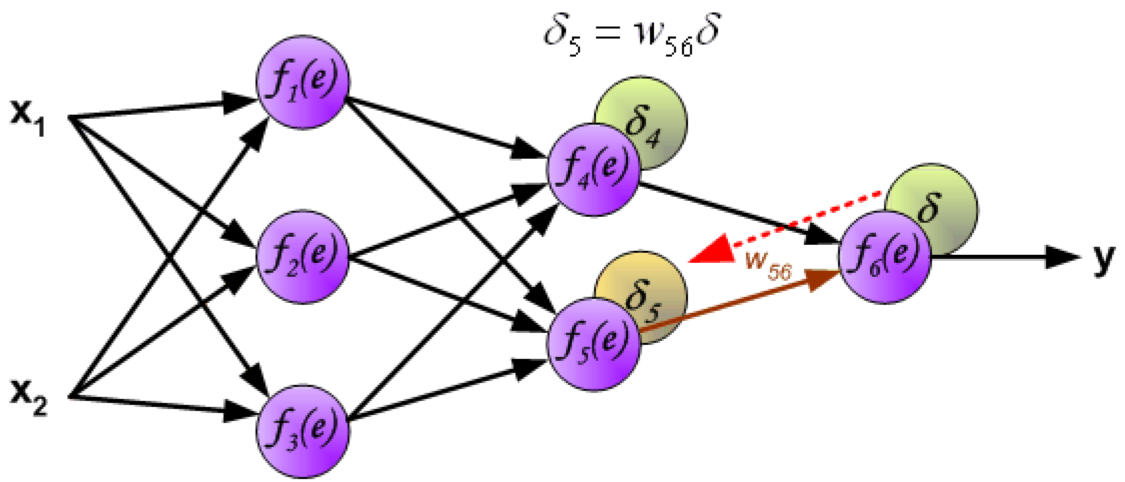


[Click to go to the source!](#)

[Mariusz Bernacki](#)
[Przemysław Włodarczyk](#)
mgr inż. Adam Gołda (2005)
Katedra Elektroniki AGH

Network Training – Backward Pass

- It is ***impossible*** to compute the ***error signal for internal neurons directly*** because the output values of these neurons are ***unknown (i.e., no labels for hidden layers)***.
- ***Error signal δ*** (computed in single teaching step) is ***propagated back*** to all neurons



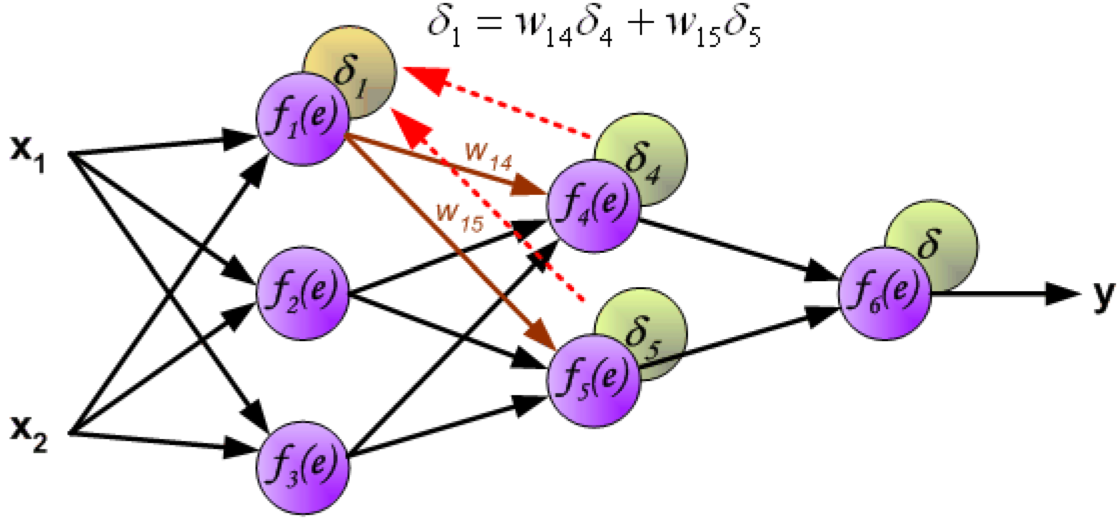
Click to go to the source!

Mariusz Bernacki
Przemysław Włodarczyk
mgr inż. Adam Gołda (2005)
Katedra Elektroniki AGH

Network Training – Backward Pass

- It is ***impossible*** to compute the ***error signal for internal neurons directly*** because the output values of these neurons are ***unknown (i.e., no labels for hidden layers)***.
- ***Error signal*** δ (computed in single teaching step) is ***propagated back*** to all neurons

$$\delta_1 = w_{14}\delta_4 + w_{15}\delta_5$$



The ***weights' coefficients*** w_{mn} used to propagate errors back are equal to those used during the forward pass to compute the output value.

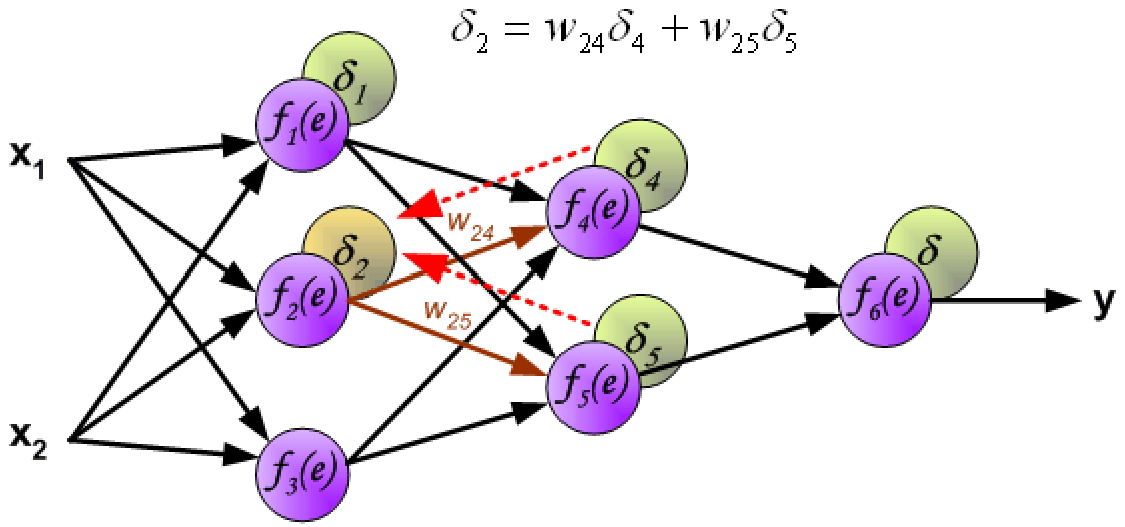
Only the direction of data flow is changed (signals are propagated from output to inputs one after the other).

Click to go to the source!

Mariusz Bernacki
Przemysław Włodarczyk
mgr inż. Adam Gołda (2005)
Katedra Elektroniki AGH

Network Training – Backward Pass

- It is ***impossible*** to compute the ***error signal for internal neurons directly*** because the output values of these neurons are ***unknown (i.e., no labels for hidden layers)***.
- ***Error signal*** δ (computed in single teaching step) is ***propagated back*** to all neurons



The ***weights' coefficients*** w_{mn} used to propagate errors back are equal to those used during the forward pass to compute the output value.

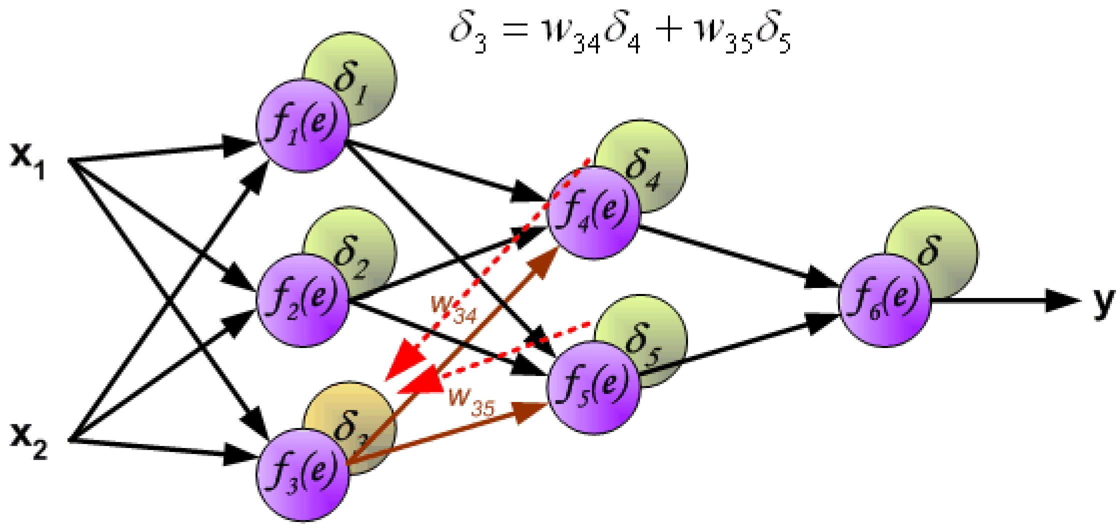
Only the direction of data flow is changed (signals are propagated from output to inputs one after the other).

[Click to go to the source!]

Mariusz Bernacki
Przemysław Włodarczyk
mgr inż. Adam Gołda (2005)
Katedra Elektroniki AGH

Network Training – Backward Pass

- It is ***impossible*** to compute the ***error signal for internal neurons directly*** because the output values of these neurons are ***unknown (i.e., no labels for hidden layers)***.
- ***Error signal*** $\boldsymbol{\delta}$ (computed in single teaching step) is ***propagated back*** to all neurons



The ***weights' coefficients*** $\boldsymbol{w_{mn}}$ used to propagate errors back are equal to those used during the forward pass to compute the output value.

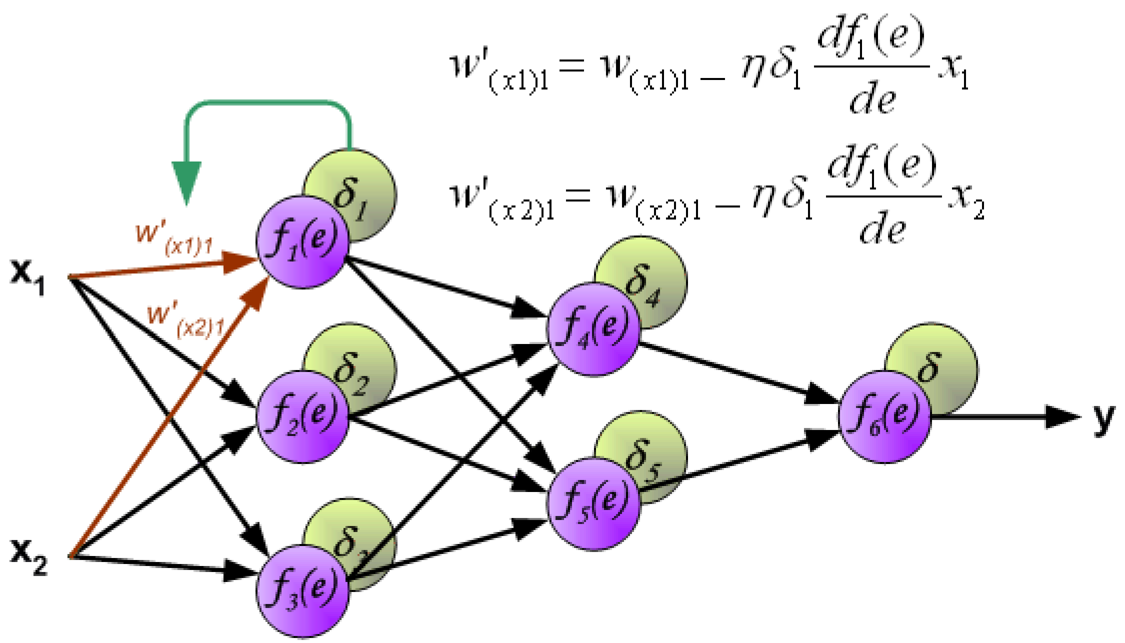
Only the direction of data flow is changed (signals are propagated from output to inputs one after the other).

[Click to go to the source!]

[Mariusz Bernacki]
[Przemysław Włodarczyk]
mgr inż. Adam Gołda (2005)
Katedra Elektroniki AGH

Network Training – Parameter Update

- When the error signal for each neuron is computed, ***the weights coefficients*** of each neuron input node may be ***modified***.
- ***Coefficient η*** affects the ***learning speed***.



[Click to go to the source!]

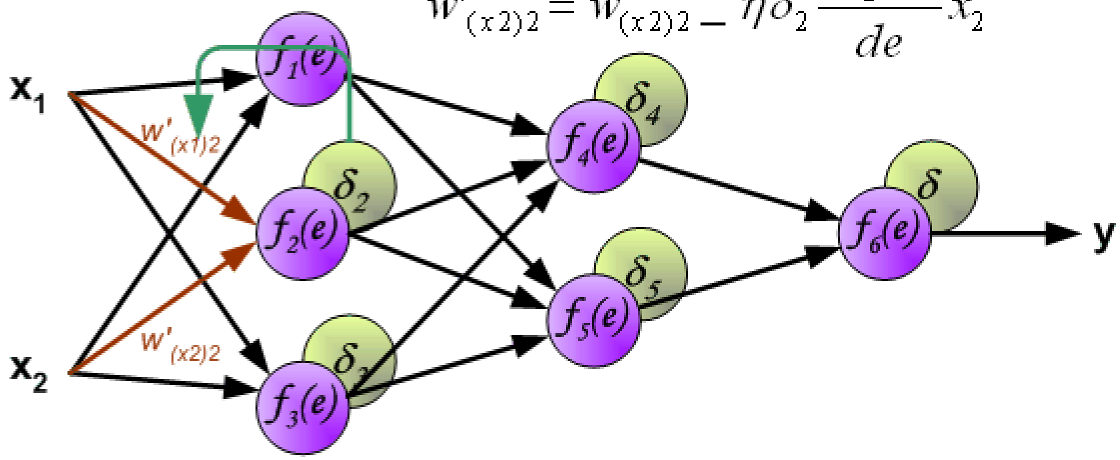
[Mariusz Bernacki]
[Przemysław Włodarczyk]
mgr inż. Adam Gołda (2005)
Katedra Elektroniki AGH

Network Training – Parameter Update

- When the error signal for each neuron is computed, ***the weights coefficients*** of each neuron input node may be ***modified***.
- ***Coefficient η*** affects the ***learning speed***.

$$w'_{(x1)2} = w_{(x1)2} - \eta \delta_2 \frac{df_2(e)}{de} x_1$$

$$w'_{(x2)2} = w_{(x2)2} - \eta \delta_2 \frac{df_2(e)}{de} x_2$$



[Click to go to the source!]

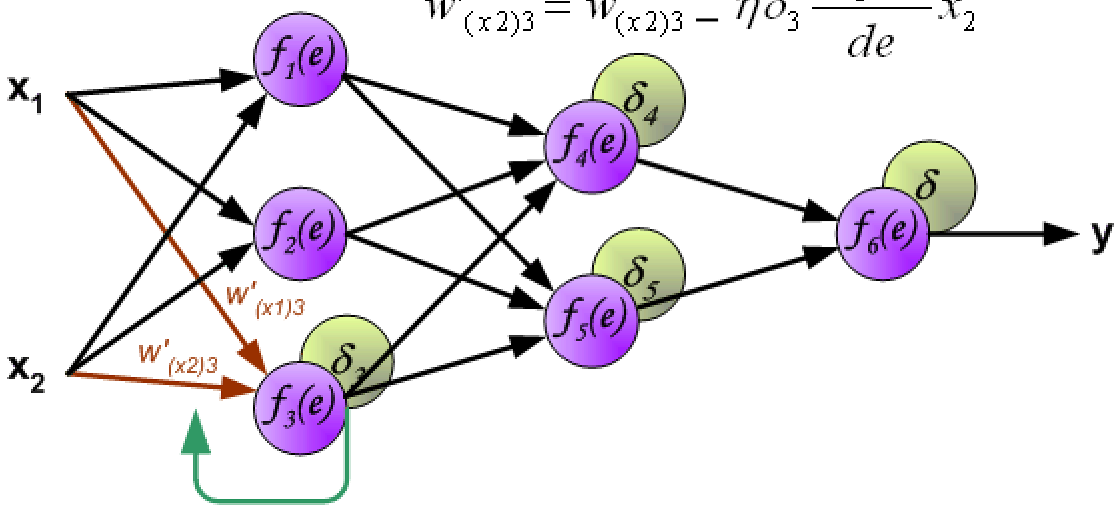
Mariusz Bernacki
Przemysław Włodarczyk
mgr inż. Adam Gołda (2005)
Katedra Elektroniki AGH

Network Training – Parameter Update

- When the error signal for each neuron is computed, ***the weights coefficients*** of each neuron input node may be ***modified***.
- ***Coefficient*** $\boldsymbol{\eta}$ affects the ***learning speed***.

$$w'_{(x1)3} = w_{(x1)3} - \eta \delta_3 \frac{df_3(e)}{de} x_1$$

$$w'_{(x2)3} = w_{(x2)3} - \eta \delta_3 \frac{df_3(e)}{de} x_2$$



[Click to go to the source!]

[Mariusz Bernacki]
[Przemysław Włodarczyk]
mgr inż. Adam Gołda (2005)
Katedra Elektroniki AGH

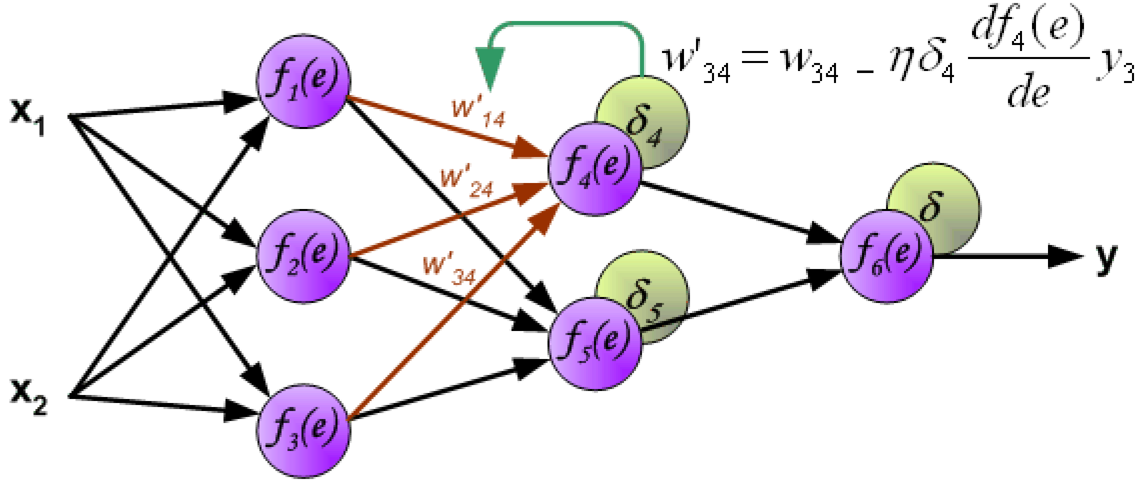
Network Training – Parameter Update

- When the error signal for each neuron is computed, ***the weights coefficients*** of each neuron input node may be ***modified***.
- ***Coefficient η*** affects the ***learning speed***.
- There are a few techniques to select ***η***, e.g., starting with a large value and then decreasing gradually while weights coefficients are being established.

$$w'_{14} = w_{14} - \eta \delta_4 \frac{df_4(e)}{de} y_1$$

$$w'_{24} = w_{24} - \eta \delta_4 \frac{df_4(e)}{de} y_2$$

$$w'_{34} = w_{34} - \eta \delta_4 \frac{df_4(e)}{de} y_3$$



[Click to go to the source!]

Mariusz Bernacki
Przemysław Włodarczyk
mgr inż. Adam Gołda (2005)
Katedra Elektroniki AGH

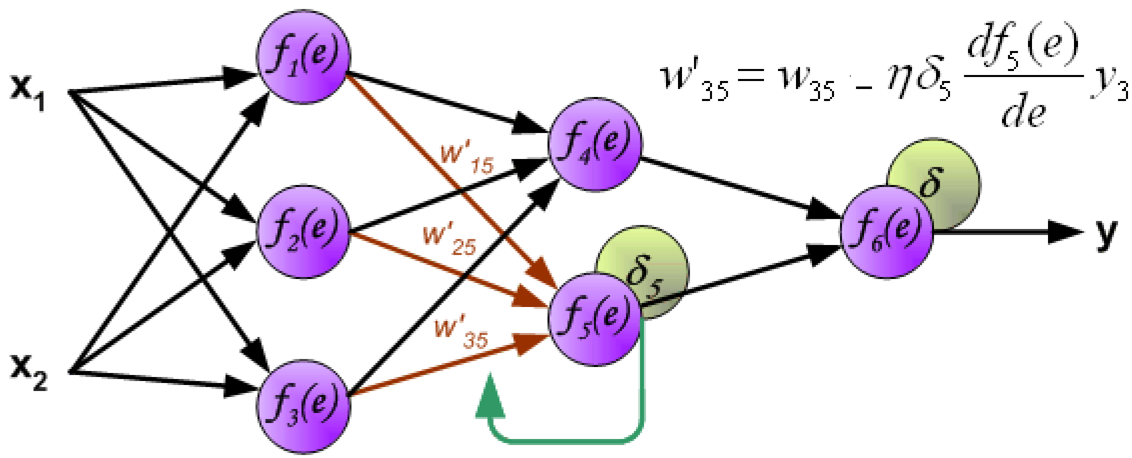
Network Training – Parameter Update

- When the error signal for each neuron is computed, ***the weights coefficients*** of each neuron input node may be ***modified***.
- ***Coefficient η*** affects the ***learning speed***.
- There are a few techniques to select η, e.g., starting with a large value and then decreasing gradually while weights coefficients are being established.

$$w'_{15} = w_{15} - \eta\delta_5 \frac{df_5(e)}{de} y_1$$

$$w'_{25} = w_{25} - \eta\delta_5 \frac{df_5(e)}{de} y_2$$

$$w'_{35} = w_{35} - \eta\delta_5 \frac{df_5(e)}{de} y_3$$



Click to go to the source!

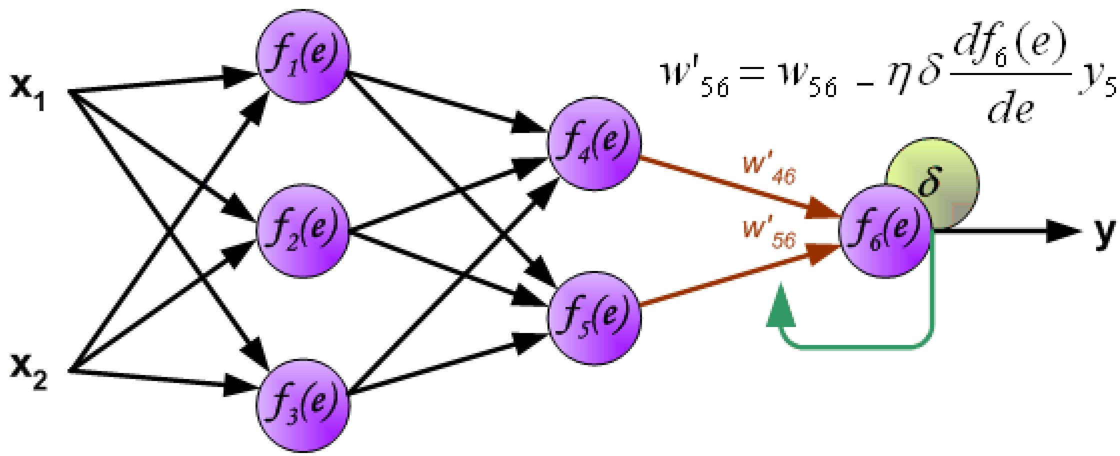
Mariusz Bernacki
Przemysław Włodarczyk
mgr inż. Adam Gołda (2005)
Katedra Elektroniki AGH

Network Training – Parameter Update

- When the error signal for each neuron is computed, ***the weights coefficients*** of each neuron input node may be ***modified***.
- ***Coefficient η*** affects the ***learning speed***.
- There are a few techniques to select η, e.g., starting with a large value and then decreasing gradually while weights coefficients are being established.

$$w'_{46} = w_{46} - \eta\delta \frac{df_6(e)}{de} y_4$$

$$w'_{56} = w_{56} - \eta\delta \frac{df_6(e)}{de} y_5$$



Click to go to the source!

Mariusz Bernacki
Przemysław Włodarczyk
mgr inż. Adam Gołda (2005)
Katedra Elektroniki AGH

Backpropagation with Computational Graphs

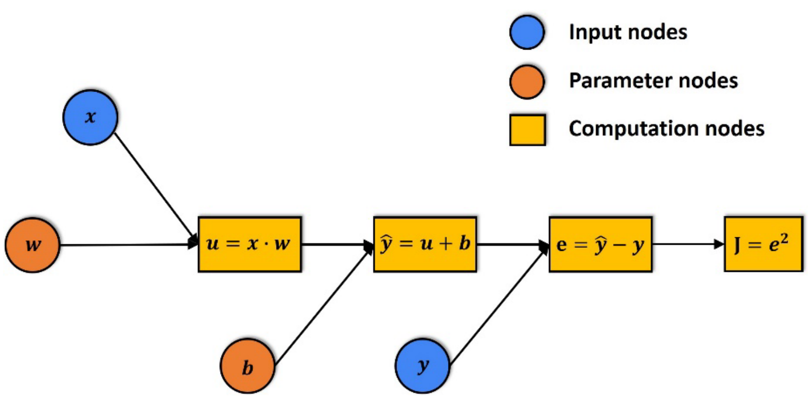
- ***Neural nets*** can have ***very deep and complicated architectures,*** e.g.,:

$$\hat{y}(x) = \sigma(W^3\sigma(W^2\sigma(W^1x + b^1) + b^2) + b^3)$$

- Lots of matrix operations and thus it becomes ***impractical to write down gradient formula by hand for all parameters***

- Instead, we can represent neural networks as ***computational graphs***

- ***A Computational Graph represents the process of computing a mathematical expression***
  - it has ***computation nodes (operations)***
  - it has ***parameter nodes***
  - it has ***input nodes***
  - it has ***edges*** that connects nodes ***(data flow)***
  - it is ***directional***
  - it can be organized into ***layers***



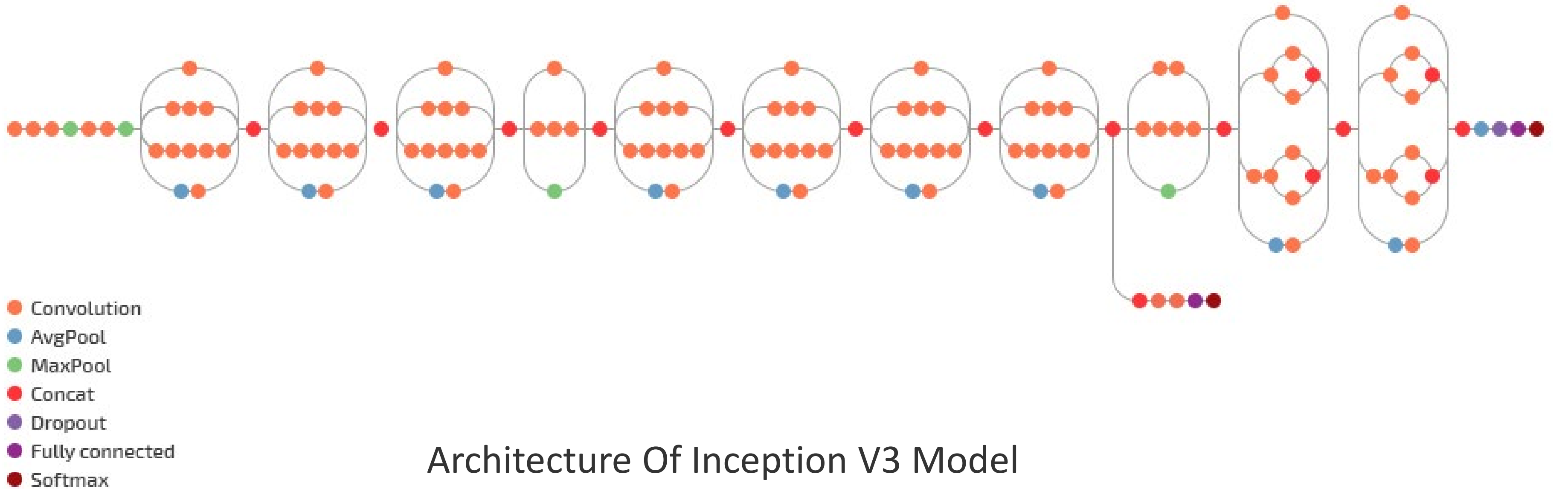
Backpropagation with Computational Graphs

- **Key Idea in Computational Graphs**:
  - ***Decompose*** complex computations in neural nets into a ***sequence of atomic assignments***
  - This sequence of assignments is called a ***computation graph***
  - The ***forward pass*** takes a training sample as input and computes the total loss
  - The ***backward (reverse) pass*** computes gradients
  - Both the forward and backward passes are efficient due to the use of dynamic programming, i.e., ***storing and reusing the intermediate results***
  - This ***decomposition and reuse of computation*** is the ***key factor*** in the success of the ***backpropagation algorithm***, thus, it is the primary workhorse of deep learning
  - The ***importance of the computation graphs*** comes from the ***backward pass***. This is used to compute the derivatives that are needed for the weight update

- **Consequently,**
  - ***Backpropagation*** is ***just the chain rule of differentiation***
  - ***Computational graphs*** help us ***reduce the computational cost*** of computing the gradient by ***caching intermediate results***
  - In addition to having GPU-based parallel architectures and large datasets, success of deep learning ***owes a lot*** to success of ***automatic differentiation (autodiff) libraries*** in, e.g., PyTorch (torch.autograd).

Backpropagation with Computational Graphs

- ***Computational Graphs:*** From a set of neurons to a ***structured compute pipeline***



Architecture Of Inception V3 Model

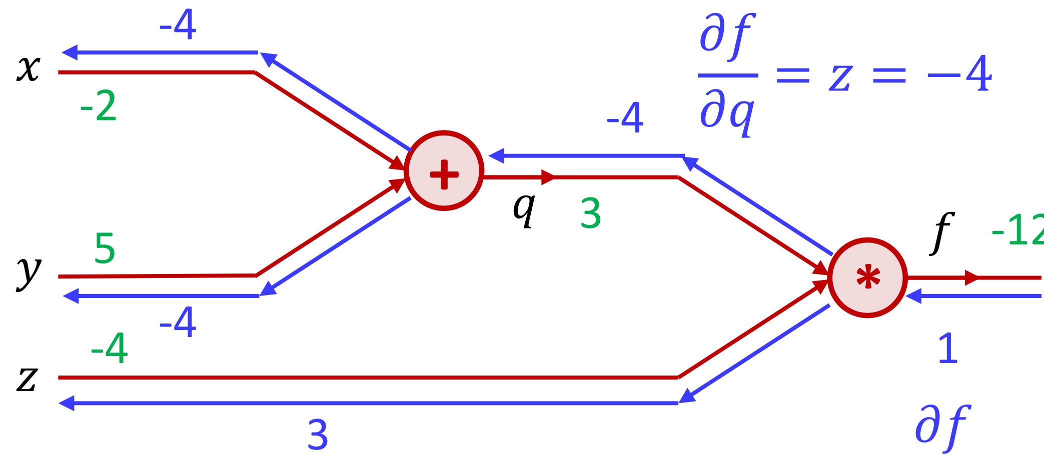
[Szegedy, et al., Rethinking the Inception Architecture for Computer Vision, 2015]()

Backpropagation with Computational Graphs

Example I

$$f(x, y, z) = (x + y)z$$

e.g., $x = -2 \quad y = 5 \quad z = -4$



$$\frac{\partial f}{\partial q} = z = -4$$

$$\frac{\partial f}{\partial f} = 1$$

$$\frac{\partial f}{\partial z} = q = x + y = 3$$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z \qquad \frac{\partial f}{\partial z} = q$$

WANTED $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Chain Rule:

Upstream gradient

Local gradient

$$\boxed{\underbrace{\frac{\partial f}{\partial y}}_{\text{Downstream gradient}} = \underbrace{\frac{\partial f}{\partial q}}_{\text{Upstream gradient}} \underbrace{\frac{\partial q}{\partial y}}_{\text{Local gradient}}} = z * 1 = -4$$

Downstream gradient

Chain Rule:

Upstream gradient

Local gradient

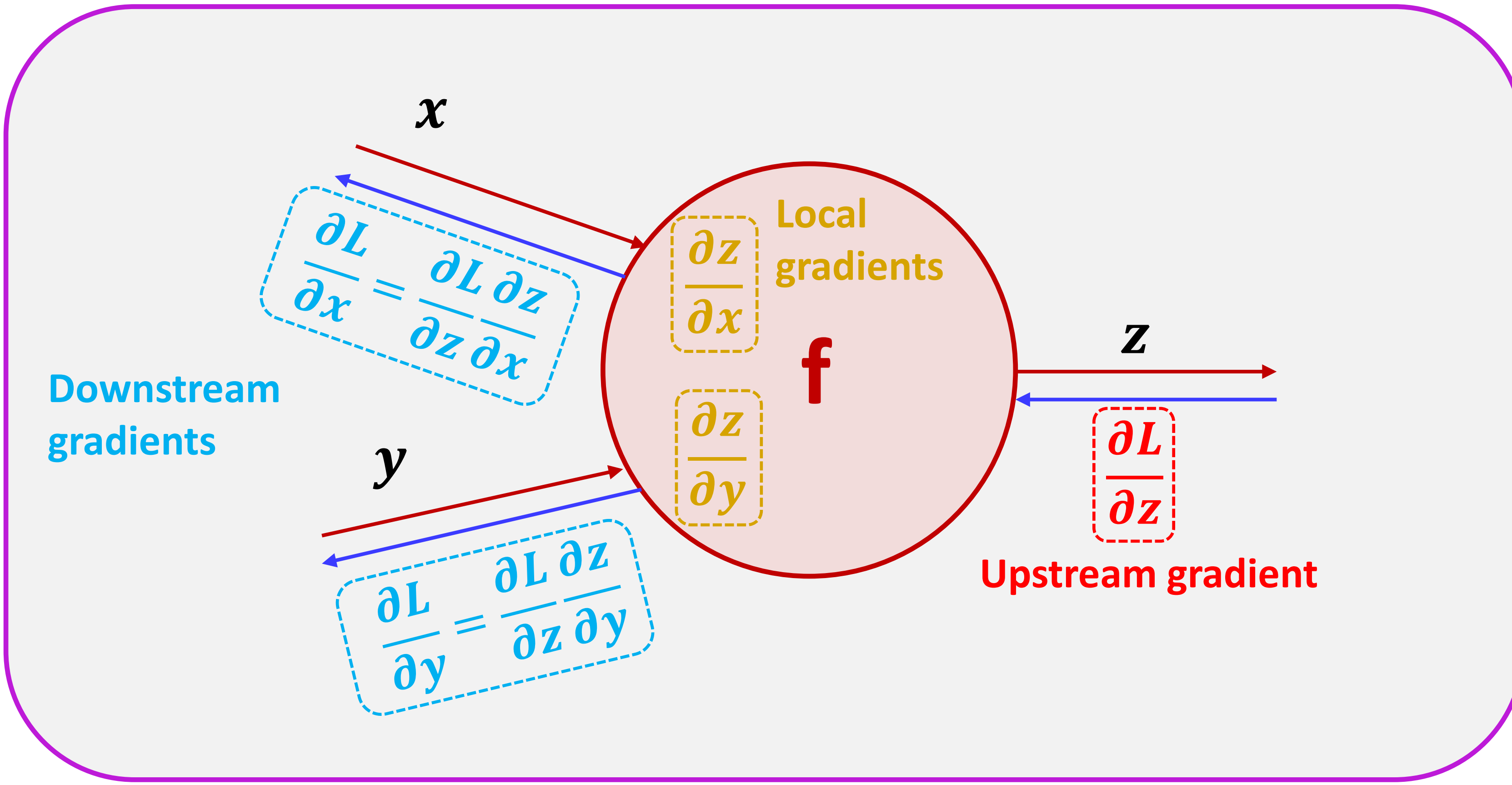
$$\boxed{\underbrace{\frac{\partial f}{\partial x}}_{\text{Downstream gradient}} = \underbrace{\frac{\partial f}{\partial q}}_{\text{Upstream gradient}} \underbrace{\frac{\partial q}{\partial x}}_{\text{Local gradient}}} = z * 1 = -4$$

Downstream gradient

Source: https://cs231n.github.io/

Backpropagation with Computational Graphs

Chain Rule ⇨ Downstream gradient = Upstream gradient * Local gradient



Source: https://cs231n.github.io/

Backpropagation with Computational Graphs

Example II

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

Derivative Cheat Sheet

$$\frac{d(ax)}{dx} = a$$

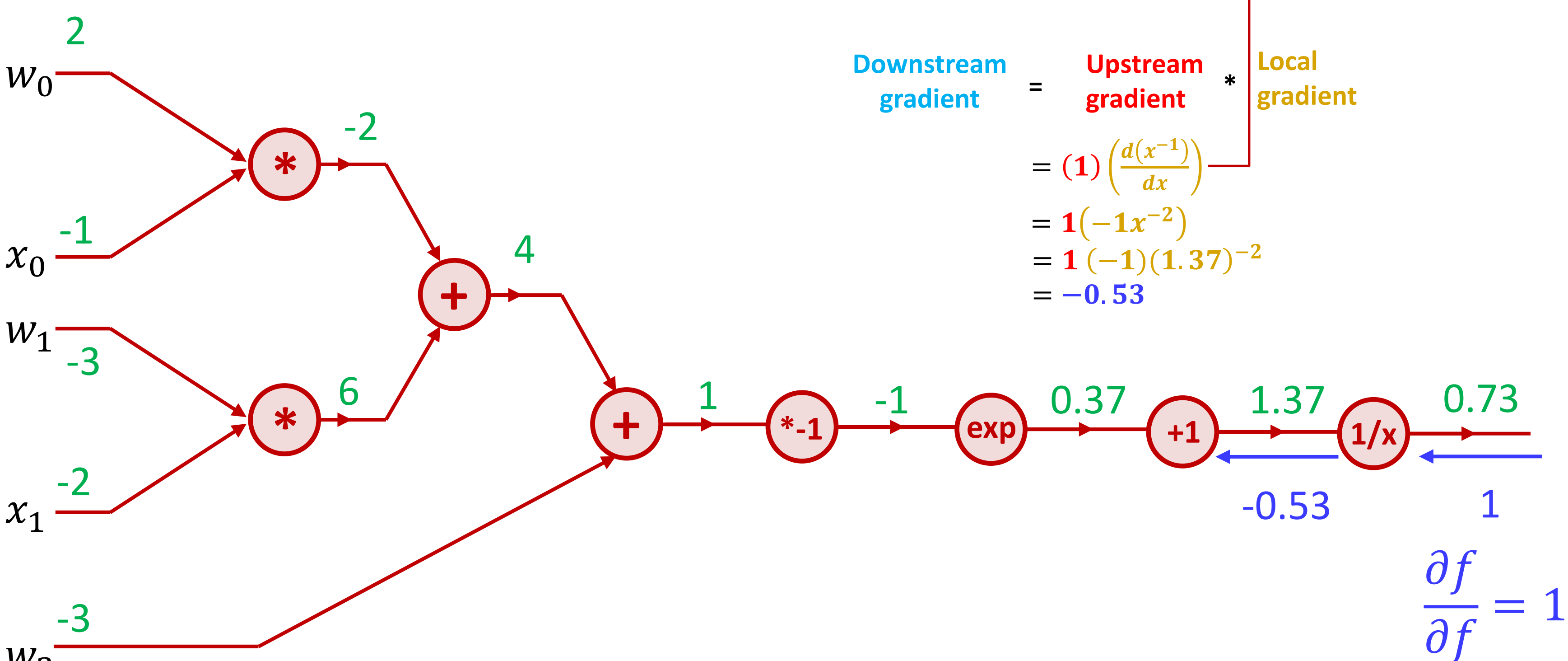
$$\frac{d(a+x)}{dx} = 1$$

$$\frac{d(x^n)}{dx} = nx^{n-1}$$

$$\frac{d(e^x)}{dx} = e^x$$

Downstream gradient = Upstream gradient * Local gradient

$$= (1)\left(\frac{d(x^{-1})}{dx}\right)$$
$$= 1(-1x^{-2})$$
$$= 1\ (-1)(1.37)^{-2}$$
$$= -0.53$$



Source: https://cs231n.github.io/

Backpropagation with Computational Graphs

Derivative Cheat Sheet

$$\frac{d(ax)}{dx} = a$$

$$\frac{d(a+x)}{dx} = 1$$

$$\frac{d(x^n)}{dx} = nx^{n-1}$$

$$\frac{d(e^x)}{dx} = e^x$$

Example II

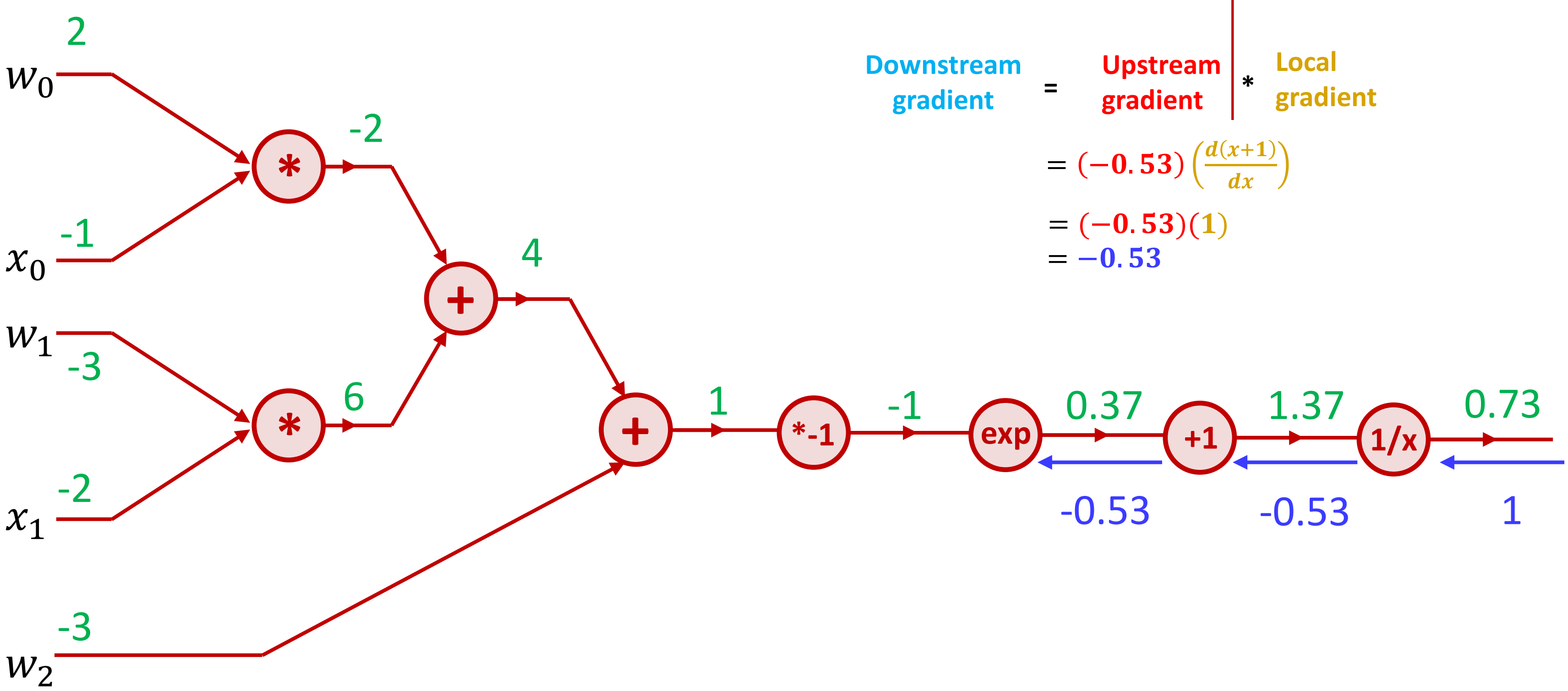
$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

Downstream gradient = Upstream gradient * Local gradient

$$= (-0.53)\left(\frac{d(x+1)}{dx}\right)$$

$$= (-0.53)(1)$$

$$= -0.53$$



Source: https://cs231n.github.io/

Backpropagation with Computational Graphs

Example II

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

Derivative Cheat Sheet

$$\frac{d(ax)}{dx} = a$$

$$\frac{d(a+x)}{dx} = 1$$

$$\frac{d(x^n)}{dx} = nx^{n-1}$$

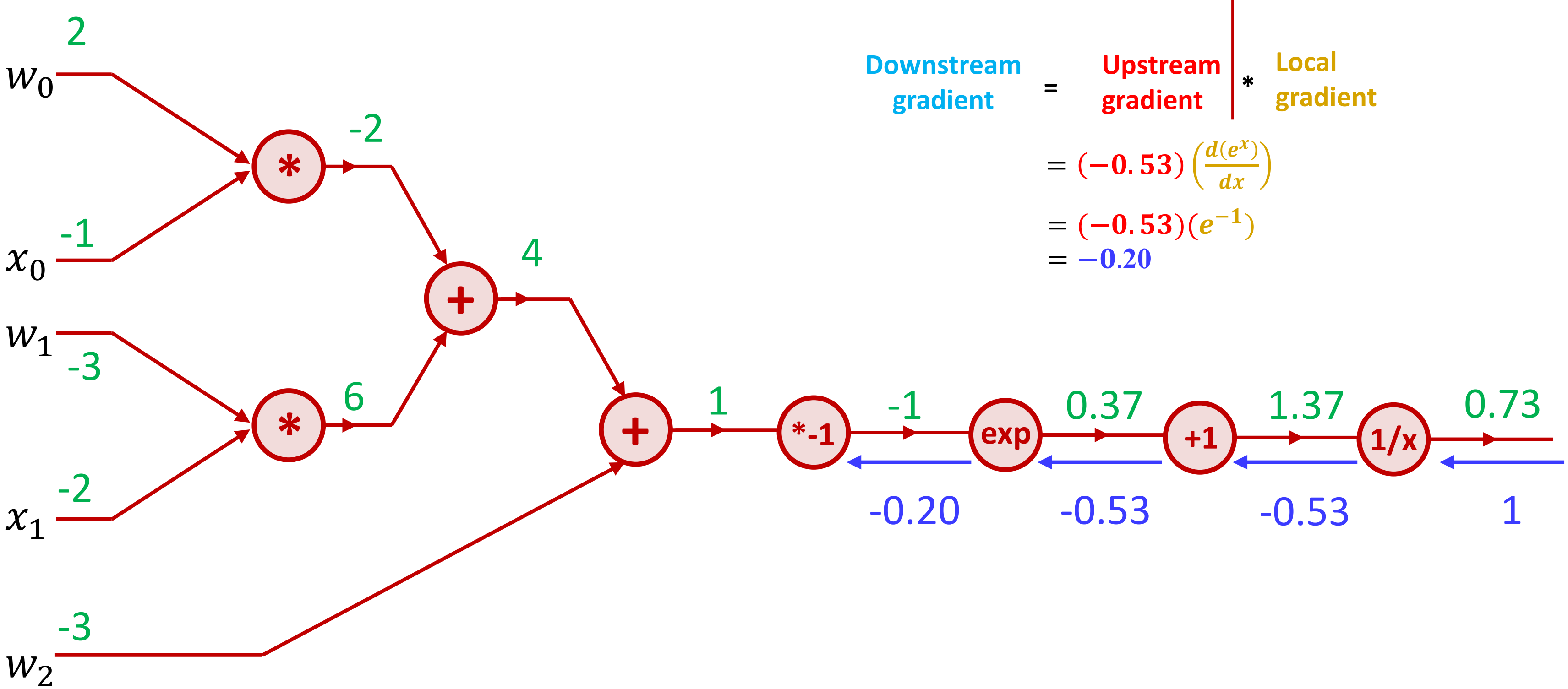
$$\frac{d(e^x)}{dx} = e^x$$

Downstream gradient = Upstream gradient * Local gradient

$$= (-0.53)\left(\frac{d(e^x)}{dx}\right)$$

$$= (-0.53)(e^{-1})$$

$$= -0.20$$



Source: https://cs231n.github.io/

Backpropagation with Computational Graphs

Derivative Cheat Sheet

$$\frac{d(ax)}{dx} = a$$

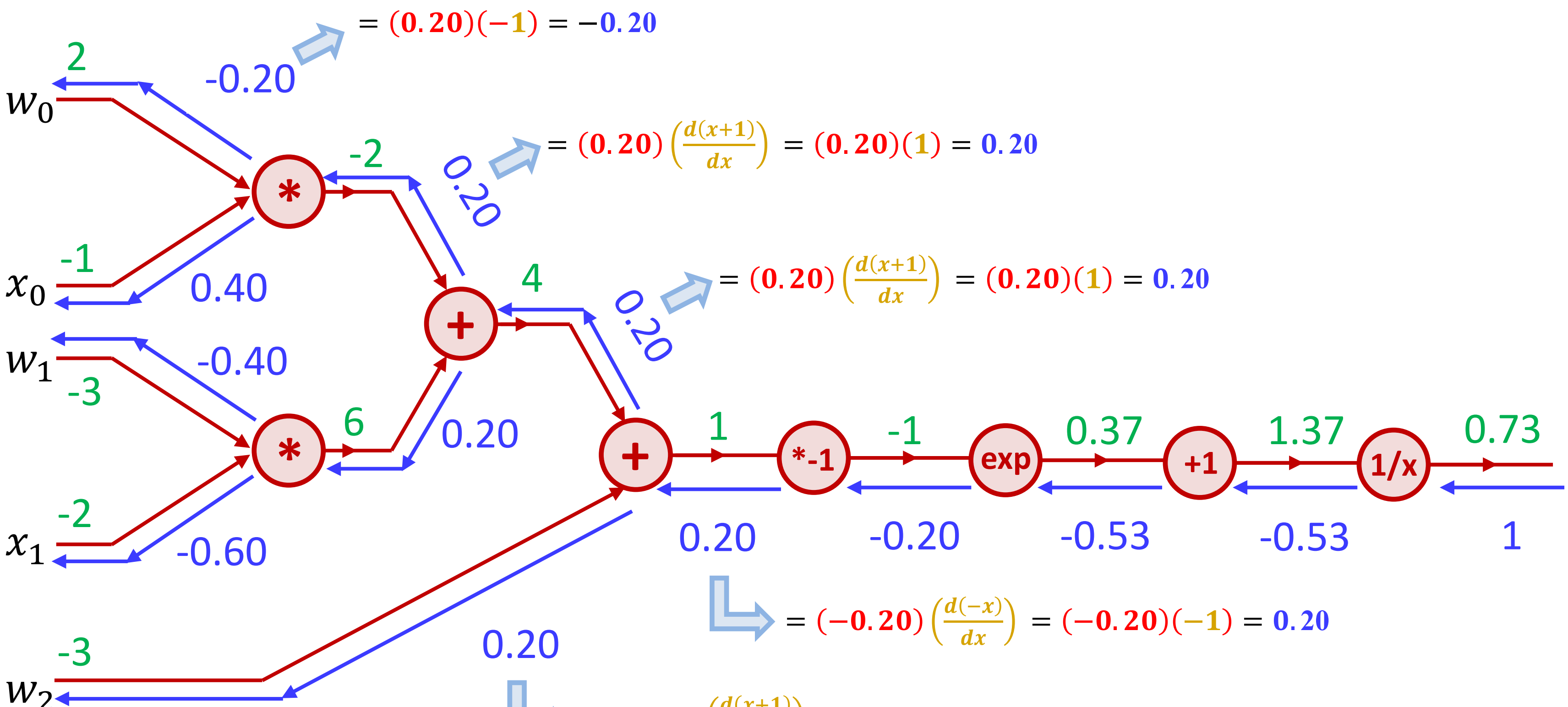
$$\frac{d(a+x)}{dx} = 1$$

$$\frac{d(x^n)}{dx} = nx^{n-1}$$

$$\frac{d(e^x)}{dx} = e^x$$

Example II

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



Source: https://cs231n.github.io/

Backpropagation with Computational Graphs

Derivative Cheat Sheet

$$\frac{d(ax)}{dx} = a$$

$$\frac{d(a+x)}{dx} = 1$$

$$\frac{d(x^n)}{dx} = nx^{n-1}$$

$$\frac{d(e^x)}{dx} = e^x$$

Example II

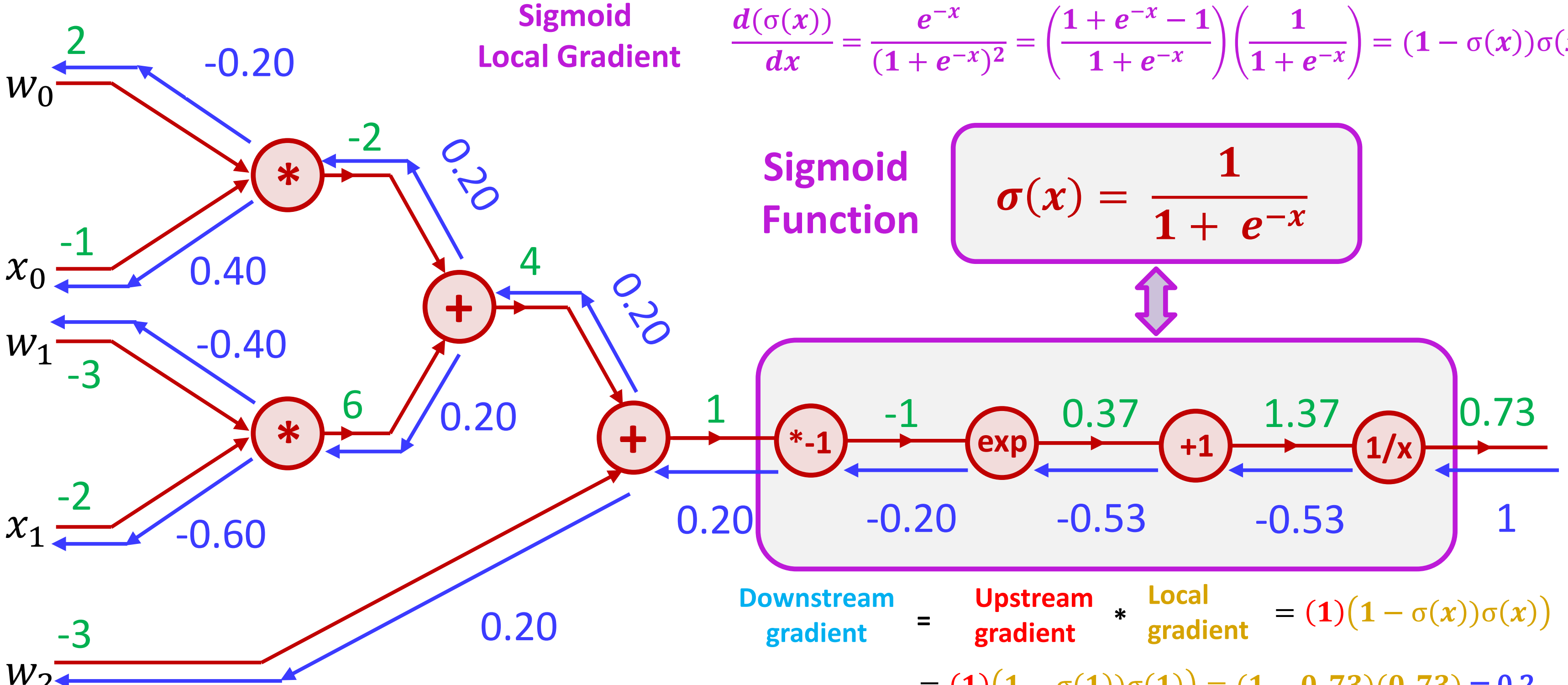
$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

Sigmoid Local Gradient

$$\frac{d(\sigma(x))}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\sigma(x)$$

Sigmoid Function

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



Downstream gradient = Upstream gradient * Local gradient $= (1)(1-\sigma(x))\sigma(x))$

$= (1)(1-\sigma(1))\sigma(1)) = (1-0.73)(0.73) = 0.2$

Backpropagation with Computational Graphs

Derivative Cheat Sheet

$$\frac{d(ax)}{dx} = a$$

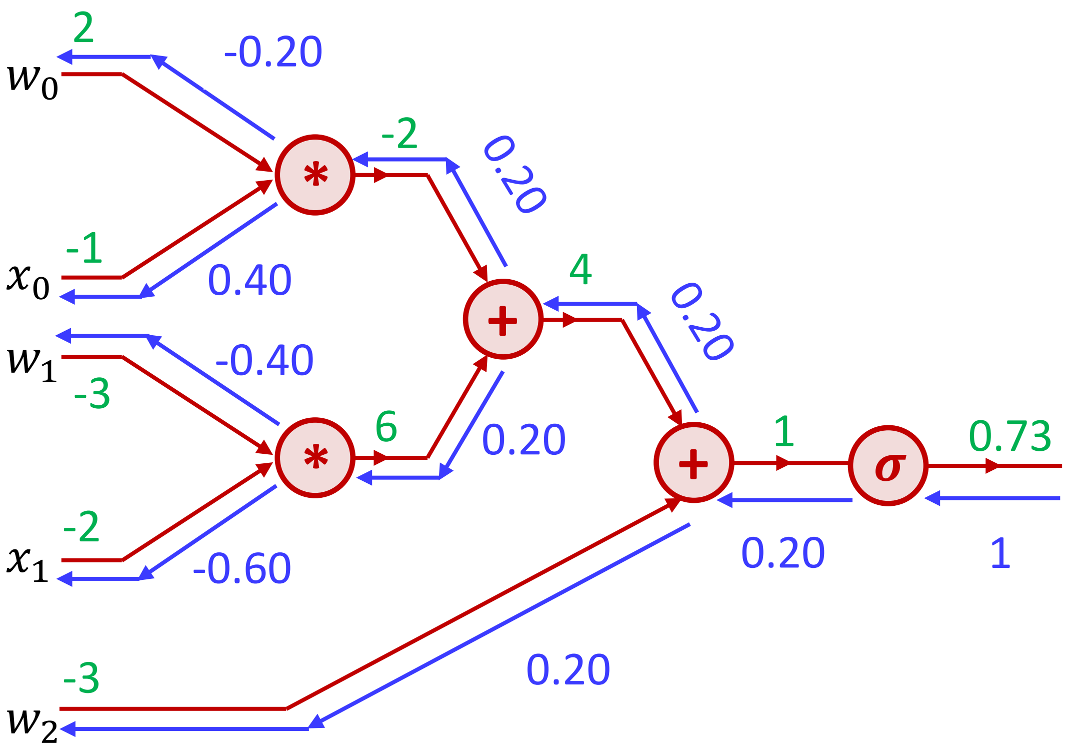
$$\frac{d(a+x)}{dx} = 1$$

$$\frac{d(x^n)}{dx} = nx^{n-1}$$

$$\frac{d(e^x)}{dx} = e^x$$

Example II

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



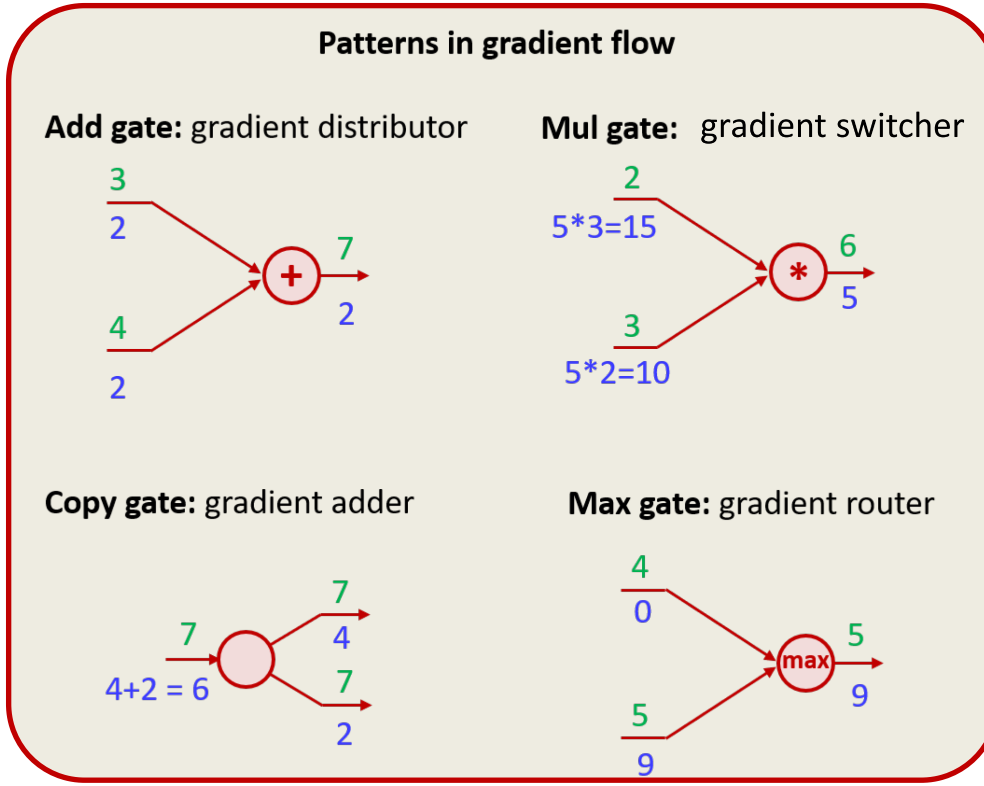
Computational Graph representation may not be unique!

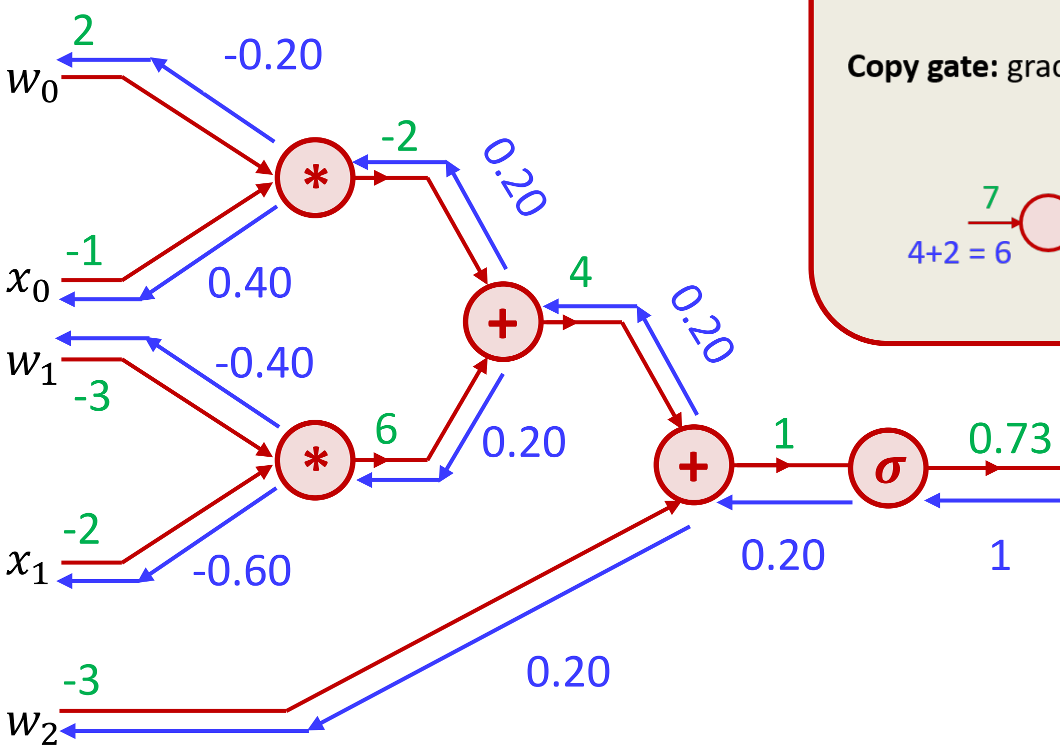
Source: https://cs231n.github.io/

Backpropagation with Computational Graphs

Example II

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$





Source: https://cs231n.github.io/

Backpropagation with **Flat Code**

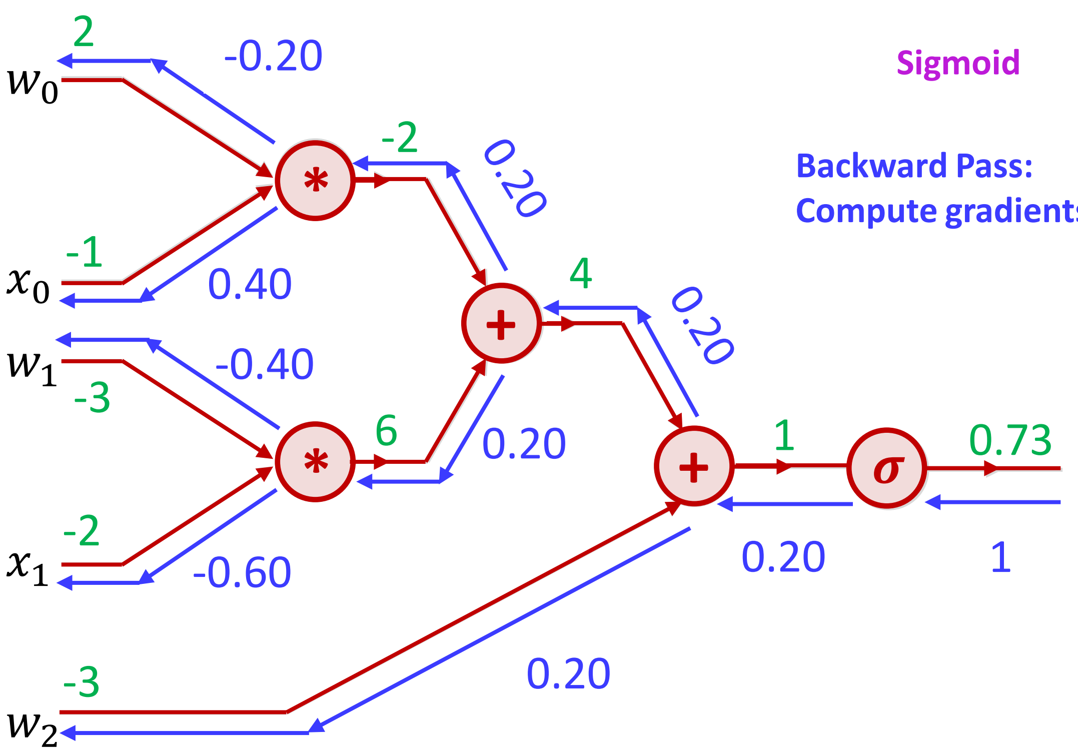
Example II

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

Forward Pass:
Compute outputs

$$\textit{def } f(w_0, x_0, w_1, x_1, w_2):$$

$$s_0 = w_0 * x_0$$
$$s_1 = w_1 * x_1$$
$$s_2 = s_0 + s_1$$
$$s_3 = s_2 + w_2$$
$$L = \sigma(s_3)$$



Backward Pass:
Compute gradients

Base case ⟹ $grad_L = 1.0$

Sigmoid ⟹ $grad_{s_3} = grad_L * (1 - L) * L$

$grad_{w_2} = grad_{s_3}$
$grad_{s_2} = grad_{s_3}$ — Add gate

$grad_{s_0} = grad_{s_2}$
$grad_{s_1} = grad_{s_2}$ — Add gate

$grad_{w_1} = grad_{s_1} * x_1$
$grad_{x_1} = grad_{s_1} * w_1$ — Multiply gate

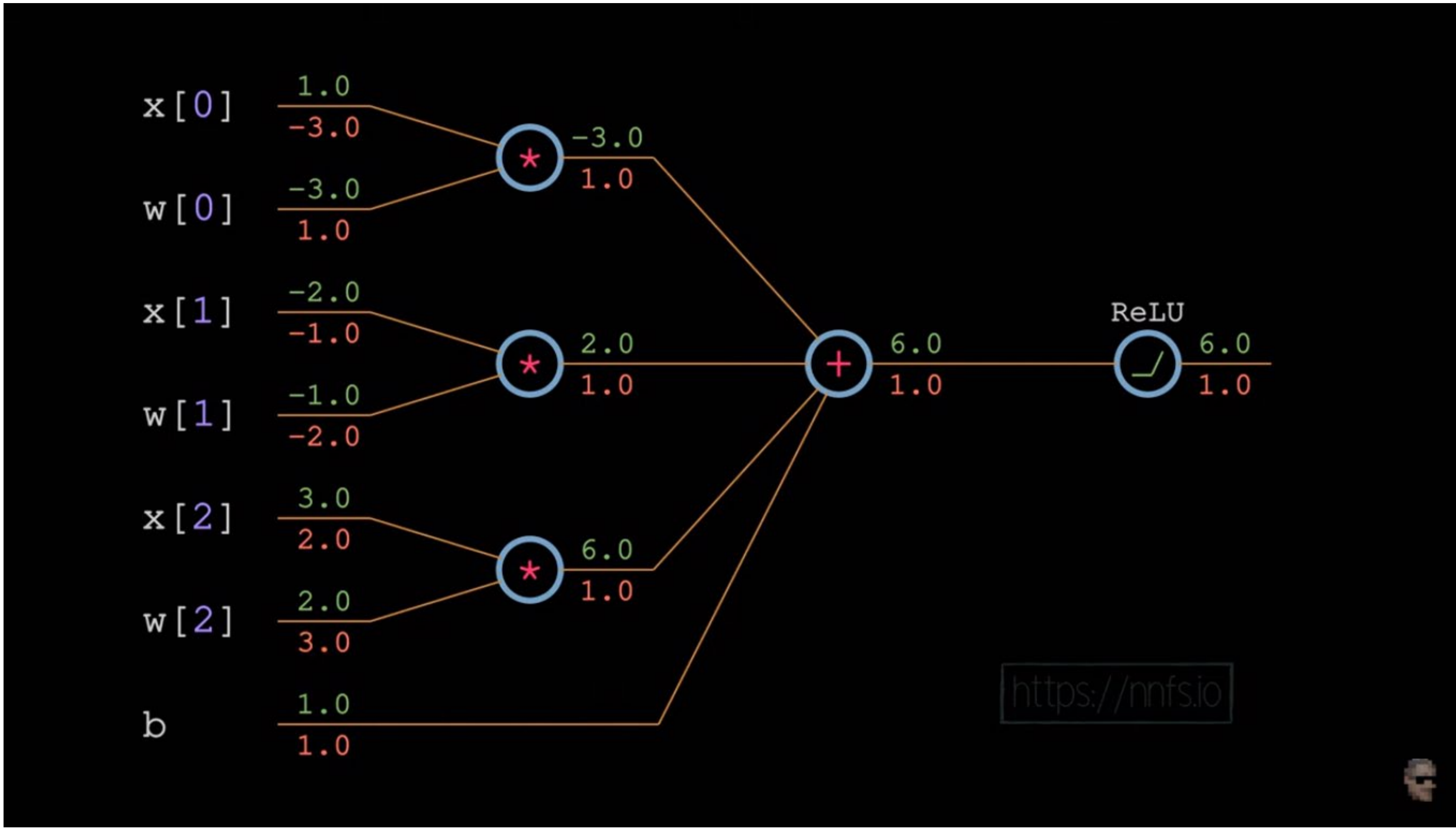
$grad_{w_0} = grad_{s_0} * x_0$
$grad_{x_0} = grad_{s_0} * w_0$ — Multiply gate

Source: https://cs231n.github.io/

Backpropagation with Computational Graphs

Example III

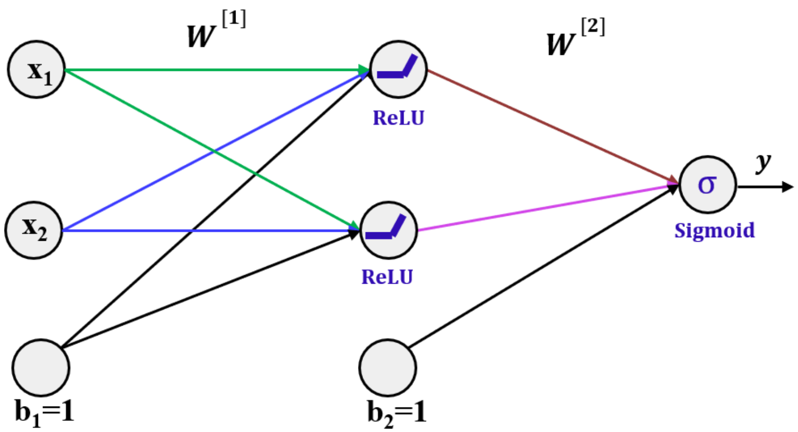
[Click to go to the solution video in YouTube]



The green numbers are the actual values, the red/orange values are the partial derivatives.

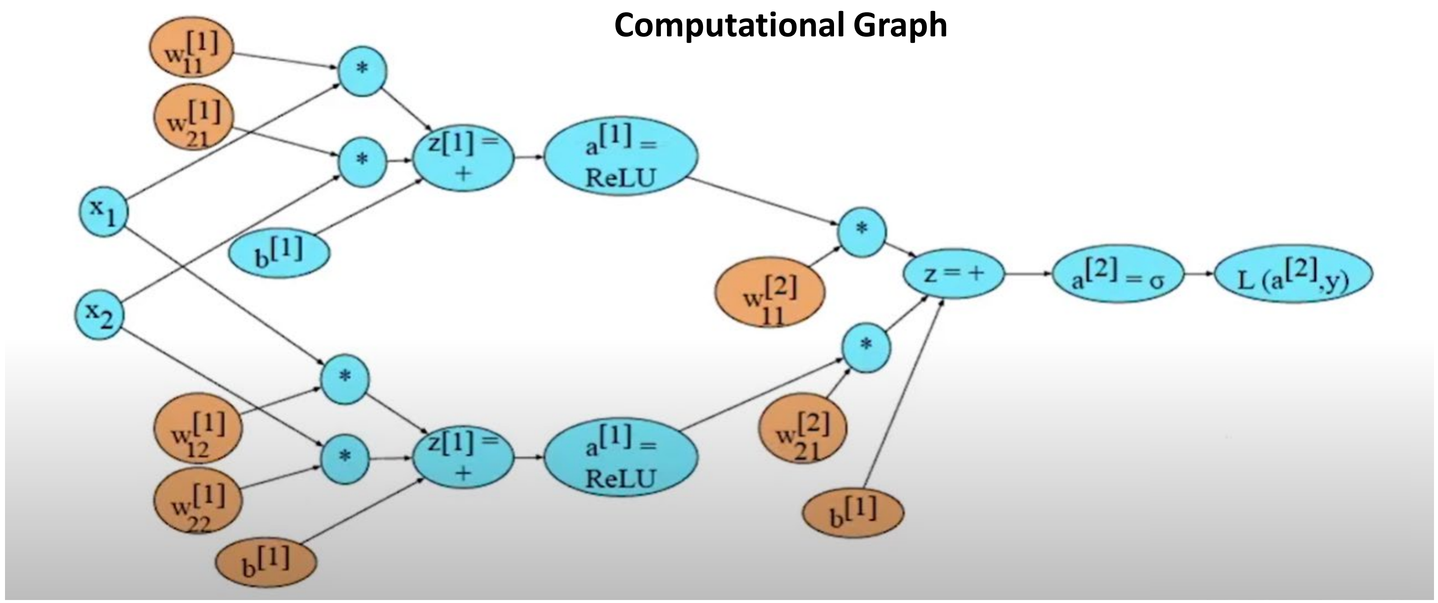
Backpropagation with Computational Graphs

Example IV – A two-layer Network



$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = ReLU(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \sigma(z^{[2]})$$
$$y = a^{[2]}$$

$$\frac{d(\sigma(z))}{dz} = (1 - \sigma(z))\sigma(z)$$
$$\frac{d(ReLU(z))}{dz} = \begin{cases} 0 \; for \; z < 0 \\ 1 \; for \; z \geq 0 \end{cases}$$



Backpropagation with **Vectors/Matrices/Tensors**

So far, we computed *backpropagation with scalars*

Scalar to Scalar

$$x \in \mathbb{R}, y \in \mathbb{R}$$

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If x changes by a small amount, how much will y change?

Vector to Scalar

$$x \in \mathbb{R}^N, y \in \mathbb{R}$$

Derivative is **Gradient**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^N \quad \left(\frac{\partial y}{\partial x}\right)_n = \frac{\partial y}{\partial x_n}$$

For each element of x, if it changes by a small amount then how much will y change?

Vector to Vector

$$x \in \mathbb{R}^N, y \in \mathbb{R}^M$$

Derivative is **Jacobian**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M} \quad \left(\frac{\partial y}{\partial x}\right)_{n,m} = \frac{\partial y_m}{\partial x_n}$$

For each element of x, if it changes by a small amount then how much will each element of y change?

Source: https://cs231n.github.io/

Discussion

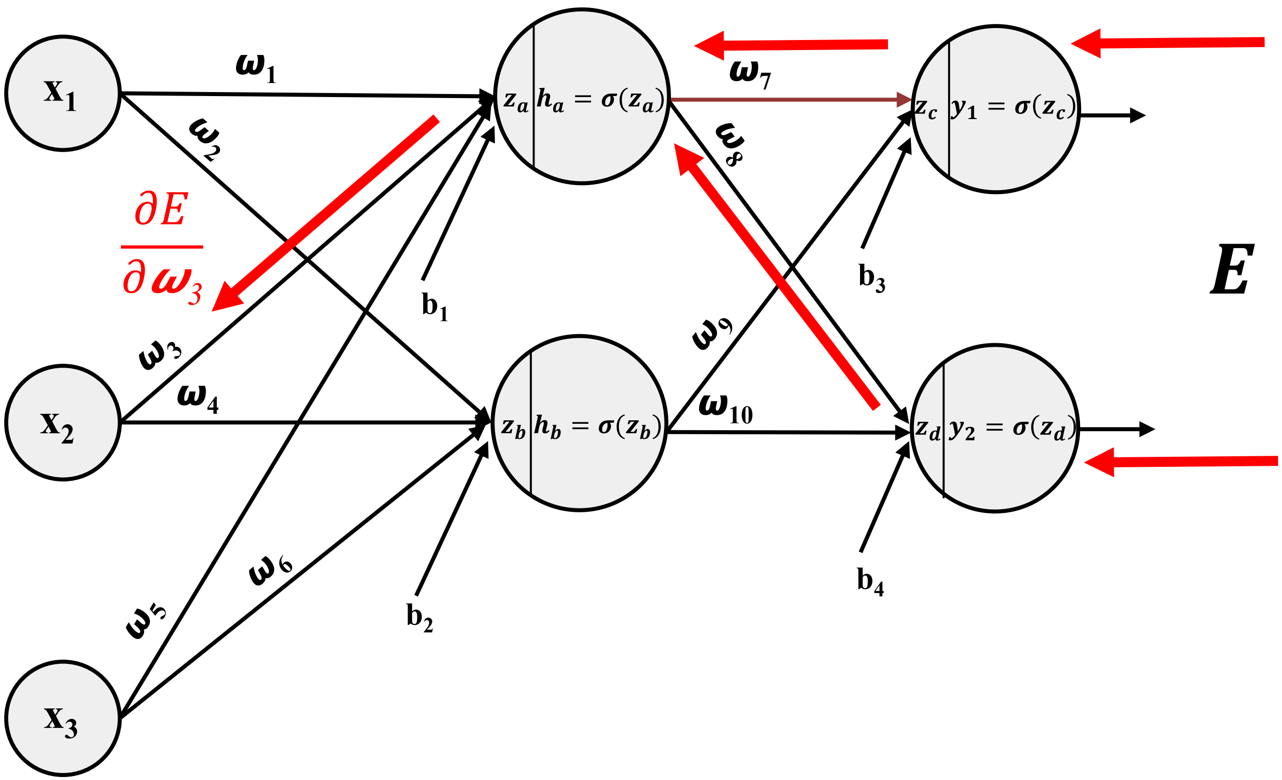
- How many steps are there for training MLPs?
  - There exist 3 fundamental steps:
    - Forward Pass
    - Error Backpropagation
    - Parameter updates

- Can Backpropagation be considered as the learning algorithm?
  - No, Backpropagation often misunderstood as the whole learning algorithm for multilayer networks
  - Backpropagation only refers to a method of computing gradient for intermediate layers

- How do you briefly describe the learning process in MLPs?
  - Learning is updating weights using gradient

Discussion

- How to compute $\frac{\partial E}{\partial \boldsymbol{\omega}_3}$?



$$\frac{\partial E}{\partial \boldsymbol{\omega}_3} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial z_c}\frac{\partial z_c}{\partial h_a}\frac{\partial h_a}{\partial z_a}\frac{\partial z_a}{\partial \boldsymbol{\omega}_3} + \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial z_d}\frac{\partial z_d}{\partial h_a}\frac{\partial h_a}{\partial z_a}\frac{\partial z_a}{\partial \boldsymbol{\omega}_3}$$

Conclusion

- ***Fully-connected Neural Networks (NNs)*** are stacks of linear functions and ***nonlinear activation functions***, thus, NNs have much more representational power than linear classifiers

- ***Backpropagation*** is a recursive application of ***the chain rule*** along a computational graph to compute the gradients of all parameters

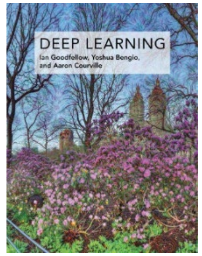
- In the ***forward operation,*** we compute ***the result of an operation*** and save any ***intermediates needed for the gradient computation in the memory***

- In the ***backward step,*** we apply ***the chain rule*** to compute the gradient of the loss function with respect to the inputs

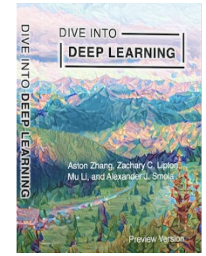
- ***Backpropagation is not guaranteed*** to find a ***"true" solution***, even if it exists, and lies within the capacity of the network to model.
  - The optimum for the loss function may not be the "true" solution

Reading Material

- Deep Learning Book
  - Chapter 6.5



- Dive into Deep Learning Book
  - Chapters 5.1 - 5.2 - 5.3



- Yann LeCun et al., 1988 “Efficient BackProp”
  - Paper (PDF)
- Video
  - Hugo Larochelle’s Neural Networks Lecture
  - Intuitively Understanding the Cross Entropy Loss
  - What is a neural network?
  - How do neural networks learn?
  - What is backpropagation really doing?
  - Backpropagation calculus
  - How Deep Neural Networks Work
  - Andrej Karpathy’s Stanford CS231 lecture
- Blogs
  - https://e2eml.school/blog.html
  - Cross-Entropy Loss Function
  - Gradient descent