Supervised Learning: Linear regression

Sometimes called “Ridge Regression” with regularizer

$$x_i \in \mathbb{R}^{D_{in}} \qquad y_i \in \mathbb{R}^{D_{out}}$$

Input and output are vectors

$$\ell(\hat{y}, y) = \frac{1}{2}\|\hat{y} - y\|_2^2$$

Loss is Euclidean distance

Linear Regression

$$f(x, W) = Wx$$

$$W \in \mathbb{R}^{D_{out} \times D_{in}}$$

Model is just a matrix multiply

$$R(W) = \lambda \|W\|^2$$

Regularizer is norm of matrix (sum of squares of entries)

Learning Problem

$$W^* = \arg\min_W \frac{1}{2N} \sum_{i=1}^{N} \|Wx_i - y\|_2^2 + \lambda \|W\|^2$$

Supervised Learning: Neural Network

Sometimes called “Fully-Connected Network” or “Multilayer Perceptron”

$$x_i \in \mathbb{R}^{D_{in}} \qquad y_i \in \mathbb{R}^{D_{out}}$$

Input and output are vectors

$$\ell(\hat{y}, y) = \frac{1}{2}\|\hat{y} - y\|_2^2$$

Loss is Euclidean distance

Linear Regression

$$f(x, W) = Wx$$

$$W \in \mathbb{R}^{D_{out} \times D_{in}}$$

Model is just a matrix multiply

New Model

$$f(x, W_1, W_2) = W_2 W_1 x$$

$$W_1 \in \mathbb{R}^{H \times D_{in}}$$

$$W_2 \in \mathbb{R}^{D_{out} \times H}$$

Model is **two** matrix multiplies

Supervised Learning: Neural Network

Sometimes called "Fully-Connected Network" or "Multilayer Perceptron"

$$x_i \in \mathbb{R}^{D_{in}} \qquad y_i \in \mathbb{R}^{D_{out}}$$

Input and output are vectors

$$\ell(\hat{y}, y) = \frac{1}{2}\|\hat{y} - y\|_2^2$$

Loss is Euclidean distance

Linear Regression

$$f(x, W) = Wx$$

$$W \in \mathbb{R}^{D_{out} \times D_{in}}$$

Model is just a matrix multiply

New Model

$$f(x, W_1, W_2) = W_2 W_1 x$$

$$W_1 \in \mathbb{R}^{H \times D_{in}}$$

$$W_2 \in \mathbb{R}^{D_{out} \times H}$$

Model is **two** matrix multiplies

Question: Is the new model "more powerful" than Linear Regression? Can it represent any functions that Linear Regression cannot?

Supervised Learning: Neural Network

Sometimes called "Fully-Connected Network" or "Multilayer Perceptron"

$$x_i \in \mathbb{R}^{D_{in}} \qquad y_i \in \mathbb{R}^{D_{out}}$$

Input and output are vectors

$$\ell(\hat{y}, y) = \frac{1}{2}\|\hat{y} - y\|_2^2$$

Loss is Euclidean distance

Linear Regression

$$f(x, W) = Wx$$

$$W \in \mathbb{R}^{D_{out} \times D_{in}}$$

Model is just a matrix multiply

New Model

$$f(x, W_1, W_2) = W_2 W_1 x$$

$$W_1 \in \mathbb{R}^{H \times D_{in}}$$

$$W_2 \in \mathbb{R}^{D_{out} \times H}$$

Model is **two** matrix multiplies

Question: Is the new model "more powerful" than Linear Regression? Can it represent any functions that Linear Regression cannot?

Answer: NO! We can write $W = W_2 W_1$
And recover Linear Regression

Supervised Learning: Neural Network

Sometimes called “Fully-Connected Network” or “Multilayer Perceptron”

$$x_i \in \mathbb{R}^{D_{in}} \qquad y_i \in \mathbb{R}^{D_{out}}$$

Input and output are vectors

$$\ell(\hat{y}, y) = \frac{1}{2}\|\hat{y} - y\|_2^2$$

Loss is Euclidean distance

Linear Regression

$$f(x, W) = Wx$$

$$W \in \mathbb{R}^{D_{out} \times D_{in}}$$

Model is just a matrix multiply

Neural Network

$$\cancel{f(x, W_1, W_2) = W_2 W_1 x}$$

$$f(x, W_1, W_2) = W_2 \sigma(W_1 x)$$

$$W_1 \in \mathbb{R}^{H \times D_{in}}$$

$$W_2 \in \mathbb{R}^{D_{out} \times H}$$

Model is **two** matrix multiplies, with an **elementwise nonlinearity**

$$\sigma : \mathbb{R}^H \to \mathbb{R}^H$$

Supervised Learning: Neural Network

Sometimes called “Fully-Connected Network” or “Multilayer Perceptron”

$$x_i \in \mathbb{R}^{D_{in}} \qquad y_i \in \mathbb{R}^{D_{out}}$$

Input and output are vectors

$$\ell(\hat{y}, y) = \frac{1}{2}\|\hat{y} - y\|_2^2$$

Loss is Euclidean distance

Linear Regression

$$f(x, W) = Wx$$

$$W \in \mathbb{R}^{D_{out} \times D_{in}}$$

Model is just a matrix multiply

Neural Network

$$\cancel{f(x, W_1, W_2) = W_2 W_1 x}$$

$$f(x, W_1, W_2) = W_2 \sigma(W_1 x)$$

$$W_1 \in \mathbb{R}^{H \times D_{in}}$$

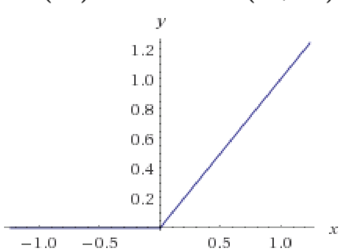
$$W_2 \in \mathbb{R}^{D_{out} \times H}$$

Model is **two** matrix multiplies, with an **elementwise nonlinearity**

$$\sigma : \mathbb{R}^H \to \mathbb{R}^H$$

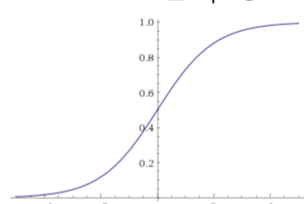
Common nonlinearities:

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



Rectified Linear (ReLU)

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



Logistic Sigmoid

Supervised Learning: Neural Network

Sometimes called "Fully-Connected Network" or "Multilayer Perceptron"

$$x_i \in \mathbb{R}^{D_{in}} \qquad y_i \in \mathbb{R}^{D_{out}}$$

Input and output are vectors

$$\ell(\hat{y}, y) = \frac{1}{2}\|\hat{y} - y\|_2^2$$

Loss is Euclidean distance

Linear Regression

$$f(x, W) = Wx$$

$$W \in \mathbb{R}^{D_{out} \times D_{in}}$$

Model is just a matrix multiply

Neural Network

$$\cancel{f(x, W_1, W_2) = W_2 W_1 x}$$

$$f(x, W_1, W_2) = W_2 \sigma(W_1 x)$$

$$W_1 \in \mathbb{R}^{H \times D_{in}}$$

$$W_2 \in \mathbb{R}^{D_{out} \times H}$$

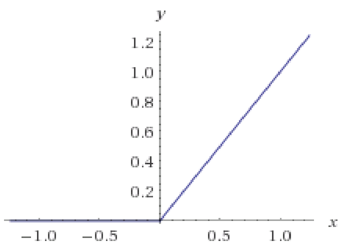
Model is **two** matrix multiplies, with an **elementwise nonlinearity**

$$\sigma : \mathbb{R}^H \to \mathbb{R}^H$$

Neural Network is more powerful than Linear Regression!

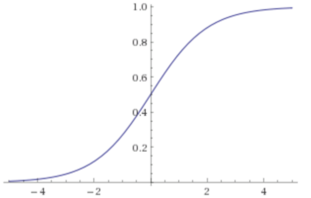
Common nonlinearities:

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



Rectified Linear (ReLU)

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



Logistic Sigmoid

Neural Networks with many layers

Sometimes called “Fully-Connected Network” or “Multilayer Perceptron”

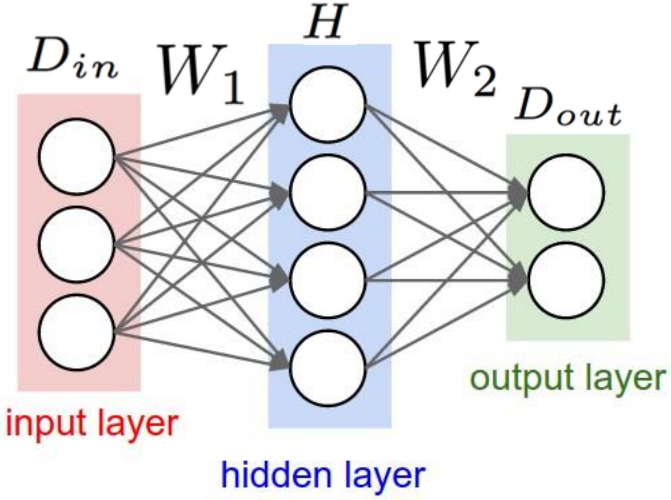
$$x_i \in \mathbb{R}^{D_{in}} \qquad y_i \in \mathbb{R}^{D_{out}}$$

Two Layer network

$$f(x, W_1, W_2) = W_2\sigma(W_1 x)$$

$$W_1 \in \mathbb{R}^{H \times D_{in}}$$

$$W_2 \in \mathbb{R}^{D_{out} \times H}$$



Neural Networks with many layers

Sometimes called “Fully-Connected Network” or “Multilayer Perceptron”

$$x_i \in \mathbb{R}^{D_{in}} \qquad y_i \in \mathbb{R}^{D_{out}}$$

Two Layer network

$$f(x, W_1, W_2) = W_2\sigma(W_1 x)$$

$$W_1 \in \mathbb{R}^{H \times D_{in}}$$

$$W_2 \in \mathbb{R}^{D_{out} \times H}$$

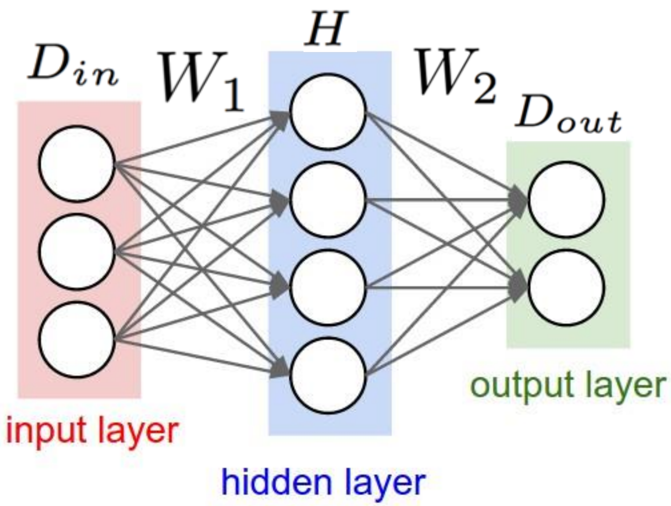
Three Layer network

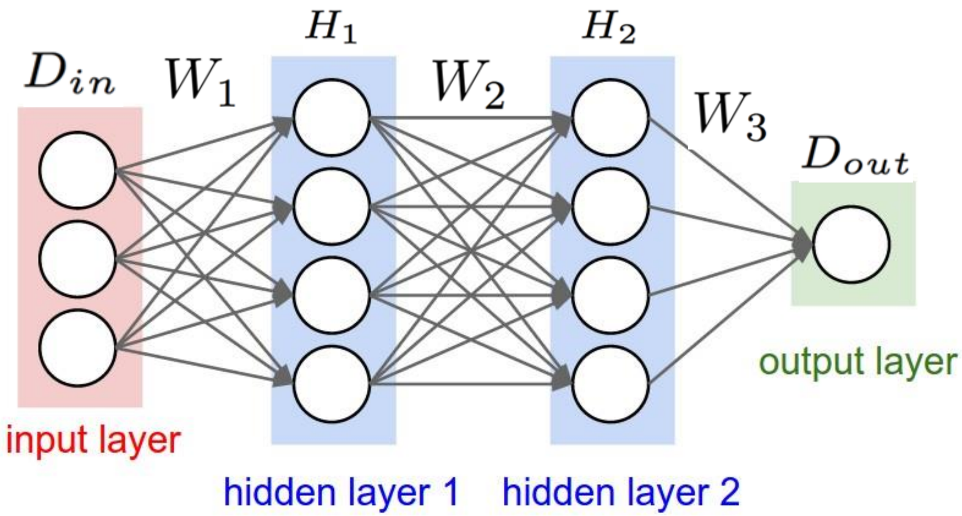
$$f(x, W_1, W_2, W_3) = W_3\sigma(W_2(\sigma(W_1 x)))$$

$$W_1 \in \mathbb{R}^{H_1 \times D_{in}}$$

$$W_2 \in \mathbb{R}^{H_2 \times H_1}$$

$$W_3 \in \mathbb{R}^{D_{out} \times H_2}$$





Neural Networks with many layers

Sometimes called “Fully-Connected Network” or “Multilayer Perceptron”

$$x_i \in \mathbb{R}^{D_{in}} \qquad y_i \in \mathbb{R}^{D_{out}}$$

Two Layer network

$$f(x, W_1, W_2) = W_2\sigma(W_1 x)$$

$$W_1 \in \mathbb{R}^{H \times D_{in}}$$

$$W_2 \in \mathbb{R}^{D_{out} \times H}$$

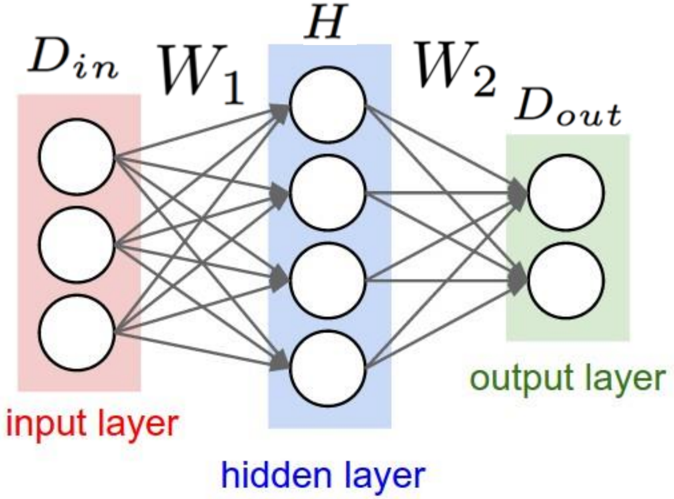
Three Layer network

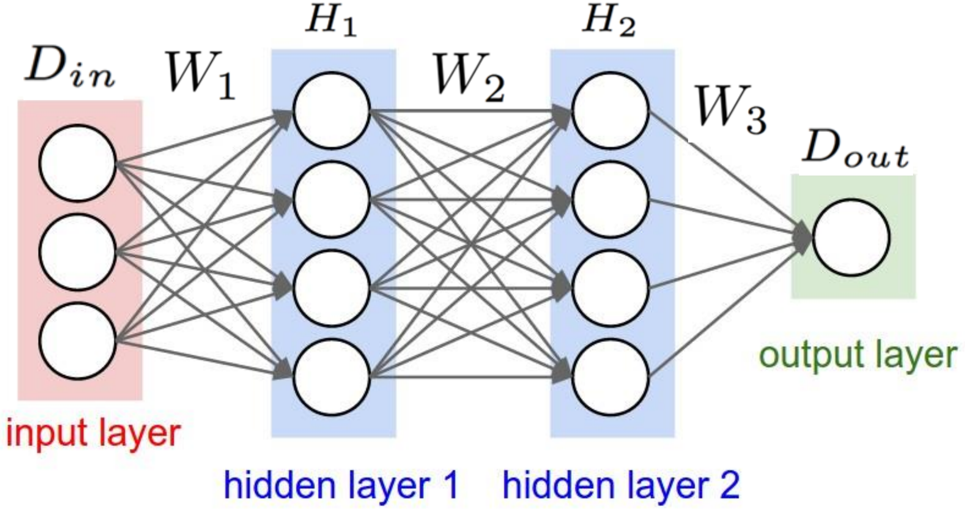
$$f(x, W_1, W_2, W_3) = W_3\sigma(W_2(\sigma(W_1 x)))$$

$$W_1 \in \mathbb{R}^{H_1 \times D_{in}}$$

$$W_2 \in \mathbb{R}^{H_2 \times H_1}$$

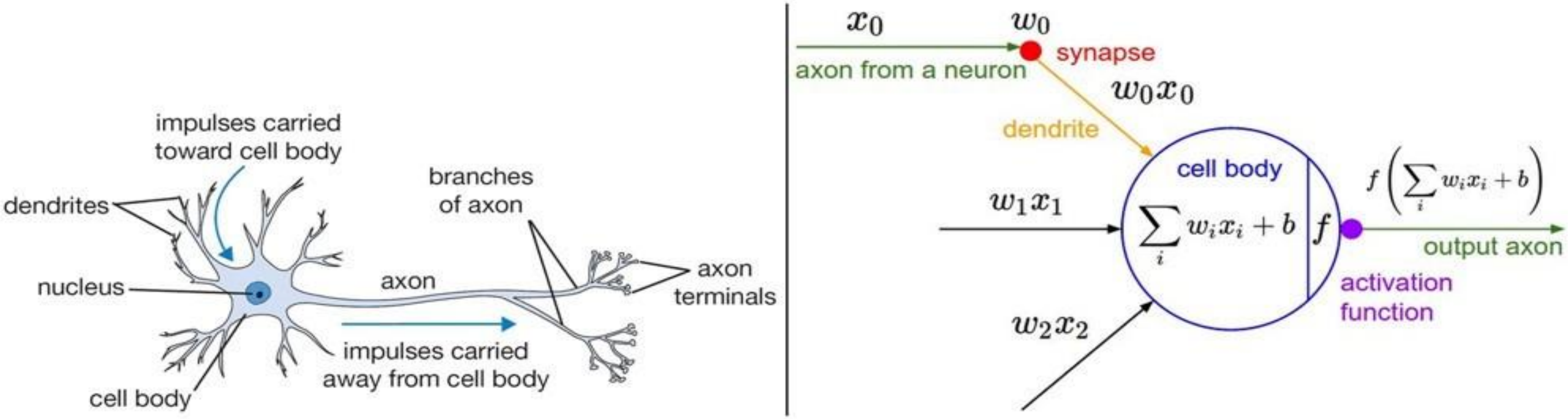
$$W_3 \in \mathbb{R}^{D_{out} \times H_2}$$





Hidden layers are **learned feature representations** of the input!

Neurons: Inspiration from Biology



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

Neural nets/perceptrons are loosely inspired by biology.

But they are NOT how the brain works, or even how neurons work.

How to compute gradients?

Work it out on paper?

Linear Regression

$$L = g(W) = \frac{1}{2N}\sum_{i=1}^{N} \|Wx_i - y_i\|_2^2 + \lambda\|W\|$$

$$\frac{\mathrm{dL}}{\mathrm{dW}} = \nabla g(W) = \ ?$$

Doable for simple models

How to compute gradients?

Work it out on paper?

Three layer network

$$g(W_1, W_2, W_3) = \frac{1}{2N} \sum_{i=1}^{N} \|W_3 \sigma(W_2 \sigma(W_1 x_i)) - y_i\|_2^2$$

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

$$\nabla_{W_1} g(W_1, W_2, W_3) = ?$$

$$\nabla_{W_2} g(W_1, W_2, W_3) = ?$$

$$\nabla_{W_3} g(W_1, W_2, W_3) = ?$$

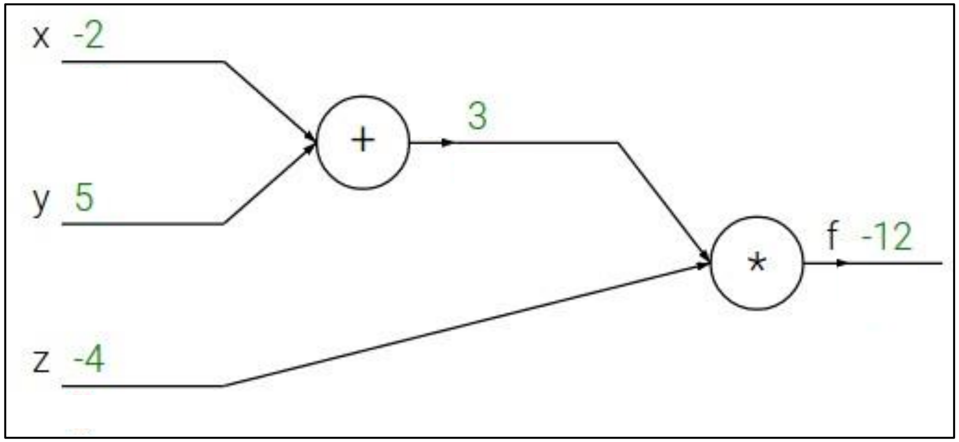
Gets hairy for more complex models

Backpropagation

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

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

Computational graph



Backpropagation

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

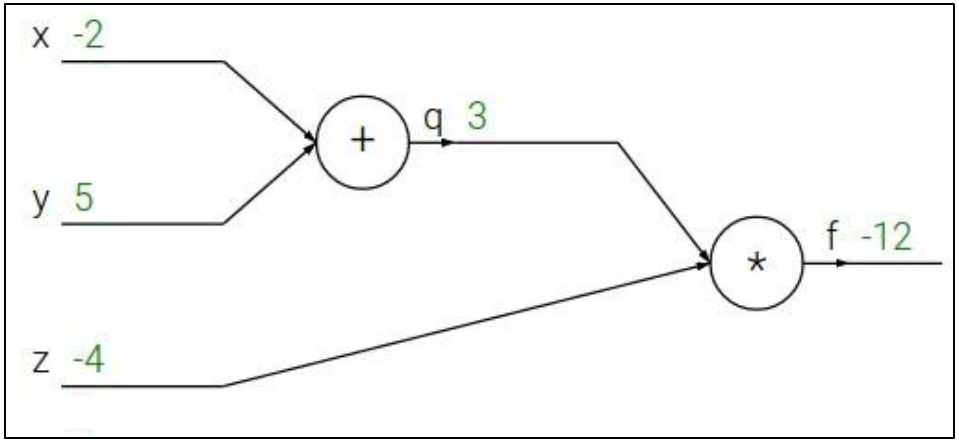
e.g. x = -2, y = 5, z = -4

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

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

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

Computational graph



Backpropagation

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

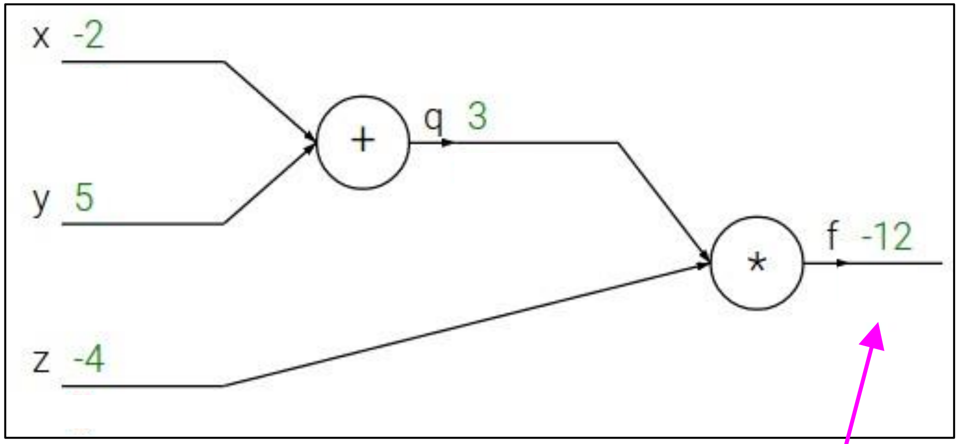
e.g. x = -2, y = 5, z = -4

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

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

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

Computational graph



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

Backpropagation

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

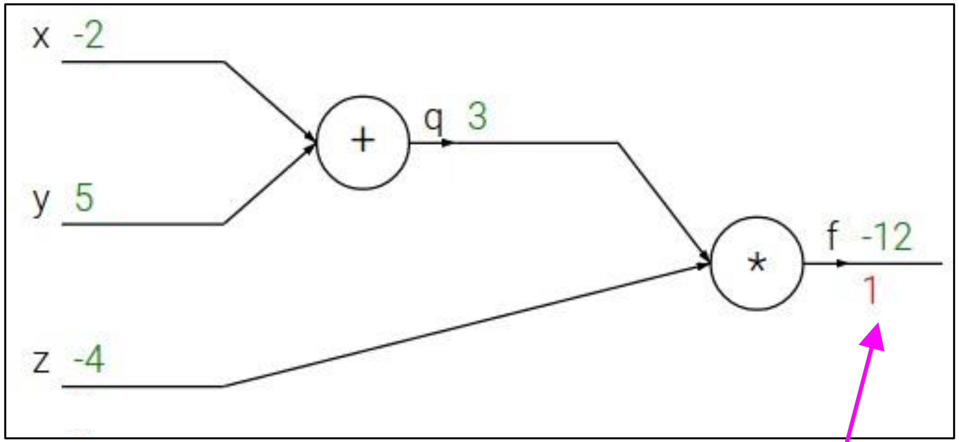
e.g. x = -2, y = 5, z = -4

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

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

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

Computational graph



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

Backpropagation

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

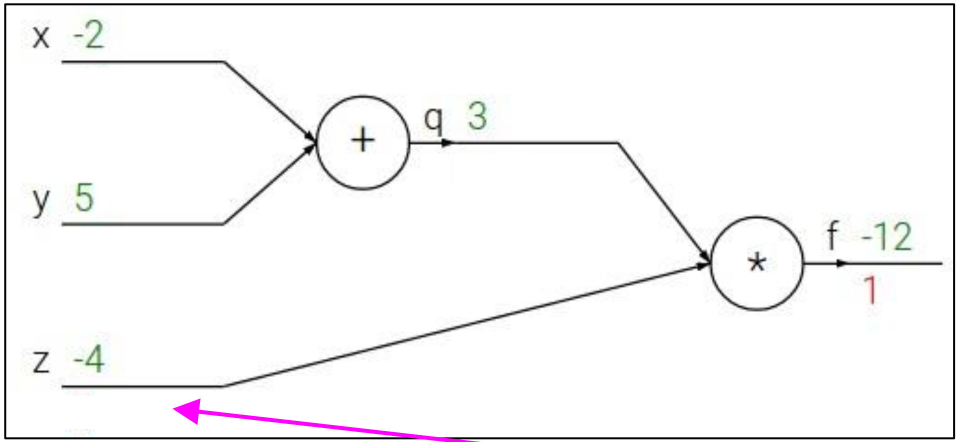
e.g. x = -2, y = 5, z = -4

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

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

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

Computational graph



$$\frac{\partial f}{\partial z}$$

Backpropagation

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

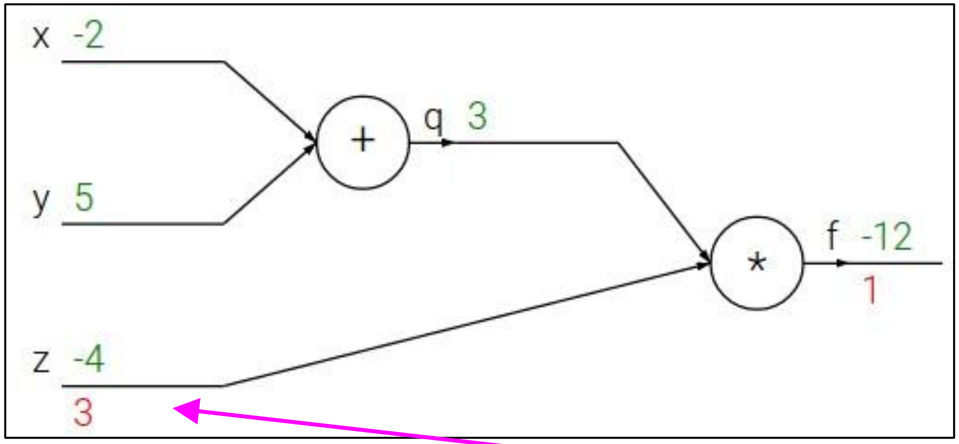
e.g. x = -2, y = 5, z = -4

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

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

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

Computational graph



$$\frac{\partial f}{\partial z}$$

Backpropagation

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

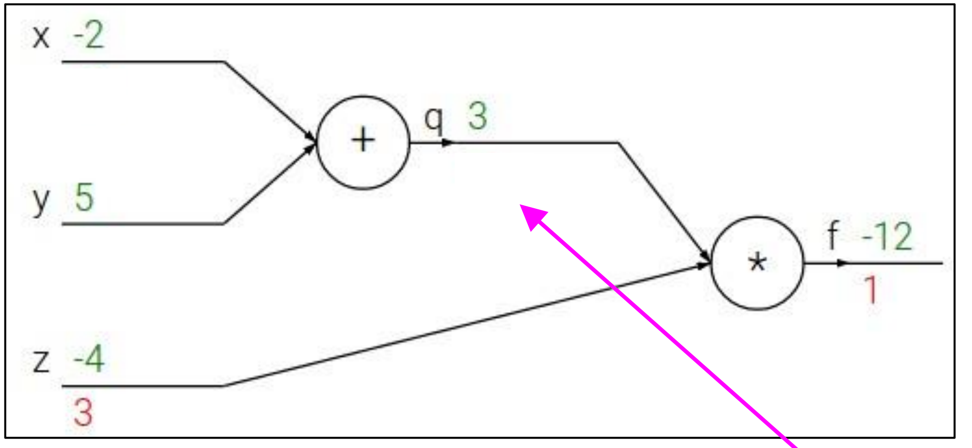
e.g. x = -2, y = 5, z = -4

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

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

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

Computational graph



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

Backpropagation

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

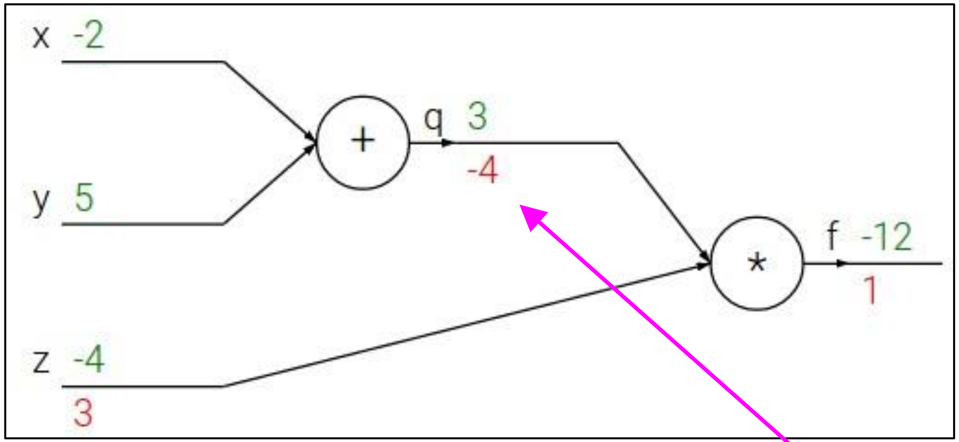
e.g. x = -2, y = 5, z = -4

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

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

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

Computational graph



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

Backpropagation

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

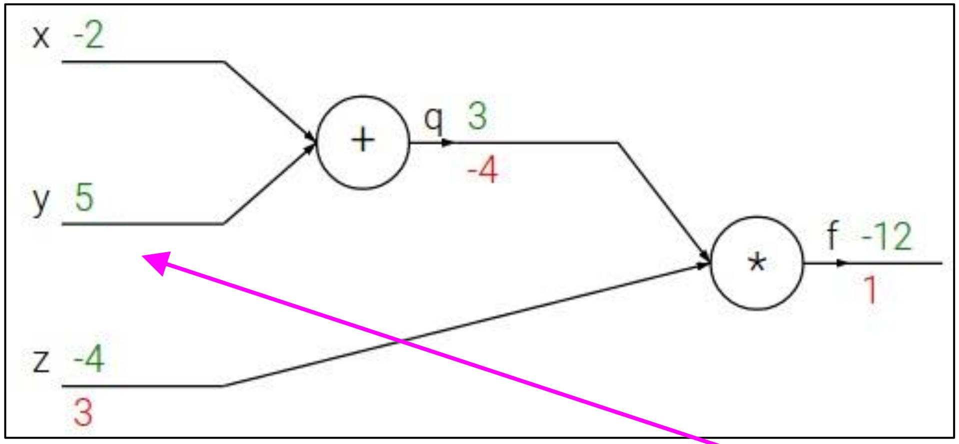
e.g. x = -2, y = 5, z = -4

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

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

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

Computational graph



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

Backpropagation

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

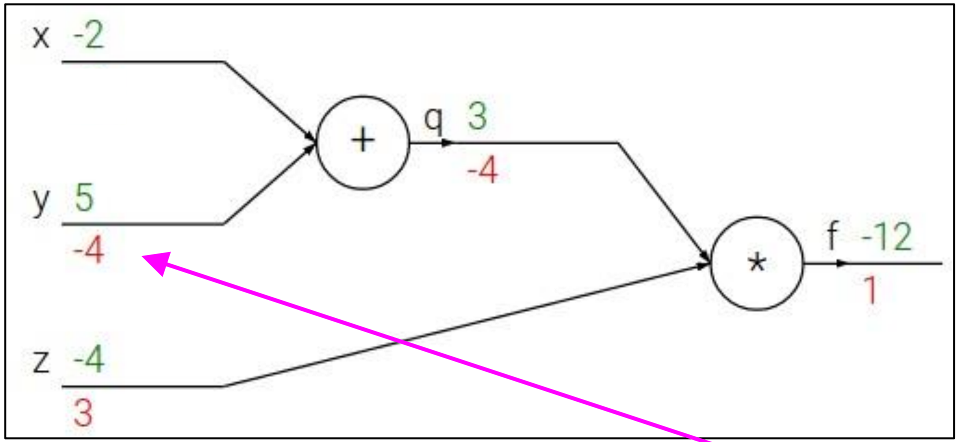
e.g. x = -2, y = 5, z = -4

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

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

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

Computational graph



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

Chain rule:

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

Backpropagation

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

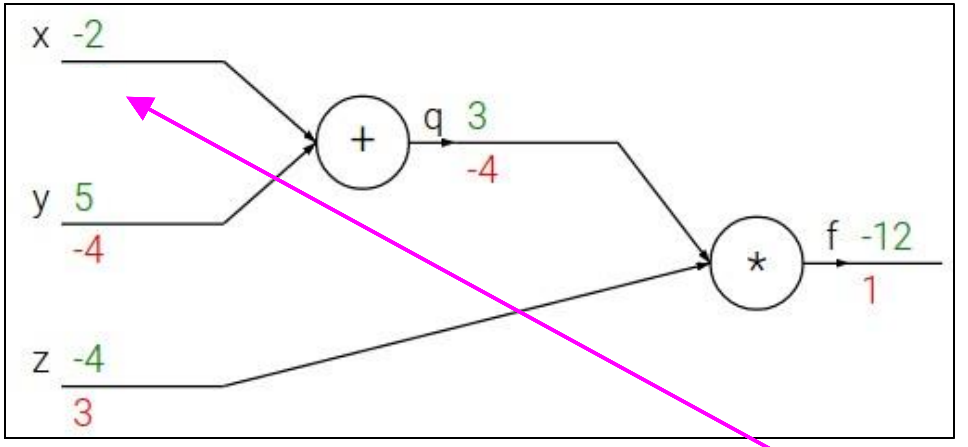
e.g. x = -2, y = 5, z = -4

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

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

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

Computational graph



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

Backpropagation

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

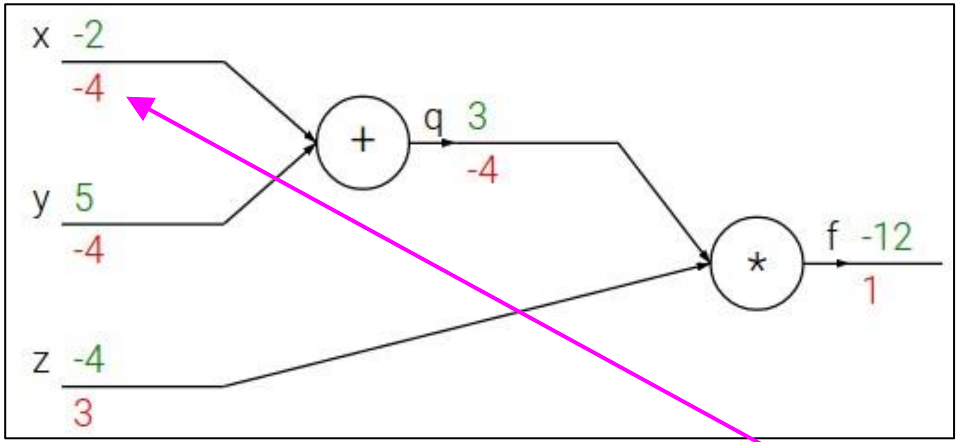
e.g. x = -2, y = 5, z = -4

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

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

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

Computational graph



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

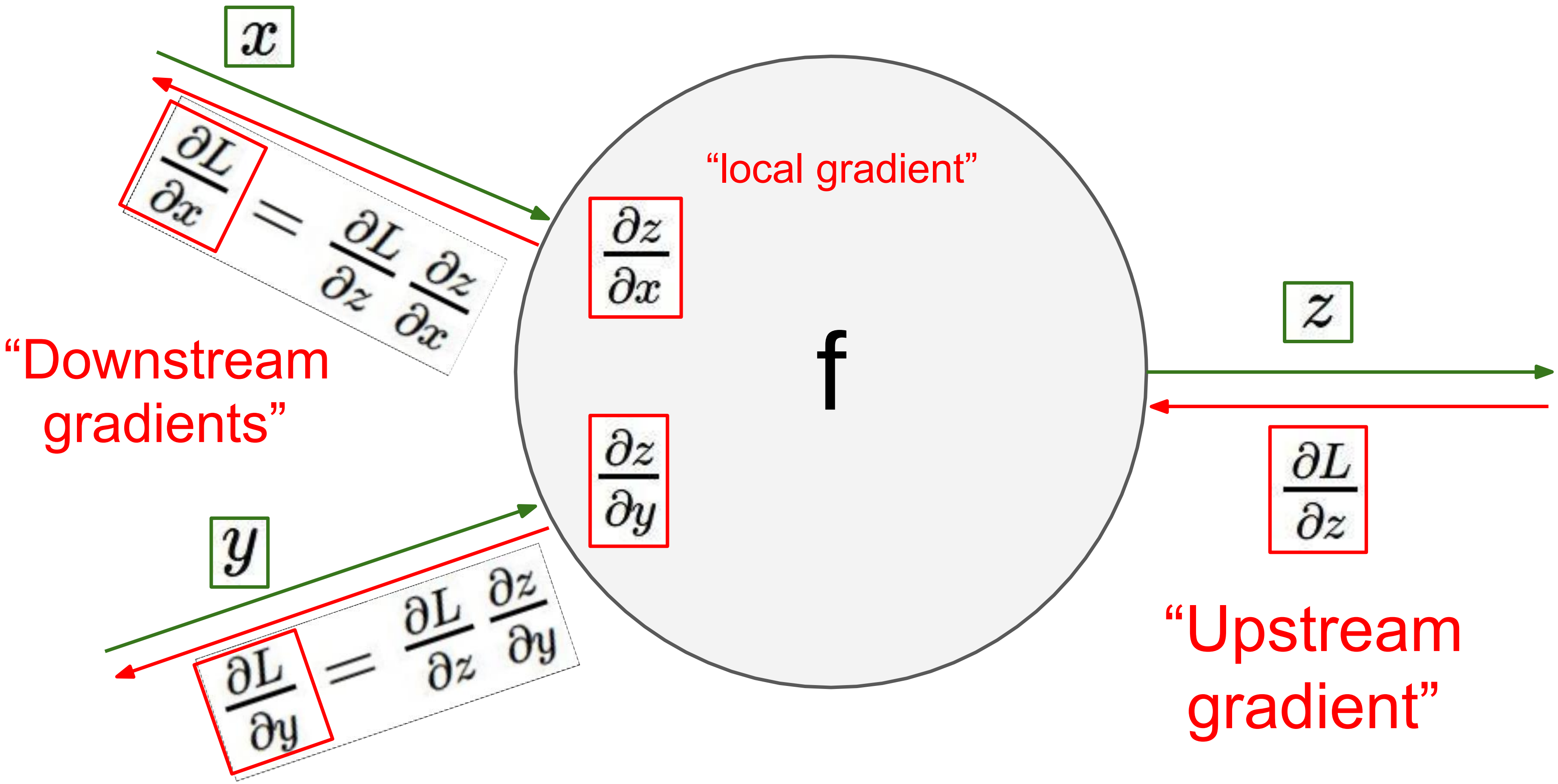
Chain rule:

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

Upstream gradient

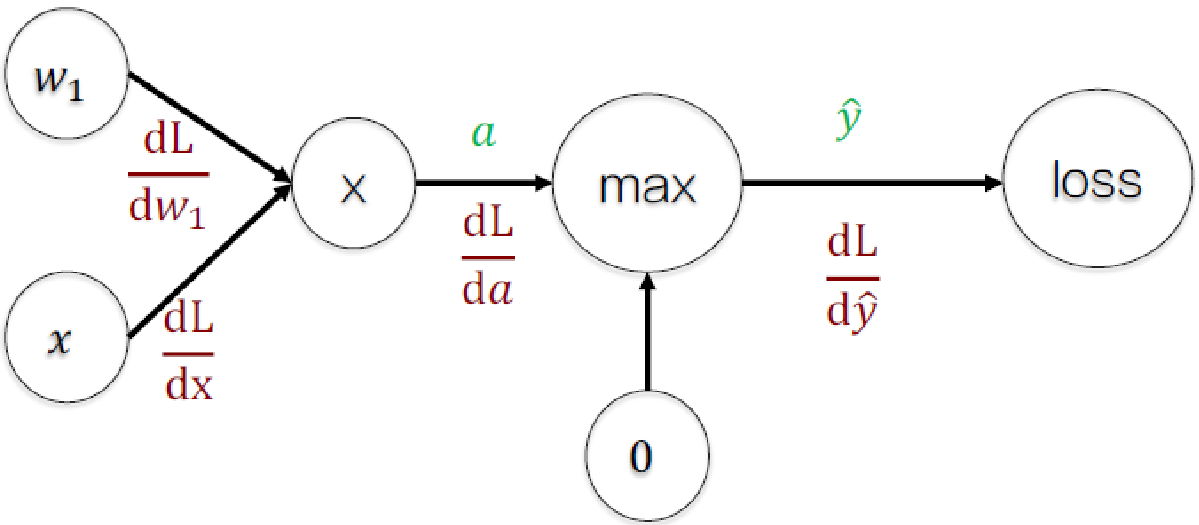
Local gradient

Backpropagation



Backpropagation

Simple example



$$a = wx$$

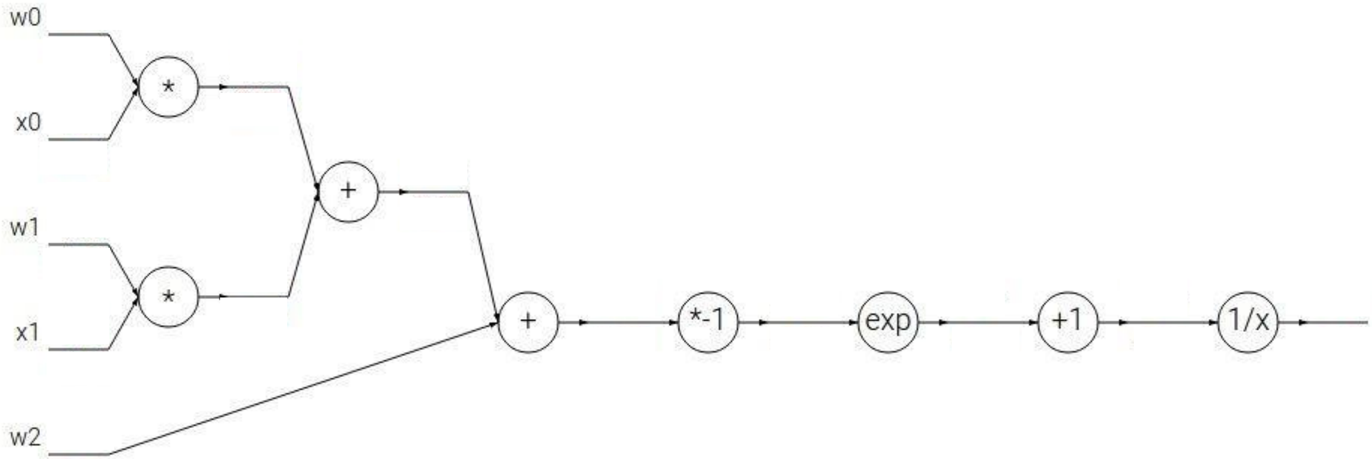
$$\hat{y} = \max(0, a)$$

$$\frac{dL}{da} = \frac{dL}{d\hat{y}}\frac{d\hat{y}}{da}$$

$$\frac{dL}{dw_1} = \frac{dL}{da}\frac{da}{dw_1}$$

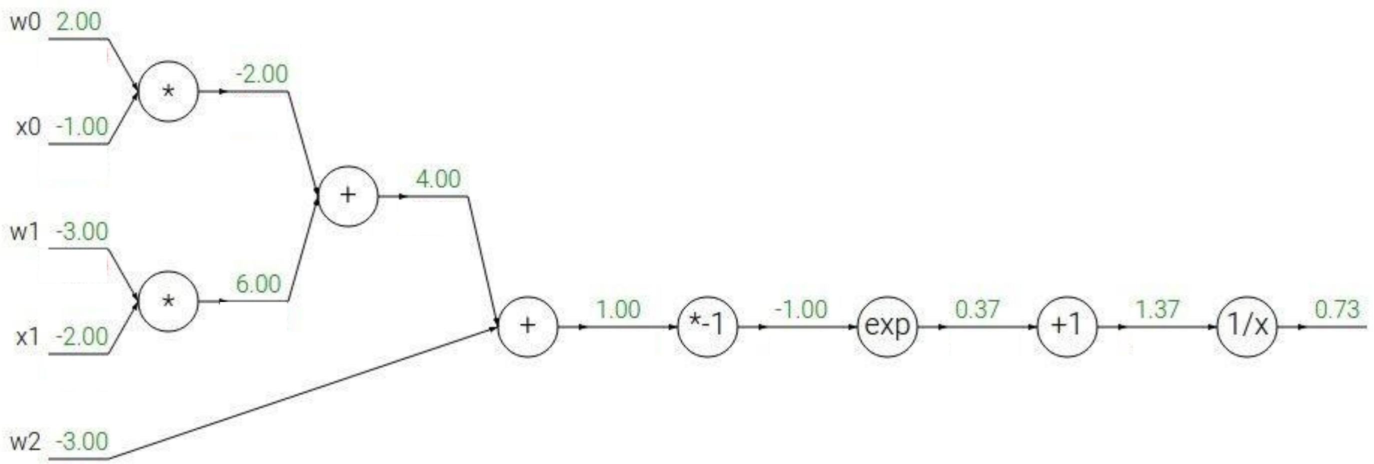
$$\frac{dL}{dx} = \frac{dL}{da}\frac{da}{dx}$$

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

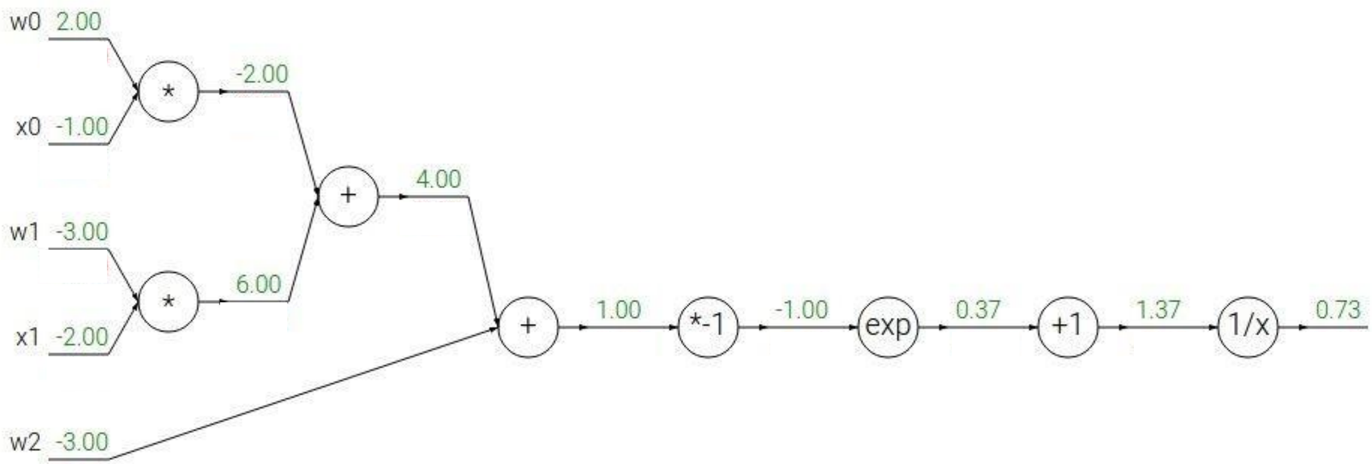


Another example:

$$f(w,x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$



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



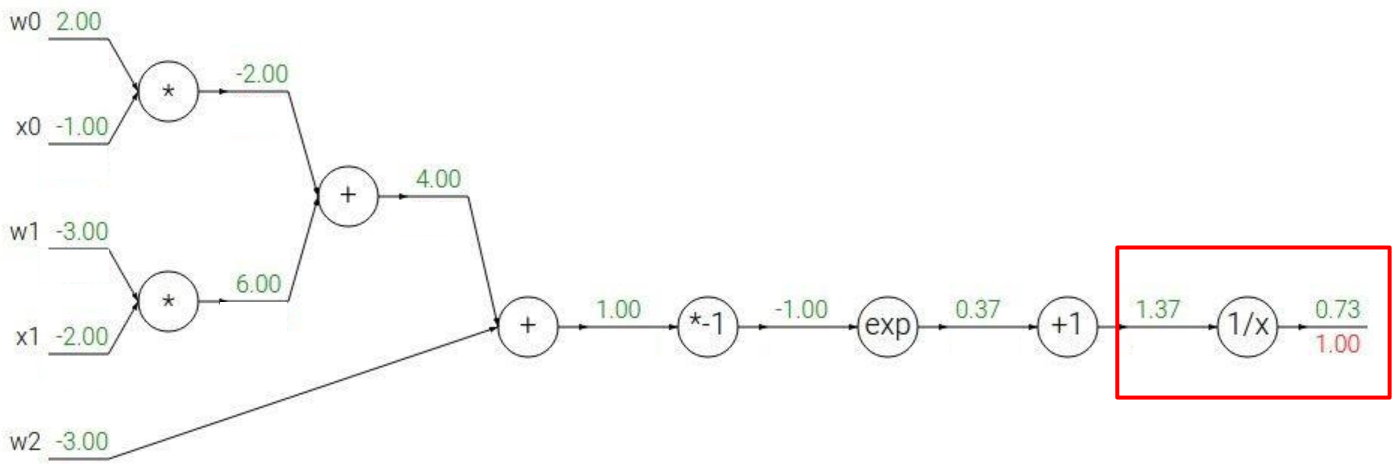
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

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

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

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

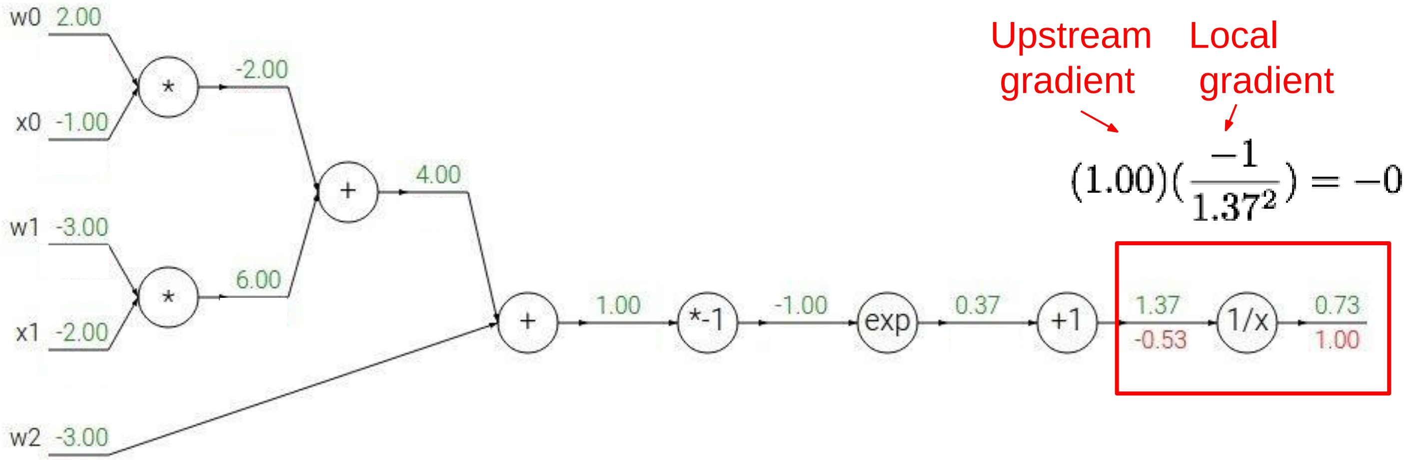
Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



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

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

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



$$(1.00)\left(\frac{-1}{1.37^2}\right) = -0.53$$

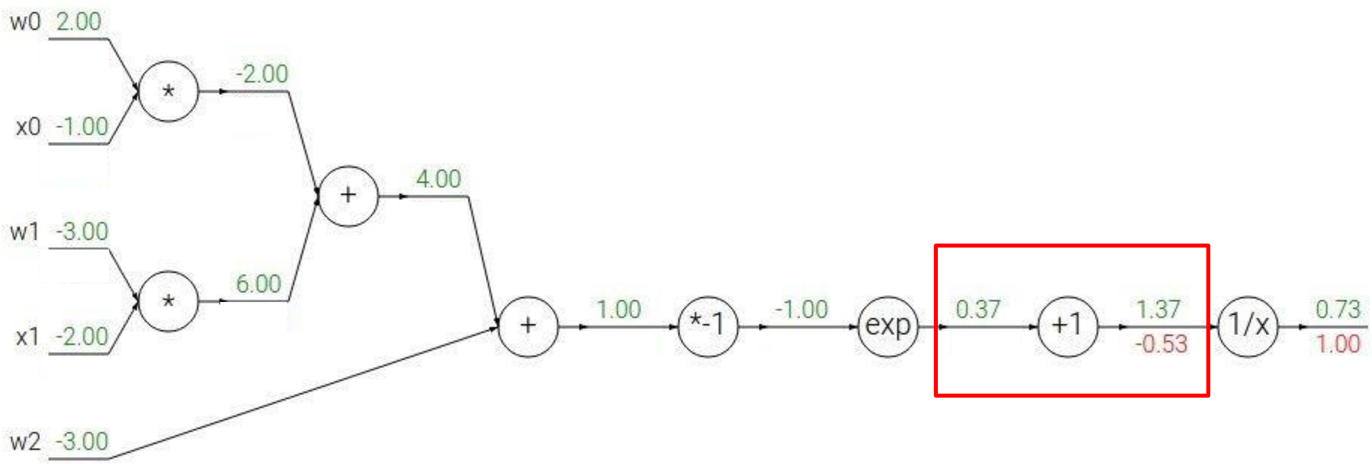
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

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

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

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

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



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

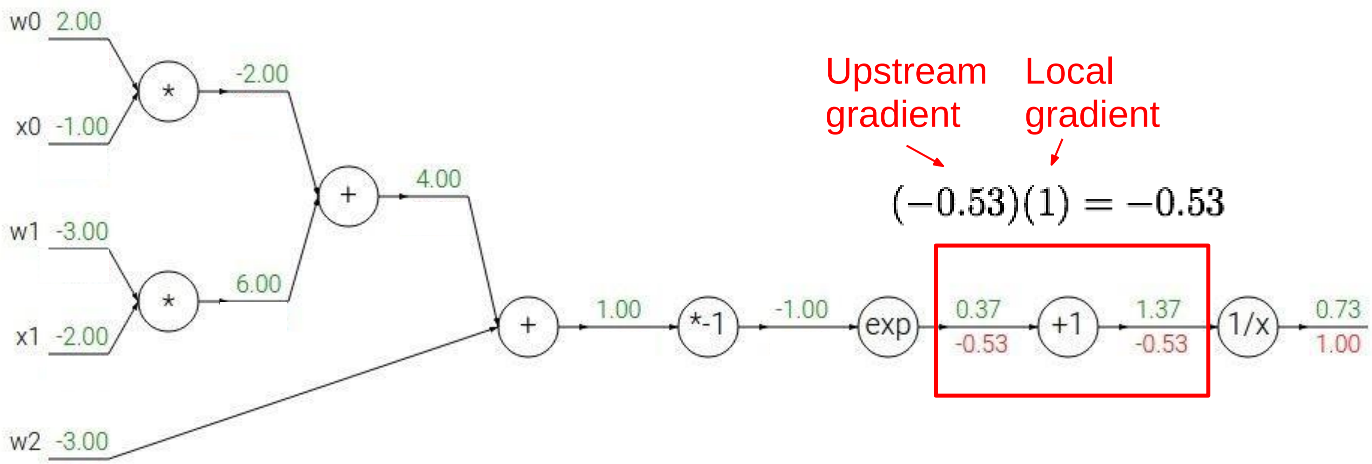
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Another example:

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



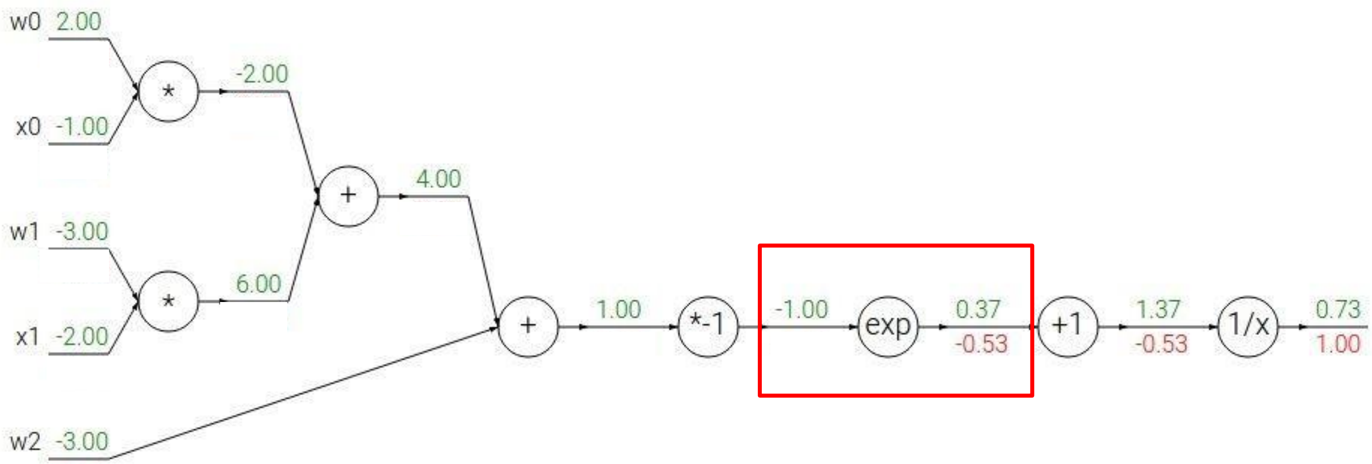
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

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

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

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

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



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

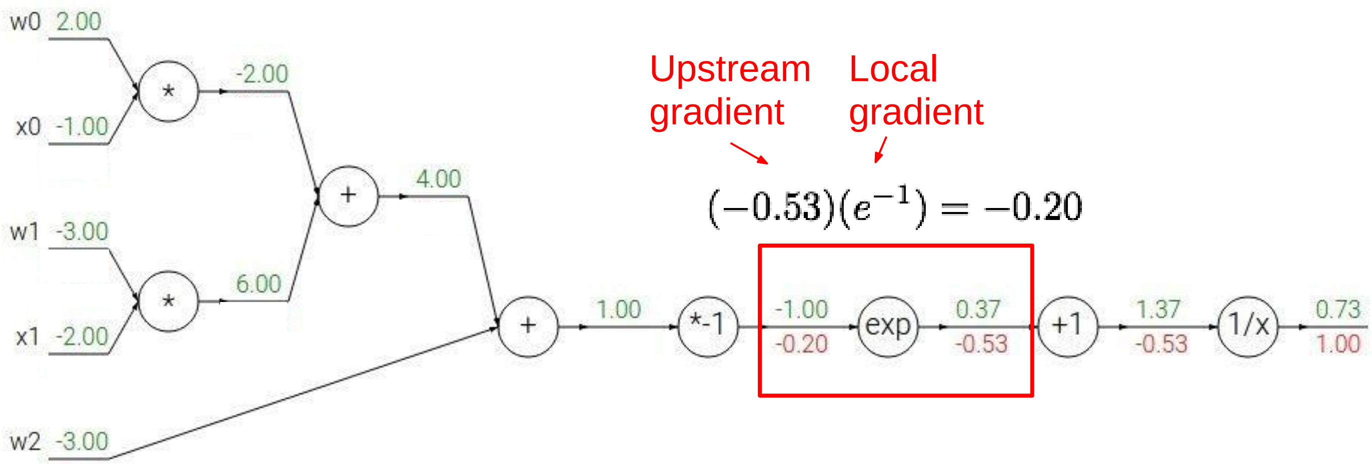
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Another example:

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



$$(-0.53)(e^{-1}) = -0.20$$

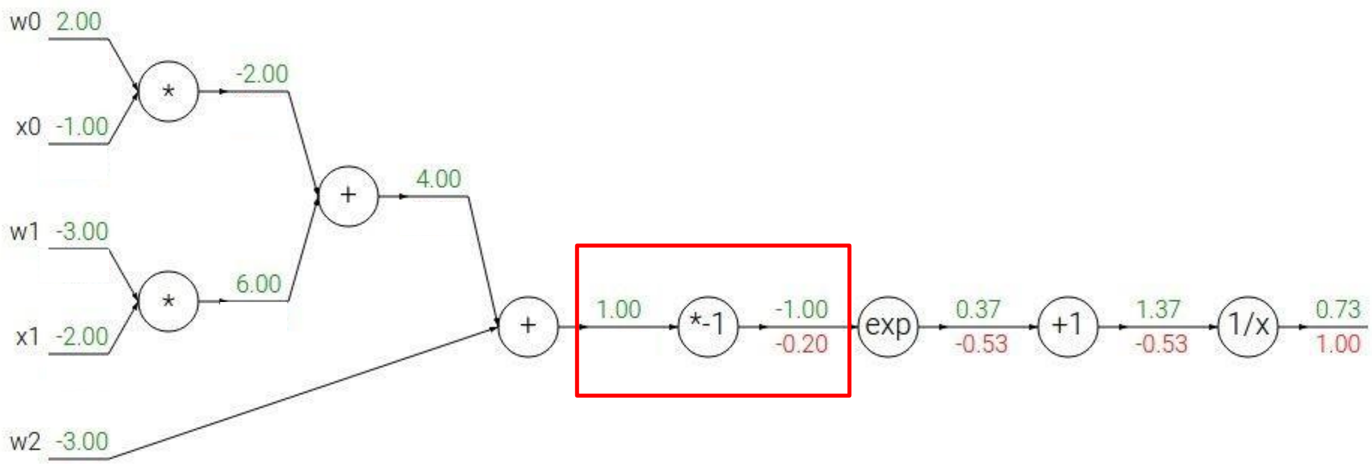
$$\boxed{f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x}$$

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

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

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

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



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

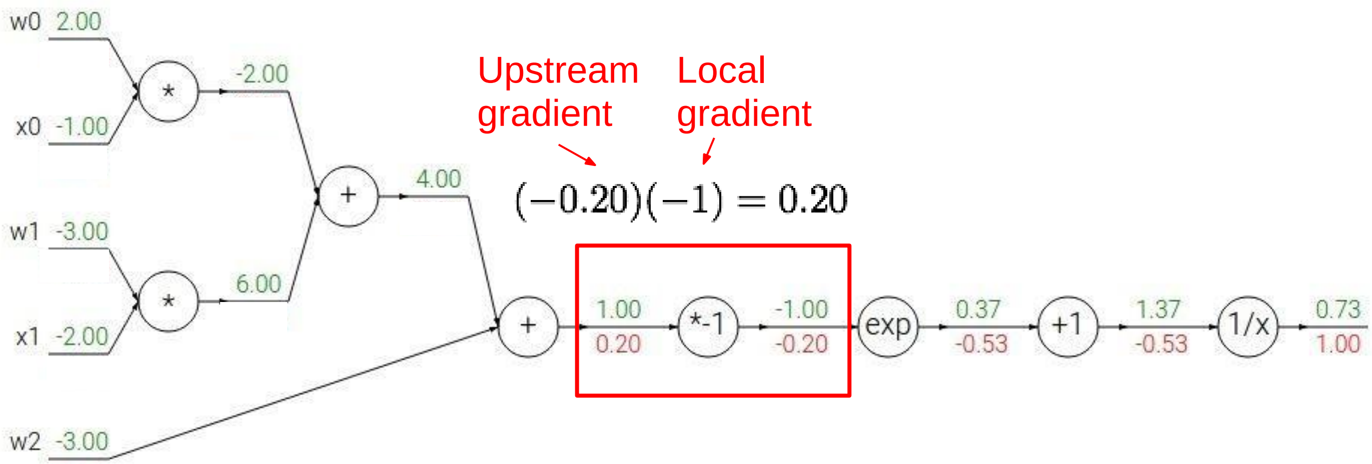
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Another example:

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



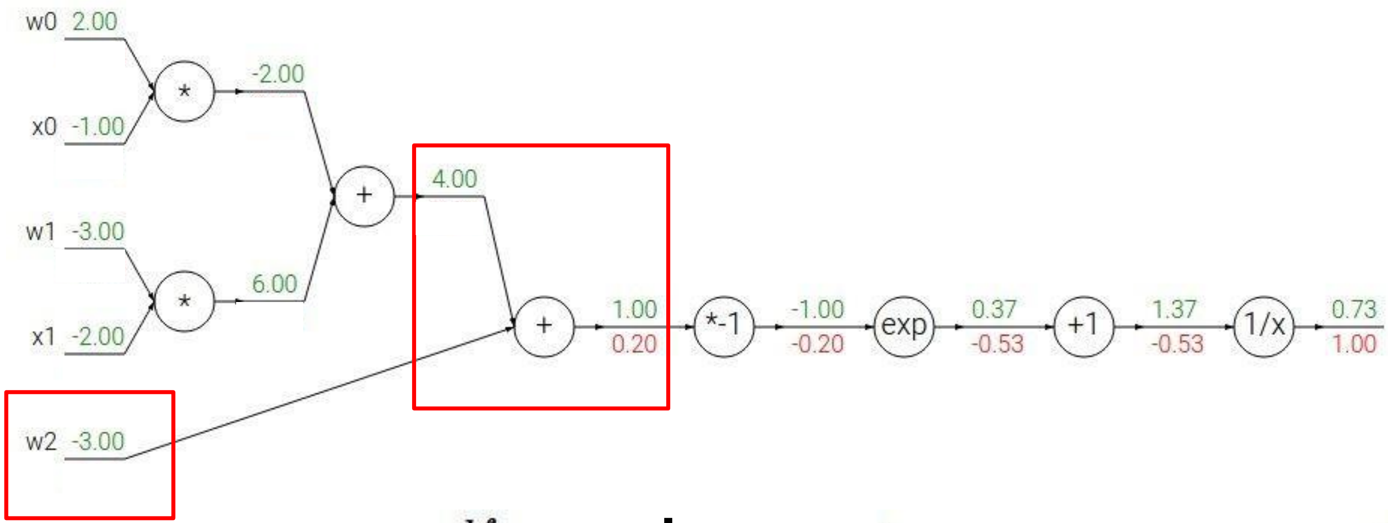
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

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

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

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

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



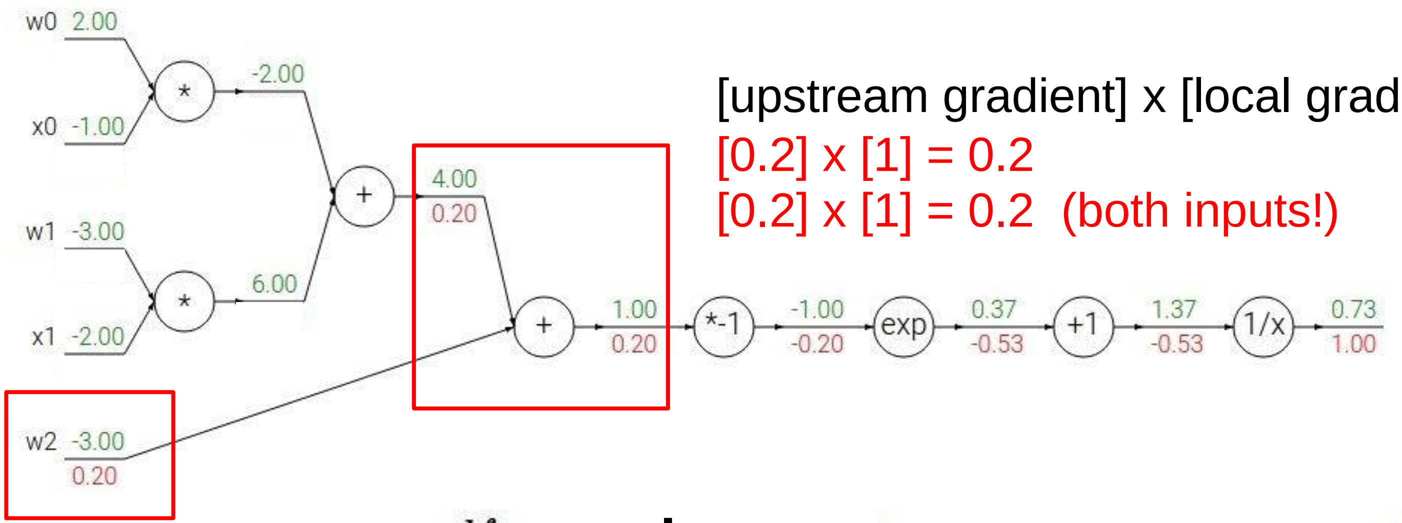
$$f(x)=e^x \quad \rightarrow \quad \frac{df}{dx}=e^x$$

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

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

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

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



[upstream gradient] x [local gradient]

[0.2] x [1] = 0.2

[0.2] x [1] = 0.2 (both inputs!)

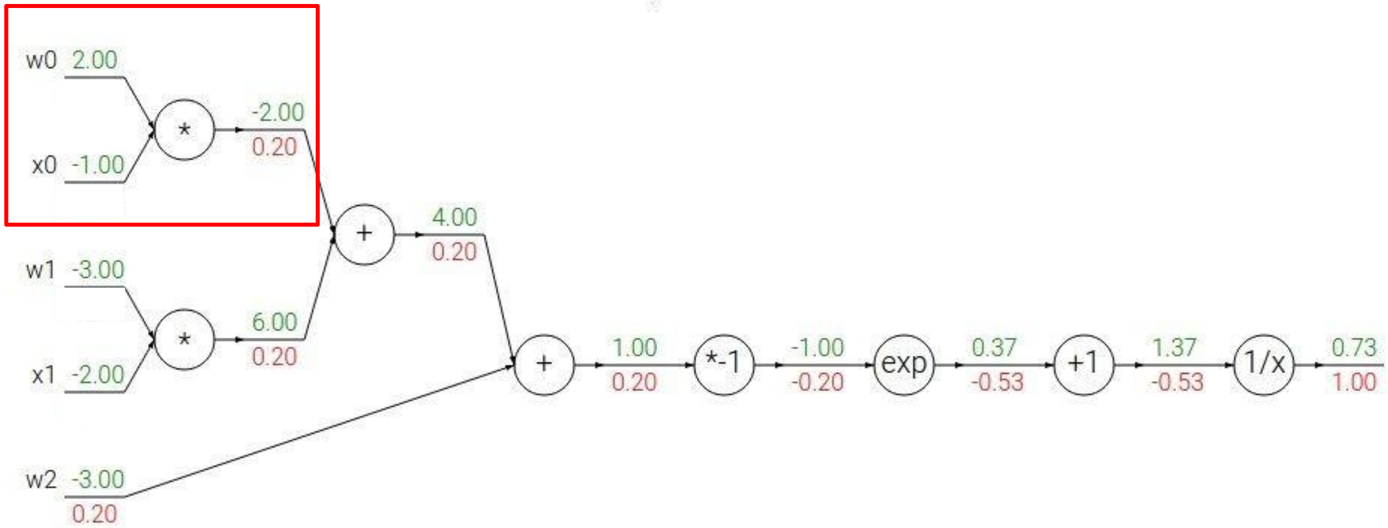
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

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

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

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

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

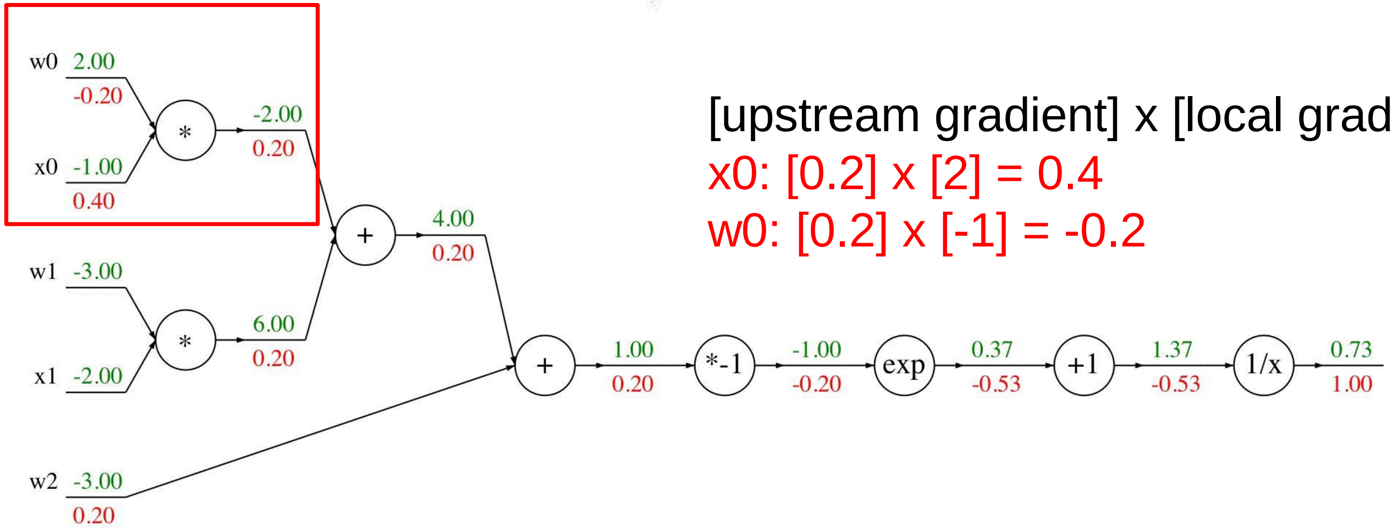


$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \qquad\qquad f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

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

Another example:

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



[upstream gradient] x [local gradient]
x0: [0.2] x [2] = 0.4
w0: [0.2] x [-1] = -0.2

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

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

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

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

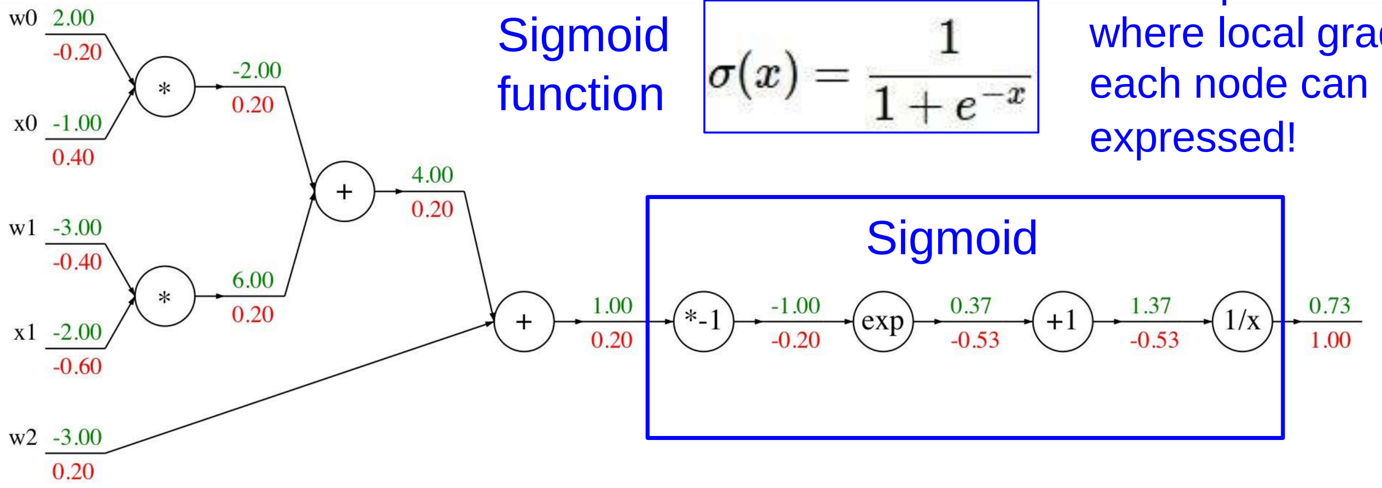
Another example:

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

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

Sigmoid function

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



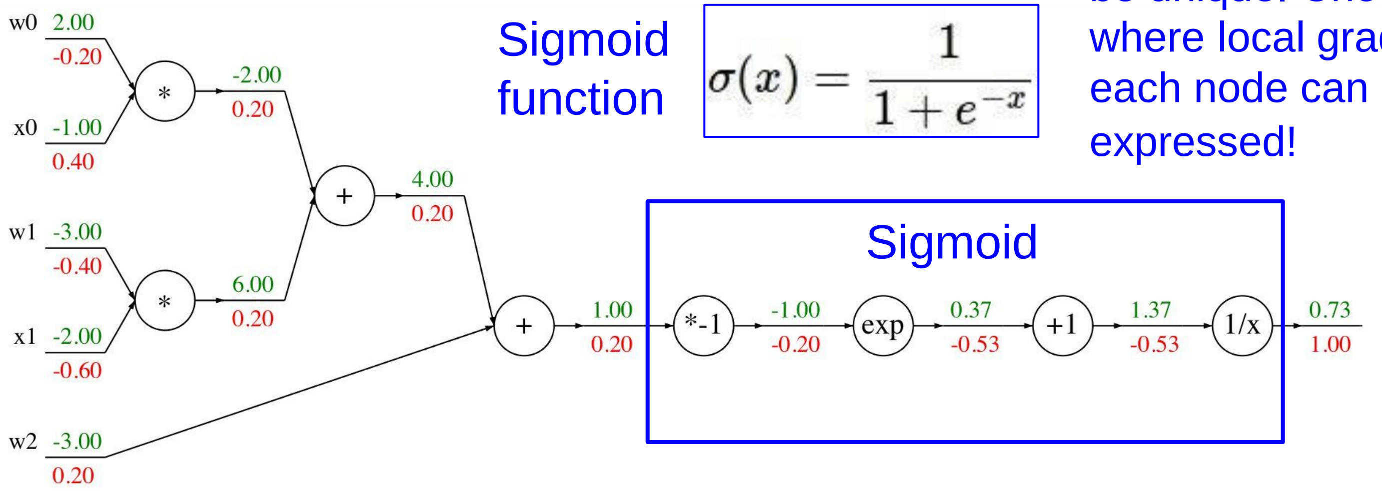
Another example:

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

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:

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

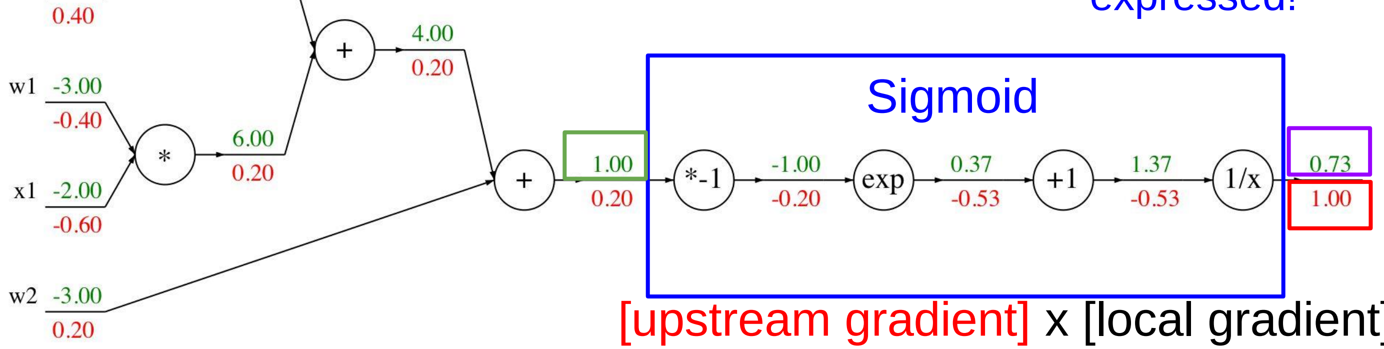
Another example:

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

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

Sigmoid function

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



[upstream gradient] x [local gradient]

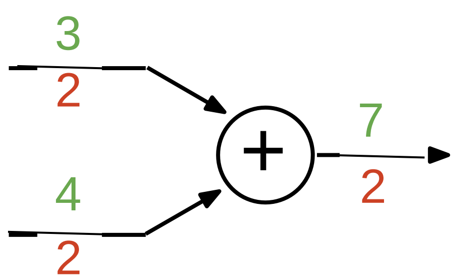
[1.00] x [(1 - 0.73) (0.73)] = 0.2

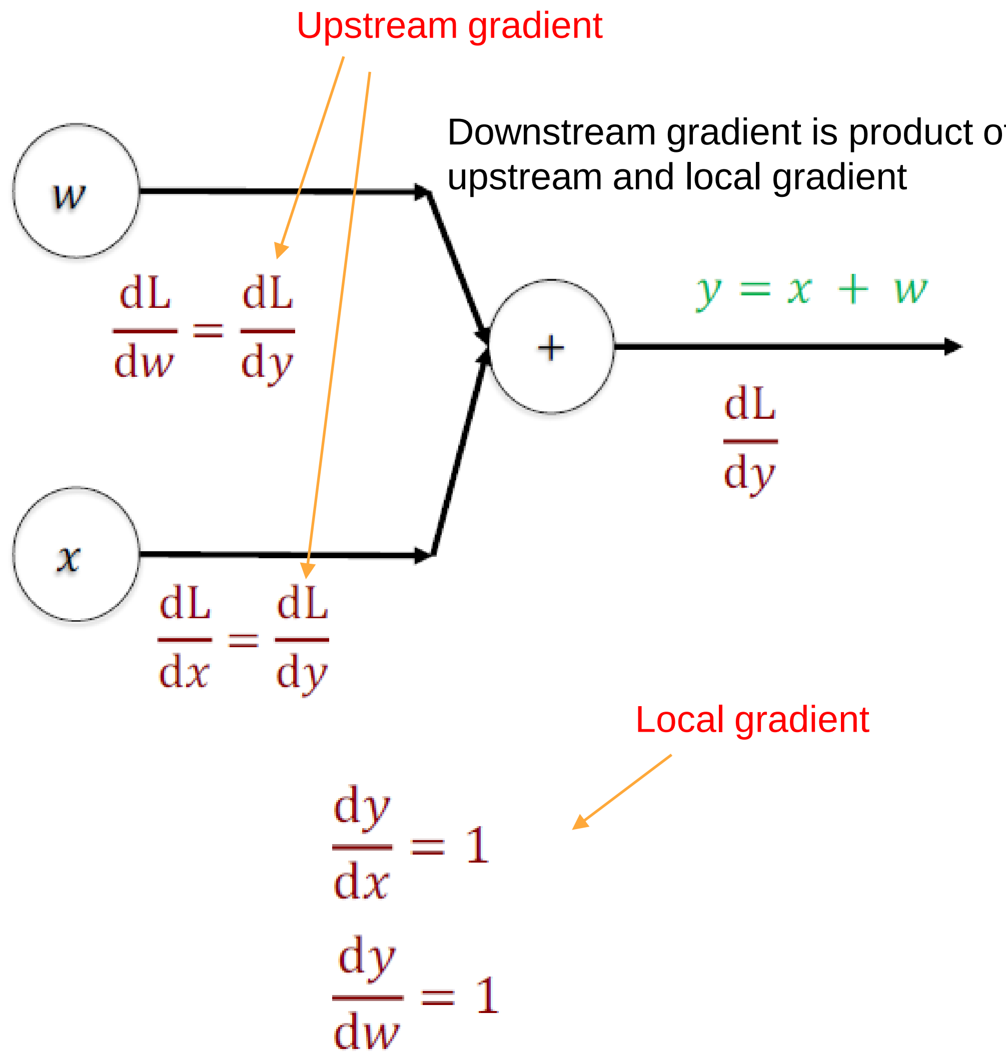
Sigmoid local gradient:

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

Patterns in gradient flow

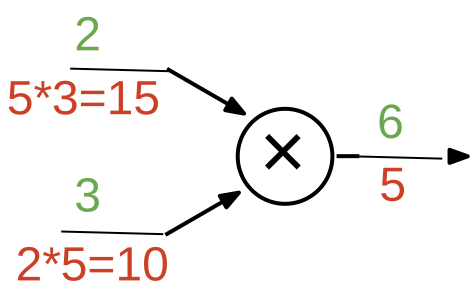
add gate: gradient distributor

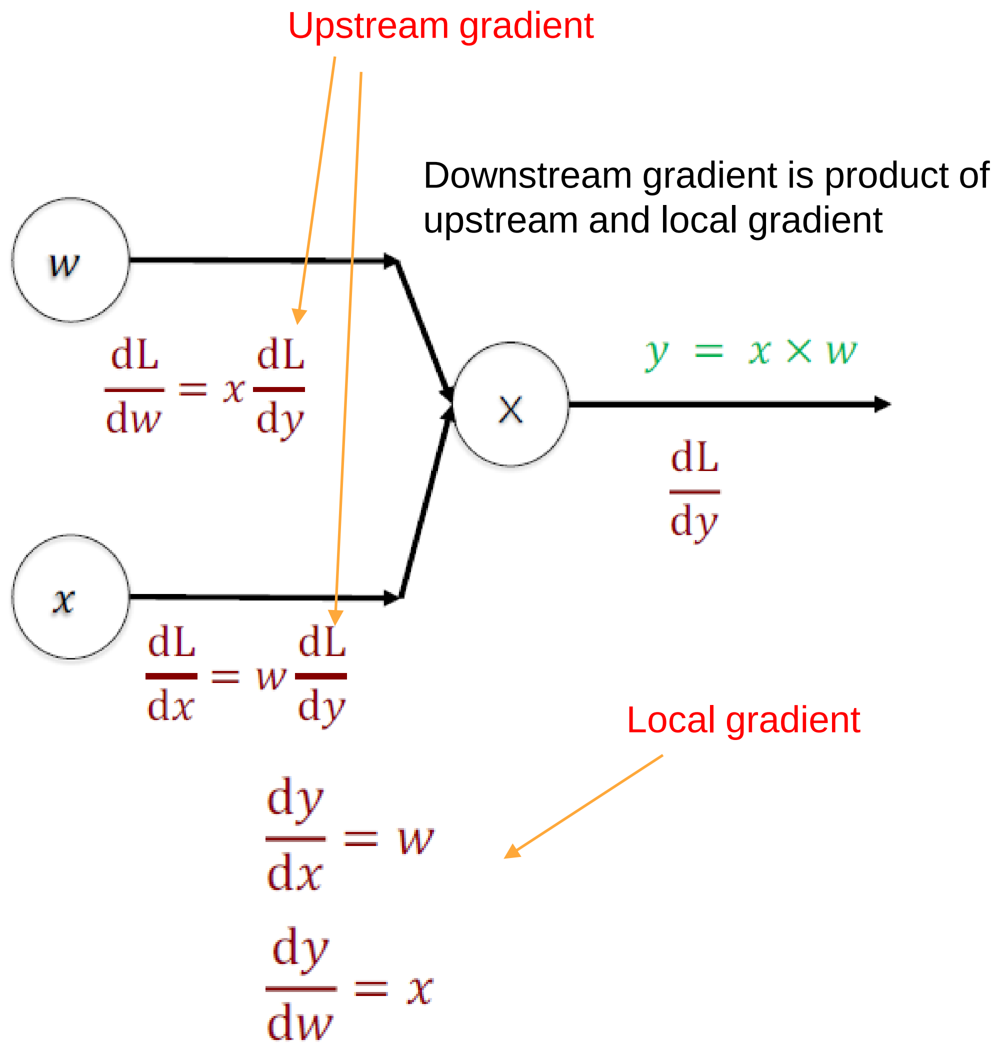




Patterns in gradient flow

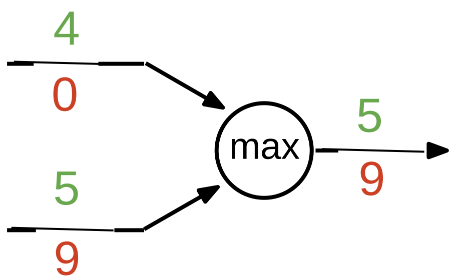
mul gate: “swap multiplier”

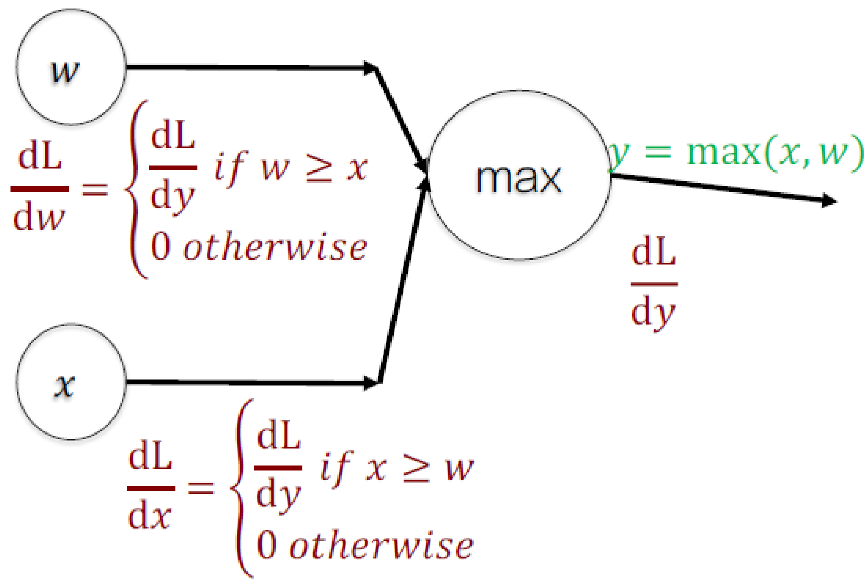




Patterns in gradient flow

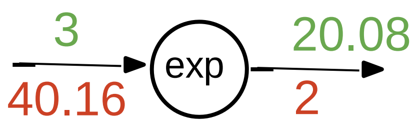
max gate: gradient router





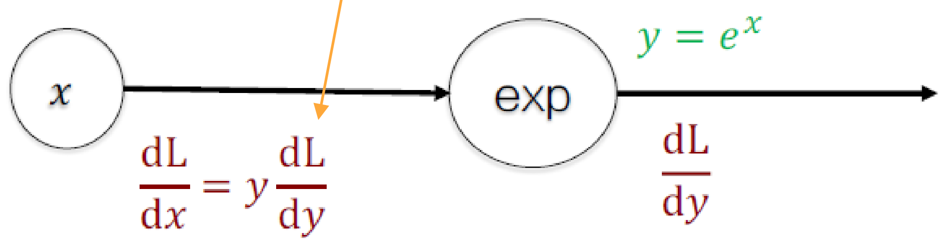
Patterns in gradient flow

exp gate: gradient output multiplier



Upstream gradient

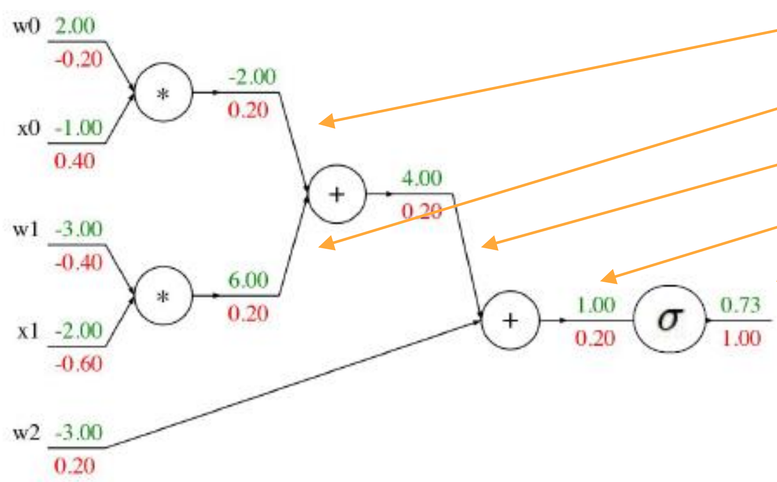
Downstream gradient is product of upstream and local gradient



Local gradient

$$\frac{dy}{dx} = e^x$$

Backprop Implementation: "Flat" code



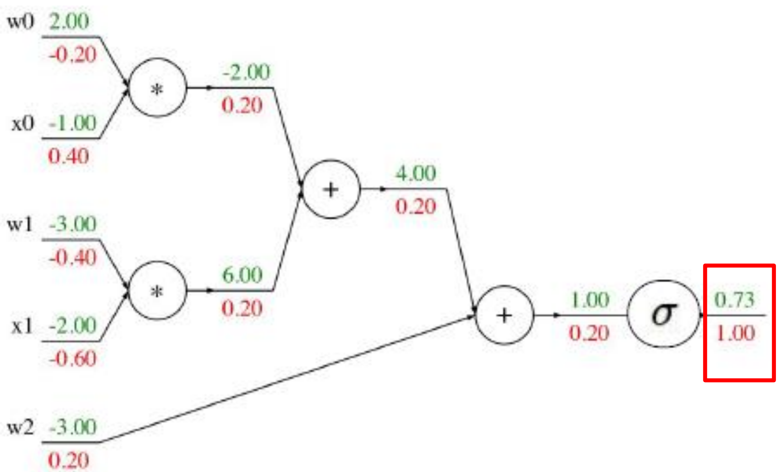
Forward pass:
Compute output

```python
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

```python
    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
```

Backprop Implementation: “Flat” code




```python
def f(w0, x0, w1, x1, w2):
```

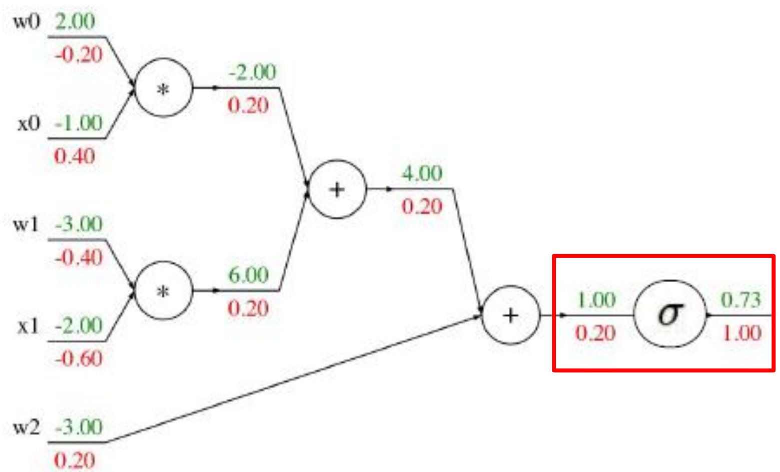
Forward pass:
Compute output

```python
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)
```

Base case

```python
    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
```

Backprop Implementation: "Flat" code



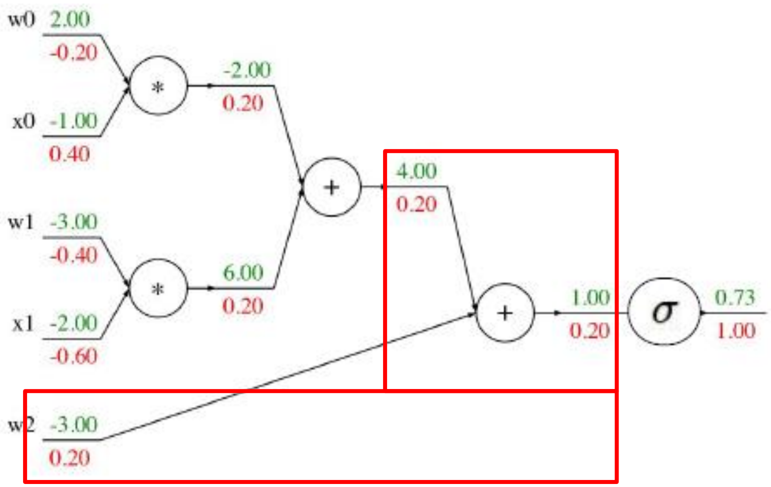
Forward pass:
Compute output

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

Sigmoid

```
  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
```

Backprop Implementation: "Flat" code



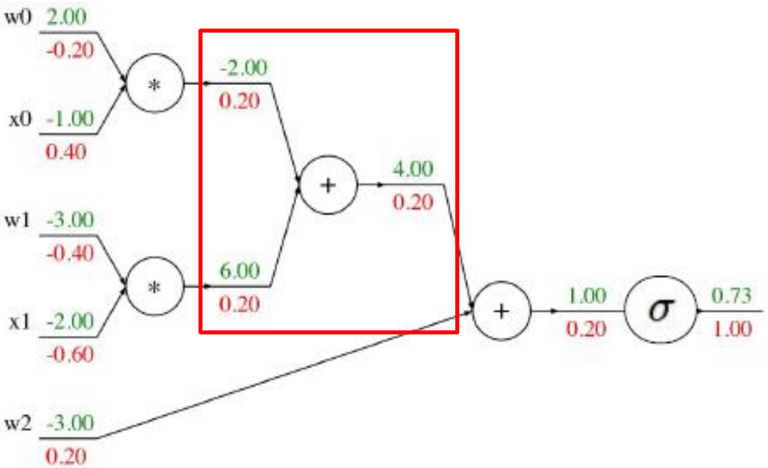
Forward pass:
Compute output

Add gate

Add gate: Gradient distributor

```
def f(w0, x0, w1, x1, w2):

    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)

    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
```

Backprop Implementation: "Flat" code



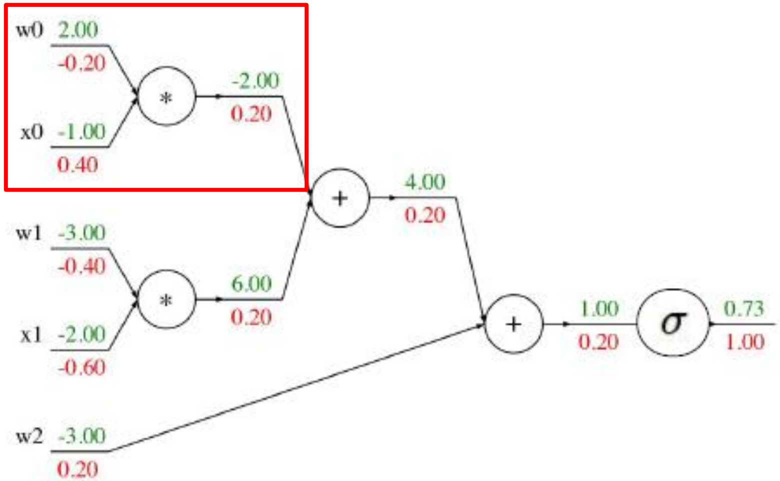
Forward pass:
Compute output

```
def f(w0, x0, w1, x1, w2):

  s0 = w0 * x0
  s1 = w1 * x1
  s2 = s0 + s1
  s3 = s2 + w2
  L = sigmoid(s3)

  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
```

Add gate

Add gate: Gradient distributor

Backprop Implementation: “Flat” code




```
def f(w0, x0, w1, x1, w2):
```

Forward pass:
Compute output

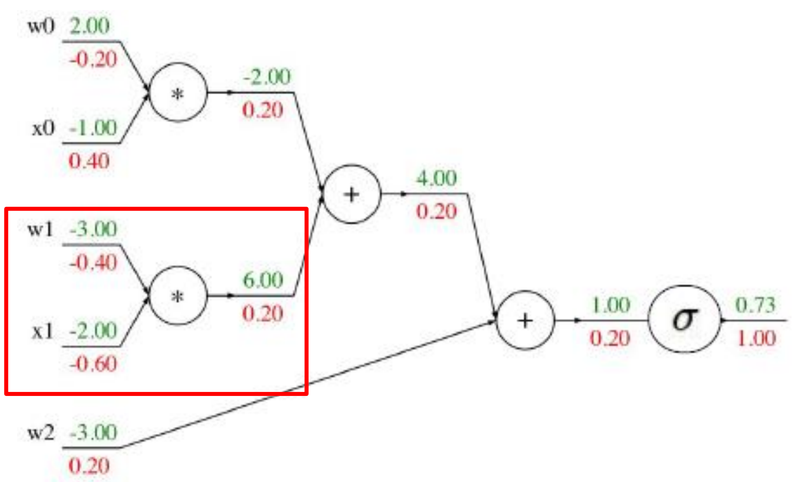
```
s0 = w0 * x0
s1 = w1 * x1
s2 = s0 + s1
s3 = s2 + w2
L = sigmoid(s3)
```

```
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
```

Multiply gate

Mul gate: Gradient
Swap Multiplier

Backprop Implementation: “Flat” code




```
def f(w0, x0, w1, x1, w2):
```

Forward pass:
Compute output

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

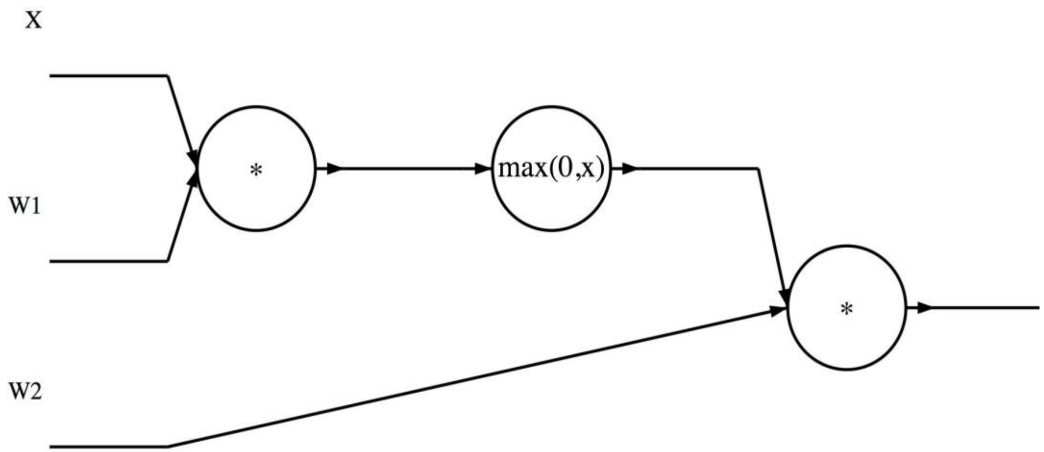
```
grad_L = 1.0
grad_s3 = grad_L * (1 - L) * L
grad_w2 = grad_s3
grad_s2 = grad_s3
grad_s0 = grad_s2
grad_s1 = grad_s2
grad_w1 = grad_s1 * x1
grad_x1 = grad_s1 * w1
grad_w0 = grad_s0 * x0
grad_x0 = grad_s0 * w0
```

Multiply gate

Mul gate: Gradient
Swap Multiplier

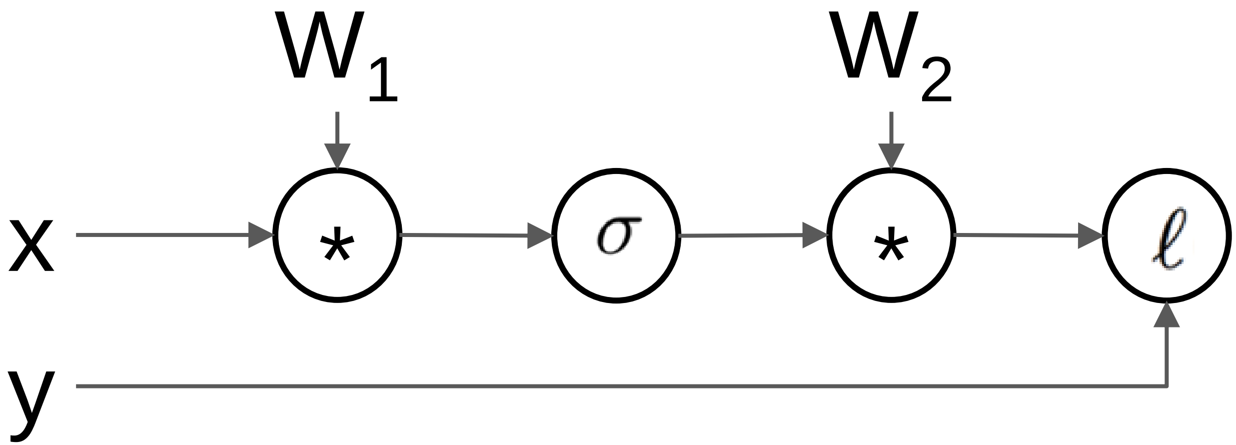
Class Participation

$$z_1 = XW_1$$
$$h_1 = \mathrm{ReLU}(z_1)$$
$$\hat{y} = h_1 W_2$$



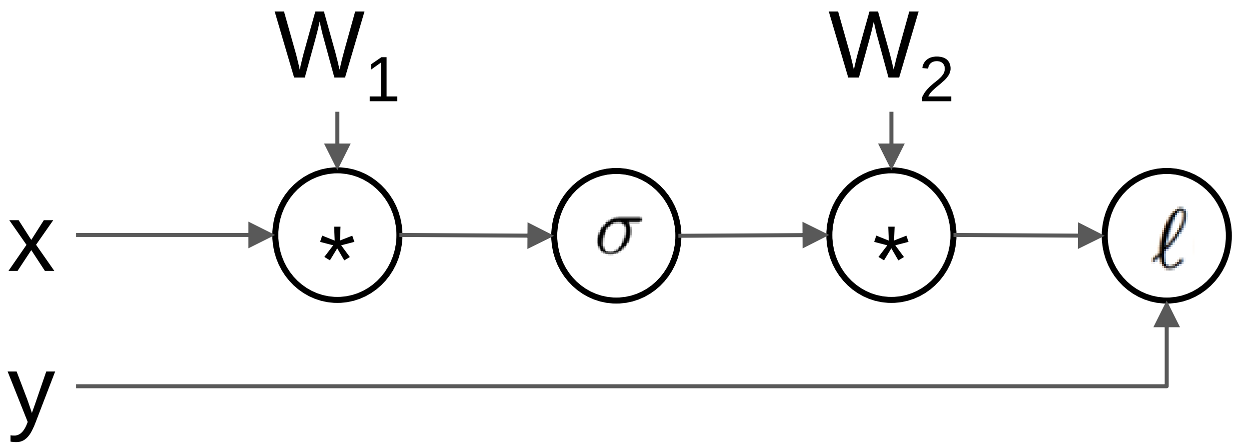
Backpropagation

Computational graph nodes can be vectors or matrices or n-dimensional tensors



Backpropagation

Computational graph nodes can be vectors or matrices or n-dimensional tensors

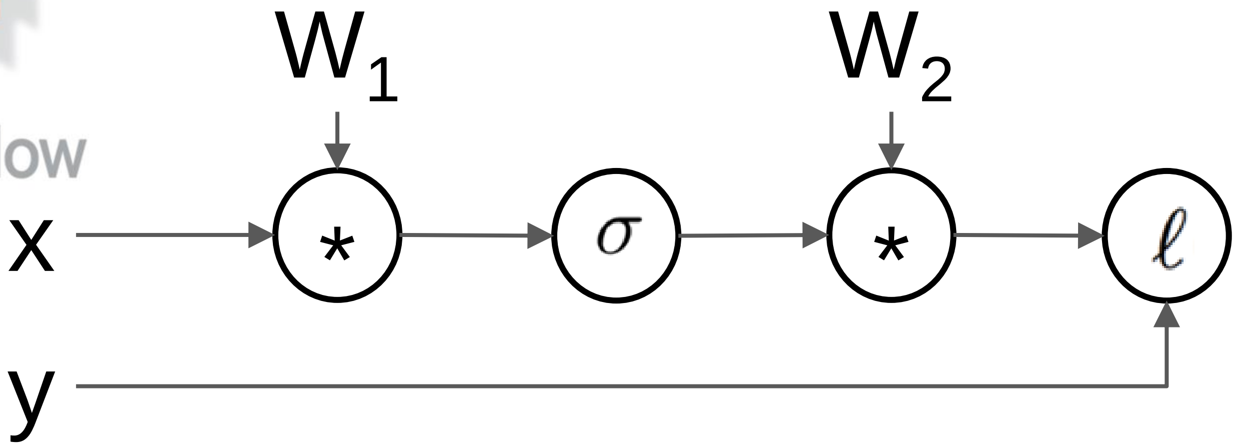


Forward pass: Run graph “forward” to compute loss
Backward pass: Run graph “backward” to compute gradients of loss function with respect to inputs

Easily compute gradients for big, complex models!

Backpropagation

Computational graph nodes can be vectors or matrices or n-dimensional tensors



Forward pass: Run graph “forward” to compute loss
Backward pass: Run graph “backward” to compute gradients of loss function with respect to inputs

Easily compute gradients for big, complex models!

Tensors **flow** along edges in the graph

A two-layer neural network in 25 lines of code

```
import numpy as np

D, H, N = 8, 64, 32
learning_rate = 0.0001

W1 = np.random.randn(D, H)
W2 = np.random.randn(H, D)
for t in xrange(10000):
  x = np.random.randn(N, D)
  y = np.sin(x)

  s = x.dot(W1)
  a = np.maximum(s, 0)
  y_hat = a.dot(W2)

  loss = 0.5 * np.sum((y_hat - y) ** 2.0)

  dy_hat = y_hat - y
  dW2 = a.T.dot(dy_hat)
  da = dy_hat.dot(W2.T)
  ds = (s > 0) * da
  dW1 = x.T.dot(ds)

  W1 -= learning_rate * dW1
  W2 -= learning_rate * dW2
```

A two-layer neural network in 25 lines of code

```
import numpy as np

D, H, N = 8, 64, 32
learning_rate = 0.0001

W1 = np.random.randn(D, H)
W2 = np.random.randn(H, D)
for t in xrange(10000):
  x = np.random.randn(N, D)
  y = np.sin(x)

  s = x.dot(W1)
  a = np.maximum(s, 0)
  y_hat = a.dot(W2)

  loss = 0.5 * np.sum((y_hat - y) ** 2.0)

  dy_hat = y_hat - y
  dW2 = a.T.dot(dy_hat)
  da = dy_hat.dot(W2.T)
  ds = (s > 0) * da
  dW1 = x.T.dot(ds)

  W1 -= learning_rate * dW1
  W2 -= learning_rate * dW2
```

Randomly initialize weights

A two-layer neural network in 25 lines of code

```python
import numpy as np

D, H, N = 8, 64, 32
learning_rate = 0.0001

W1 = np.random.randn(D, H)
W2 = np.random.randn(H, D)
for t in xrange(10000):
  x = np.random.randn(N, D)
  y = np.sin(x)

  s = x.dot(W1)
  a = np.maximum(s, 0)
  y_hat = a.dot(W2)

  loss = 0.5 * np.sum((y_hat - y) ** 2.0)

  dy_hat = y_hat - y
  dW2 = a.T.dot(dy_hat)
  da = dy_hat.dot(W2.T)
  ds = (s > 0) * da
  dW1 = x.T.dot(ds)

  W1 -= learning_rate * dW1
  W2 -= learning_rate * dW2
```

Get a batch of (random) data

A two-layer neural network in 25 lines of code

```python
import numpy as np

D, H, N = 8, 64, 32
learning_rate = 0.0001

W1 = np.random.randn(D, H)
W2 = np.random.randn(H, D)
for t in xrange(10000):
  x = np.random.randn(N, D)
  y = np.sin(x)

  s = x.dot(W1)
  a = np.maximum(s, 0)
  y_hat = a.dot(W2)

  loss = 0.5 * np.sum((y_hat - y) ** 2.0)

  dy_hat = y_hat - y
  dW2 = a.T.dot(dy_hat)
  da = dy_hat.dot(W2.T)
  ds = (s > 0) * da
  dW1 = x.T.dot(ds)

  W1 -= learning_rate * dW1
  W2 -= learning_rate * dW2
```

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

ReLU nonlinearity

Forward pass: compute loss

$$\ell(\hat{y}, y) = \frac{1}{2}\|\hat{y} - y\|_2^2$$

Loss is Euclidean distance

A two-layer neural network in 25 lines of code

```
import numpy as np

D, H, N = 8, 64, 32
learning_rate = 0.0001

W1 = np.random.randn(D, H)
W2 = np.random.randn(H, D)
for t in xrange(10000):
  x = np.random.randn(N, D)
  y = np.sin(x)

  s = x.dot(W1)
  a = np.maximum(s, 0)
  y_hat = a.dot(W2)

  loss = 0.5 * np.sum((y_hat - y) ** 2.0)

  dy_hat = y_hat - y
  dW2 = a.T.dot(dy_hat)
  da = dy_hat.dot(W2.T)
  ds = (s > 0) * da
  dW1 = x.T.dot(ds)

  W1 -= learning_rate * dW1
  W2 -= learning_rate * dW2
```

Backward pass: compute gradients

A two-layer neural network in 25 lines of code

```python
import numpy as np

D, H, N = 8, 64, 32
learning_rate = 0.0001

W1 = np.random.randn(D, H)
W2 = np.random.randn(H, D)
for t in xrange(10000):
  x = np.random.randn(N, D)
  y = np.sin(x)

  s = x.dot(W1)
  a = np.maximum(s, 0)
  y_hat = a.dot(W2)

  loss = 0.5 * np.sum((y_hat - y) ** 2.0)

  dy_hat = y_hat - y
  dW2 = a.T.dot(dy_hat)
  da = dy_hat.dot(W2.T)
  ds = (s > 0) * da
  dW1 = x.T.dot(ds)

  W1 -= learning_rate * dW1
  W2 -= learning_rate * dW2
```

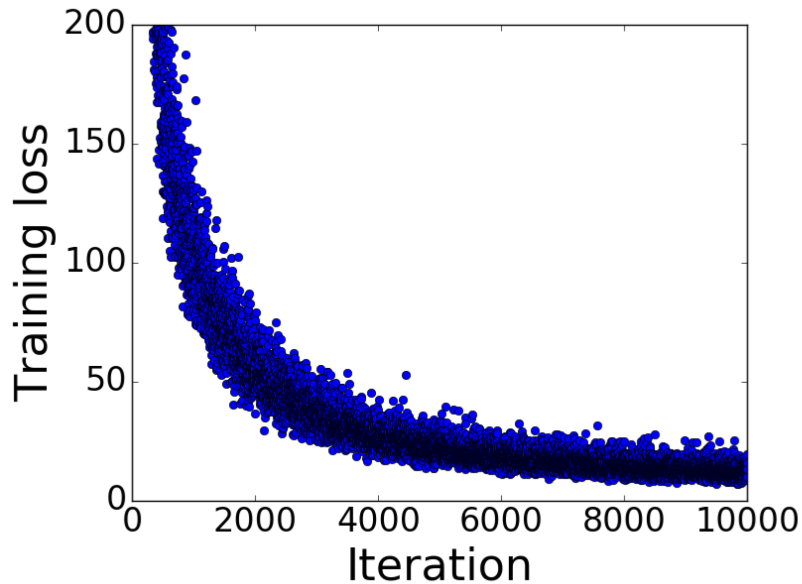
$$W := W - \alpha \frac{dL}{dW}$$

Update weights

A two-layer neural network in 25 lines of code

```python
import numpy as np

D, H, N = 8, 64, 32
learning_rate = 0.0001

W1 = np.random.randn(D, H)
W2 = np.random.randn(H, D)
for t in xrange(10000):
  x = np.random.randn(N, D)
  y = np.sin(x)

  s = x.dot(W1)
  a = np.maximum(s, 0)
  y_hat = a.dot(W2)

  loss = 0.5 * np.sum((y_hat - y) ** 2.0)

  dy_hat = y_hat - y
  dW2 = a.T.dot(dy_hat)
  da = dy_hat.dot(W2.T)
  ds = (s > 0) * da
  dW1 = x.T.dot(ds)

  W1 -= learning_rate * dW1
  W2 -= learning_rate * dW2
```

When you run this code: loss goes down!



For vector valued functions: Vector derivatives

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

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

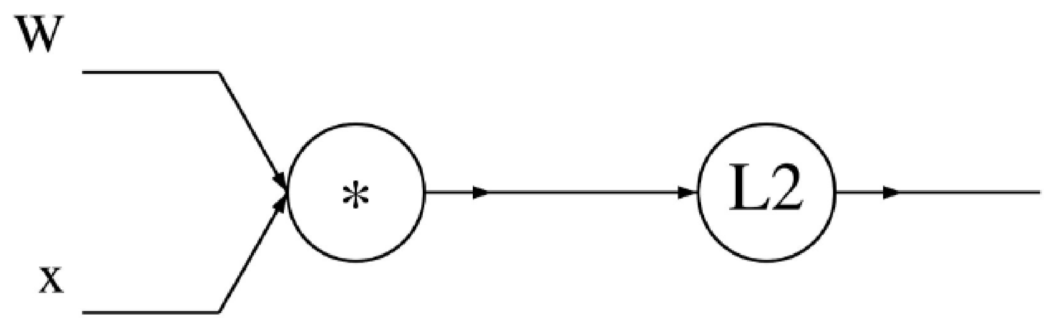
Derivative is **Jacobian**:

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

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

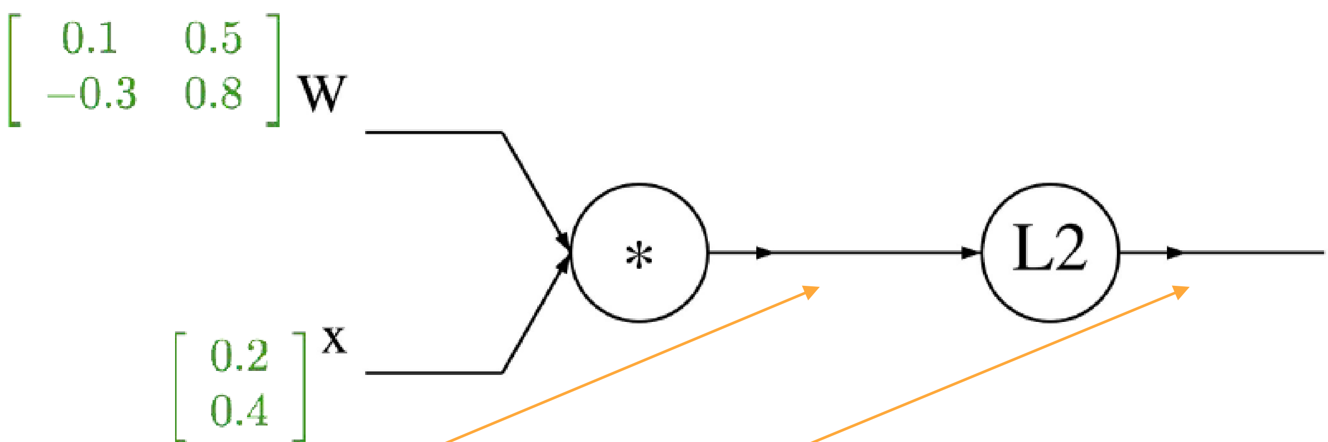
A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

$\in \mathbb{R}^n \quad \in \mathbb{R}^{n \times n}$



A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

$\in \mathbb{R}^n \in \mathbb{R}^{n \times n}$

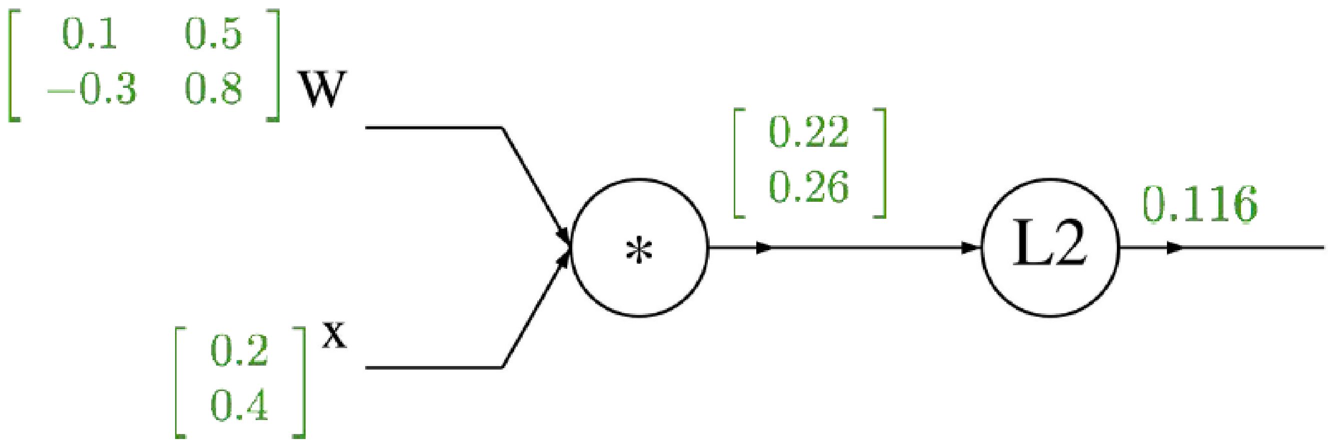


$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

$\in \mathbb{R}^n \quad \in \mathbb{R}^{n \times n}$

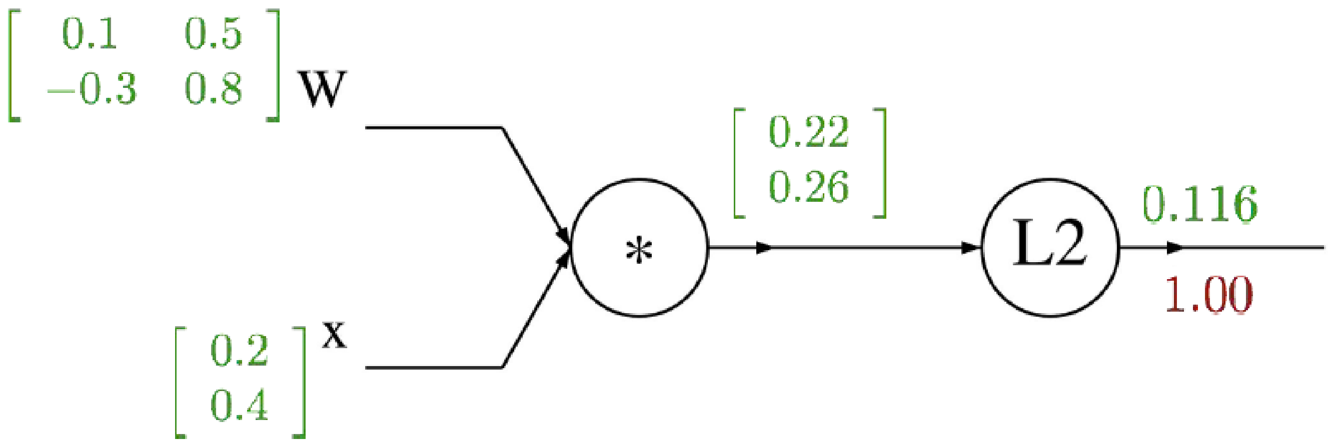


$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

$\in \mathbb{R}^n \quad \in \mathbb{R}^{n \times n}$

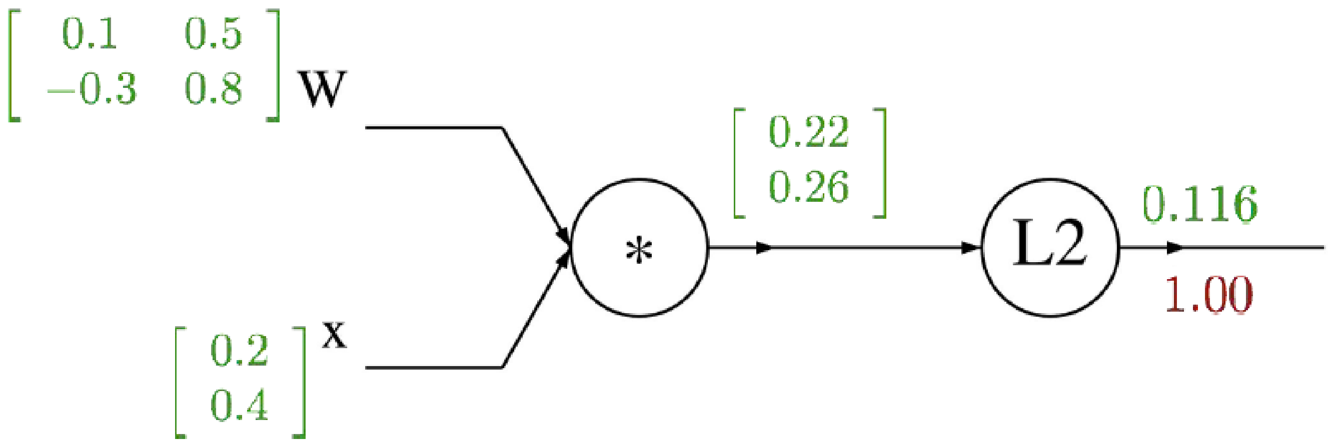


$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

$\in \mathbb{R}^n$ $\in \mathbb{R}^{n \times n}$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

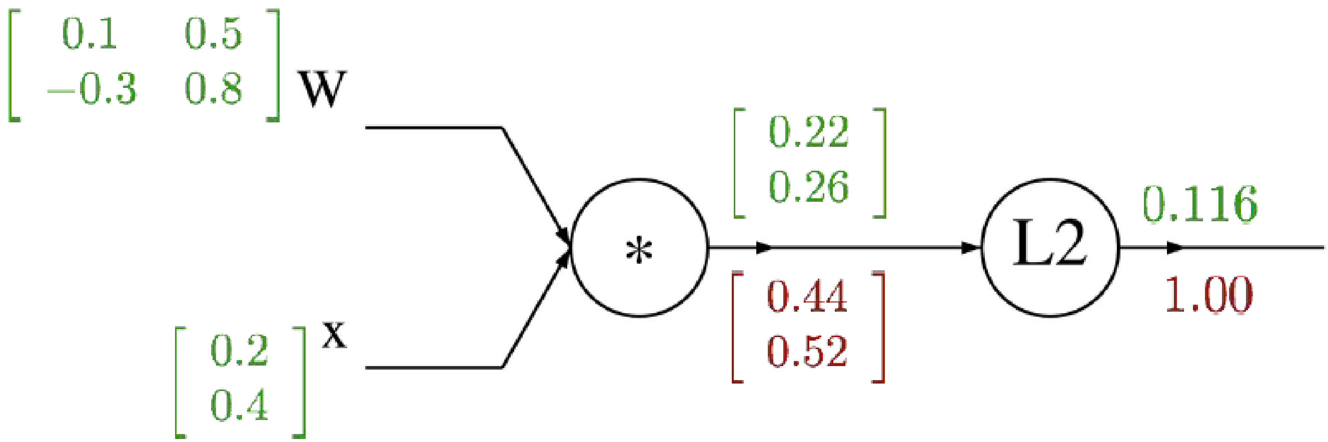
$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial f}{\partial q_i} = 2q_i$$

$$\boxed{\nabla_q f = 2q}$$

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

$\in \mathbb{R}^n \quad \in \mathbb{R}^{n \times n}$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

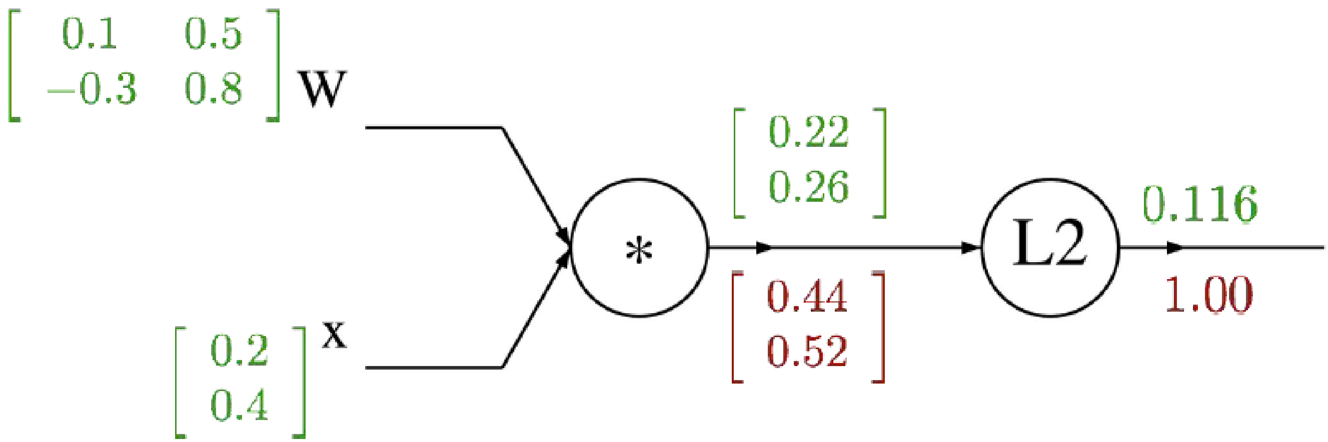
$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial f}{\partial q_i} = 2q_i$$

$$\boxed{\nabla_q f = 2q}$$

A vectorized example: $f(x,W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

$\in \mathbb{R}^n \quad \in \mathbb{R}^{n \times n}$



$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

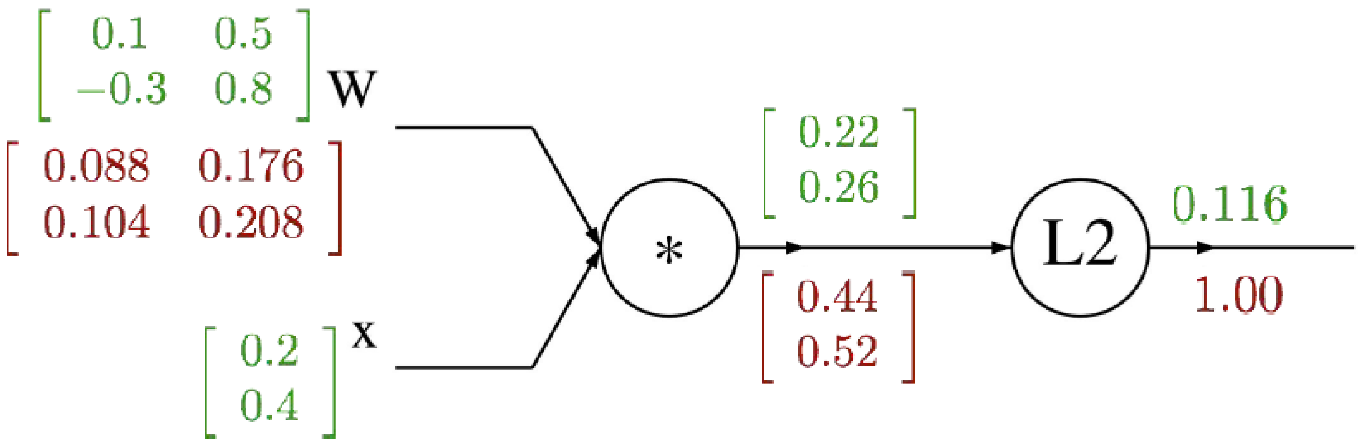
$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\begin{aligned} \frac{\partial f}{\partial W_{i,j}} &= \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}} \\ &= \sum_k (2q_k)(\mathbf{1}_{k=i} x_j) \\ &= 2q_i x_j \end{aligned}$$

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

$\in \mathbb{R}^n \in \mathbb{R}^{n \times n}$



$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

$$\frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}}$$

$$= \sum_k (2q_k)(\mathbf{1}_{k=i} x_j)$$

$$= 2q_i x_j$$

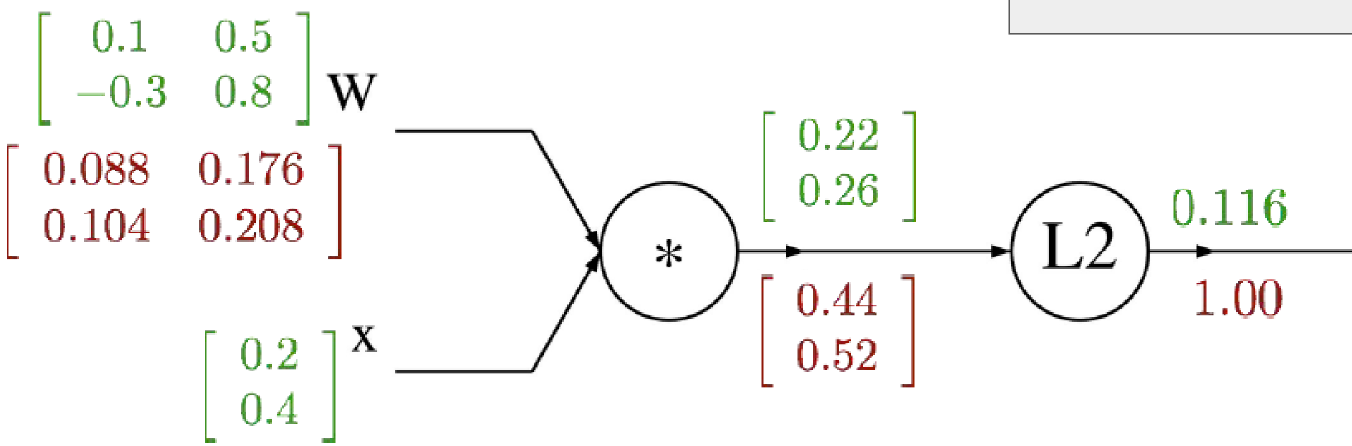
$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

$\in \mathbb{R}^n \quad \in \mathbb{R}^{n \times n}$

$$\boxed{\nabla_W f = 2q \cdot x^T}$$



$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

$$\frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}}$$

$$= \sum_k (2q_k)(\mathbf{1}_{k=i} x_j)$$

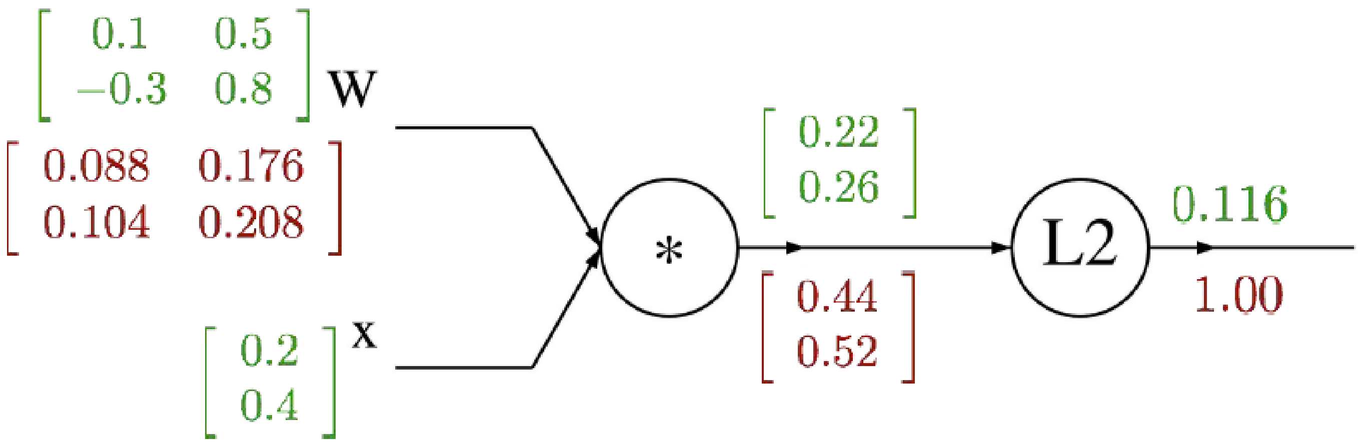
$$= 2q_i x_j$$

$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

$\in \mathbb{R}^n \quad \in \mathbb{R}^{n \times n}$



$$\frac{\partial q_k}{\partial x_i} = W_{k,i}$$

$$\frac{\partial f}{\partial x_i} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial x_i} = \sum_k 2 q_k W_{k,i}$$

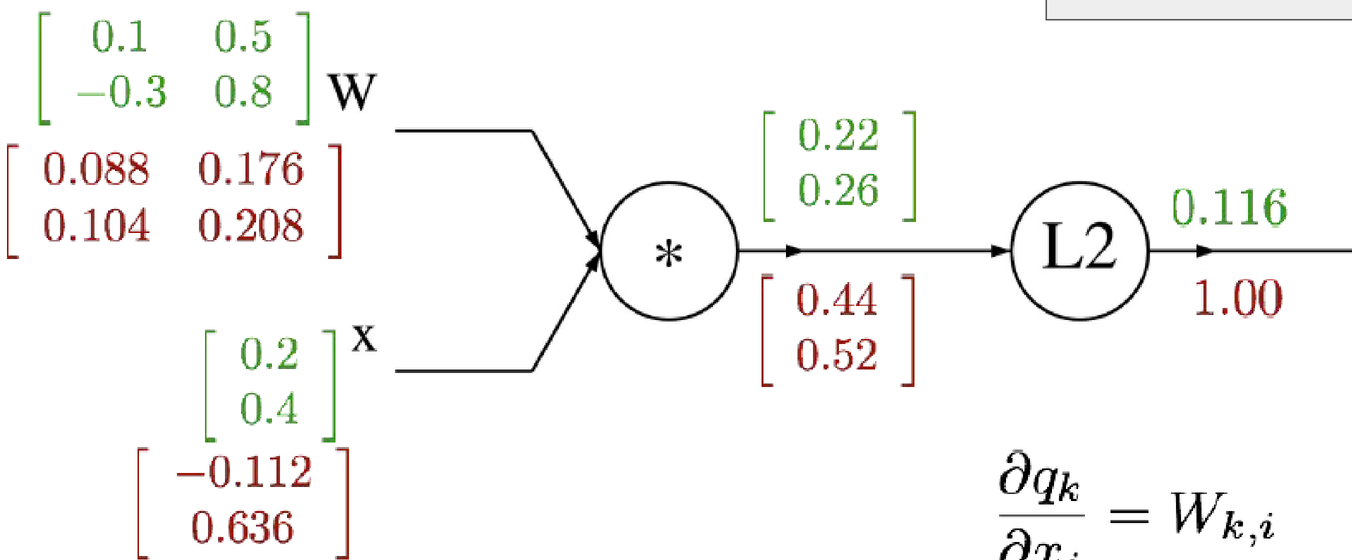
$$q = W \cdot x = \begin{pmatrix} W_{1,1} x_1 + \cdots + W_{1,n} x_n \\ \vdots \\ W_{n,1} x_1 + \cdots + W_{n,n} x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

$x \in \mathbb{R}^n$, $W \in \mathbb{R}^{n \times n}$

$$\boxed{\nabla_x f = 2W^T \cdot q}$$



$$\frac{\partial q_k}{\partial x_i} = W_{k,i}$$

$$\frac{\partial f}{\partial x_i} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial x_i} = \sum_k 2 q_k W_{k,i}$$

$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

Remember the dimensions of a variable and its gradients have to be the same.

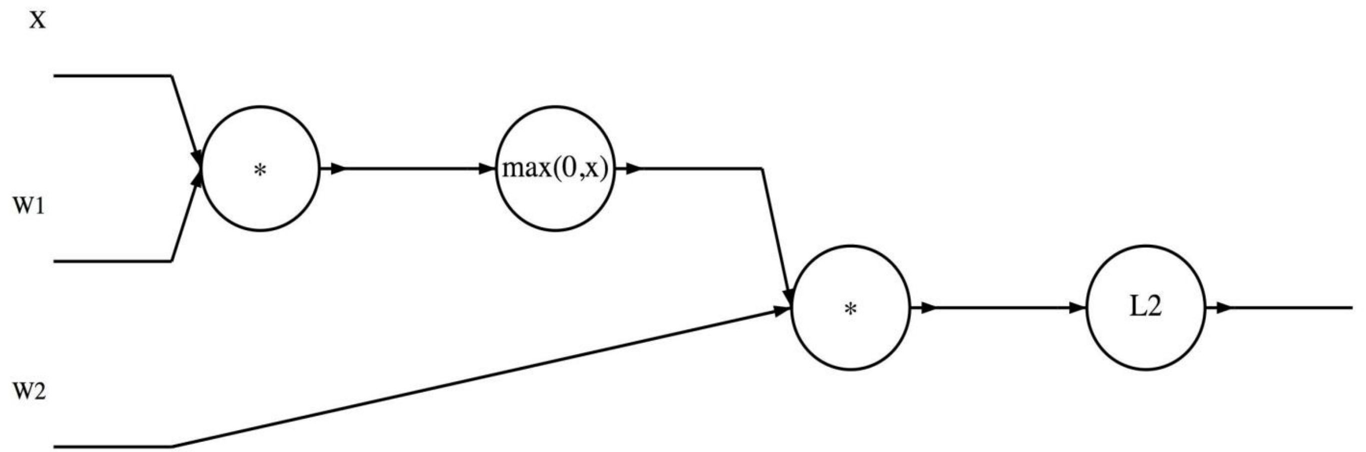
Class Participation:

$$z_1 = XW_1$$

$$h_1 = \mathrm{ReLU}(z_1)$$

$$\hat{y} = h_1 W_2$$

$$L = ||\hat{y}||_2^2$$



Now, what Deep Learning is?

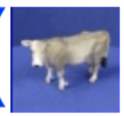
- Learning hierarchical representations from data
- End-to-end learning: raw inputs to predictions
- Can use a small set of simple tools to solve many problems
- Has led to rapid progress on many problems
- (**Very loosely!**) Inspired by the brain

Recall: Image Classification

Write a function that maps **images** to **labels**
(also called **object recognition**)

f() = “apple”

f() = “tomato”

f() = “cow”

Dataset: ETH-80, by B. Leibe; Slide credit: L. Lazebnik

```
def predict(image):
    # ????
    return class_label
```

No obvious way to
implement this function!

Recall: Image Classification

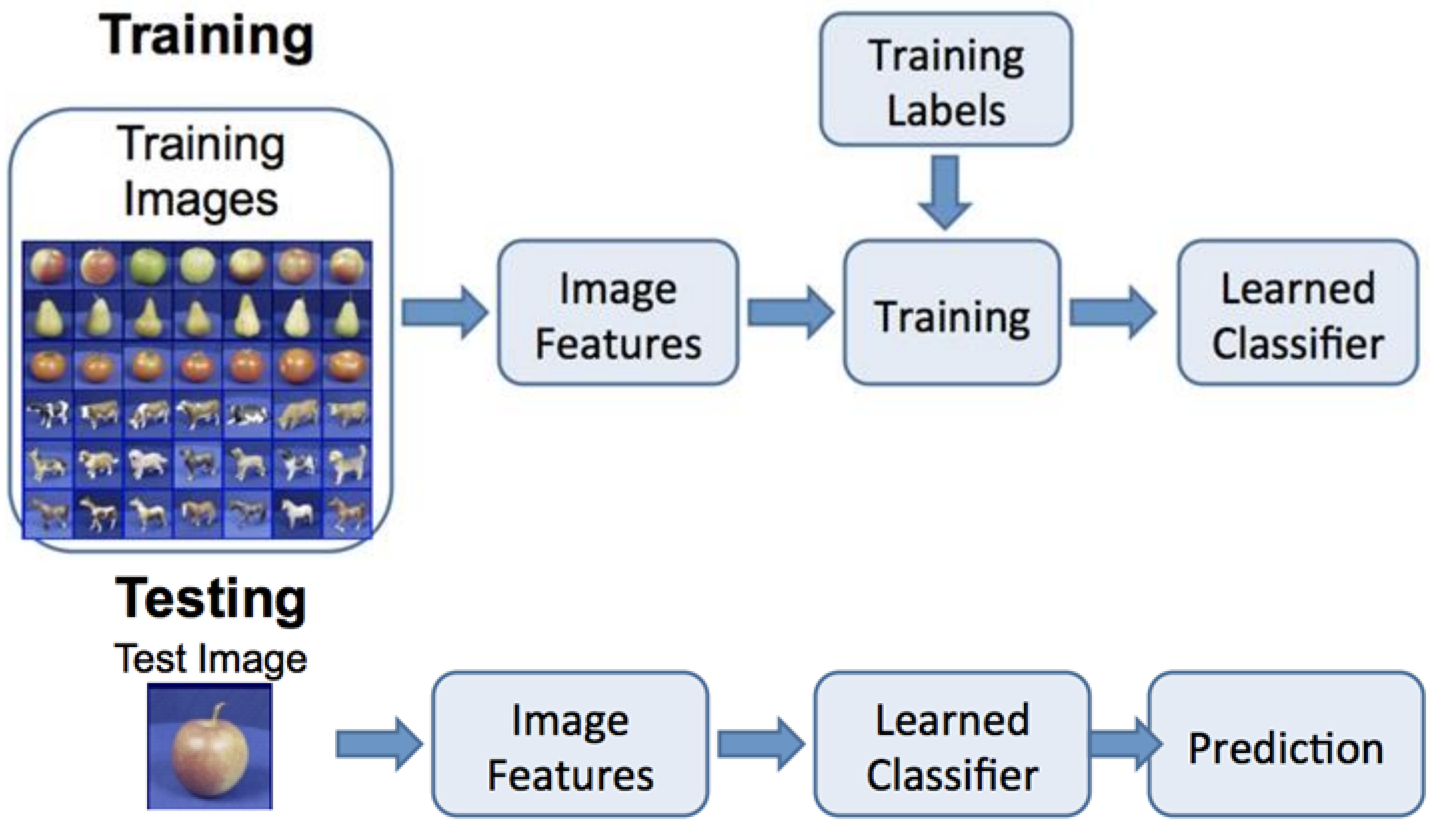
Data-driven approach:

1. Collect a dataset of images and labels
2. Use Machine Learning to train an image classifier
3. Evaluate the classifier on a withheld set of test images

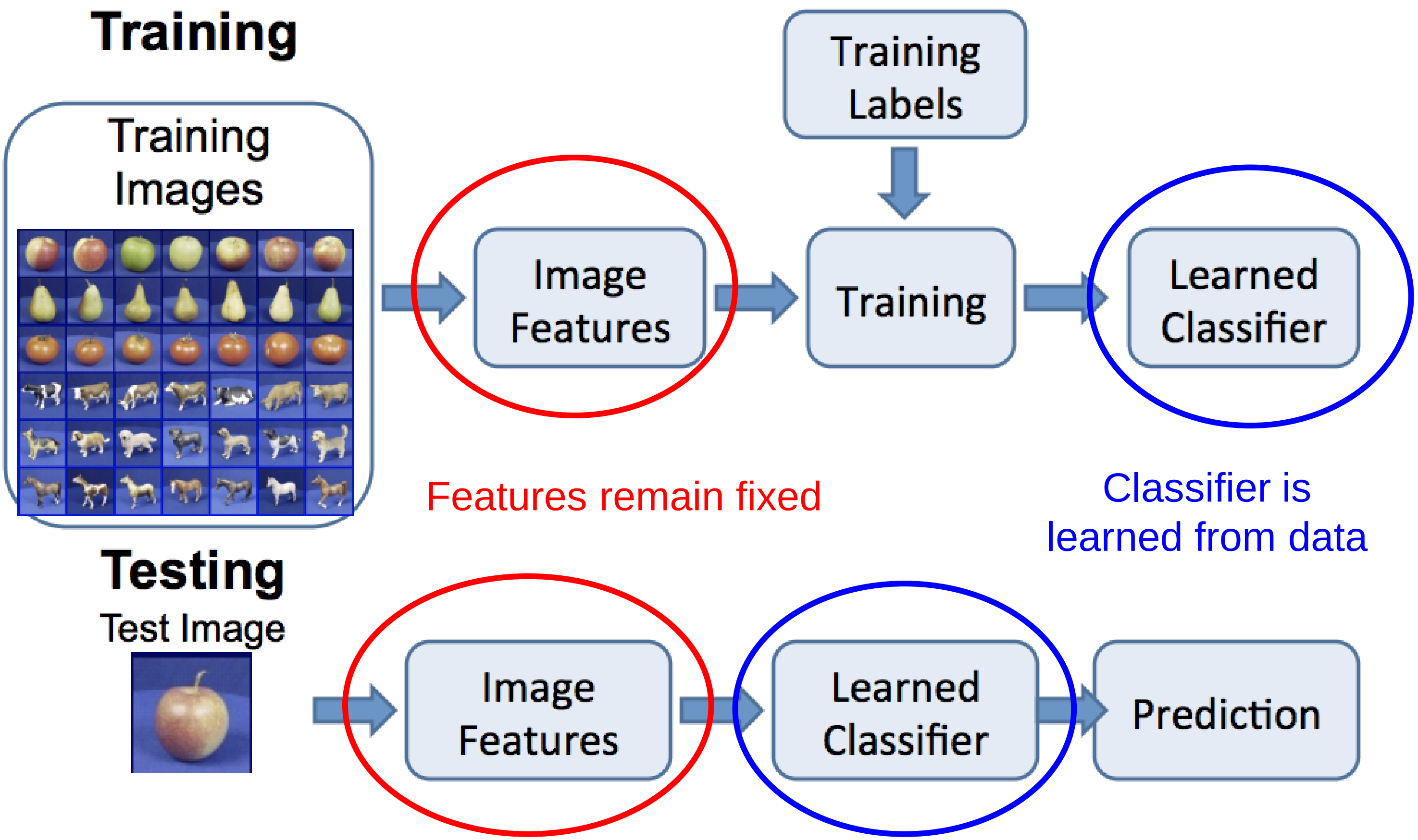
```python
def train(train_images, train_labels):
  # build a model for images -> labels...
  return model

def predict(model, test_images):
  # predict test_labels using the model...
  return test_labels
```

Example training set

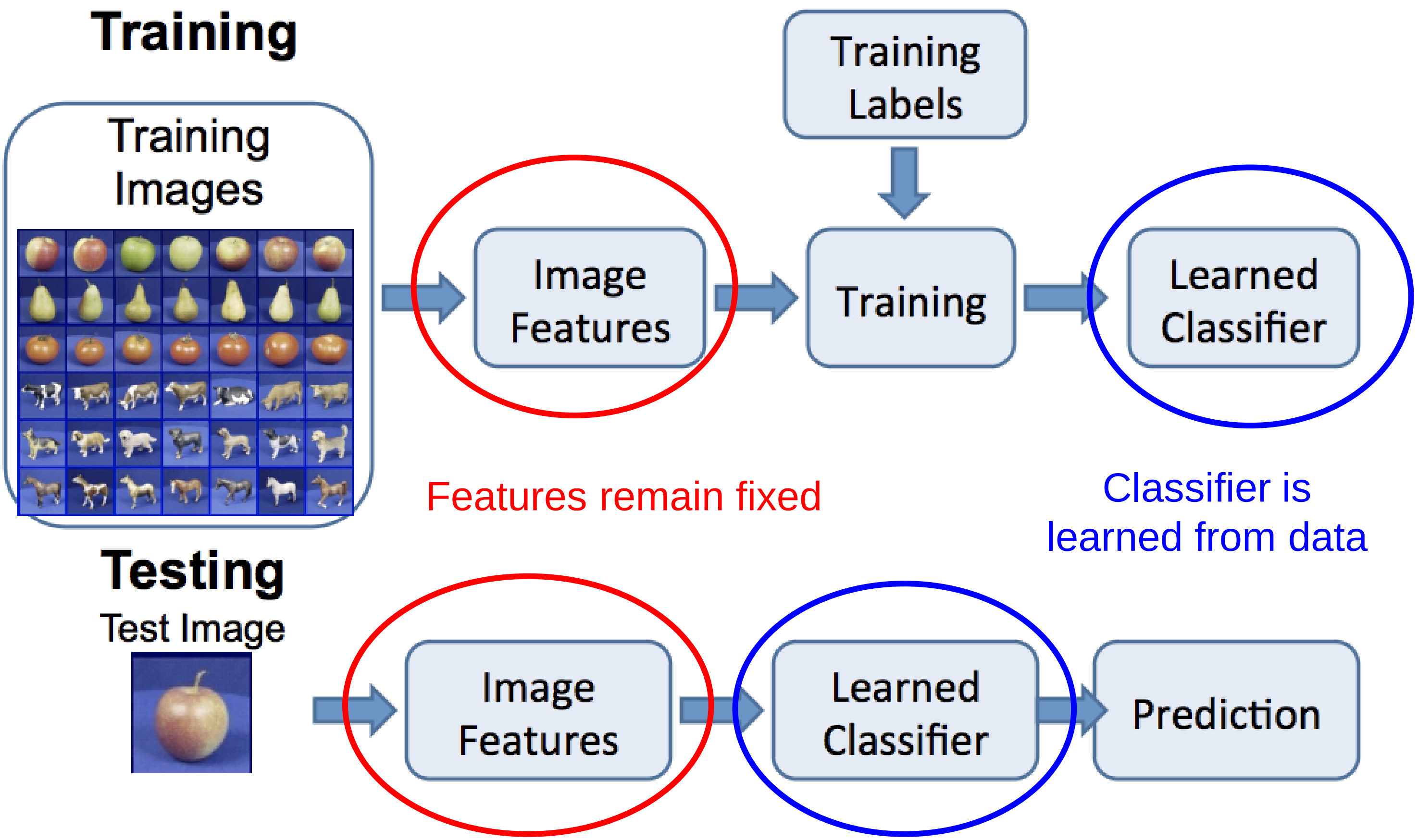
Recall: Image Classification



Recall: Image Classification

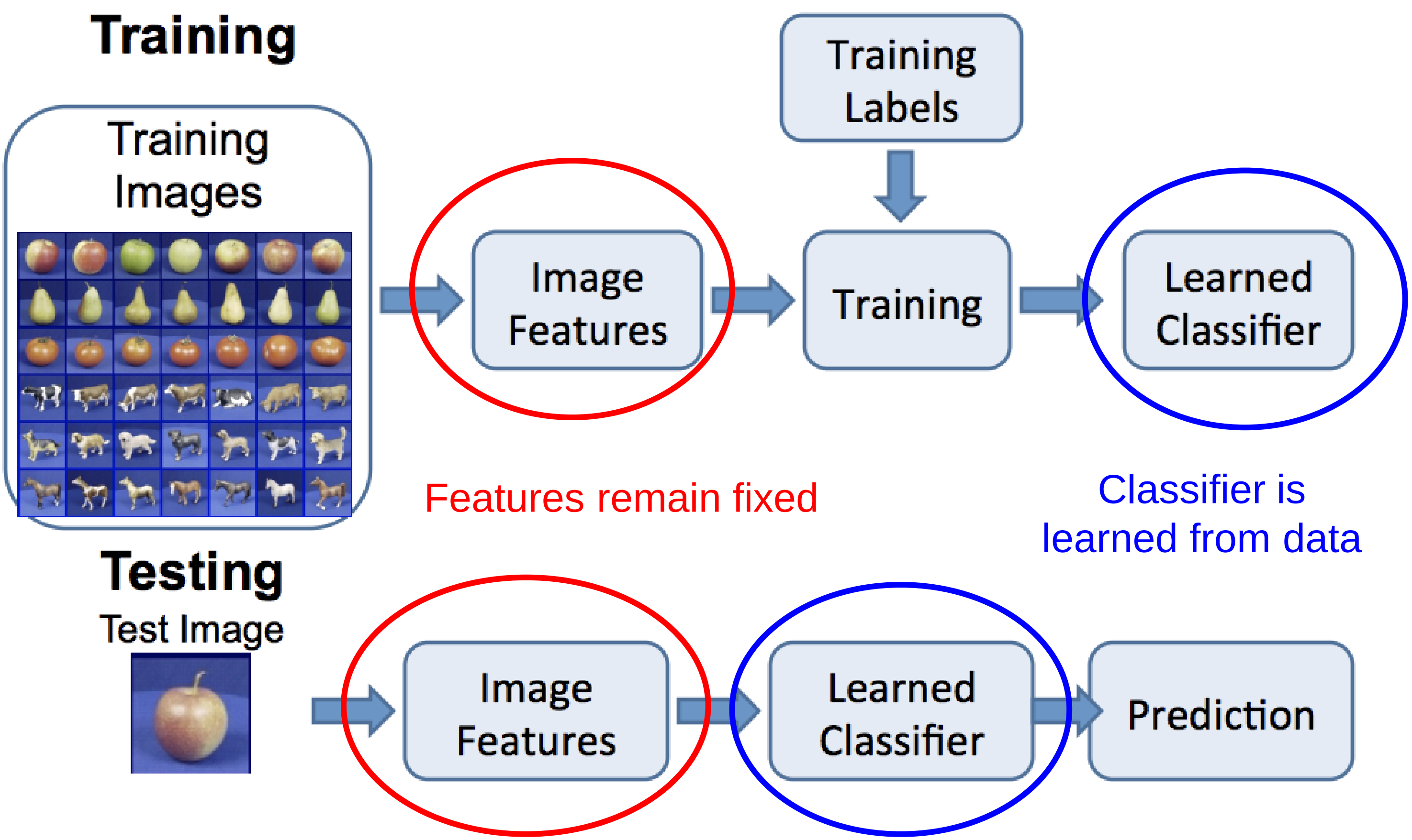


Recall: Image Classification



Problem:
How do we know which features to use? We may need different features for each problem!

Recall: Image Classification



Problem:
How do we know which features to use? We may need different features for each problem!

Solution:
Learn the features jointly with the classifier!

Image Classification: Feature Learning

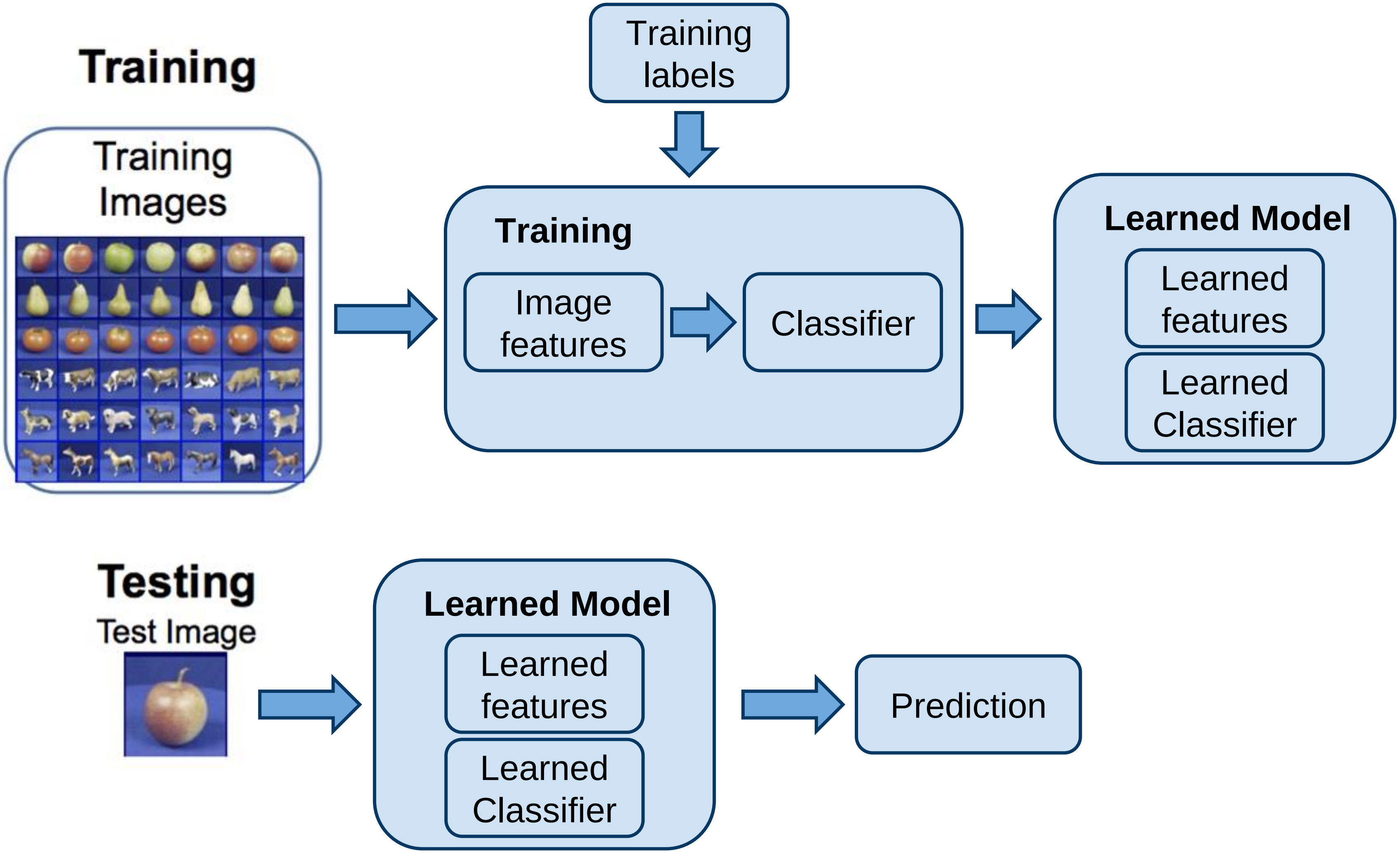
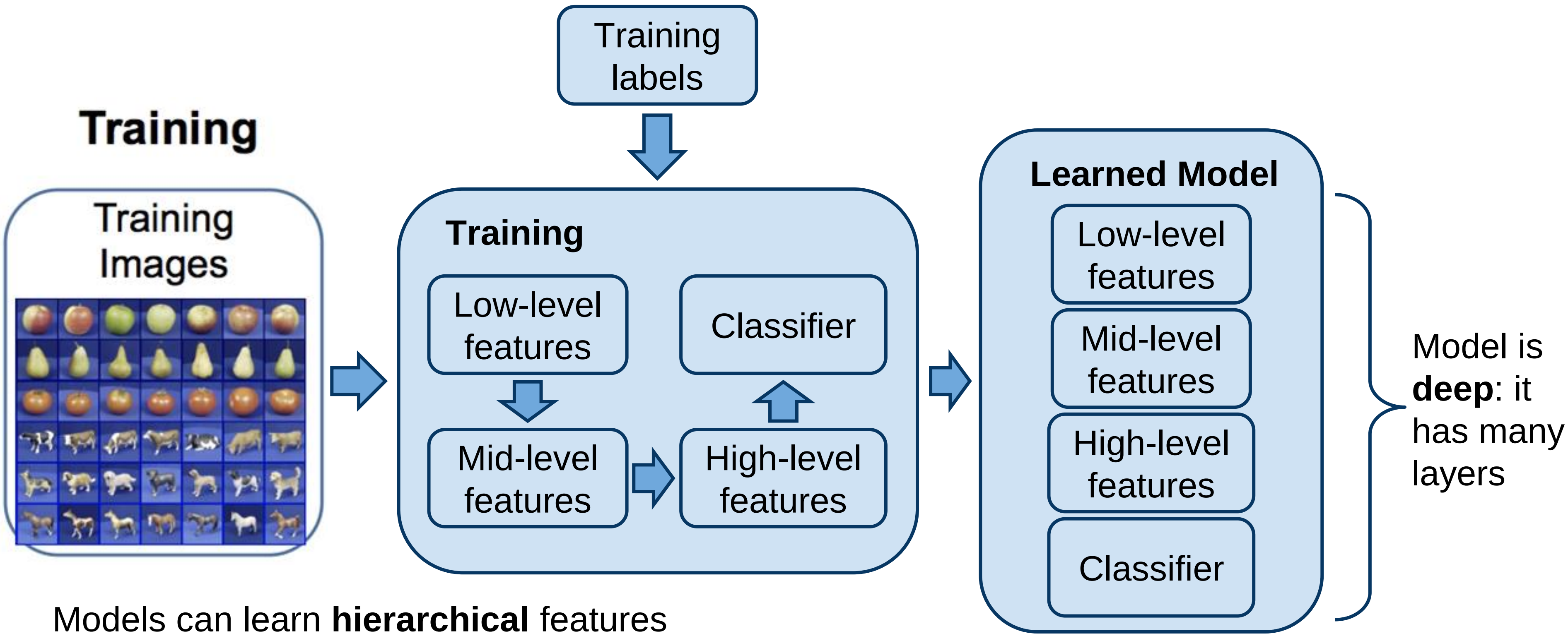
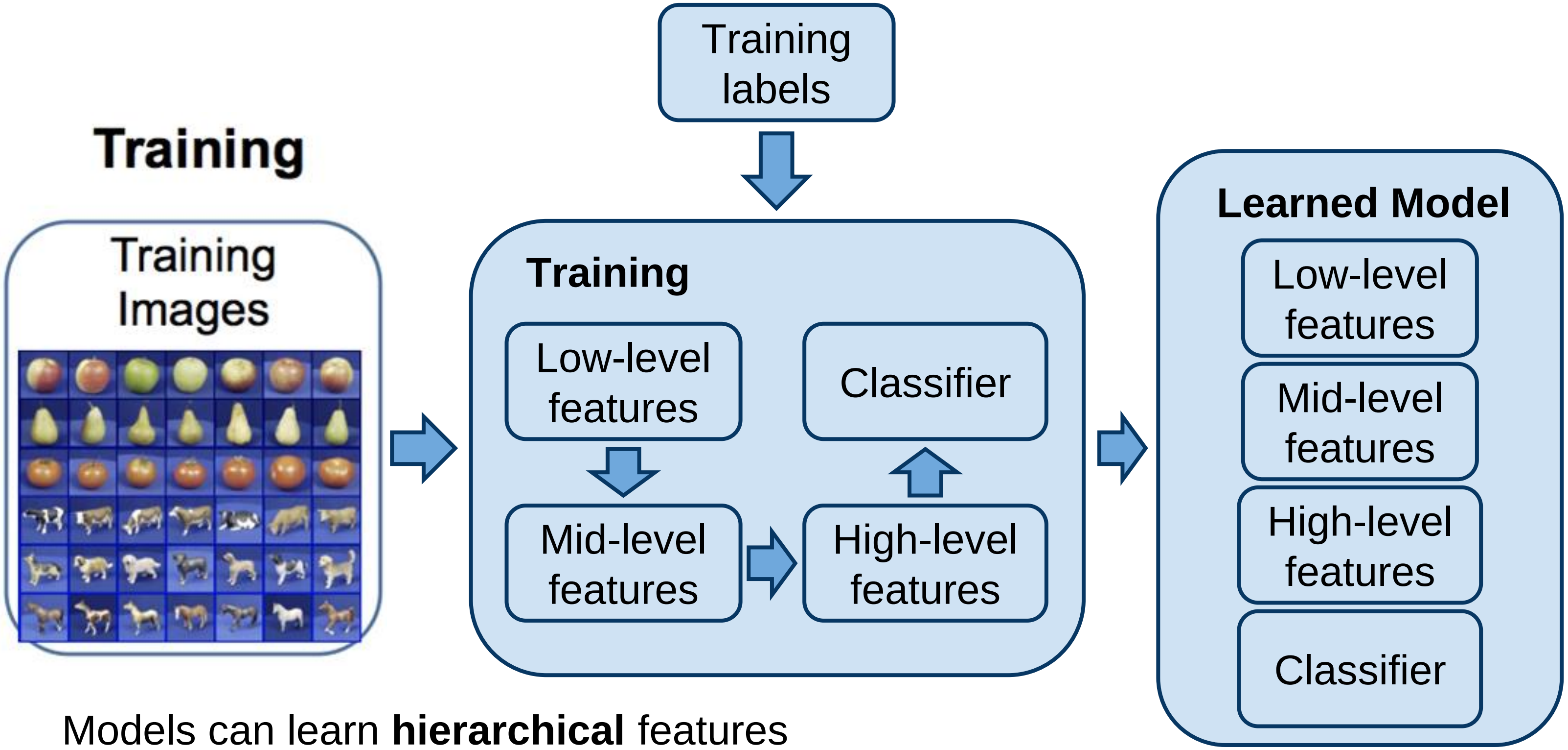


Image Classification: Deep Learning



Models can learn **hierarchical** features

Image Classification: Deep Learning



Model is **deep**: it has many layers

Models can learn **hierarchical** features