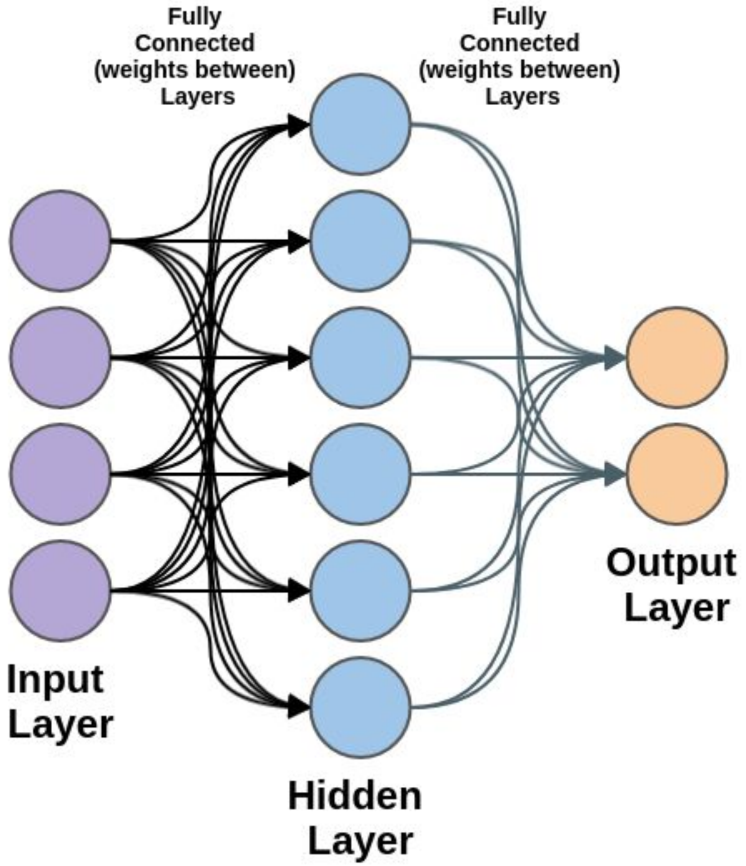
Multi-Class Classification

Solution

- Traditional Method: 1-vs-other method
  - Too slow. If we have n-classes, we need to train n models
  - Performance is not great, because the sample size is different for positive and negative classes
- Multiple Neurons
  - Use n output neuron to correspond n classes.
  - Easy, fast, and robust
  - Problem: how to model the probability? The values in the neural network can be negative or greater than 1.



Softmax: normalized exponential

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

Input: vector of reals

Output: **probability** distribution

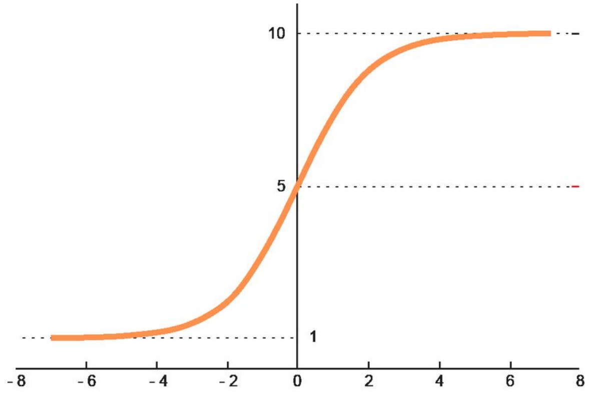
softmax([1,2,7,3,2]):

Calculate e^x: [2.72, 7.39, 1096.63, 20.09, 7.39]

Calculate sum(e^x): 2.72+7.39+1096.63+20.09+7.39 = 1134.22

Normalize: e^x/sum(e^x) = [0.002, 0.007, 0.967, 0.017, 0.007]

Result is a vector of reals.

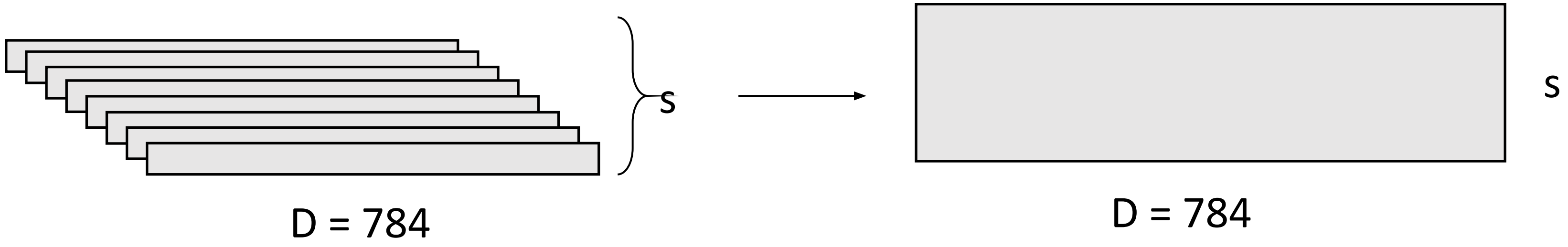


A Simple Example

Here, we will go over a simple 2-layer neural network (no bias).

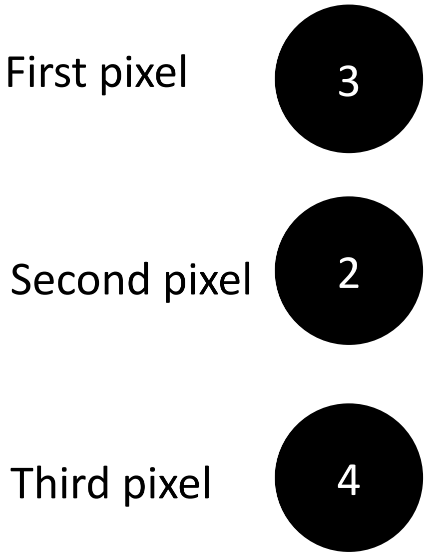
Mini-batch for Machine Learning

- Let’s say we have S images of size 28x28
- We use a matrix to represent data.



- When s is very large RAM overload
- In HW4, we will use a batch size of 128

Neural Network Easy Example

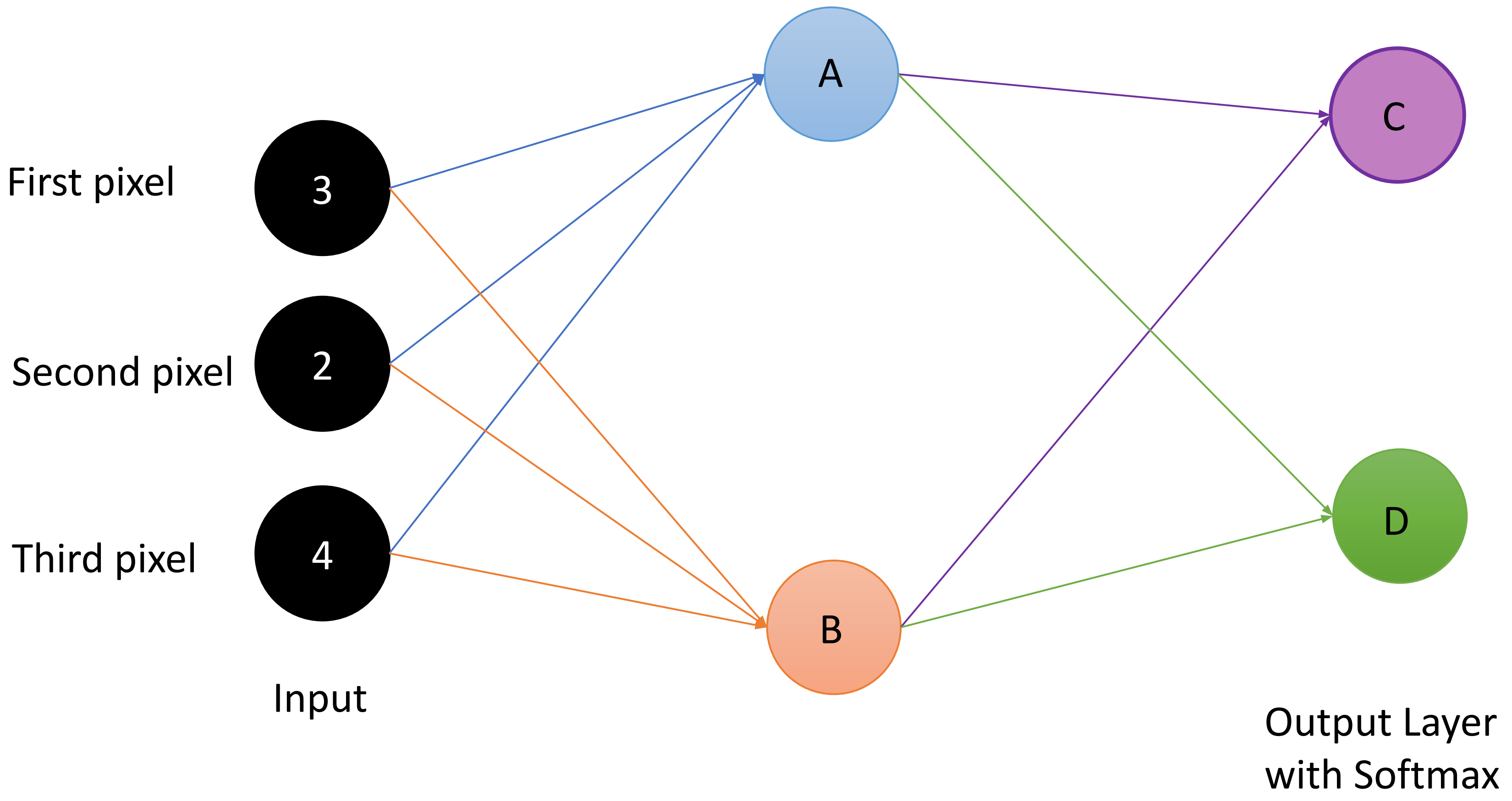


First Batch

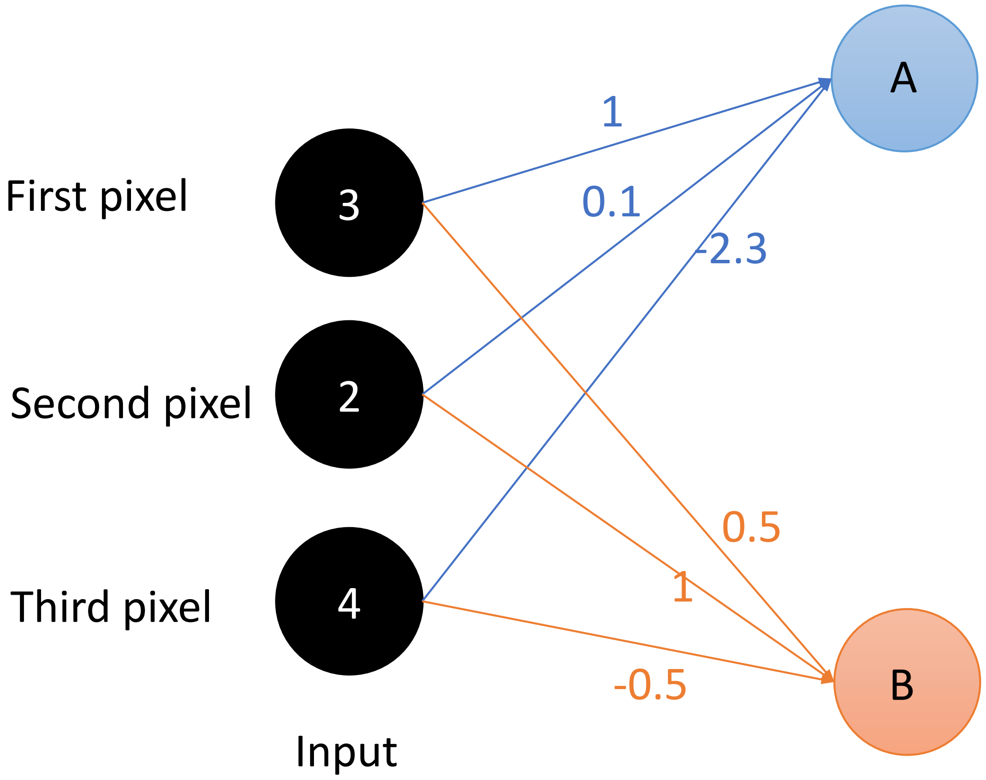
Input

$$X_{in} \quad \left.\begin{matrix} 3 & 2 & 4 \\ . & . & . \\ . & . & . \end{matrix}\right\} \text{Bsz} = 3$$

Neural Network Easy Example



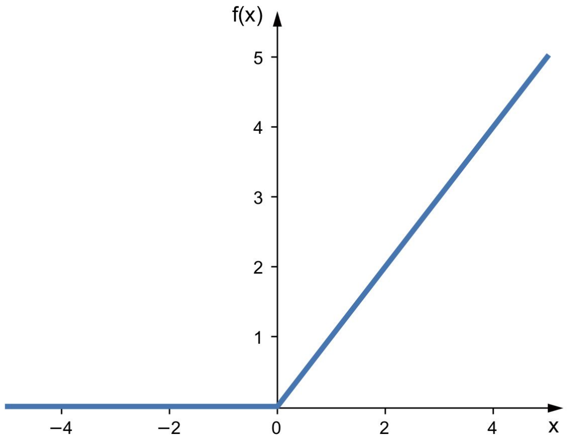
Neural Network Easy Example



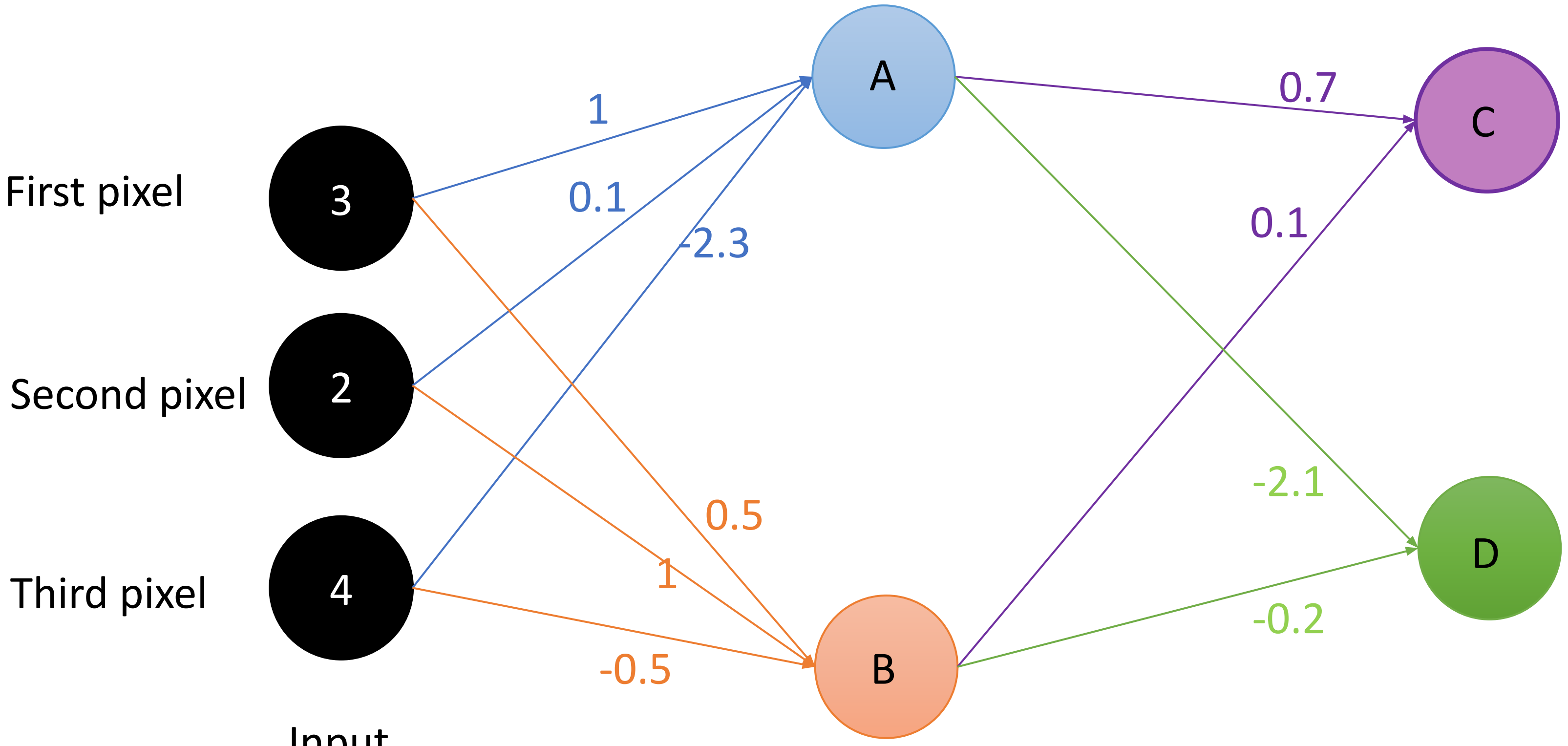
	3	2	4
X_{in}	.	.	.
	.	.	.

1-st Layer (ReLU)

	A	B
w_1	1	0.5
	0.1	1
	-2.3	-0.5



Neural Network Easy Example



Input

$$X_{in} \quad \begin{matrix} 3 & 2 & 4 \\ \vdots & \vdots & \vdots \end{matrix}$$

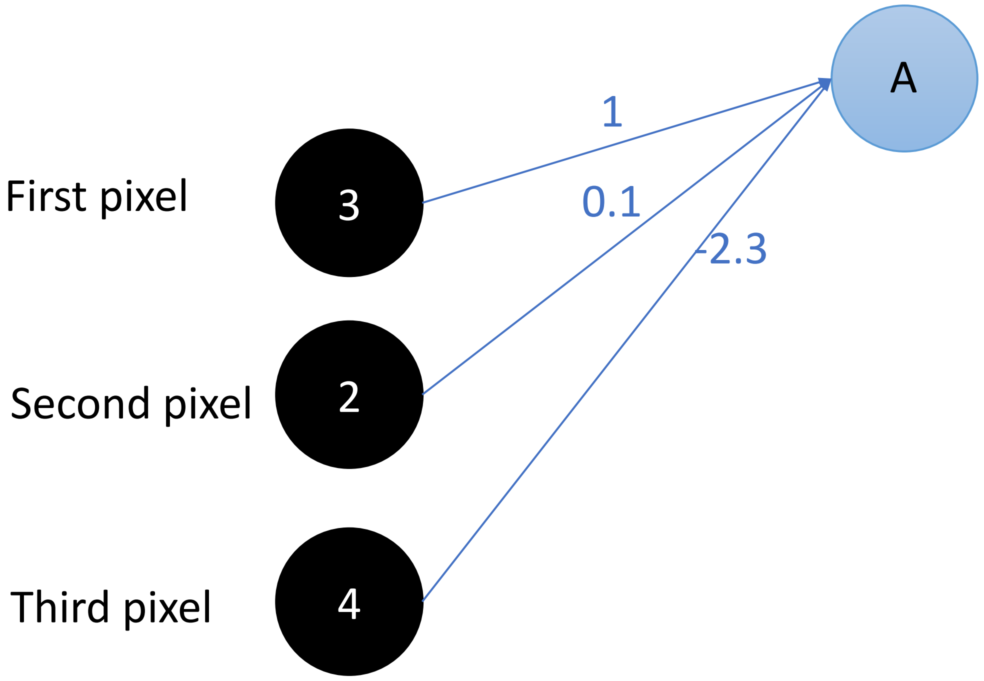
1-st Layer (ReLU)

$$w_1 \quad \begin{matrix} 1 & 0.5 \\ 0.1 & 1 \\ -2.3 & -0.5 \end{matrix}$$

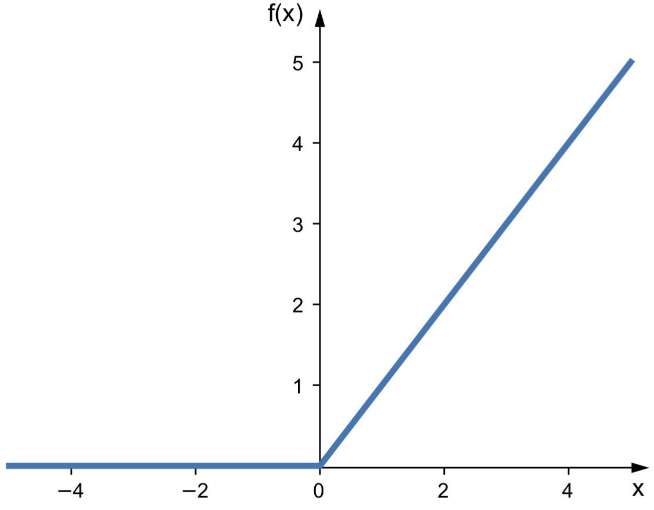
Output Layer with Softmax

$$w_2 \quad \begin{matrix} 0.7 & -2.1 \\ 0.1 & -0.2 \end{matrix}$$

1st Layer with ReLU

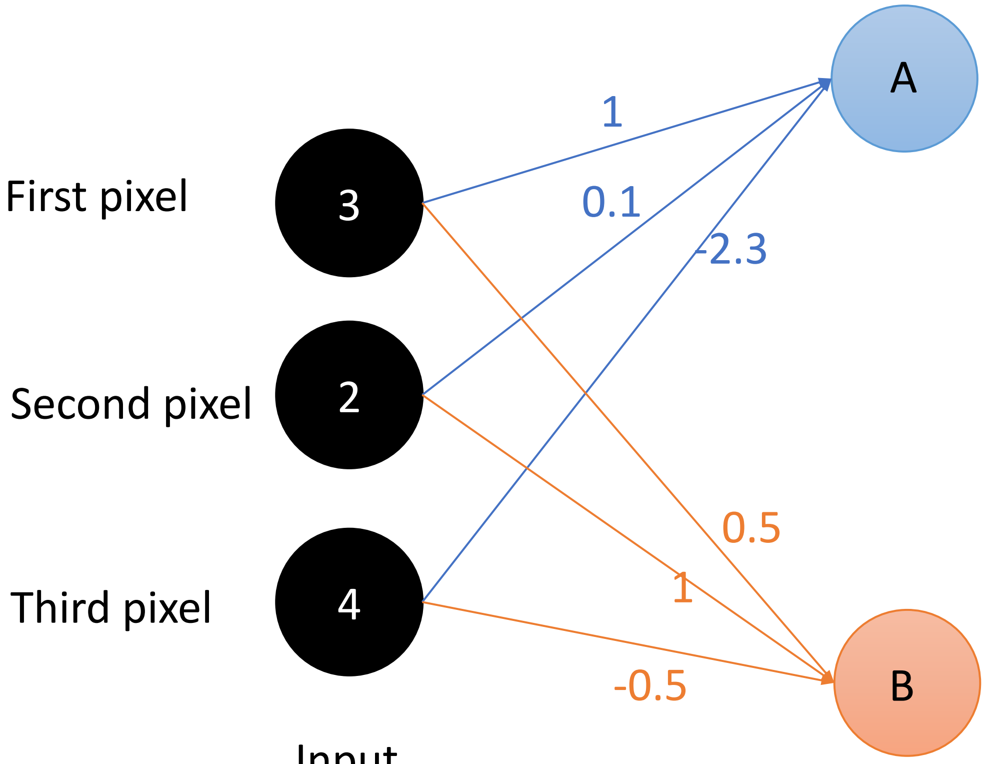


$$Output_A = 3 * 1 + 2 * 0.1 + 4 * -2.3 = -6.0$$
$$Output_A = 0\ (after\ RELU)$$



RELU

Neural Network Easy Example



$$\boldsymbol{A = 0}$$

$$\boldsymbol{B = 3 * 0.5 + 2 * 1 + 4 * -0.5 = 1.5}$$

Input

X_{in}

3	2	4
.	.	.
.	.	.

1-st Layer (ReLU)

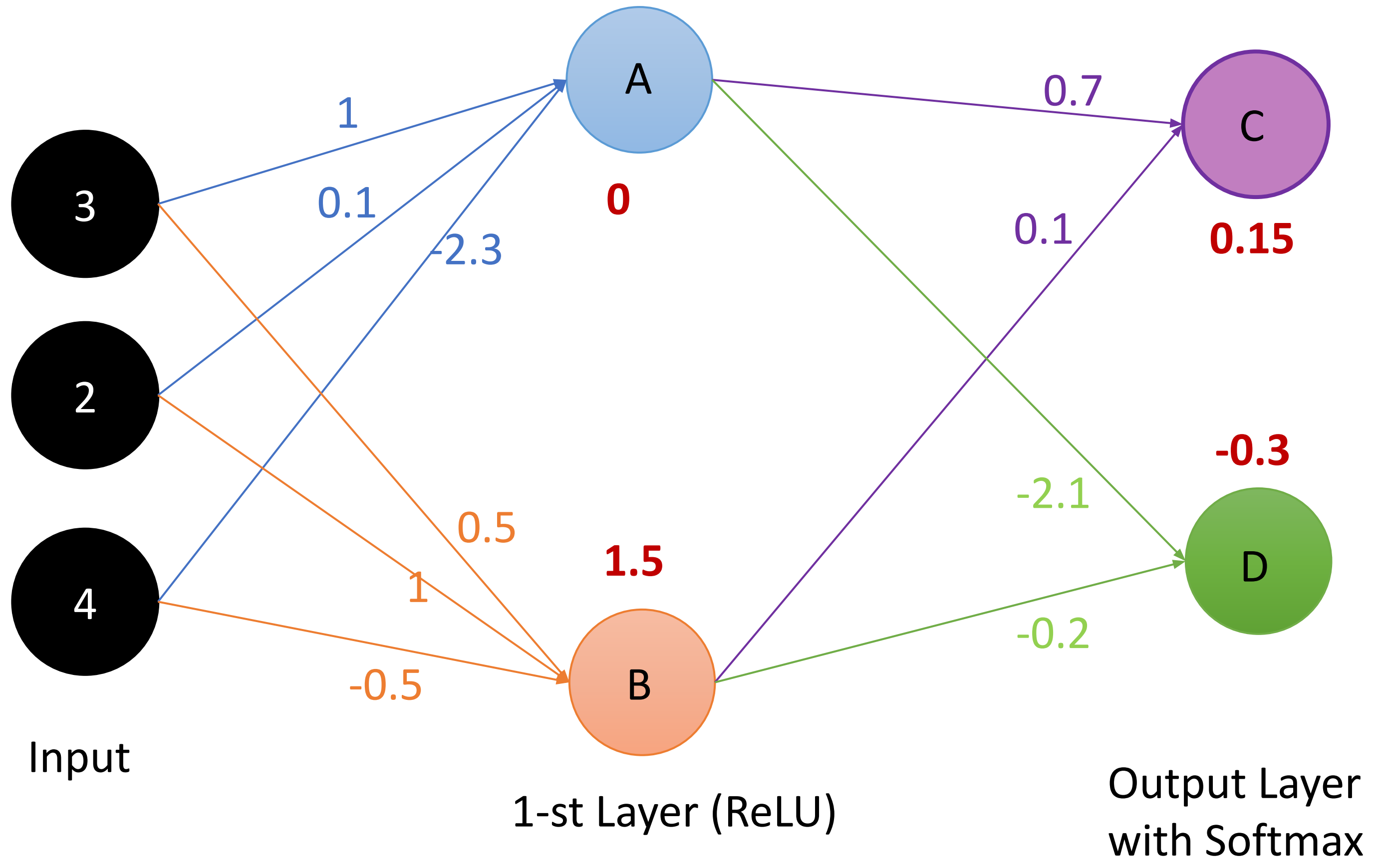
w_1

1	0.5
0.1	1
-2.3	-0.5

o_1

0	1.5
.	.
.	.

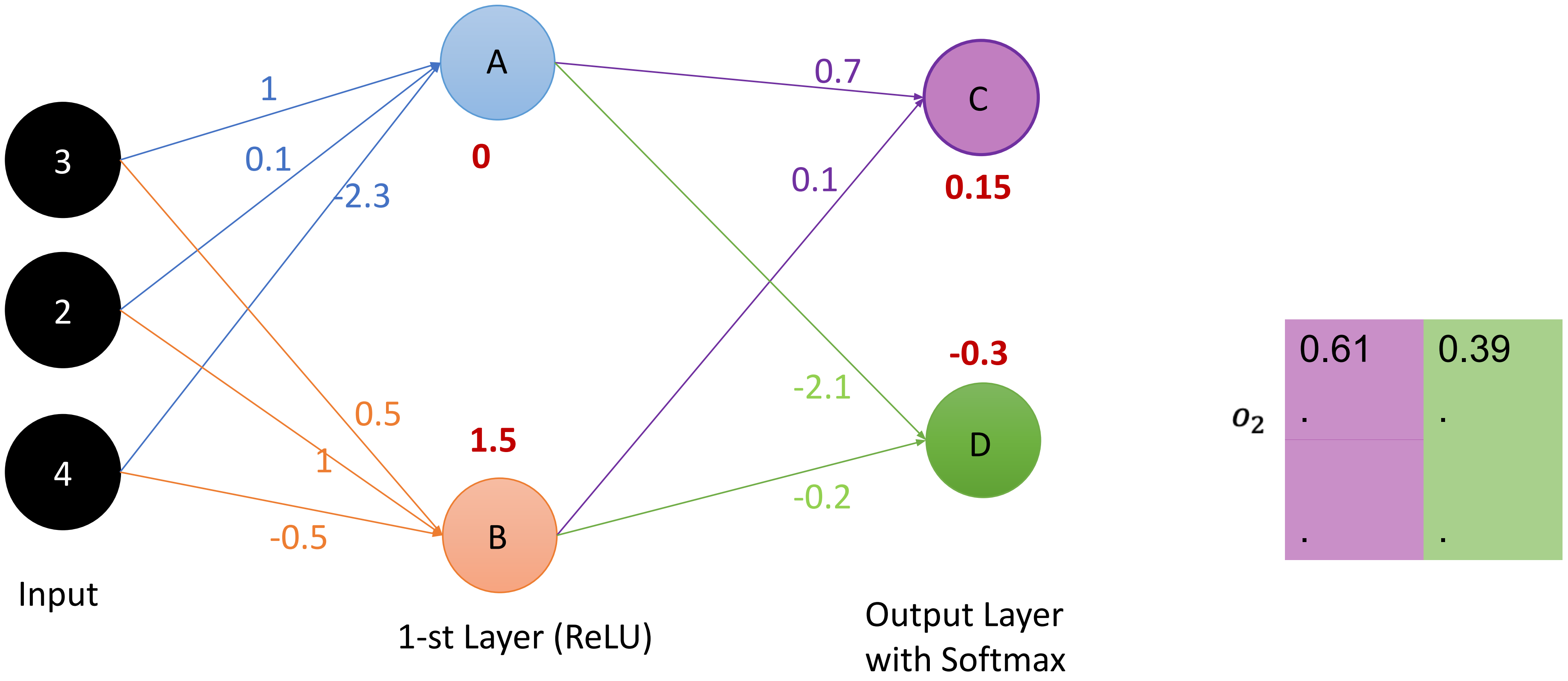
Neural Network Easy Example



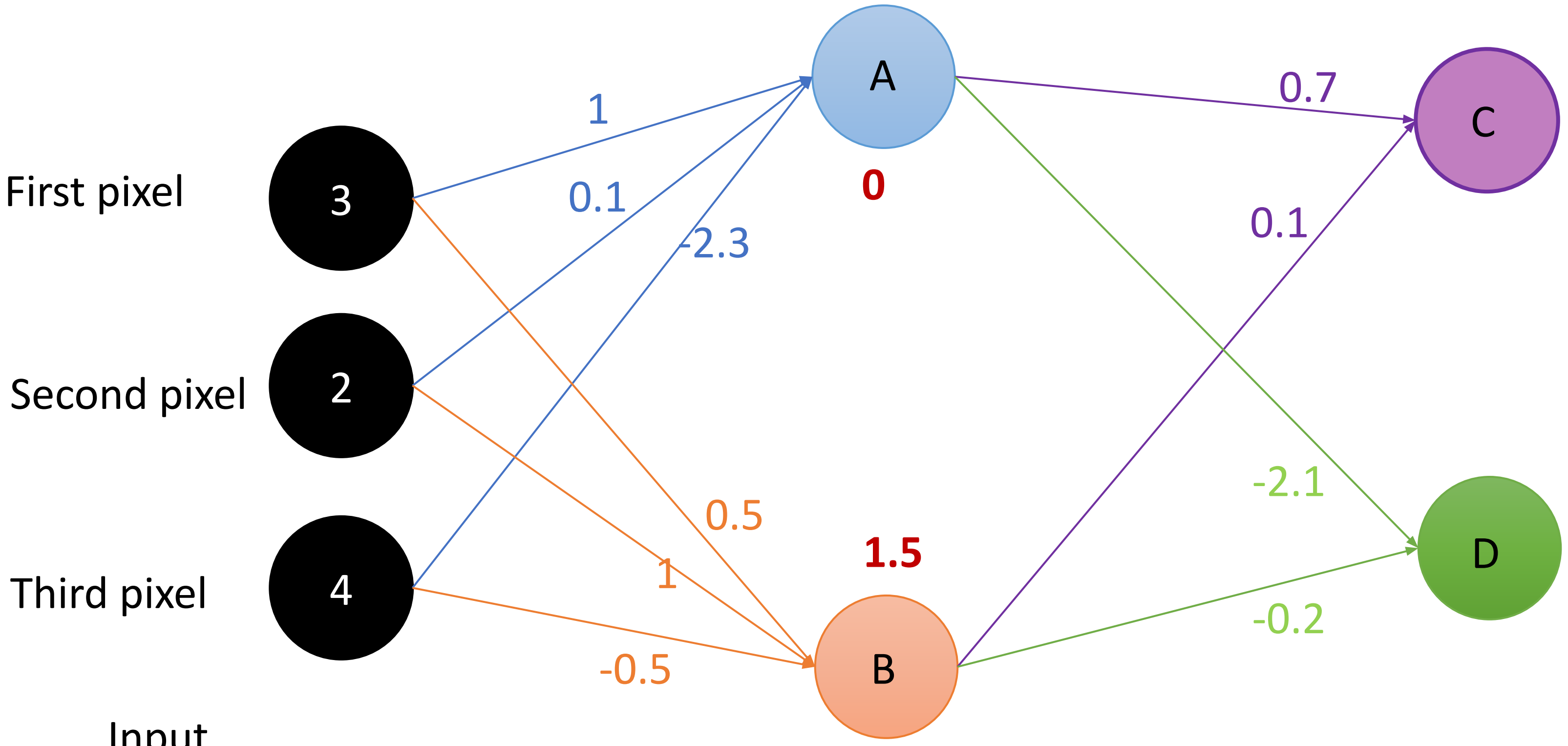
$$output_C = \frac{e^{0.15}}{e^{0.15} + e^{-0.3}} \approx \frac{1.16}{1.16 + 0.74}$$
$$= 0.61$$

$$output_D = \frac{e^{-0.3}}{e^{0.15} + e^{-0.3}} \approx \frac{0.74}{1.16 + 0.74}$$
$$= 0.39$$

Neural Network Easy Example



Neural Network Easy Example



Input

X_{in}

3	2	4
.	.	.
.	.	.

1-st Layer (ReLU)

w_1

1	0.5
0.1	1
-2.3	-0.5

o_1

0	1.5
.	.
.	.

Output Layer
with Softmax

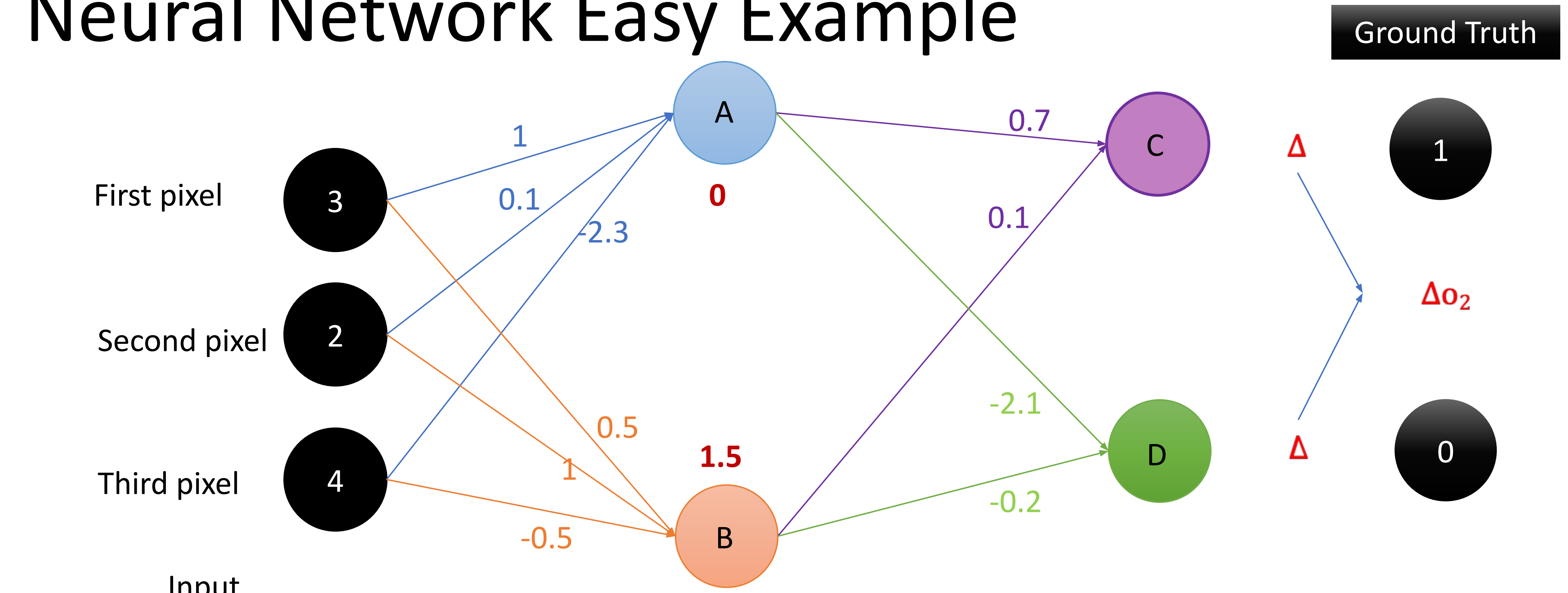
w_2

0.7	-2.1
0.1	-0.2

o_2

0.61	0.39
.	.
.	.

Neural Network Easy Example



Input

$$X_{in} \quad \begin{matrix} 3 & 2 & 4 \\ . & . & . \\ \\ . & . & . \end{matrix}$$

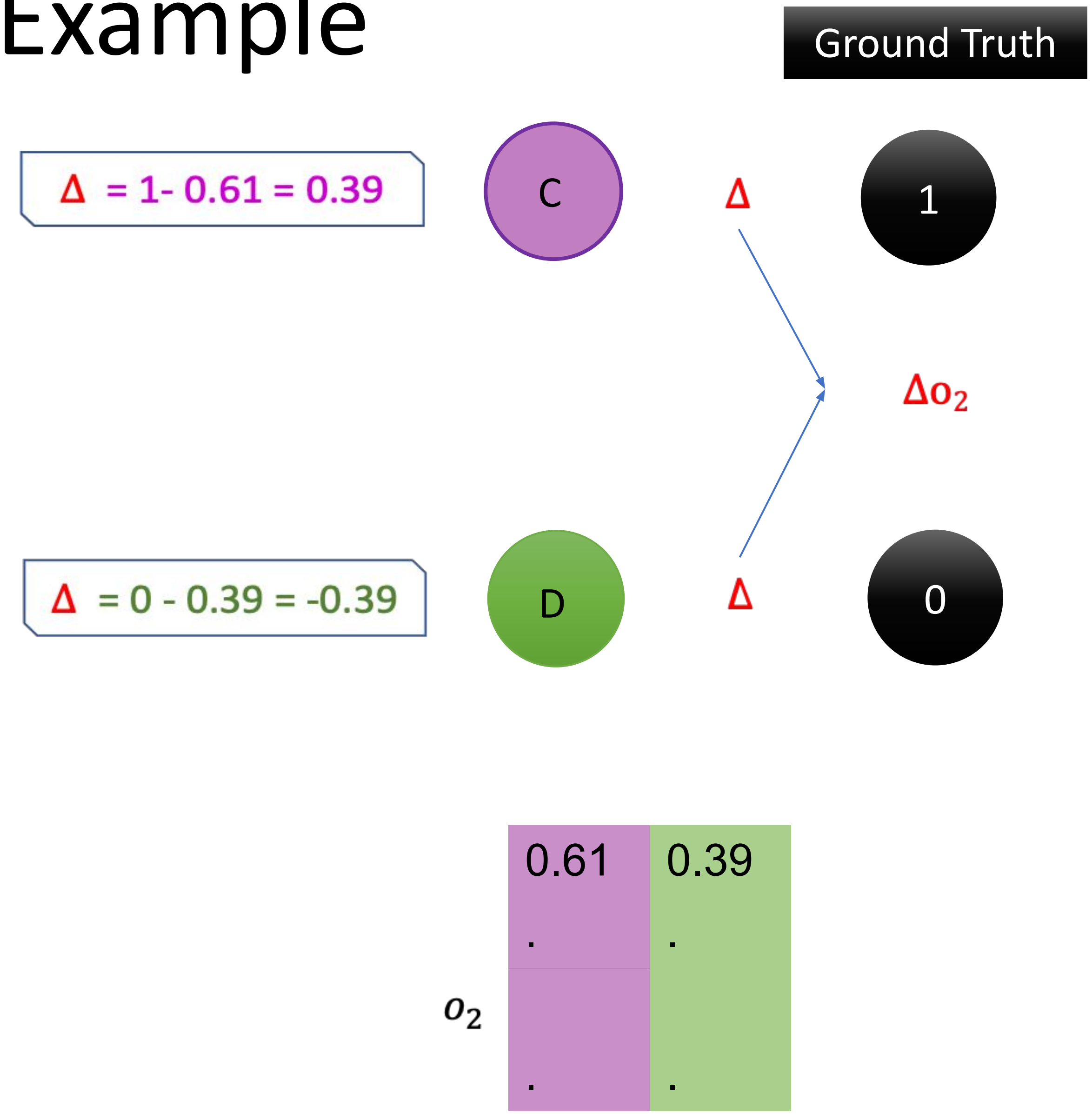
1-st Layer (ReLU)

$$w_1 \quad \begin{matrix} 1 & 0.5 \\ 0.1 & 1 \\ -2.3 & -0.5 \end{matrix} \qquad o_1 \quad \begin{matrix} 0 & 1.5 \\ . & . \\ \\ . & . \end{matrix}$$

Output Layer
with Softmax

$$w_2 \quad \begin{matrix} 0.7 & -2.1 \\ 0.1 & -0.2 \end{matrix} \qquad o_2 \quad \begin{matrix} 0.61 & 0.39 \\ . & . \\ \\ . & . \end{matrix}$$

Neural Network Easy Example



What to do with this error?

What to do with this error?

Backpropagate it to update the weights!

How?

A Computational Graph Perspective

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

Figure taken from Ranjay Krishna's course CSE493

A Computational Graph Perspective

Backpropagation: a simple example

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

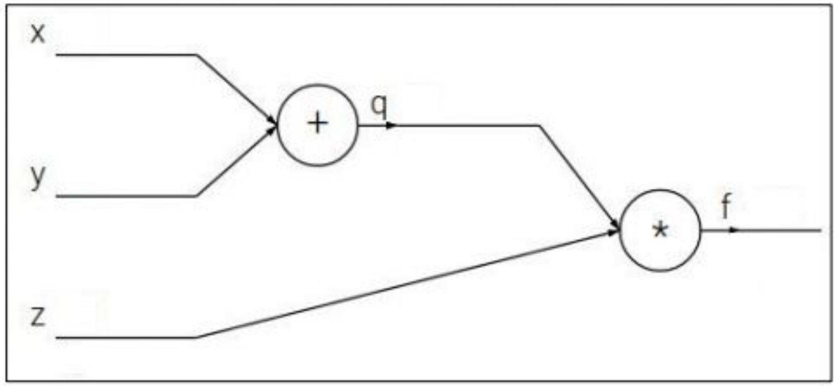


Figure taken from Ranjay Krishna’s course CSE493

A Computational Graph Perspective

Backpropagation: a simple example

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

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

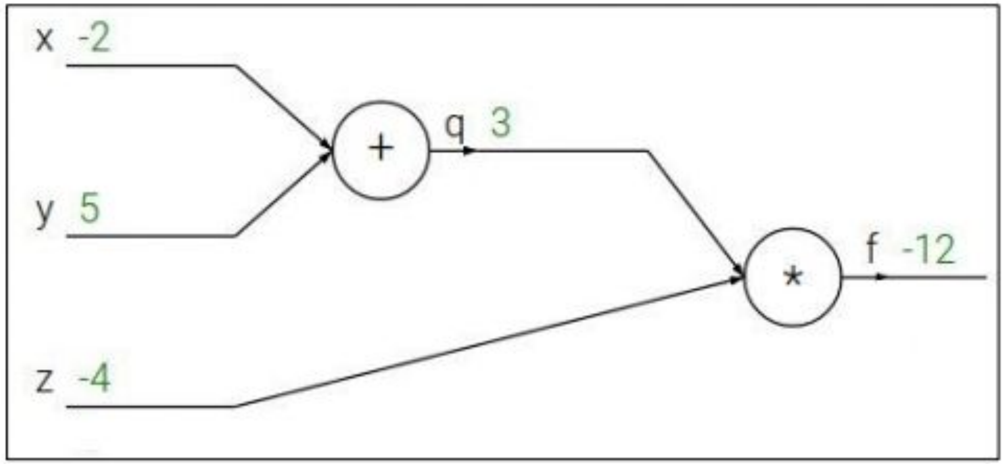


Figure taken from Ranjay Krishna's course CSE493

A Computational Graph Perspective

Backpropagation: a simple example

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

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

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

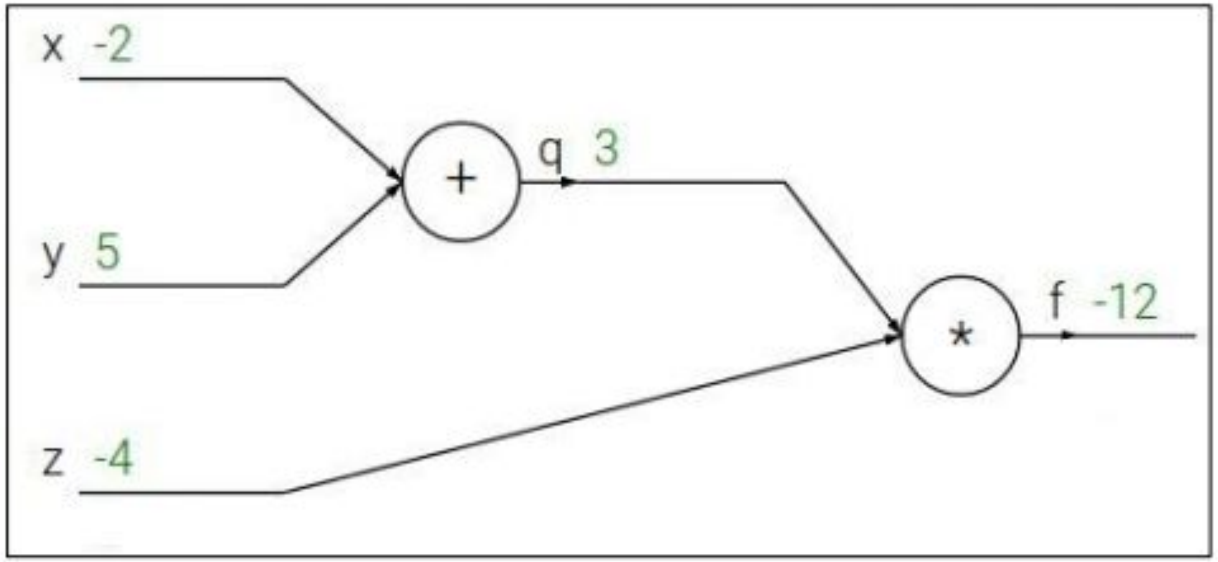


Figure taken from Ranjay Krishna's course CSE493

A Computational Graph Perspective

Backpropagation: a simple example

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

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

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

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

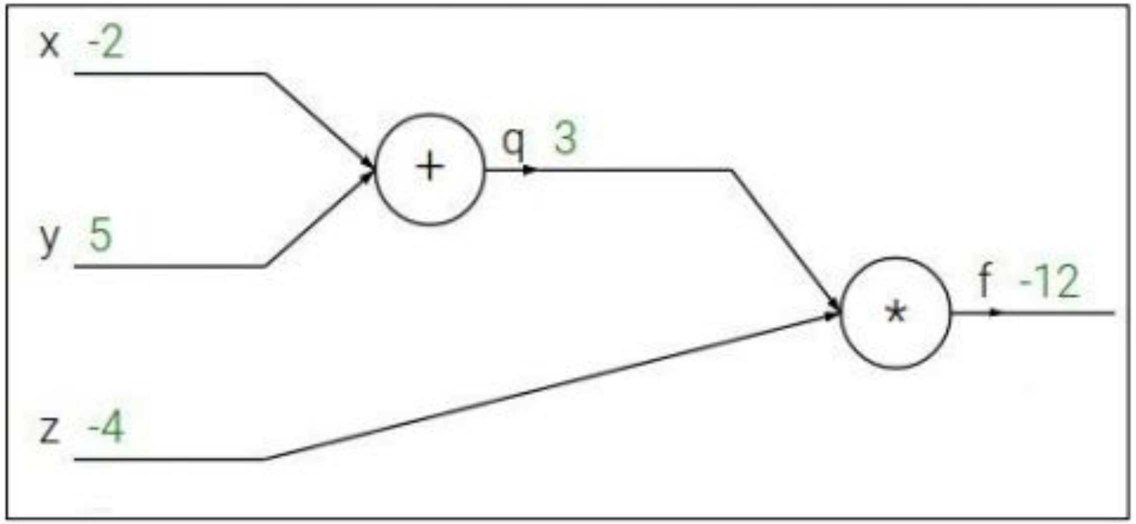


Figure taken from Ranjay Krishna’s course CSE493

A Computational Graph Perspective

Backpropagation: a simple example

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

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

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

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

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

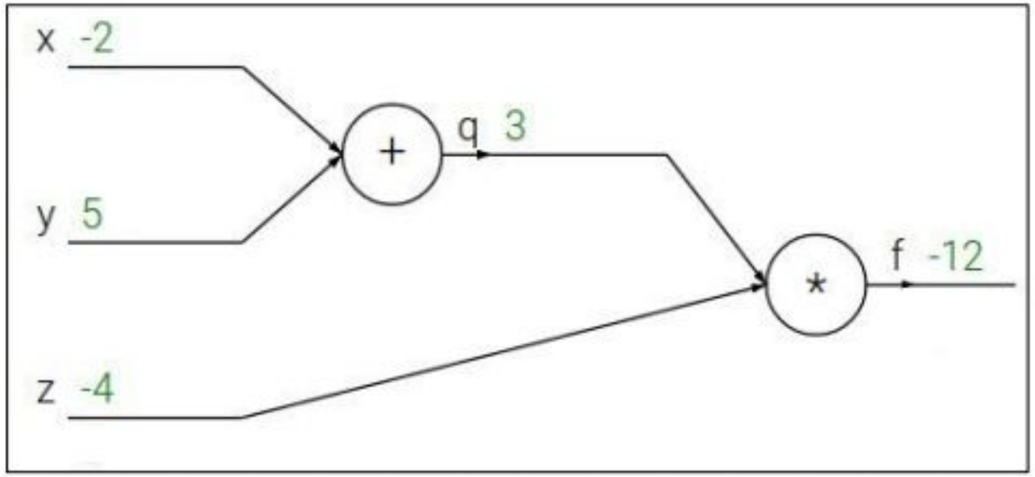


Figure taken from Ranjay Krishna's course CSE493

A Computational Graph Perspective

Backpropagation: a simple example

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

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

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

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

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

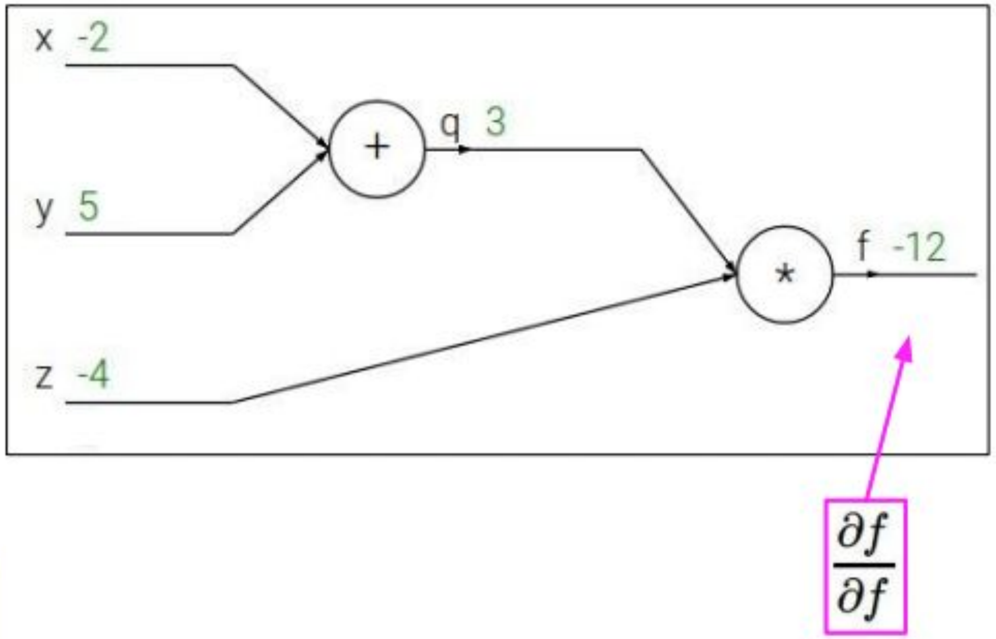


Figure taken from Ranjay Krishna’s course CSE493

A Computational Graph Perspective

Backpropagation: a simple example

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

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

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

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

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

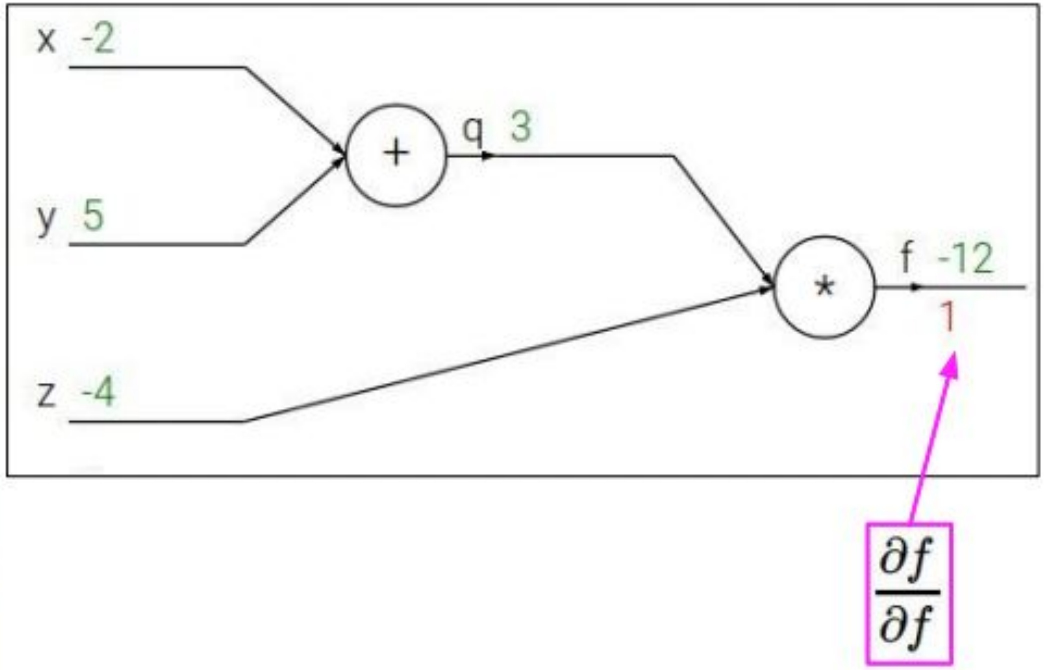


Figure taken from Ranjay Krishna’s course CSE493

A Computational Graph Perspective

Backpropagation: a simple example

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

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

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

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

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

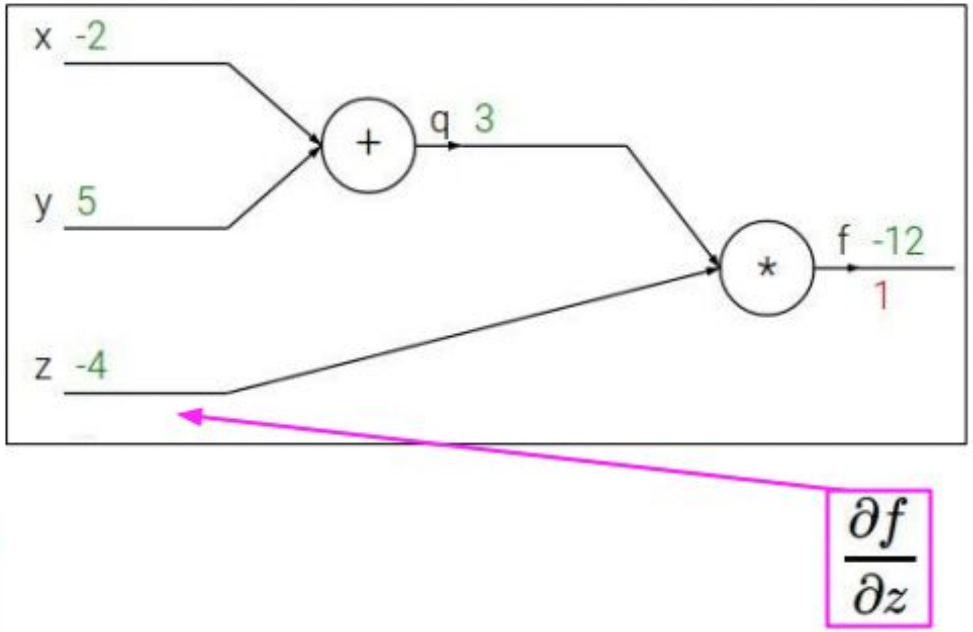


Figure taken from Ranjay Krishna’s course CSE493

A Computational Graph Perspective

Backpropagation: a simple example

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

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

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

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

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

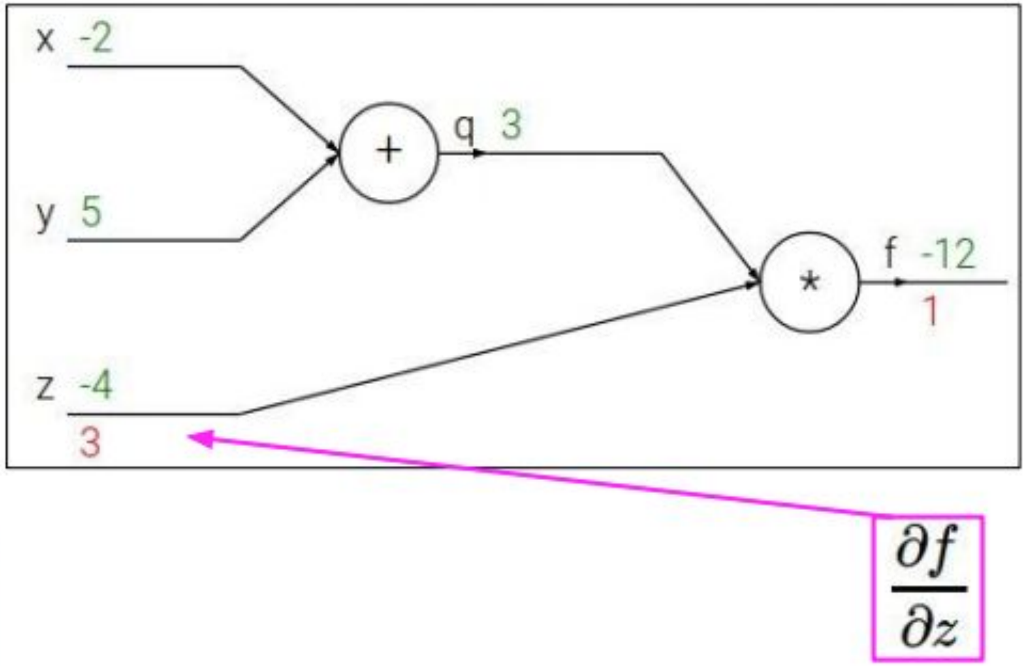


Figure taken from Ranjay Krishna’s course CSE493

A Computational Graph Perspective

Backpropagation: a simple example

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

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

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

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

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

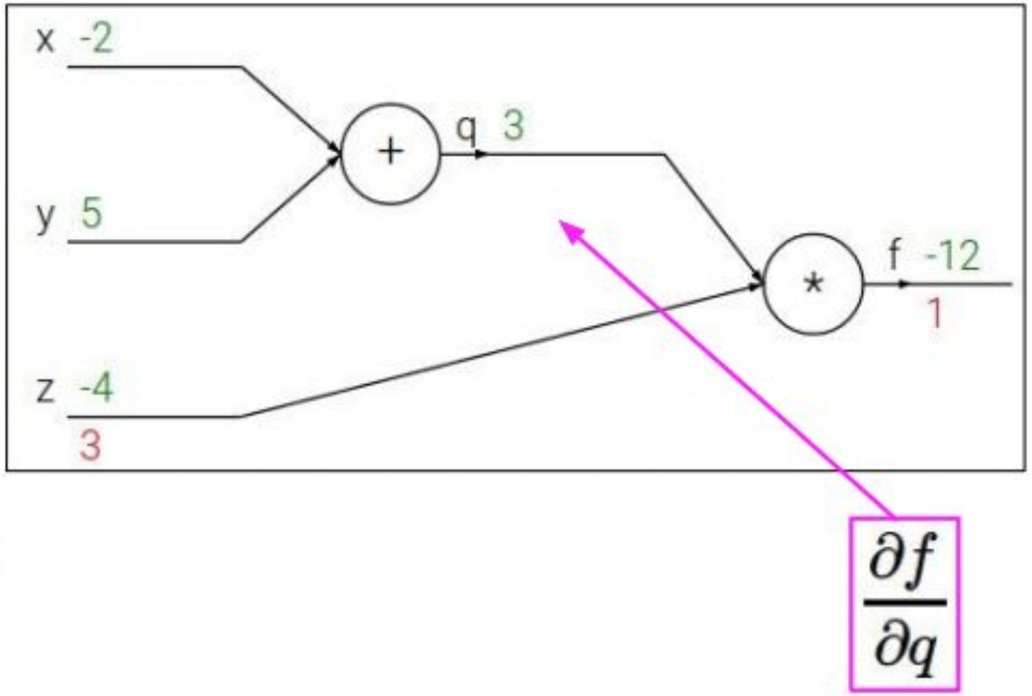


Figure taken from Ranjay Krishna’s course CSE493

A Computational Graph Perspective

Backpropagation: a simple example

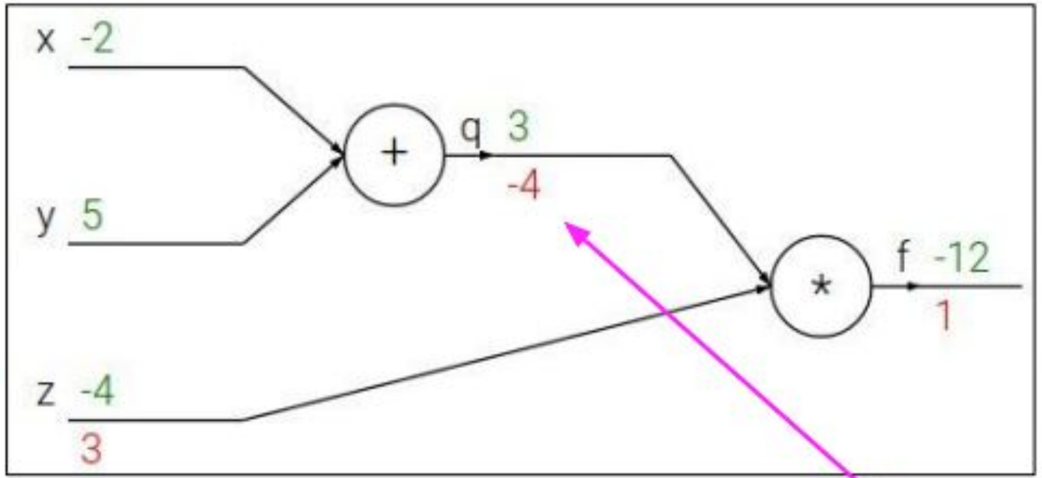
$$f(x, y, z) = (x + y)z$$

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

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

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

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial q}$$

Figure taken from Ranjay Krishna’s course CSE493

A Computational Graph Perspective

Backpropagation: a simple example

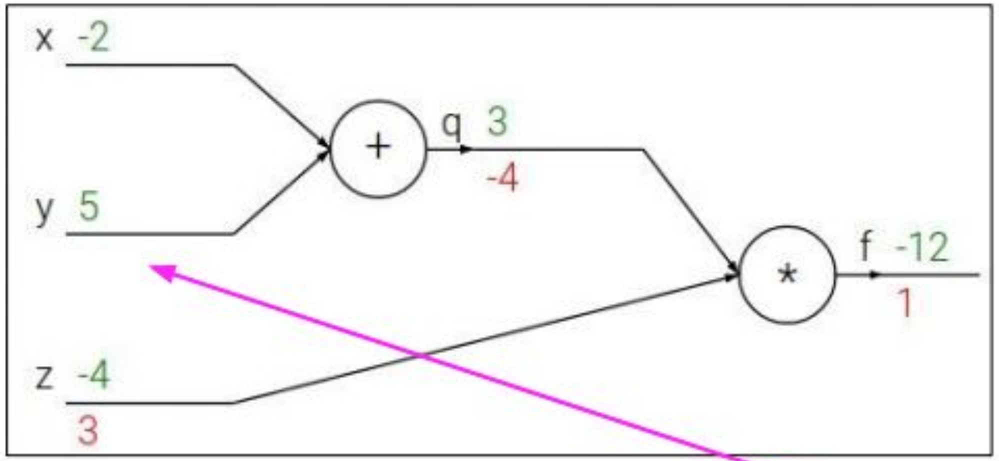
$$f(x,y,z) = (x+y)z$$

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

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

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

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial y}$$

Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$$

Upstream gradient

Local gradient

Figure taken from Ranjay Krishna’s course CSE493

A Computational Graph Perspective

Backpropagation: a simple example

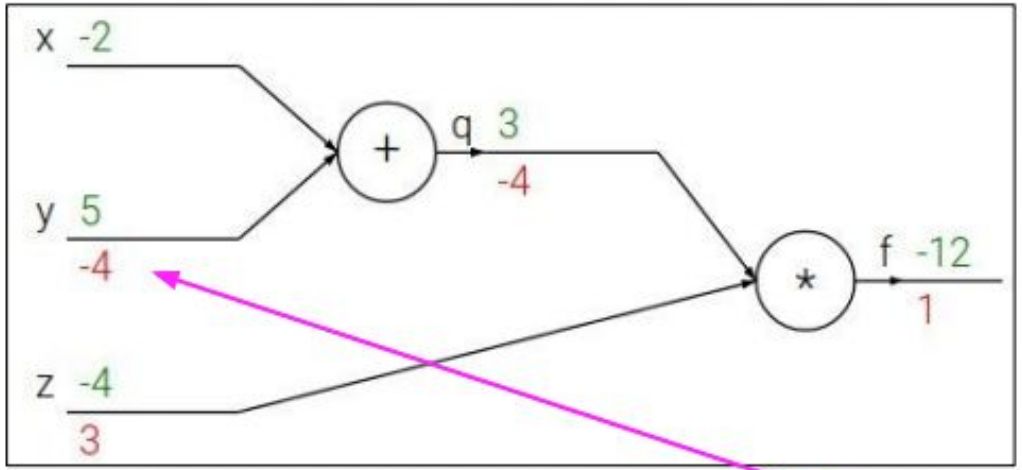
$$f(x, y, z) = (x + y)z$$

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

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

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

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial y}$$

Chain rule:

$$\boxed{\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}}$$

Upstream gradient

Local gradient

Figure taken from Ranjay Krishna’s course CSE493

A Computational Graph Perspective

Backpropagation: a simple example

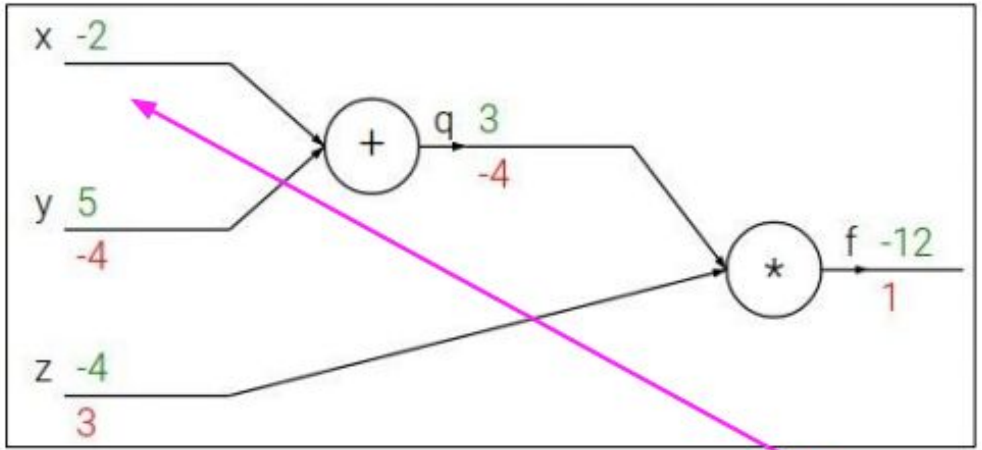
$$f(x, y, z) = (x + y)z$$

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

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

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

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\boxed{\frac{\partial f}{\partial x}}$$

Chain rule:

$$\boxed{\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}}$$

Upstream gradient

Local gradient

Figure taken from Ranjay Krishna’s course CSE493

A Computational Graph Perspective

Backpropagation: a simple example

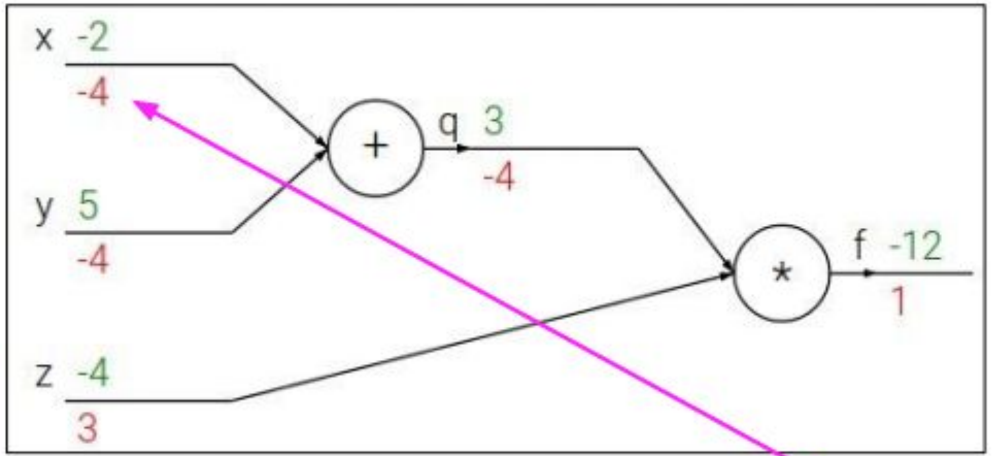
$$f(x, y, z) = (x + y)z$$

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

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

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

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$\boxed{\frac{\partial f}{\partial x}}$

Chain rule:

$$\boxed{\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}}$$

Upstream gradient   Local gradient

Figure taken from Ranjay Krishna's course CSE493

One more example

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

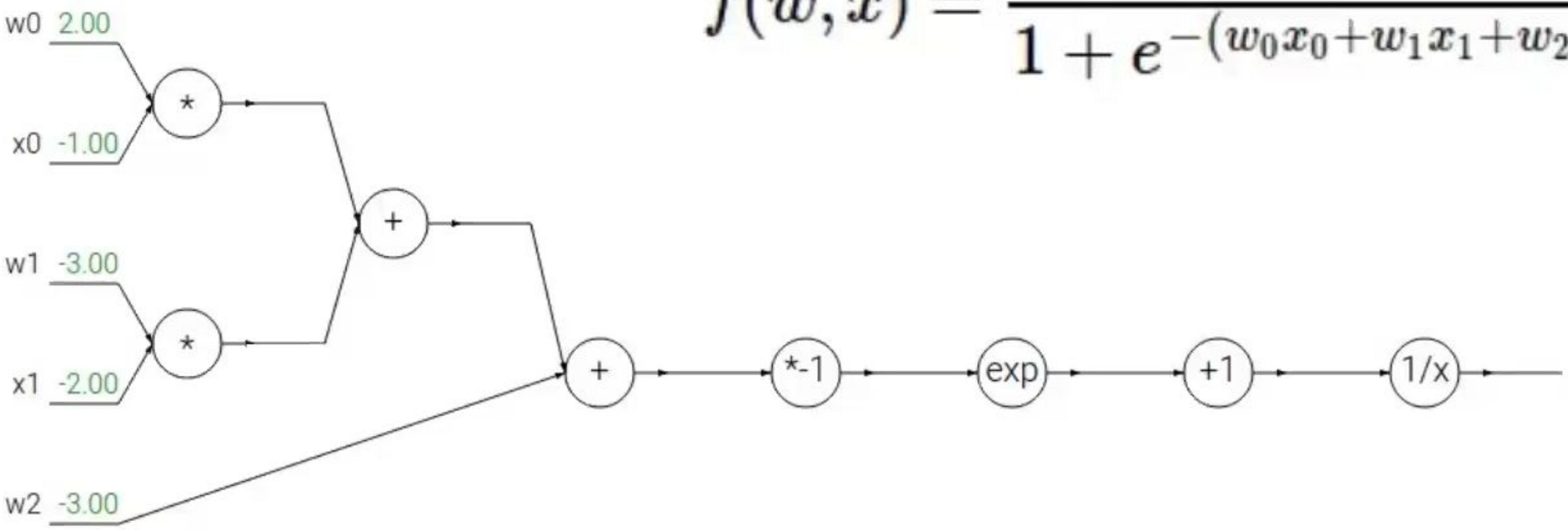


Figure taken from Ranjay Krishna's course CSE493

One more example

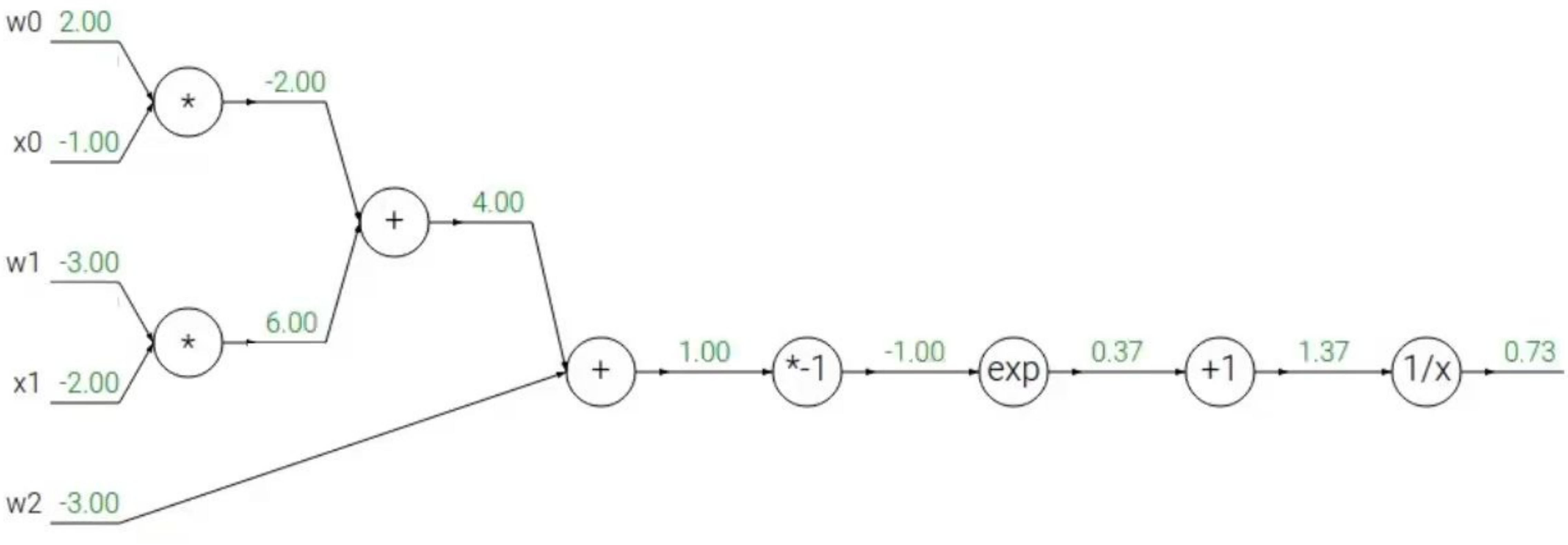
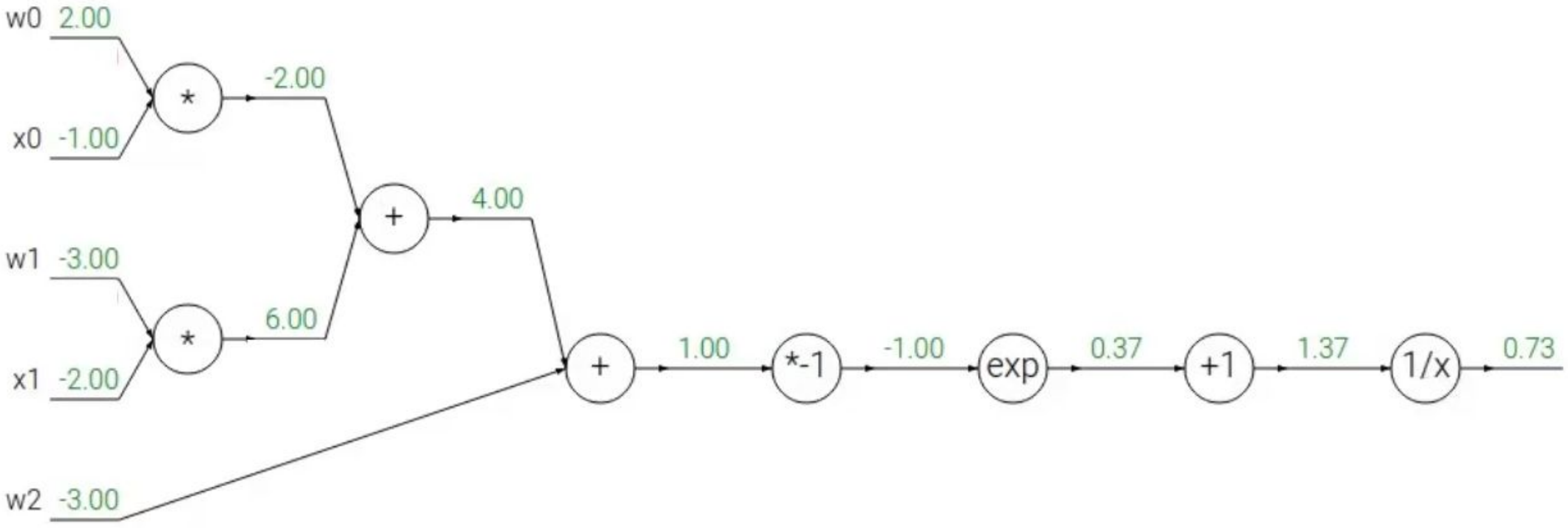


Figure taken from Ranjay Krishna’s course CSE493

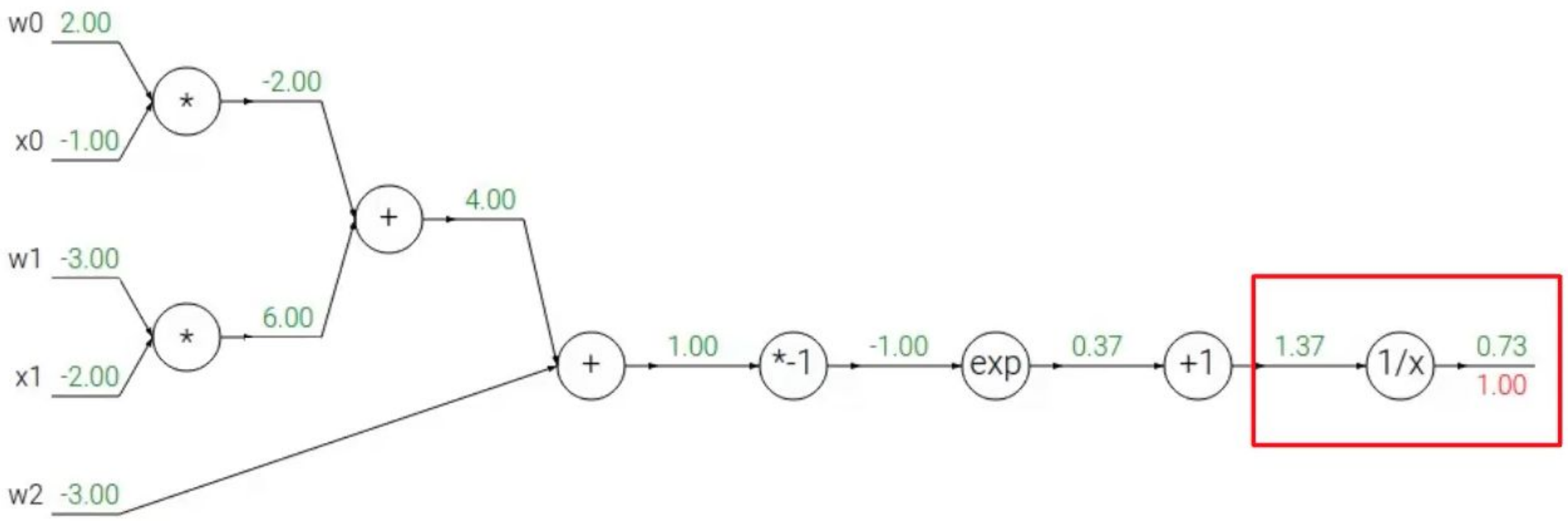
One more example



$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$	$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$
$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$	$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$

Figure taken from Ranjay Krishna’s course CSE493

One more example



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

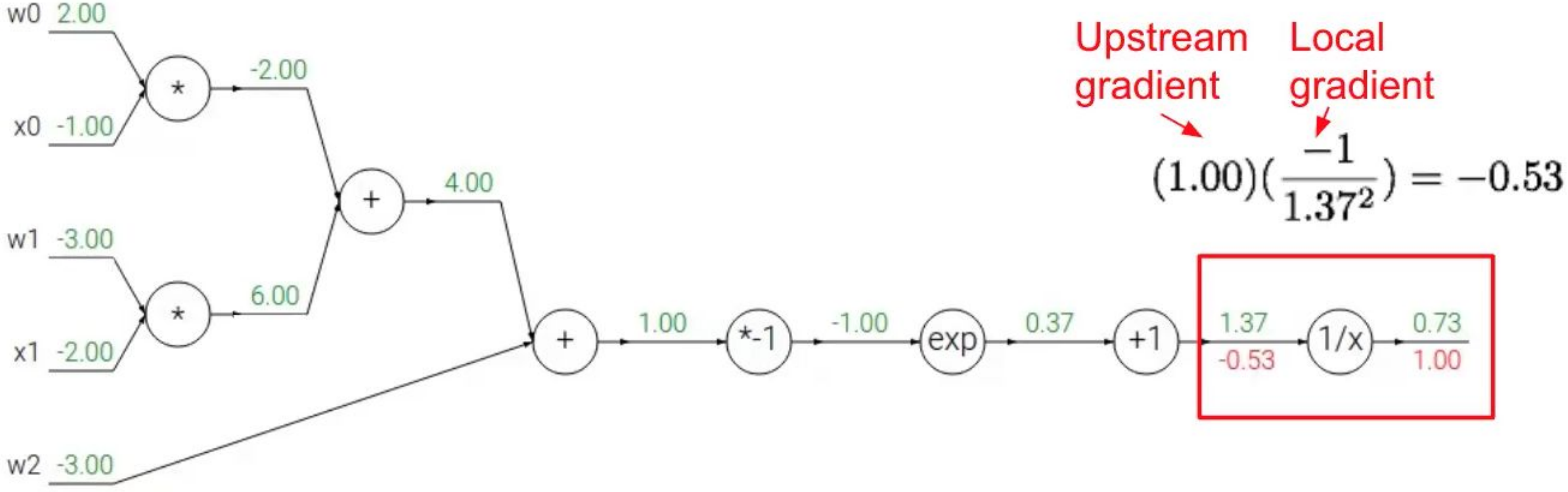
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$\boxed{f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2}$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Figure taken from Ranjay Krishna’s course CSE493

One more example



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

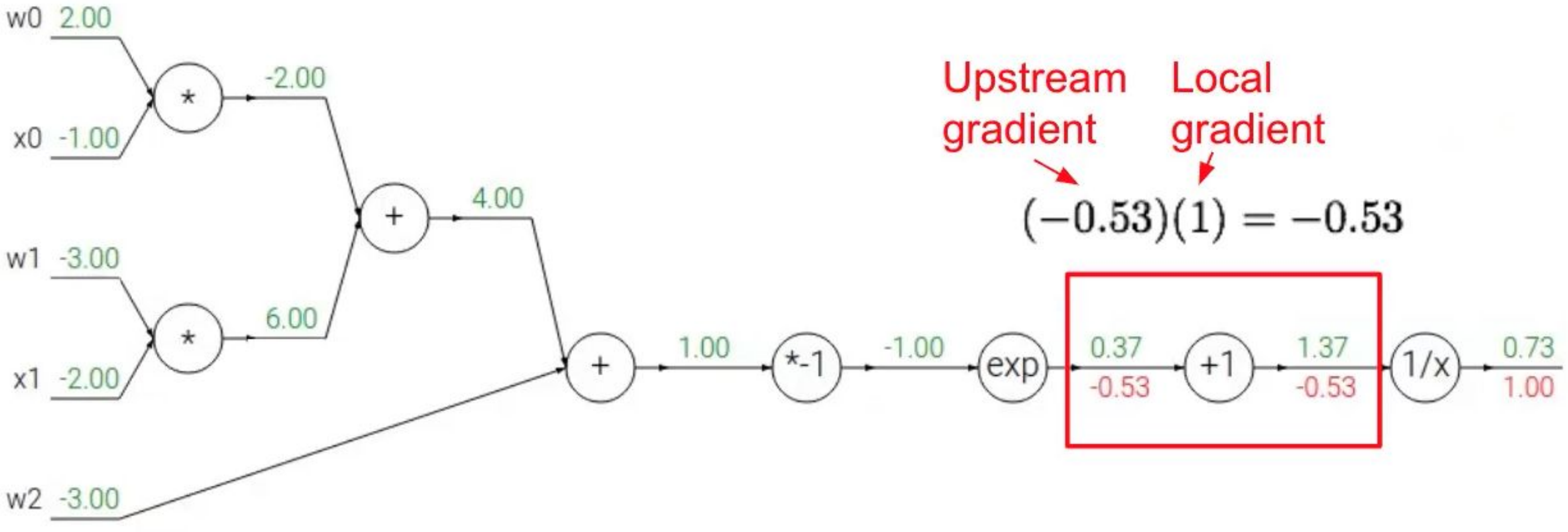
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$\boxed{f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2}$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Figure taken from Ranjay Krishna's course CSE493

One more example



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

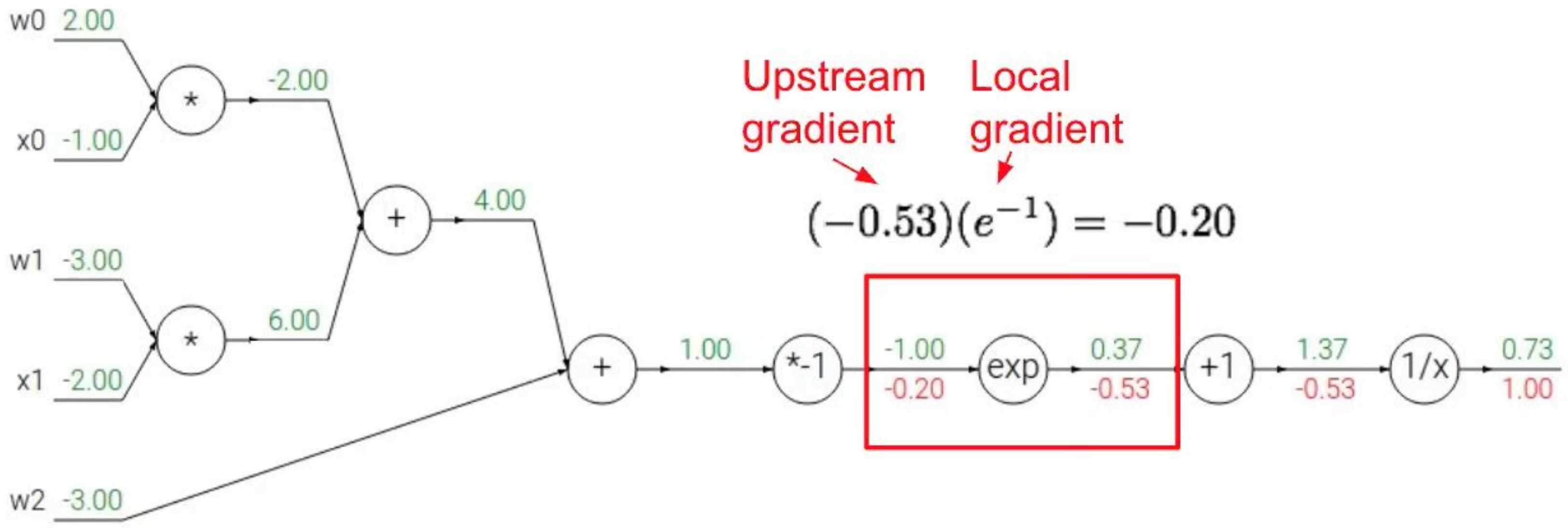
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$\boxed{f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1}$$

Figure taken from Ranjay Krishna’s course CSE493

One more example



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

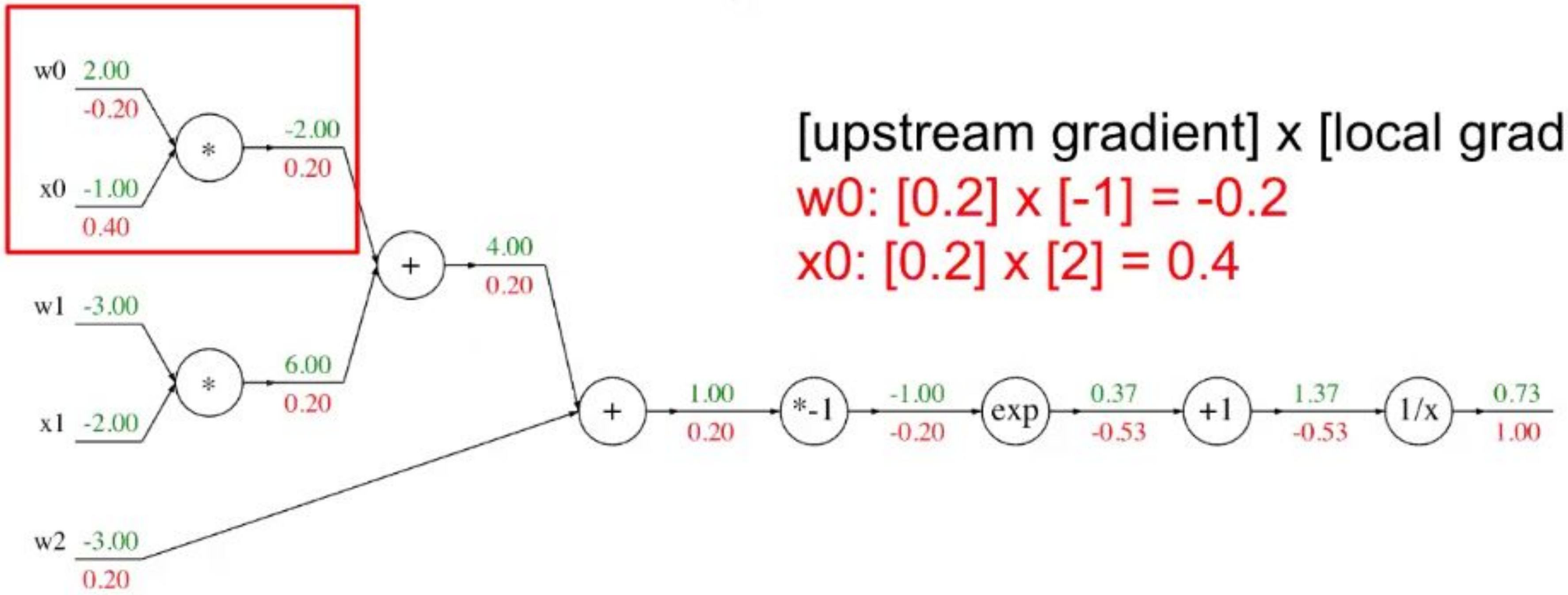
$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Figure taken from Ranjay Krishna’s course CSE493

One more example



[upstream gradient] x [local gradient]

w0: [0.2] x [-1] = -0.2

x0: [0.2] x [2] = 0.4

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Figure taken from Ranjay Krishna's course CSE493

One more example

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

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

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

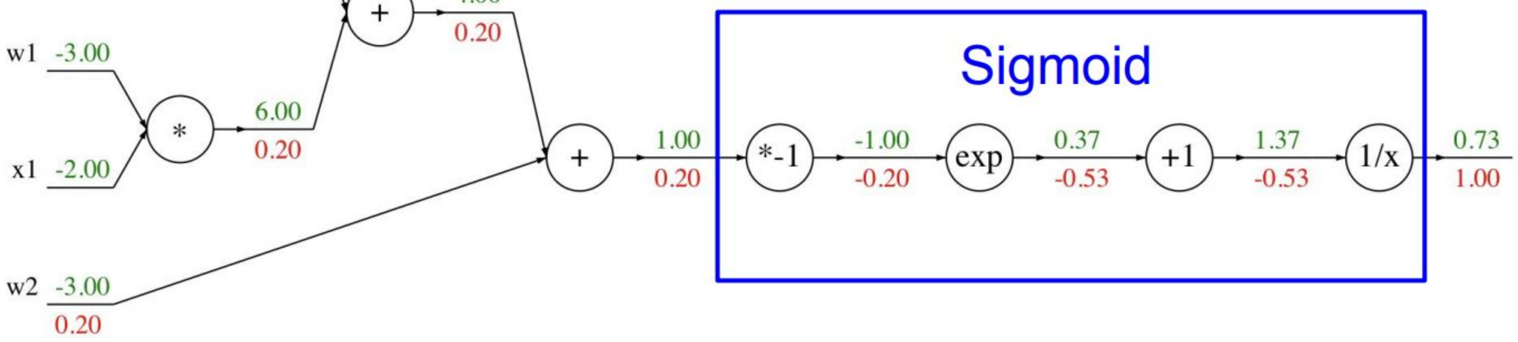


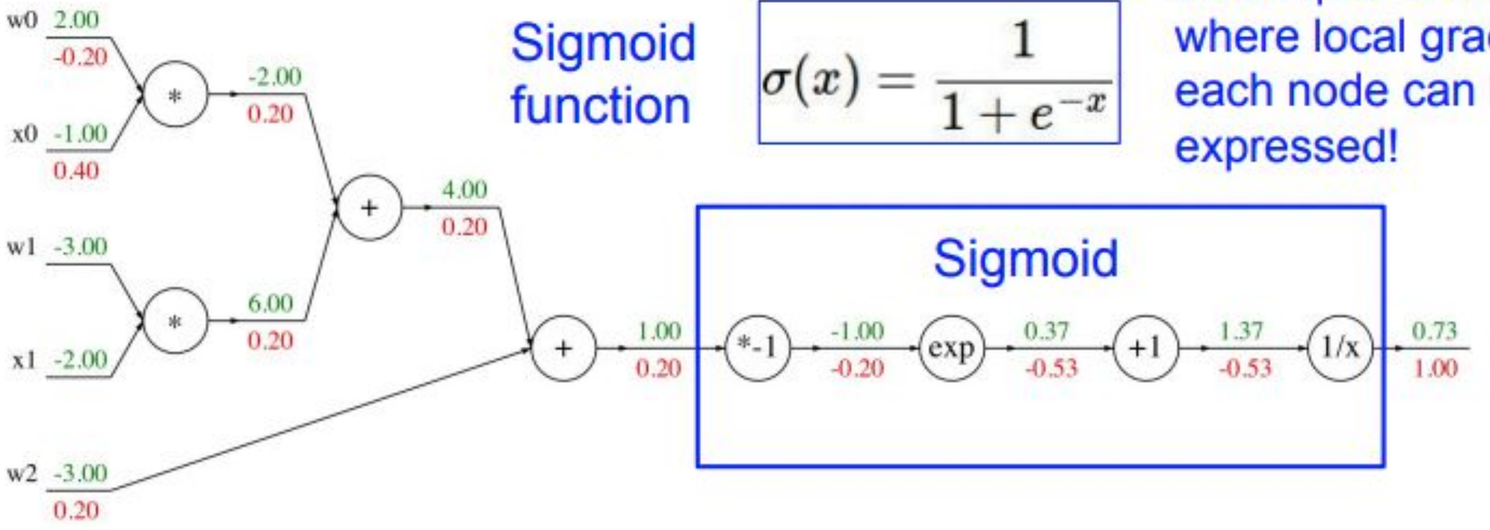
Figure taken from Ranjay Krishna’s course CSE493

One more example

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

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

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



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

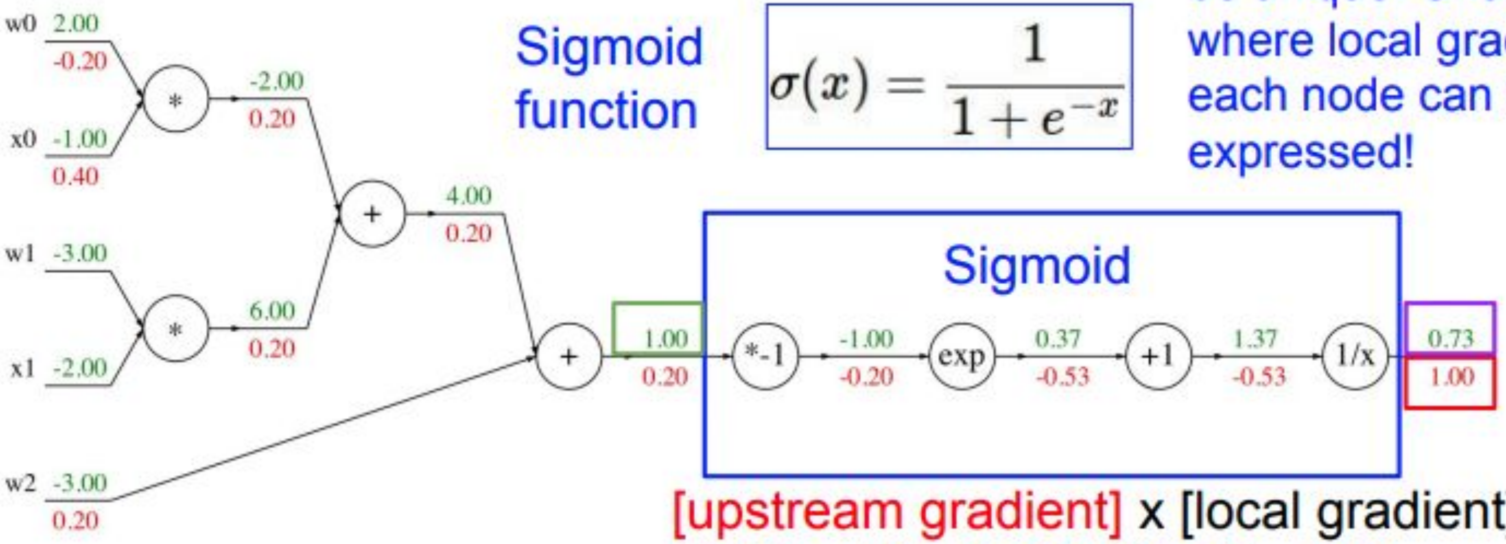
Figure taken from Ranjay Krishna’s course CSE493

One more example

Another example: $f(w,x) = \dfrac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

Sigmoid function $\boxed{\sigma(x) = \dfrac{1}{1+e^{-x}}}$



[upstream gradient] x [local gradient]

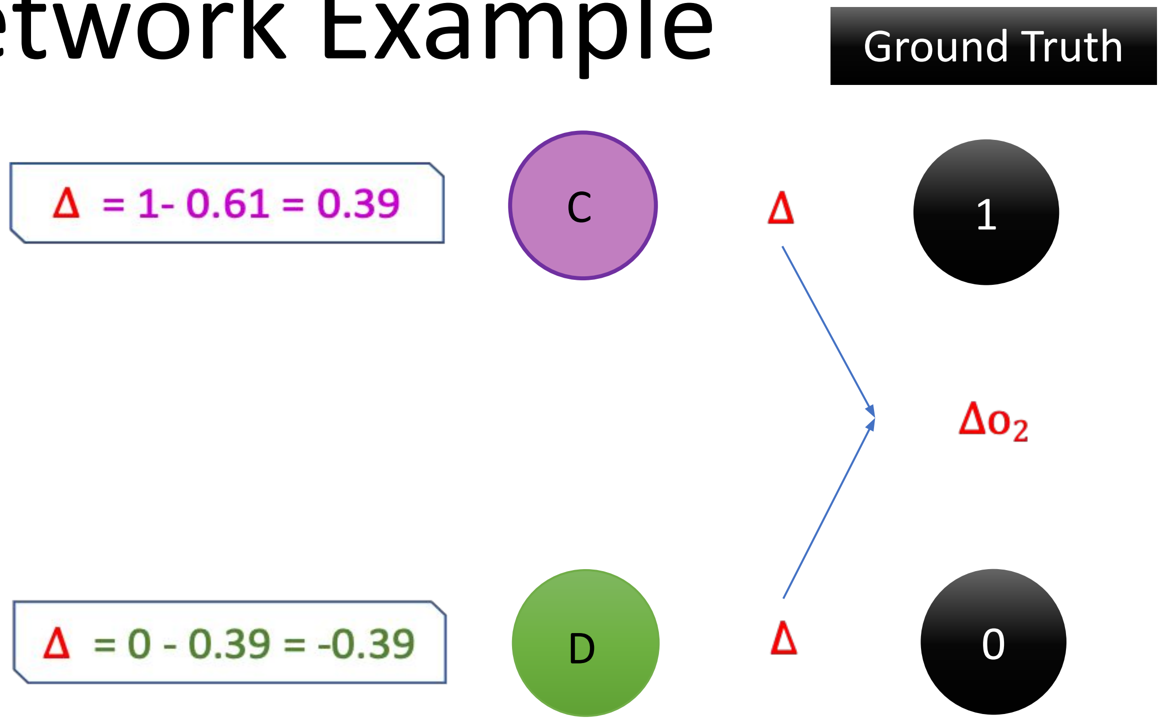
[1.00] x [(1 - 0.73) (0.73)] = 0.2

Sigmoid local gradient: $\dfrac{d\sigma(x)}{dx} = \dfrac{e^{-x}}{(1+e^{-x})^2} = \left(\dfrac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\dfrac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$

Figure taken from Ranjay Krishna’s course CSE493

So far: backprop with scalars, let’s go back to our initial example of vectors

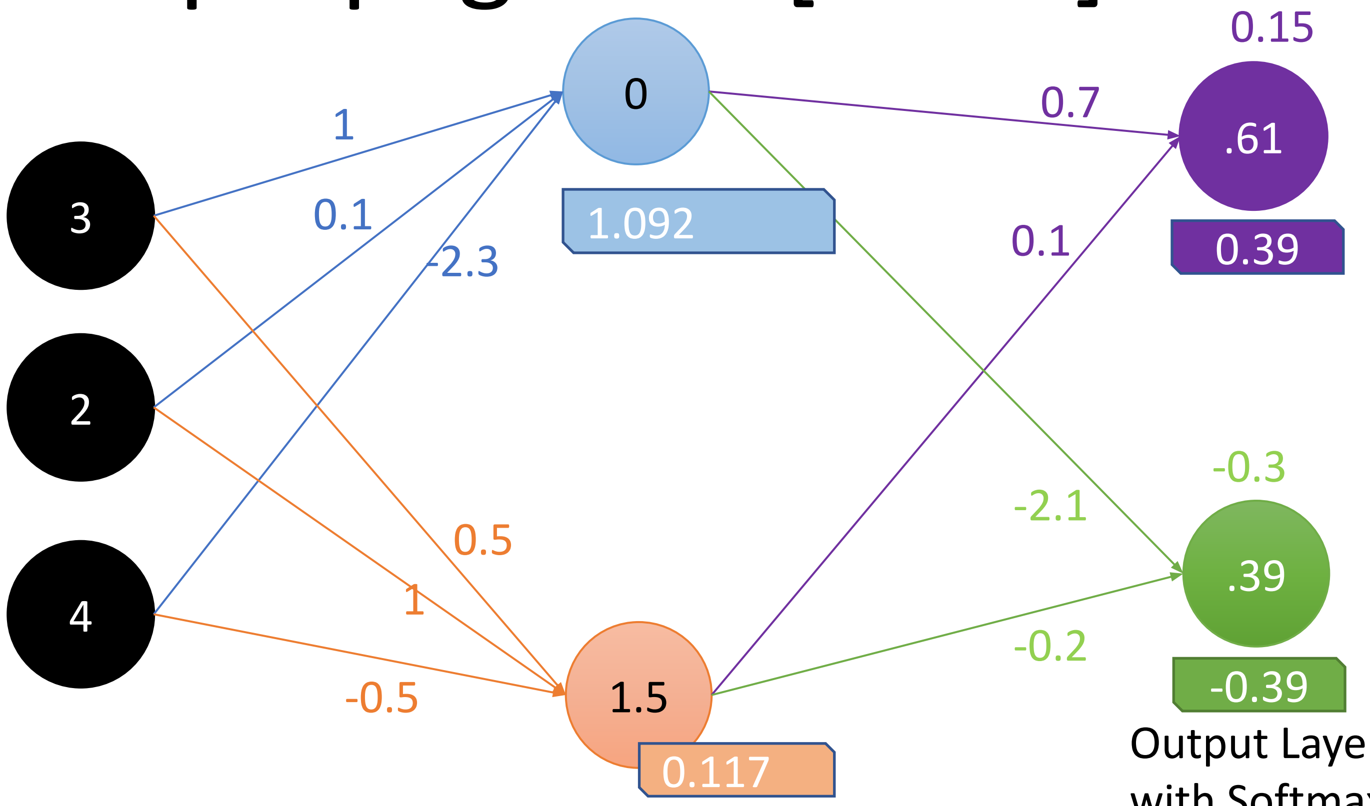
Back to our Neural Network Example



o_2

0.61	0.39
.	.
.	.

Backpropagation [Cont.]



Output Layer with Softmax

Input

$$X_{in} = \begin{array}{ccc} 3 & 2 & 4 \\ . & . & . \\ . & . & . \end{array}$$

1-st Layer (ReLU)

$$W_1 = \begin{array}{cc} 1 & 0.5 \\ 0.1 & 1 \\ -2.3 & -0.5 \end{array} \qquad o_1 = \begin{array}{cc} 0 & 1.5 \\ . & . \\ . & . \end{array}$$

$$W_2 = \begin{array}{cc} 0.7 & -2.1 \\ 0.1 & -0.2 \end{array} \qquad o_2 = \begin{array}{cc} 0.61 & 0.39 \\ . & . \\ . & . \end{array}$$

$g'(o_2) \circ \Delta o_2$

$$\begin{array}{cc} 0.39 & -0.39 \\ . & . \\ . & . \end{array}$$

$\Delta w_2 = o_1^T g'(o_2) \circ \Delta o_2$

$$\begin{array}{cc} 0 & 0 \\ 0.585 & -0.585 \end{array}$$

$\Delta o_1 = g'(o_2) \circ \Delta o_2 w_2^T$

$$\begin{array}{cc} 1.092 & 0.117 \\ . & . \\ . & . \end{array}$$

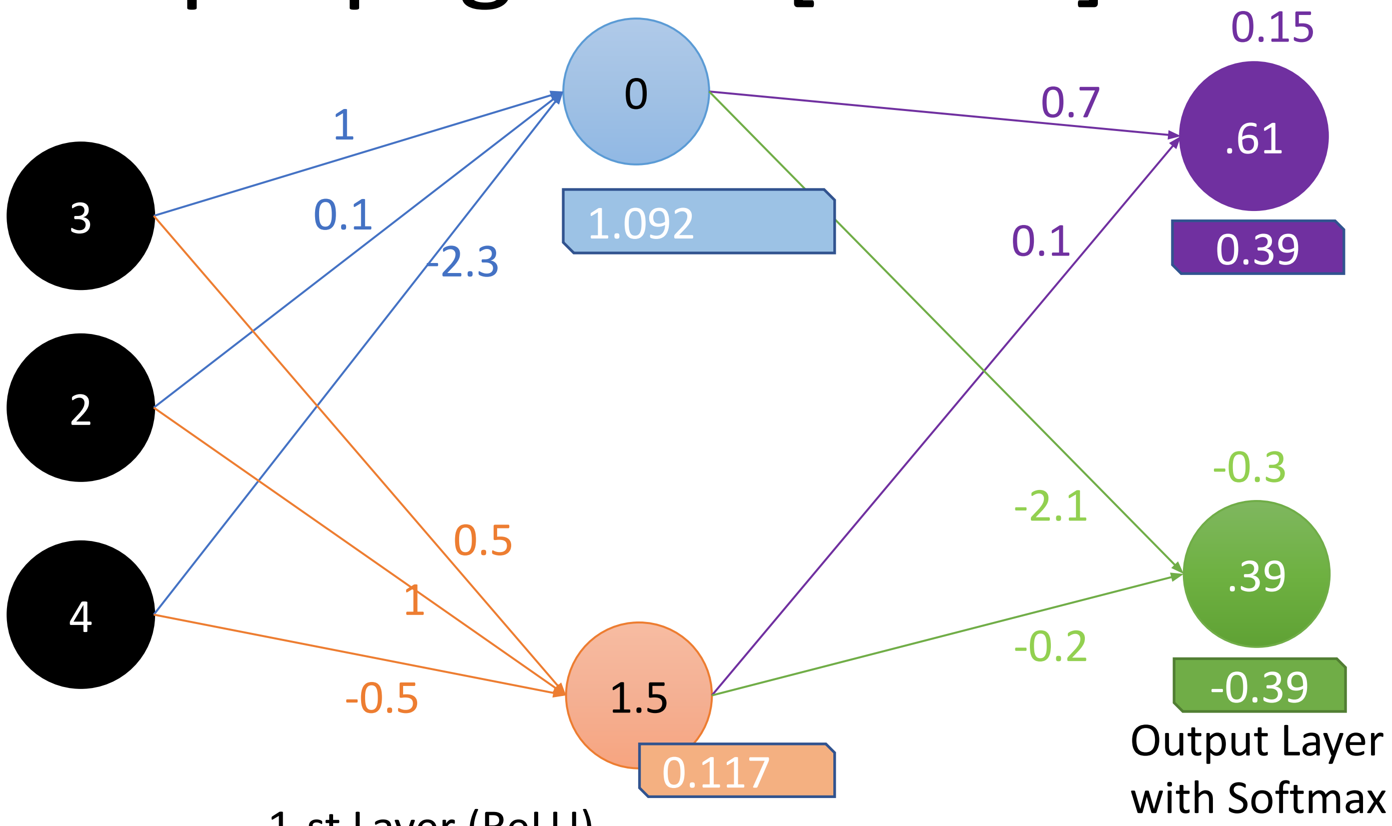
Backpropagation [Cont.]

$g'(o_1) \circ \Delta o_1$

0	0.117
.	.
.	.

$\Delta w_1 = o_0^T g'(o_1) \circ \Delta o_1$

0	0.351
0	0.234
0	0.468



Input

X_{in}

3	2	4
.	.	.
.	.	.

1-st Layer (ReLU)

w_1

1	0.5
0.1	1
-2.3	-0.5

o_1

0	1.5
.	.
.	.

w_2

0.7	-2.1
0.1	-0.2

o_2

0.61	0.39
.	.
.	.

$g'(o_2) \circ \Delta o_2$

0.39	-0.39
.	.
.	.

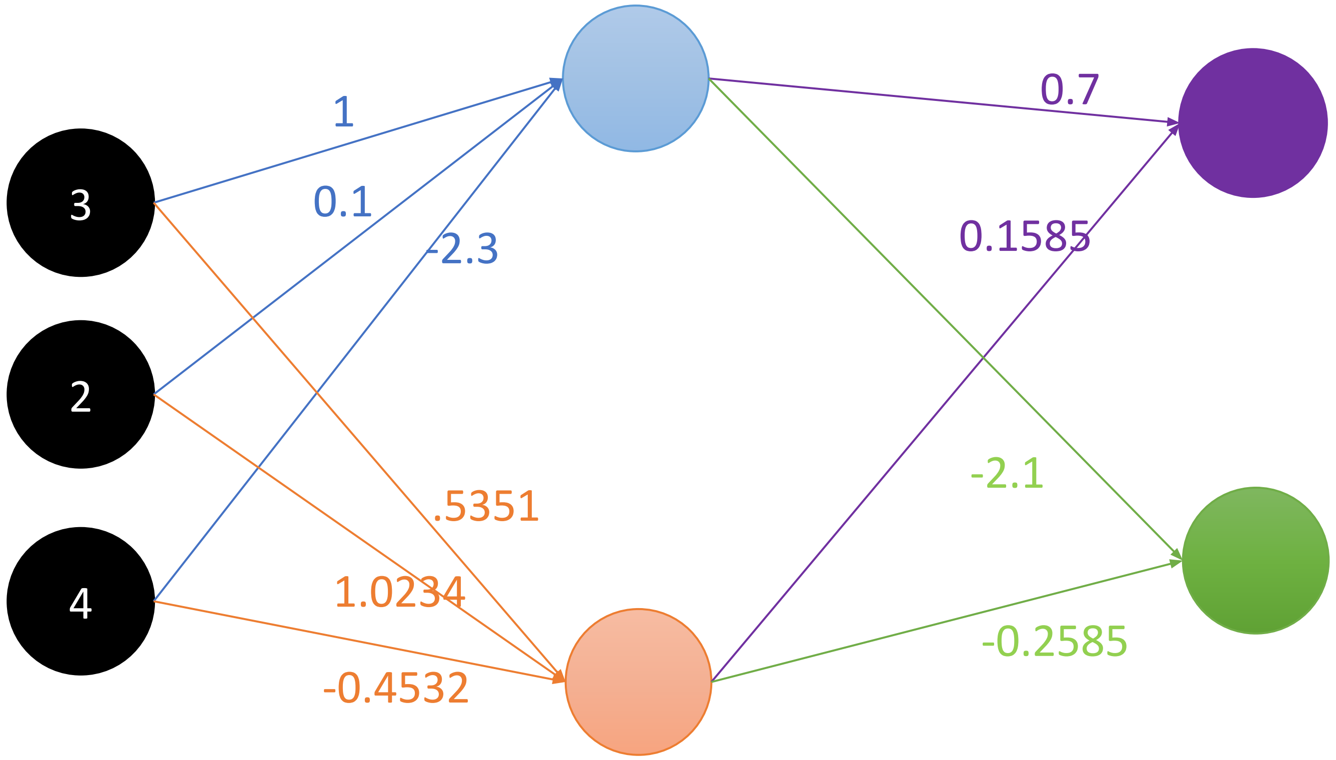
$\Delta w_2 = o_1^T g'(o_2) \circ \Delta o_2$

0	0
0.585	-0.585

$\Delta o_1 = g'(o_2) \circ \Delta o_2 w_2^T$

1.092	0.117
.	.
.	.

Now we backpropagate with a learning rate of 0.1



$$w_1 = w_1 + \alpha \Delta w_1$$

$$w_2 = w_2 + \alpha \Delta w_2$$

Input Layer

$$X_{in} \quad \begin{matrix} 3 & 2 & 4 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{matrix}$$

1-st Layer (ReLU)

w_1

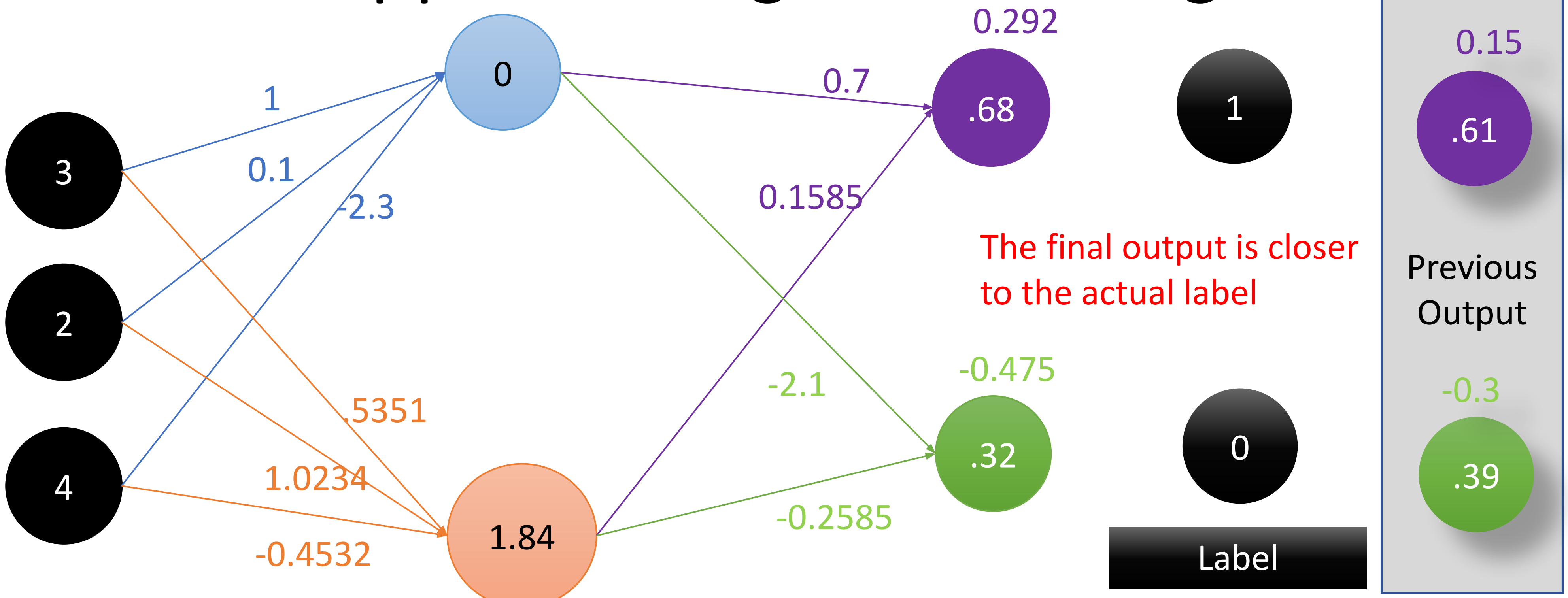
1	0.5351
0.1	1.0234
-2.3	-0.453 2

Output Layer with Softmax

w_2

0.7	-2.1
0.1585	-0.2585

Think: What will happen if we go forward again?



Input Layer

	3	2	4
X_{in}	.	.	.
	.	.	.

1-st Layer (ReLU)

w_1

1	0.5351
0.1	1.0234
-2.3	-0.453 2

Output Layer with Softmax

w_2

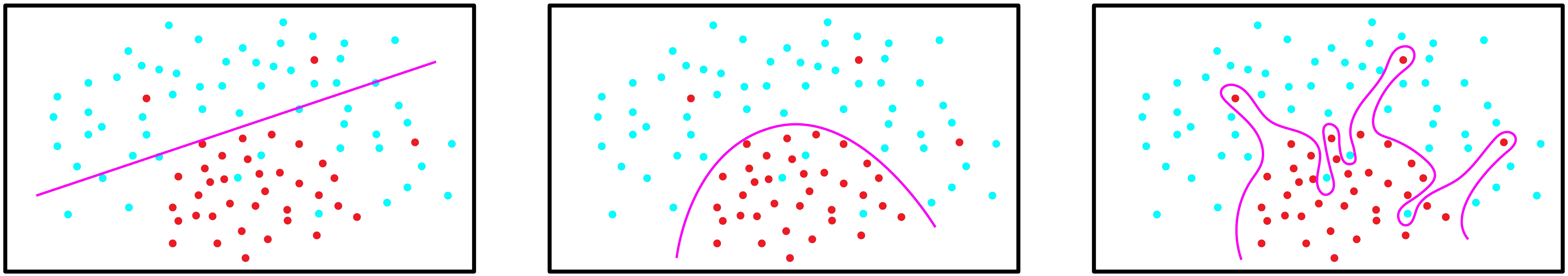
0.7	-2.1
0.1585	-0.2585

Tricks for Training Neural Networks

Problem: Under and Overfitting

Underfitting: model not powerful enough, too much bias

Overfitting: model too powerful, fits to noise, doesn't generalize well



Weight decay: neural network regularization

We use Δw_t to represent the weight gradient for timepoint **t** (the current step).

We want the weights to be close to 0.

Let L be the "loss" function; (e.g. $L = |y - g(in)|$, $L = (y - g(in))^2$, etc.)

λ is a regularization parameter (for decay)

Higher: more penalty for large weights, less powerful model

Lower: less penalty, more overfitting

Subtract a little bit of weight every iteration

Before:

$$\Delta w_t = -\partial/\partial w_t\, L(w_t)$$

$$w_{t+1} = w_t + \alpha\, \Delta w_t$$

Now:

$$w_{t+1} = w_t - \alpha[\partial/\partial w_t\, L(w_t) + \lambda w_t] = w_t - \alpha[- \Delta w_t + \lambda w_t]$$

$$= w_t - \alpha\, \partial/\partial w_t\, L(w_t) - \alpha\, \lambda w_t = w_t + \alpha\, \Delta w_t - \alpha\, \lambda w_t$$

Momentum: speeding up SGD

If we keep moving in same direction we should move further every round

Before:

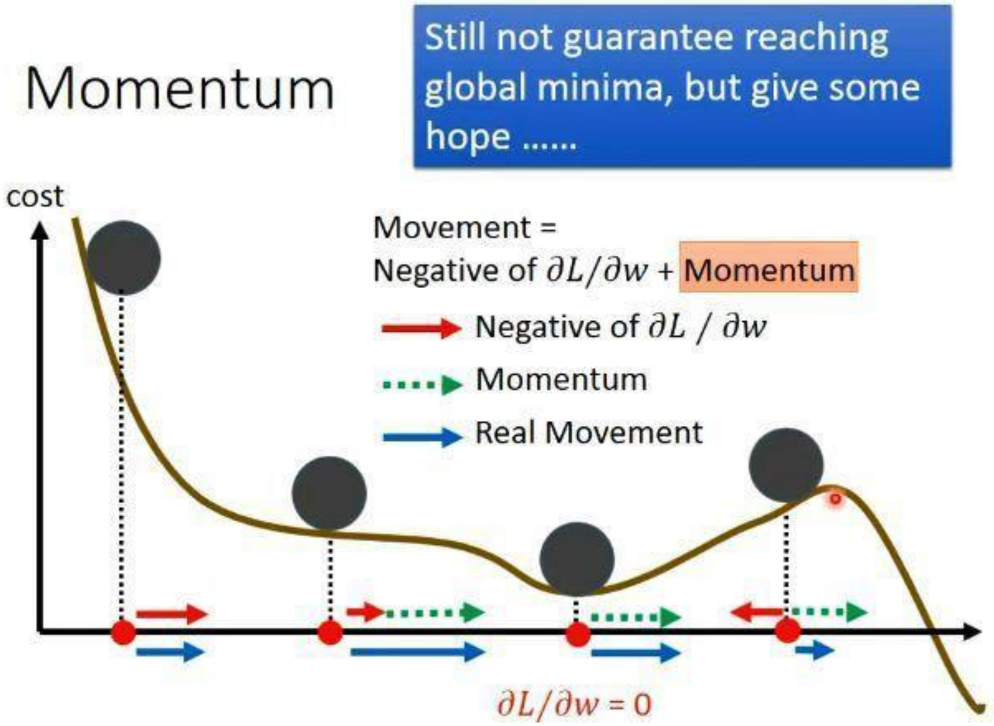
$$\Delta w_t = -\partial/\partial w_t \, L(w_t)$$

Now:

$$\Delta w_t = -\partial/\partial w_t \, L(w_t) + m\Delta w_{t-1}$$

$$w_{t+1} = w_t + \alpha \, \Delta w_t$$

Side effect: **smooths** out updates if gradient is in different directions



Δw_{t-1} represent the gradient calculated in the previous step.

NN updates with weight decay and momentum

$$\Delta w'_t = -\partial/\partial w_t\, L(w_t) - \lambda w_t + m \Delta w'_{t-1}$$

Gradient of loss

Weight decay

Momentum

$$w_{t+1} = w_t + \alpha \Delta w'_t$$

Learning rate

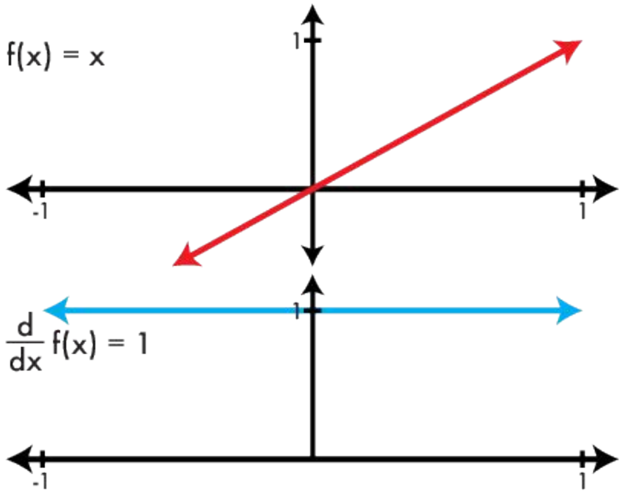
Dropout

Randomly eliminate some nodes during training

Activations

Linear Activation

- $$g(x) = x$$
  $$g'(x) = 1$$



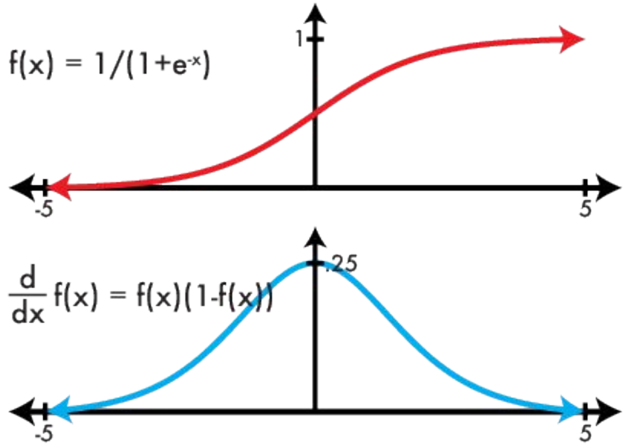
- Only offers linear effects.
- For a 2-layer NN with linear activations for both layers.

$$f(X) = g(g(Xw_1)w_2) = Xw_1w_2 = Xw$$

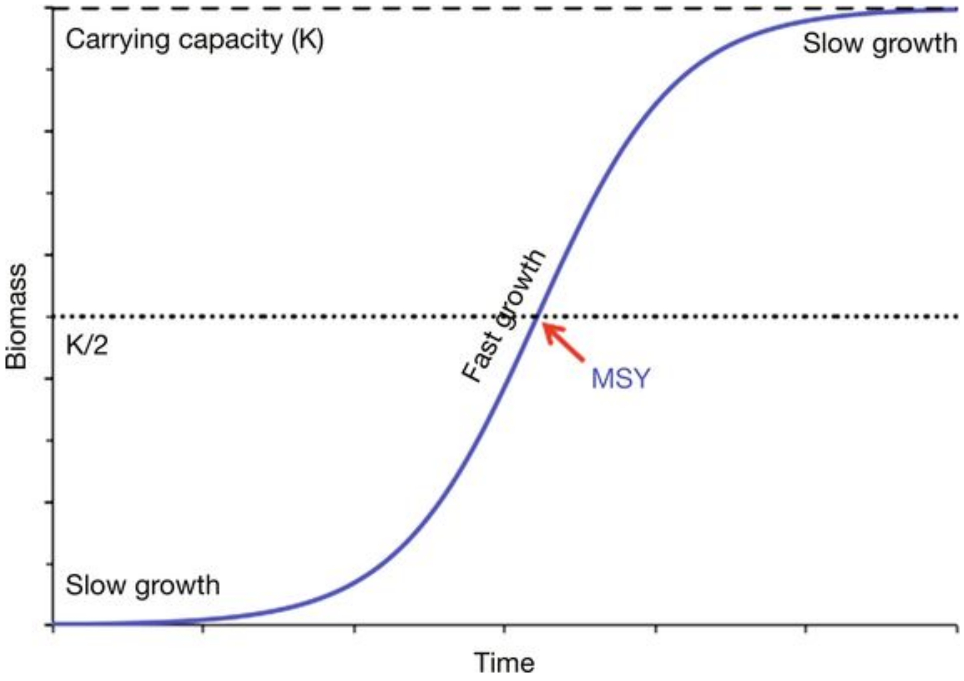
- Not so great, need Non-Linear activations to learn more complex data distribution.

Logistic Activation

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



- Aka Sigmoid function (S-shape)
- Used in Logistic regression.
- The result is in range (0, 1),
- It can represent probability.
- A special case of logistic growth (population model).



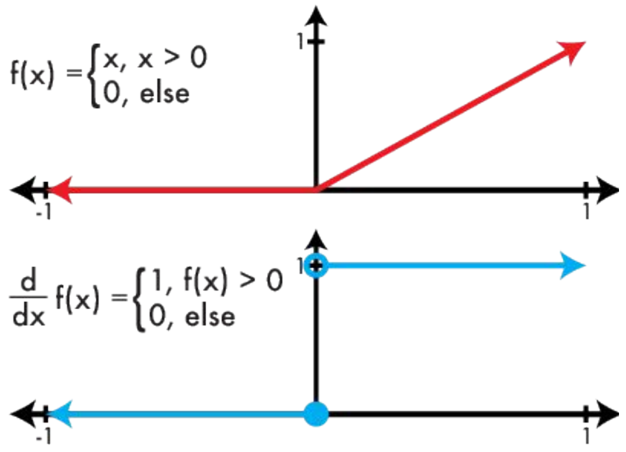
Do you see the problem with Sigmoidal activations?

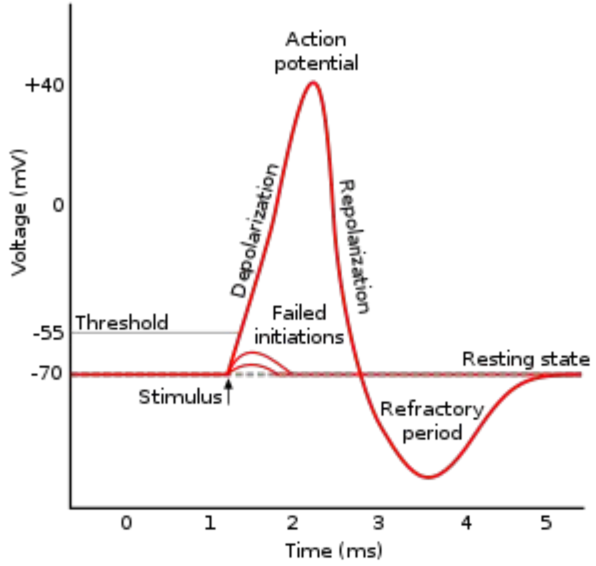
Do you see the problem with Sigmoidal activations?

Vanishing gradients!

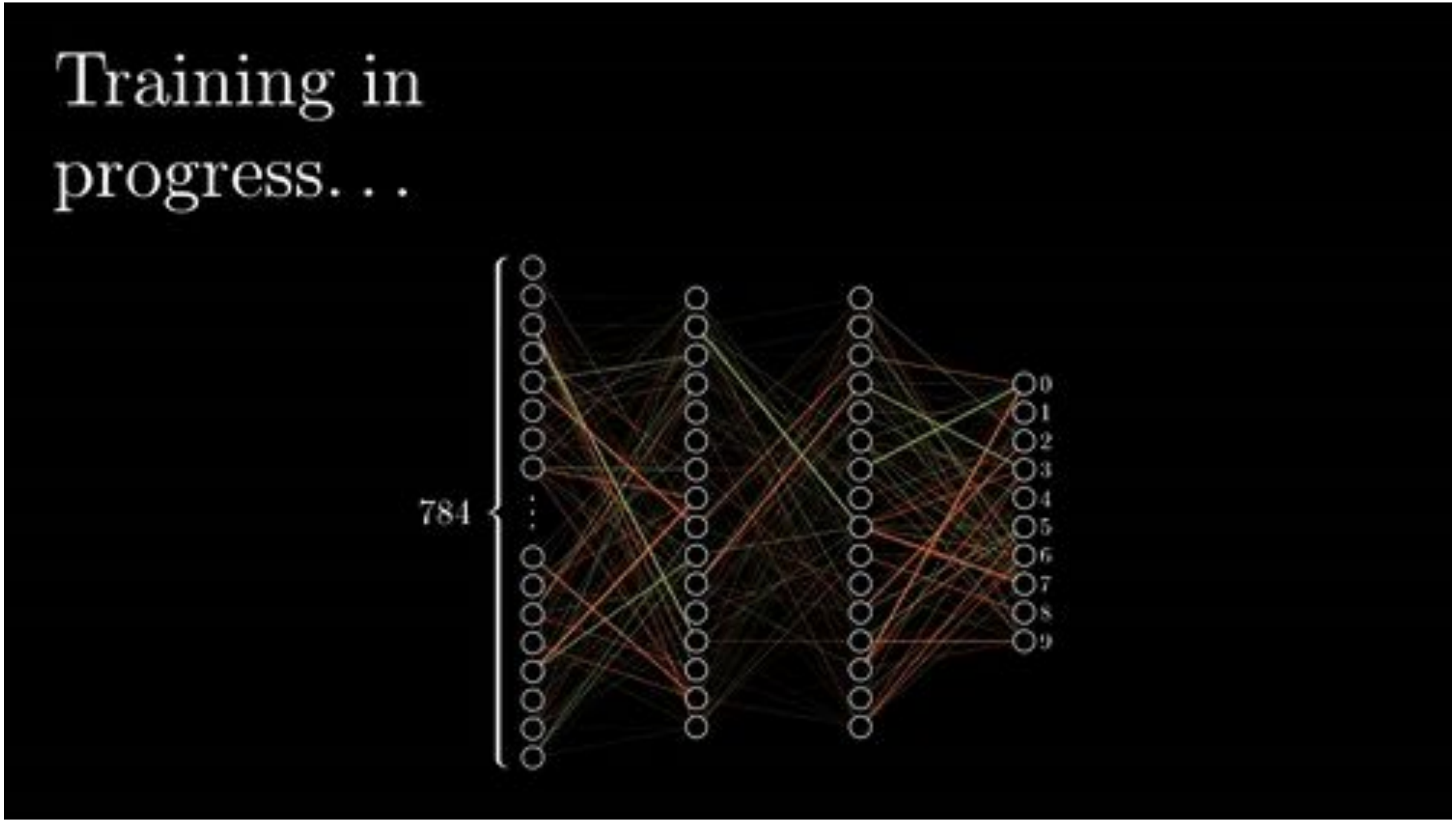
ReLU Activation

- $$g(x) = \max(0, x)$$
$$g'(x) = \mathbf{1}_{g(x)>0}$$
- Rectified linear unit
- ***Fast!*** In backpropagation, 1 when positive, 0 otherwise.
- Optimizes important (positive) values and ignore the others.
- Analog to neurons
- Information loss is small (other neurons will carry information)



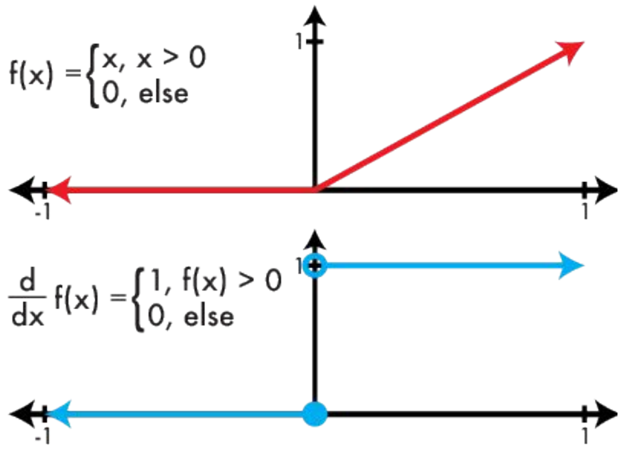


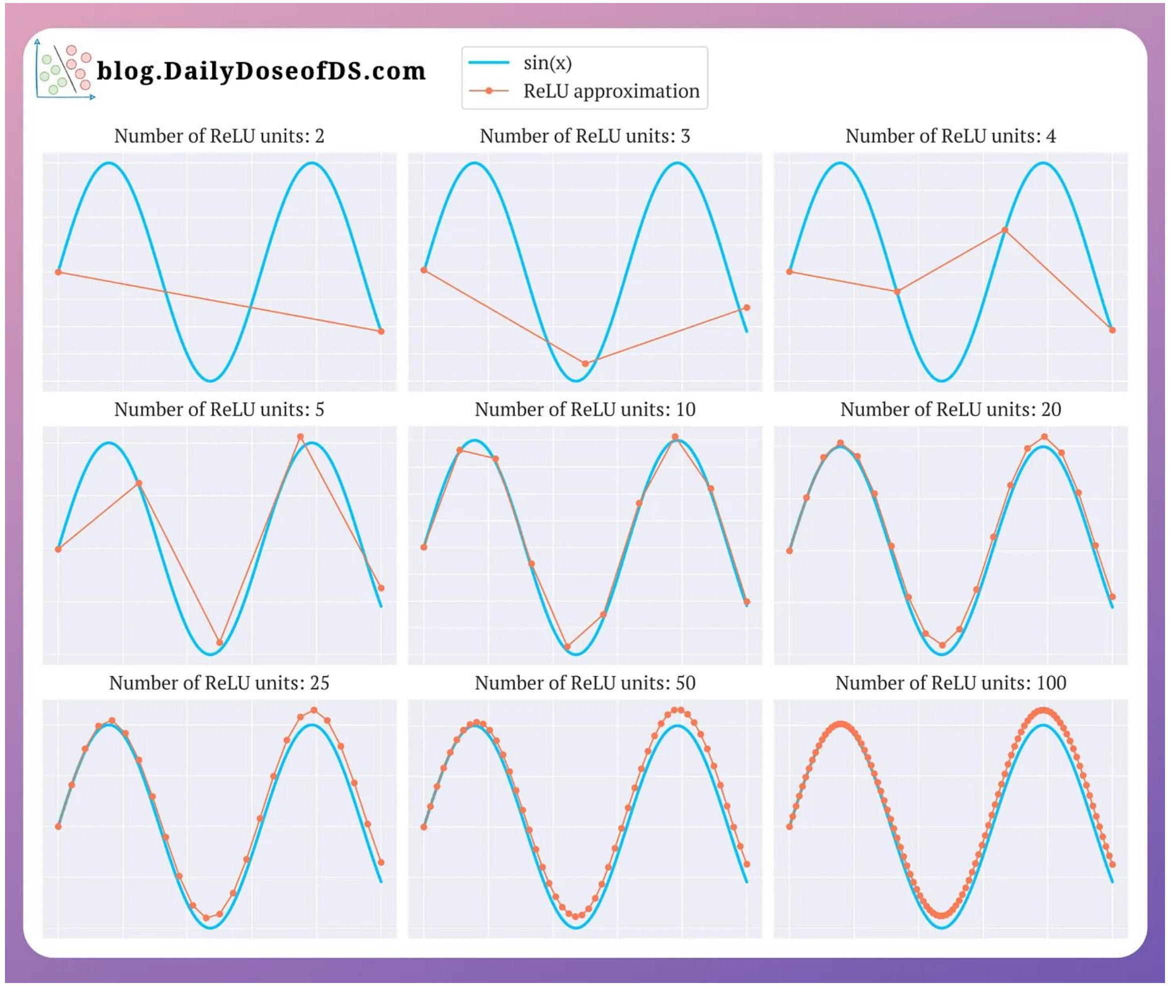
Visualization with ReLU



https://www.youtube.com/channel/UCYO_jab_esuFRV4b17AJtAw

Why ReLU provides non-linearity?

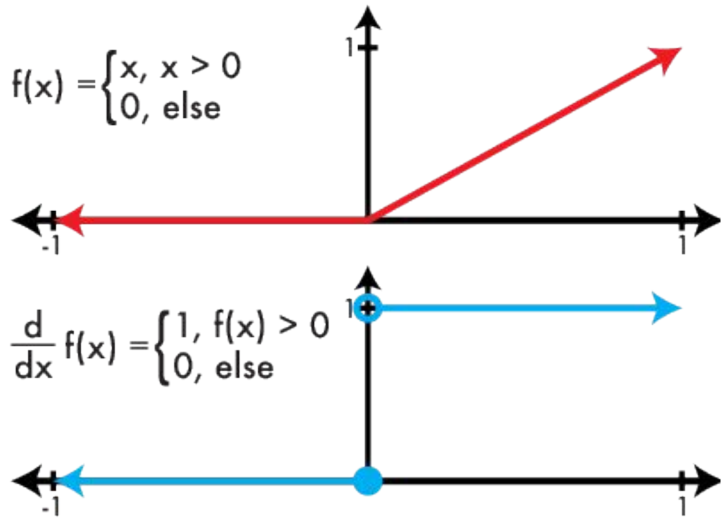




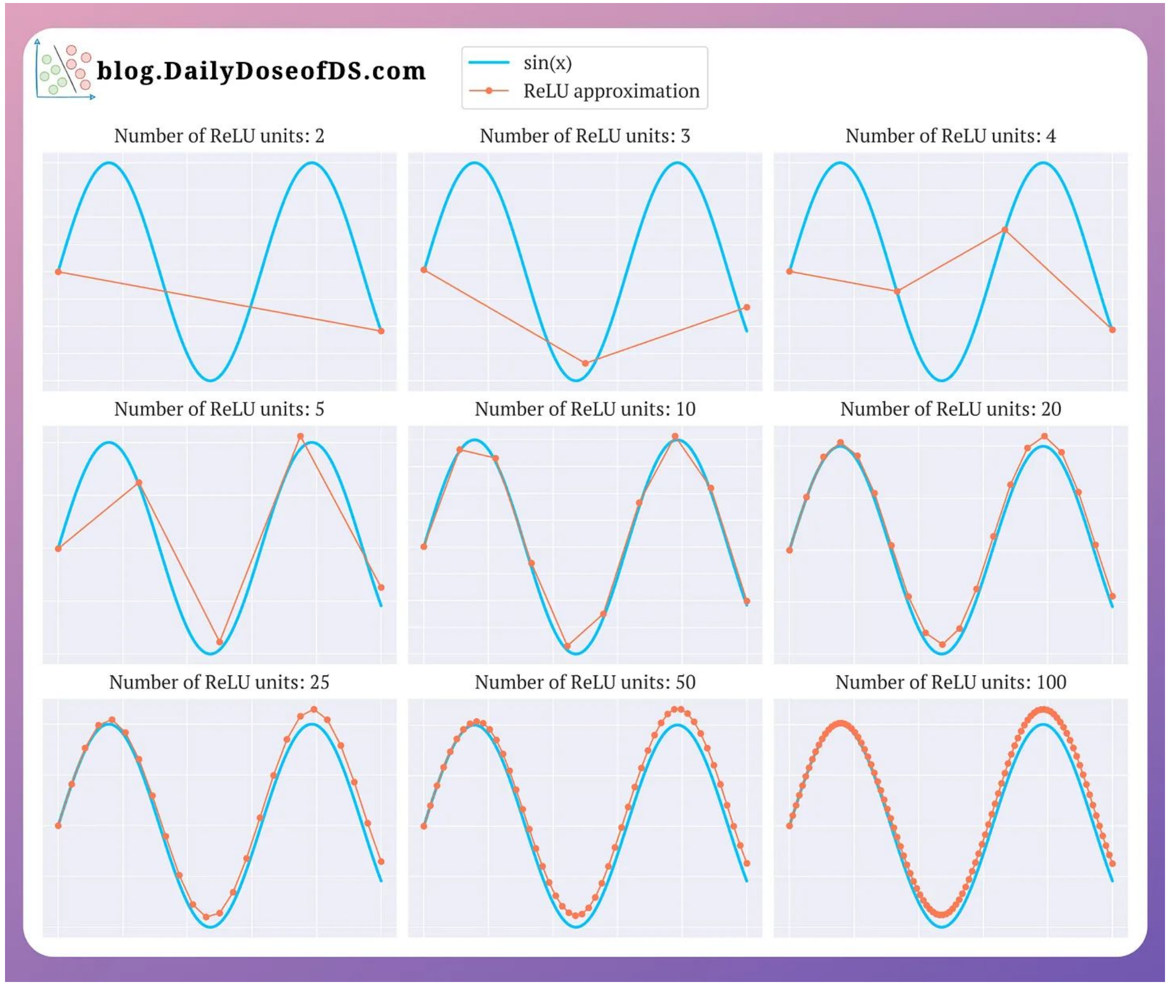
Why ReLU provides non-linearity?

The function is not linear if it does not satisfy the superposition principles: 1) additivity, 2) homogeneity

1) $F(x1 + x2) = F(x1) + F(x2)$
2) $F(ax) = aF(x)$

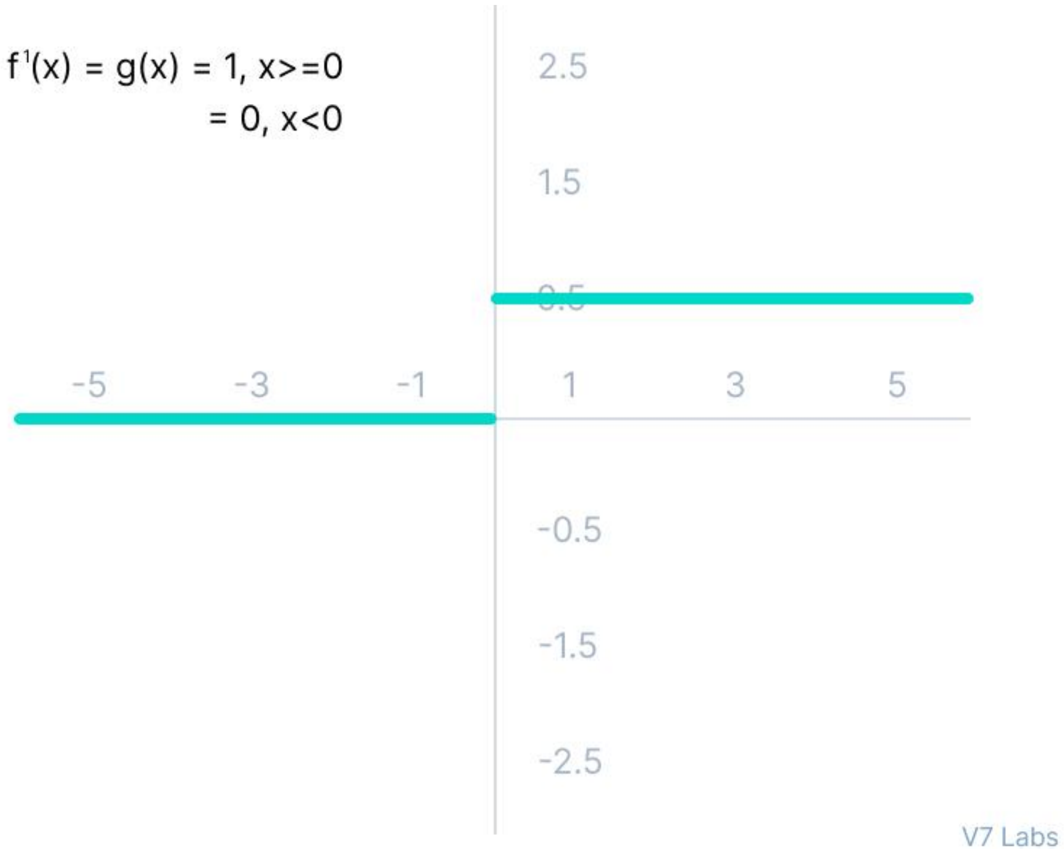


Why ReLU provides non-linearity?

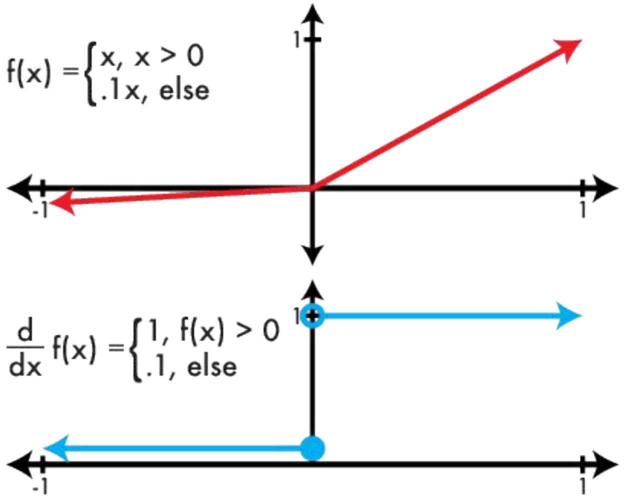


Taken from https://www.blog.dailydoseofds.com/p/a-visual-and-intuitive-guide-to-what#:~:text=The%20core%20point%20to%20understand,of%20a%20non%2Dlinear%20curve.

The Dying ReLU Problem

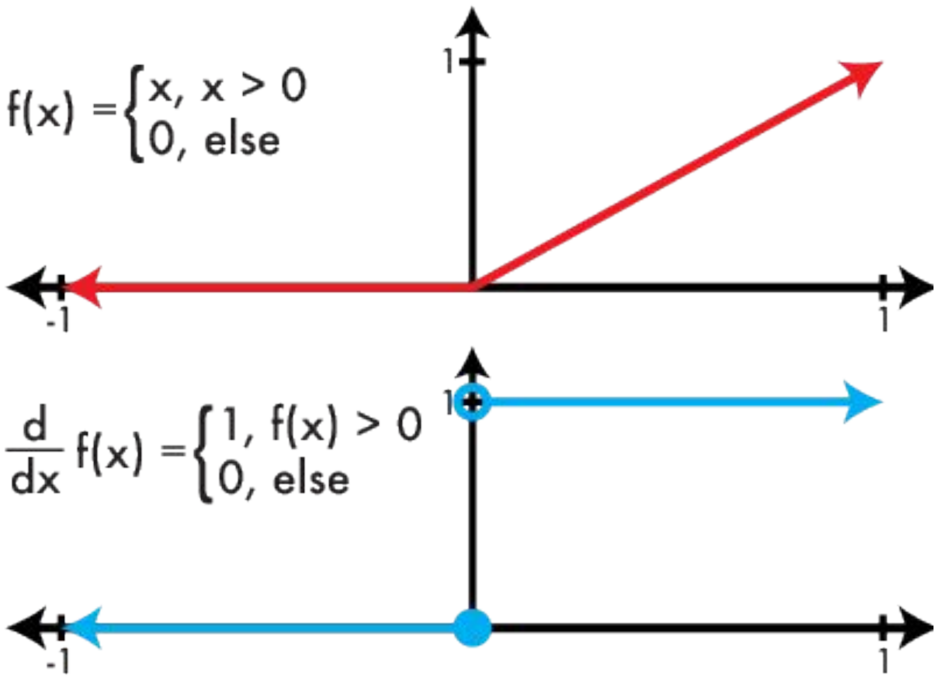


LeakyReLU Activation



- No information loss (compared to ReLU)
- Solves "dying ReLU" problem (i.e. all neurons output 0)
- Similar to ReLU, pays less attention to less important neurons
- Not always better than ReLU

Do you see any other issues with ReLU?



Do you see any other issues with ReLU?

- Exploding gradients: Exact opposite phenomena we had with Sigmoid (vanishing gradients)

Do you see any other issues with ReLU?

- Exploding gradients: Exact opposite phenomena we had with Sigmoid (vanishing gradients)
  - Do gradient clipping, or layer-wise normalization

Homework 4

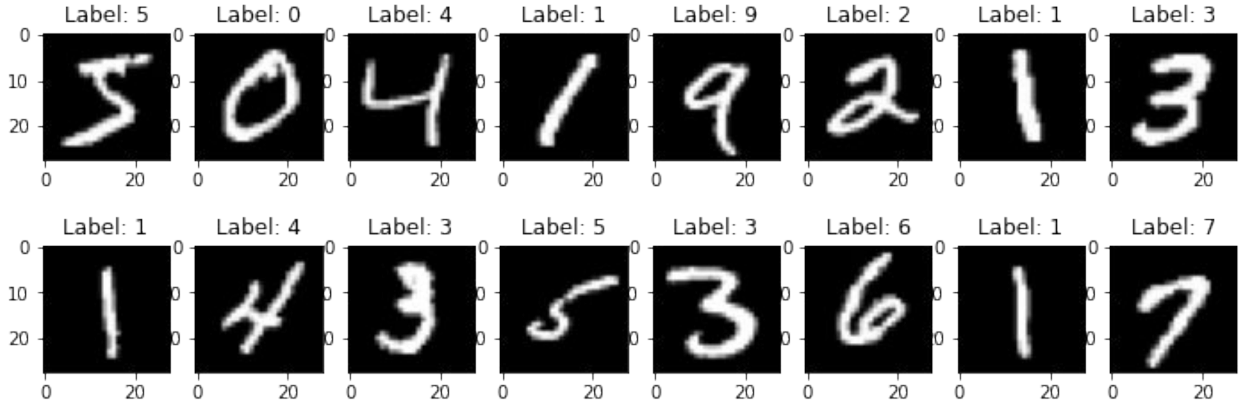
Neural Network

MNIST: Handwriting recognition

50,000 images of handwriting

Input: 28 x 28 x 1 (grayscale) 784 pixel values

Numbers 0-9 10 classes



Train a linear softmax model

> 95% accuracy

Functions You need to Code

Functions You need to Code (**classifier.c**)

```
void activate_matrix(matrix m, ACTIVATION a)
void gradient_matrix(matrix m, ACTIVATION a, matrix d)
matrix forward_layer(layer *l, matrix in)
matrix backward_layer(layer *l, matrix delta)
void update_layer(layer *l, double rate, double momentum, double decay)
```

Run Experiments and Write a Report (**hw4.pdf**)

Play around with tryhw4.py file, and answer the questions.

Save your question to a PDF file and submit to Canvas for grading.

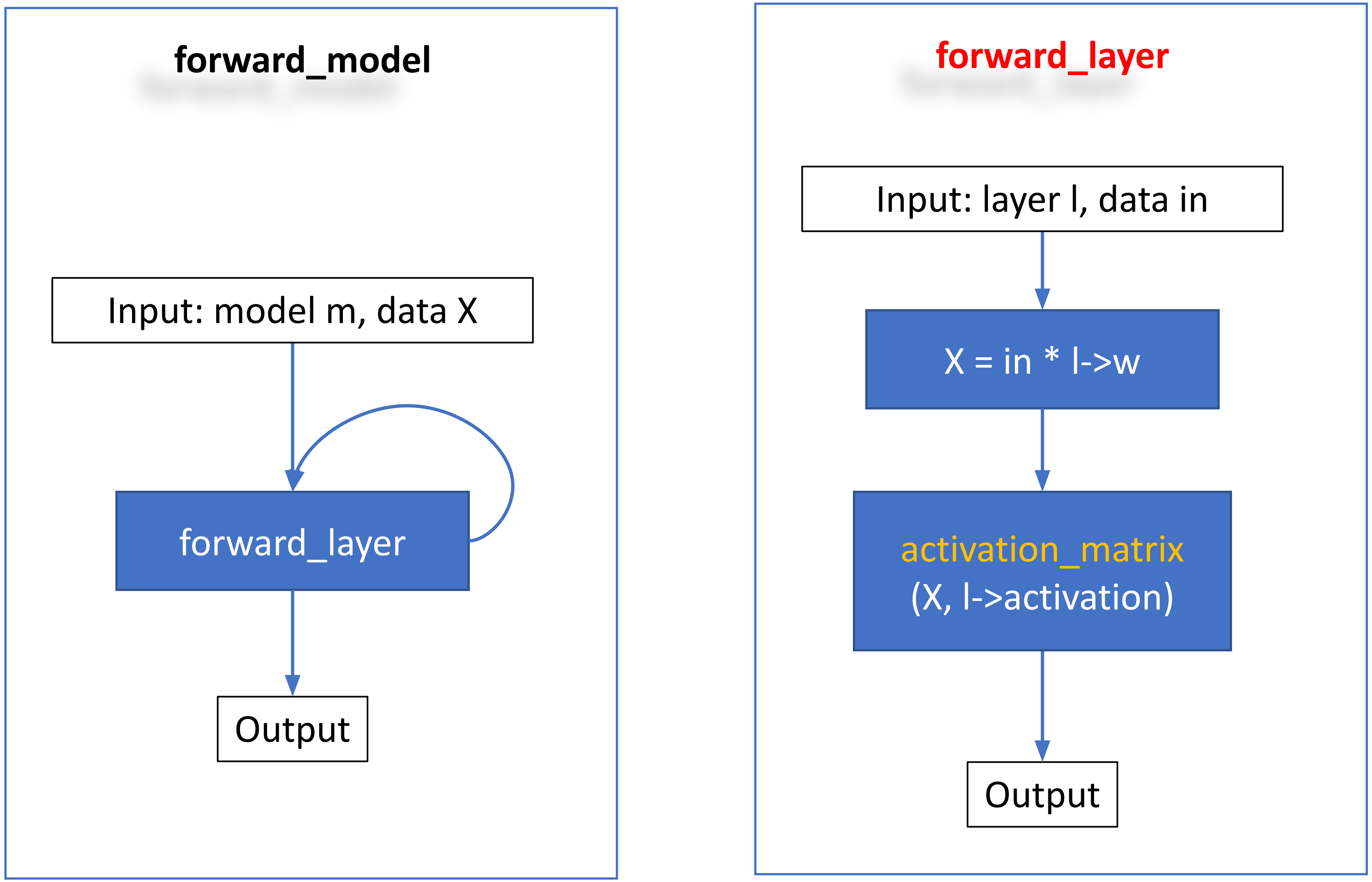
Important Data Structure (image.h)

```
typedef enum{LINEAR, LOGISTIC, RELU, LRELU, SOFTMAX} ACTIVATION;

typedef struct {
    matrix in;              // Saved input to a layer
    matrix w;               // Current weights for a layer
    matrix dw;              // Current weight updates
    matrix v;               // Past weight updates (for use with momentum)
    matrix out;             // Saved output from the layer
    ACTIVATION activation;  // Activation the layer uses
} layer;

typedef struct {
    layer *layers;
    int n;
} model;
```

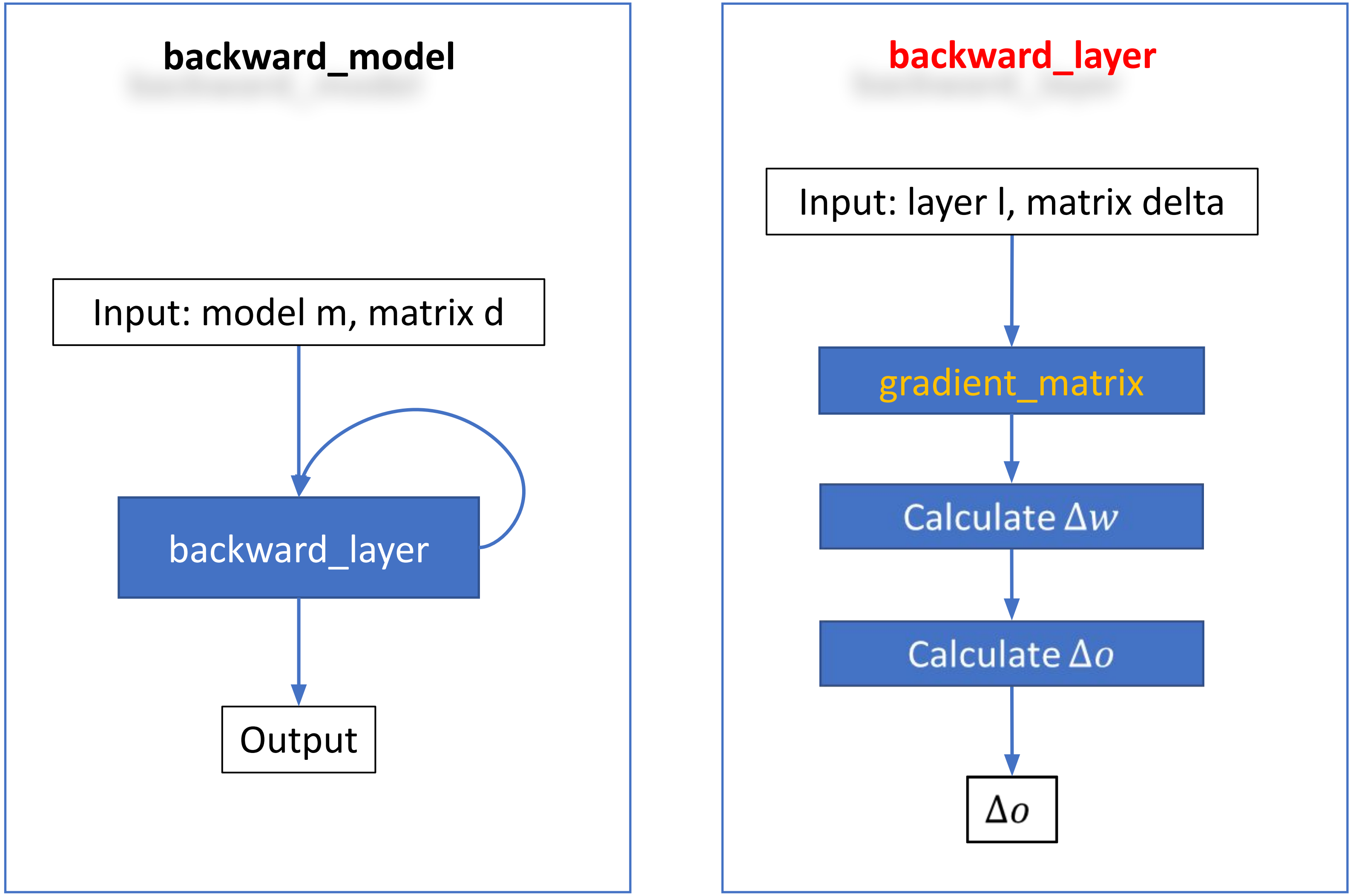
Useful Matrix manipulation functions (matrix.c)

```
matrix matrix_mult_matrix(matrix a, matrix b);
matrix transpose_matrix(matrix m);
matrix axpy_matrix(double a, matrix x, matrix y); // a * x + y
```

Forward Pass in Homework

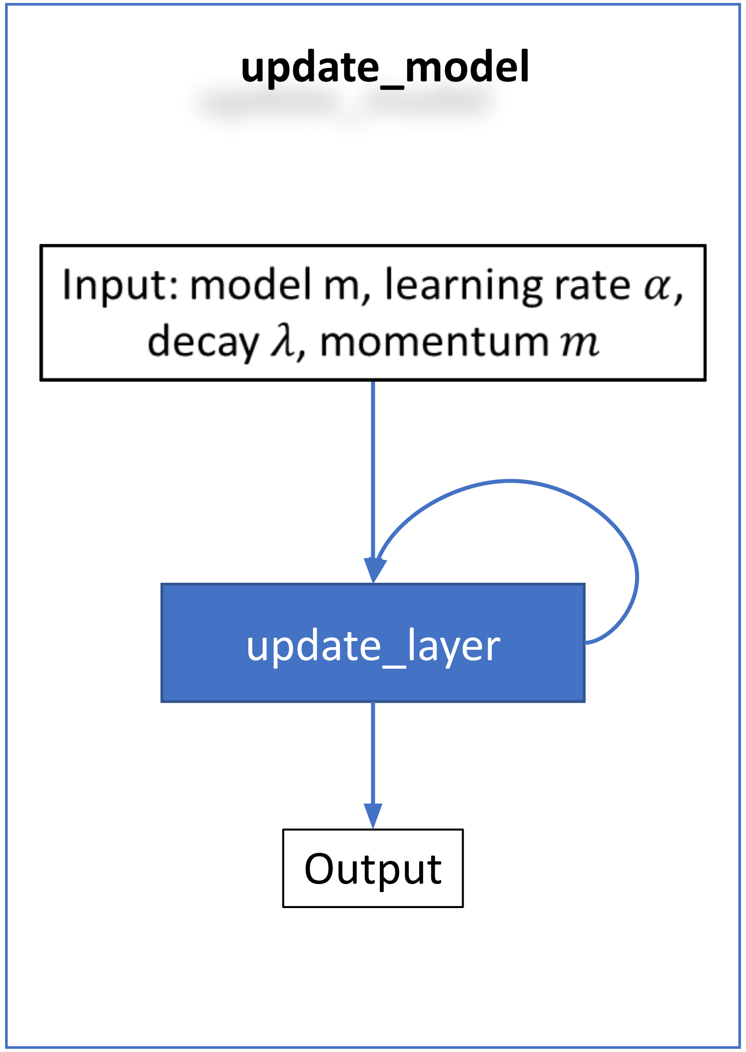


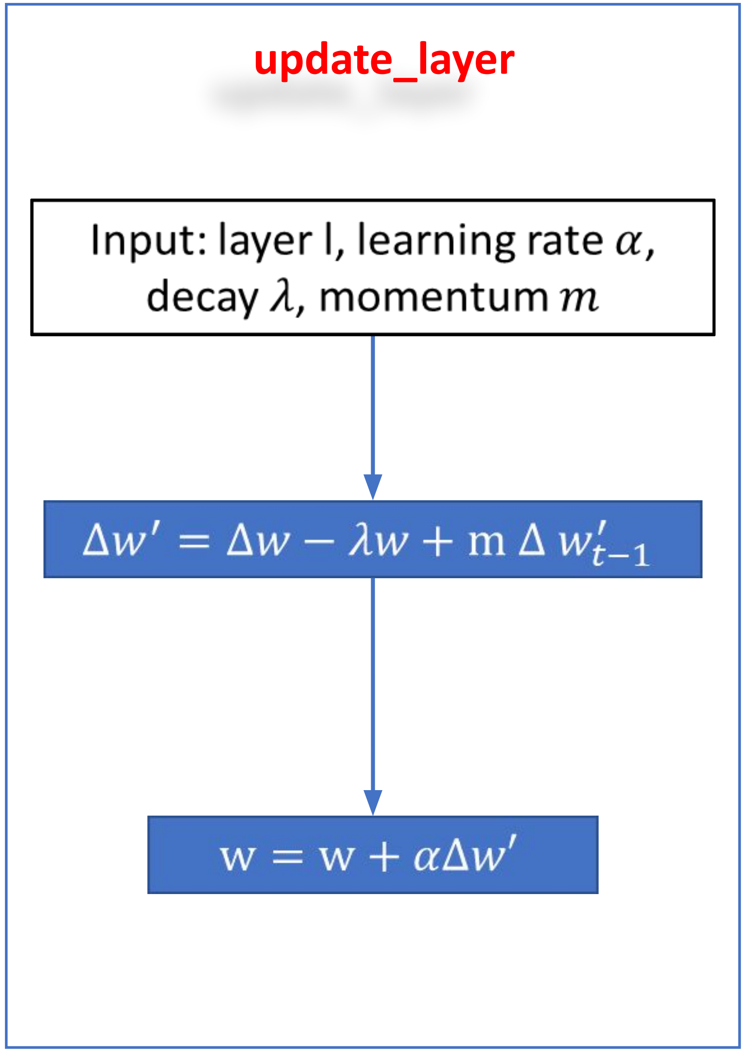
Backward Pass in Homework



Weight Update in Homework

$\Delta w'_{t-1}$ represent the regularized gradient from the previous step. In the code, we use “l->v” to store this value.





TODO void activate_matrix(matrix m, ACTIVATION a)

Apply activation "a" to the matrix "m"

```
for(i = 0; i < m.rows; ++i){
    double sum = 0;
    for(j = 0; j < m.cols; ++j){
        double x = m.data[i][j];
        if(a == LOGISTIC){
            // TODO  m.data[i][j] should equals 1 / (1 + exp(-x));
        } else if (a == RELU){
            // TODO  m.data[i][j] should equals x if x > 0; otherwise, it should equal 0
        } else if (a == LRELU){
            // TODO  m.data[i][j] should equals x if x > 0; otherwise, it should equal 0.1 * x.
        } else if (a == SOFTMAX){
            // TODO  m.data[i][j] should equals exp(x) here, and we will normalize it later.
        }
        sum += m.data[i][j];
    }
    if (a == SOFTMAX) {
        // TODO: have to normalize by sum if we are using SOFTMAX
         // for all the possible j, we should normalize it as m.data[i][j] /= sum;
    }
}
```

TODO void gradient_matrix(matrix m, ACTIVATION a, matrix d)

Calculate g'(m) * d, and store in-place to matrix d.
The matrix "m" is the output of a layer, and matrix "d" is the Δ of output.

```
int i, j;
for(i = 0; i < m.rows; ++i){
    for(j = 0; j < m.cols; ++j){
        double x = m.data[i][j];
        // TODO: multiply the correct element of d by the gradient
        // if a is SOFTMAX or a is LINEAR, we should do nothing (multiply by 1)
        // if a is LOGISTIC,  d.data[i][j] should times x * (1.0 - x);
        // if a is RELU and x <= 0, d.data[i][j] should be zero
        // if a is LRELU and x <= 0, d.data[i][j] should multiple 0.1
    }
}
```

TODO `matrix forward_layer(layer *l, matrix in)`

Given the input data “in” and layer “l”, calculate the output data.

```
l->in = in;   // Save the input for backpropagation

// TODO: multiply input by weights and apply activation function.
// Calculate out = in * l->w (note: matrix multiplication here)
// Then, apply activate_matrix function to out with l->activation

free_matrix(l->out);// free the old output
l->out = out;        // Save the current output for gradient calculation
return out;
```

```
TODO matrix backward_layer(layer *l, matrix delta)
```

Given the layer “l” and delta, perform backward step:

- 1.4.1: Calculate the delta after considering the activation
- 1.4.2: Calculate Δw
- 1.4.3: Calculate and Return Δo (aka “dx”).

```
// delta is Δout
// TODO: modify it in place to be “g'(out) * delta” out with // gradient_matrix function.
// You can use gradient_matrix function with “l->out” and “l->activation” to “delta”

// TODO: then calculate dL/dw and save it in l->dw
free_matrix(l->dw);
// Calculate xt as the transpose matrix of “l->in”
// Calculate dw as xt times delta (matrix multiplication)
// free matrix xt to avoid memory leak
l->dw = dw;

// TODO: finally, calculate dL/dx and return it. (Similar to 1.4.2. Care memory leak)
// Calculate dx = delta * (l->w)^T,   where * is matrix multiplication and ^T is matrix transpose
return dx;
```

TODO `void update_layer(layer *l, double rate, double momentum, double decay)`

Given a layer "l", learning rate, momentum, and decay rate, Update the weight (i.e. l->w)

```
// Calculate Δw_t = dL/dw_t - λw_t + mΔw_{t-1}
// save it to l->v
// Note that You can use axpy_matrix to perform the matrix summation/subtraction


// Update l->w
// l->w = rate * l->v + l->w
```

Note the multiplication and summation in this slides all mean matrix multiplication or matrix summation.

Functions You Need to Know before Experiments

For simplicity, we already filled the following functions for you. You should read and understand these functions (classifier.c) before running experiments.

```
layer make_layer(int input, int output, ACTIVATION activation)
matrix forward_model(model m, matrix X)
void backward_model(model m, matrix dL)
void update_model(model m, double rate, double momentum, double decay)
double accuracy_model(model m, data d)
double cross_entropy_loss(matrix y, matrix p)
void train_model(model m, data d, int batch, int iters, double rate, double momentum, double decay)
```

Get the Data

1. Download, Unzip, and Prepare the MNIST Dataset

```
wget https://pjreddie.com/media/files/mnist_train.tar.gz
wget https://pjreddie.com/media/files/mnist_test.tar.gz
tar xzf mnist_train.tar.gz
tar xzf mnist_test.tar.gz
find train -name \*.png > mnist.train
find test -name \*.png > mnist.test
```

2. Download, Unzip, and Prepare the CIFAR-10 Dataset

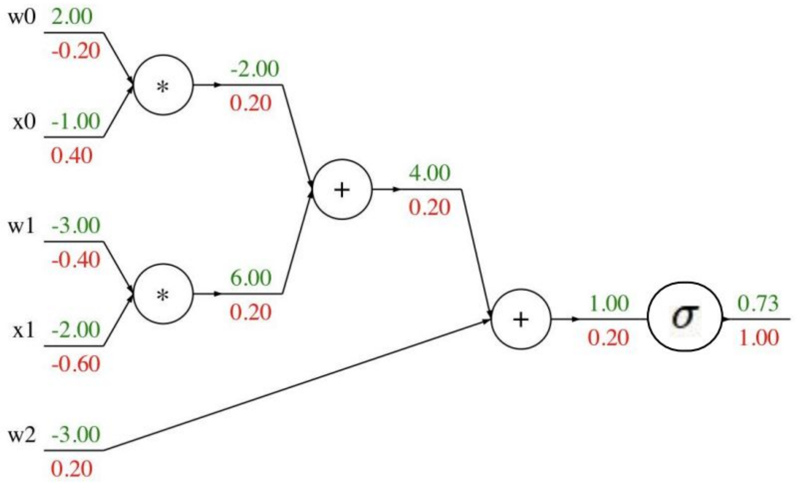
```
wget http://pjreddie.com/media/files/cifar.tgz
tar xzf cifar.tgz
find cifar/train -name \*.png > cifar.train
find cifar/test -name \*.png > cifar.test
```

Experiments (Write Your Answers to **hw4.pdf**)

1. Coding and Data prepare
2. MNIST Experiments
   1. Linear Softmax Model (1-layer)
      1. Run the basic model
      2. Tune the learning rate
      3. Tune the decay
   2. Neural Network (2-layer NNs and 3-layer NNs)
      1. Find the best activation
      2. Tune the learning rate
      3. Tune the decay
      4. Tune the decay for 3-layer Neural Network
3. Experiments for CIFAR-10
   1. Neural Network (3-layer NNs)
      1. Tune the learning rate and decay

How to write this in code

Backprop Implementation:
“Flat” code



Forward pass:
Compute output

```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)
```

Backward pass:
Compute grads

```
    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```

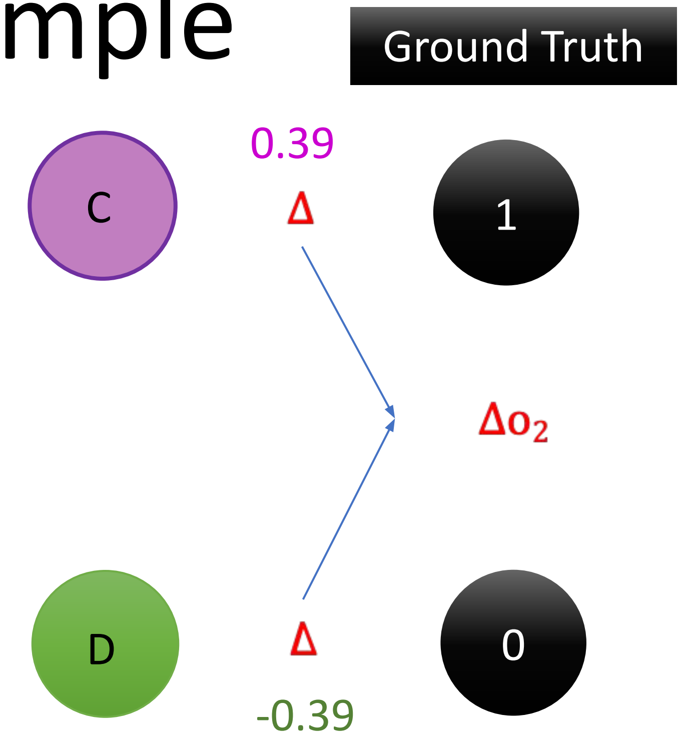
Figure taken from Ranjay Krishna’s course CSE493

Back to our Neural Network Example

$$\Delta w_t = -\frac{\partial \mathrm{E}}{\partial w_t} = o_1^T\, g'(\mathrm{o}_2) \circ \Delta o_2$$
$$w_{t+1} = w_t + \alpha\, \Delta w_t$$

Assume g'(.) = 1

"∘" represents elementwise multiplication for matrix



	0.61 .	0.39 .
o_2	.	.

Neural Network Easy Example

$g'(o_2) \circ \Delta o_2$

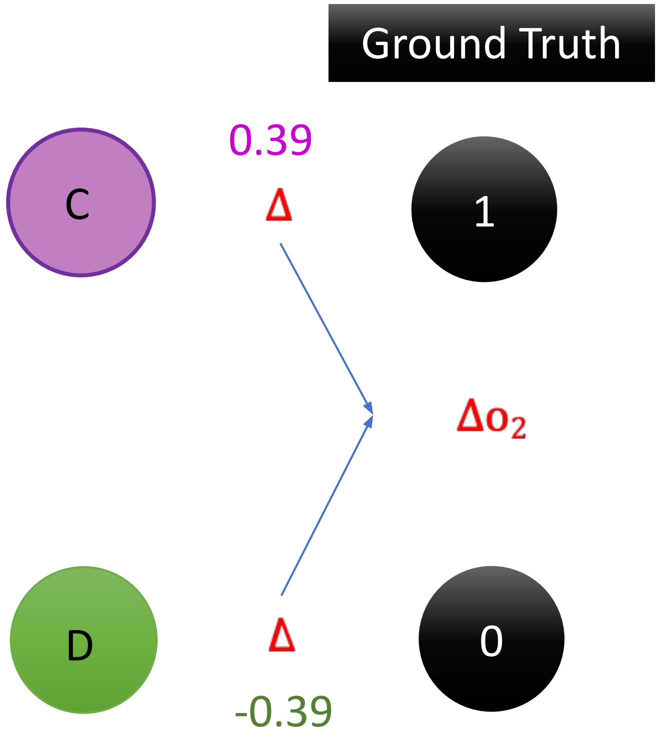
0.39	-0.39
.	.
.	.

$\Delta w_2 = o_1^T g'(o_2) \circ \Delta o_2$

0	0
0.585	-0.585

$\Delta o_1 = g'(o_2) \circ \Delta o_2 w_2^T$

1.092	0.117
.	.
.	.



"∘" represents elementwise multiplication for matrix

$$\Delta w_t = -\frac{\partial E}{\partial w_t} = o_1^T\, g'(o_2) \circ \Delta o_2$$

$$w_{t+1} = w_t + \alpha\, \Delta w_t$$

Assume g'(.) = 1

o_2

0.61	0.39
.	.
.	.