

# Deep Learning

Week - 4

# Announcement

- CLab-1 Due 23:59 this Friday
- CLab-2 will be released by early Monday next week.
- Move our lectures to Melville Hall in Weak 5.
- Regarding the high pitch noisy: LSSG team will look into this.

# Announcement

- Pass on the following info to students if they are using Safari to play the recordings:

"This message is to let you know that LSSG has recently received user reports about a loading issue of Echo recording in Safari.

- Issue description:

Users may experience lag, stuttering, or whitescreen during playback on products using iOS (such as iPhones and iPads) and some desktop devices running the Safari browser.

- Current status:

The issue has been acknowledged by Echo, and they are actively working on a solution.

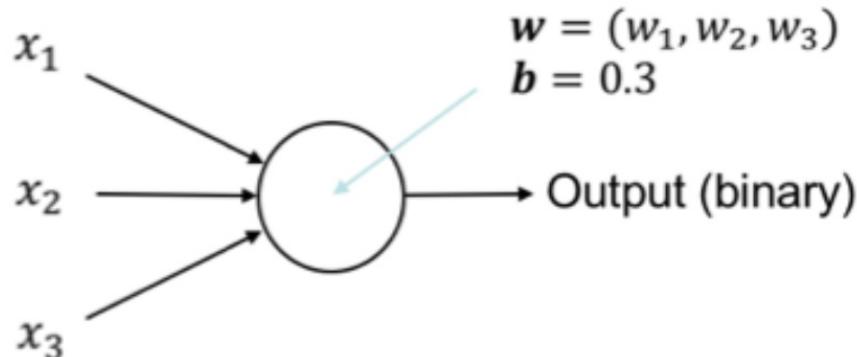
Workaround:

- While awaiting a fix, we recommend using the Firefox browser when attempting to view and load Echo recordings

# Fundamentals for Artificial Intelligence network

# Recall: Perceptron as linear classifier

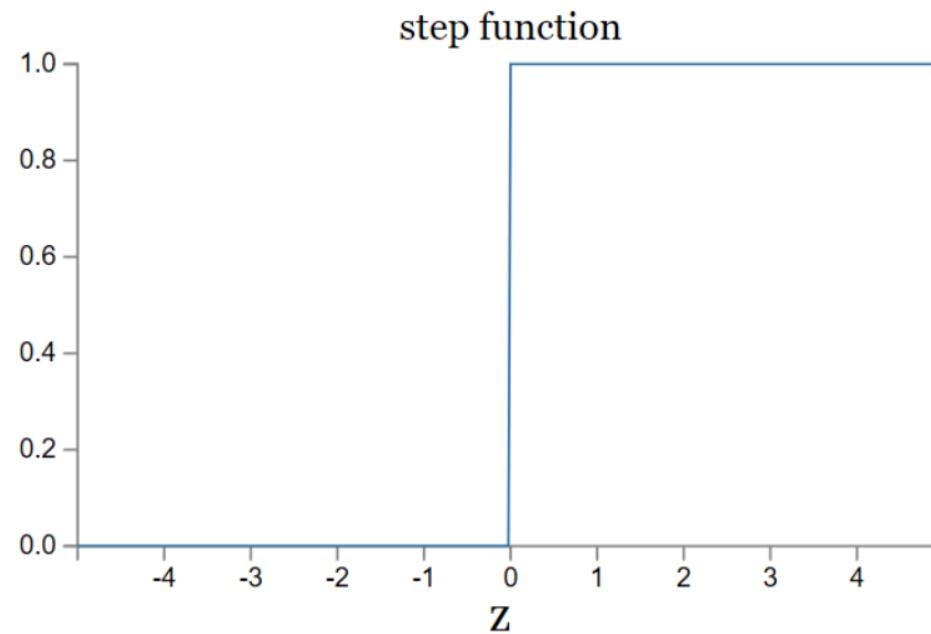
- Basic building block for composition is **perceptron** (Rosenblatt c.1960)
- Linear neuron classifier – vector of weights  $w$  and a ‘bias’  $b$



$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases} \quad w \cdot x \equiv \sum_j w_j x_j$$

# Perceptron's activation function

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$



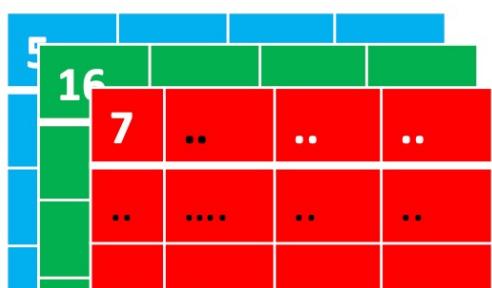
In the context of neural networks, a perceptron is an artificial neuron using the step function as the activation function.

# Binary Classification Example

- Is it a Kangaroo?



{0,1}  
1 – kangaroo  
0 – not a kangaroo



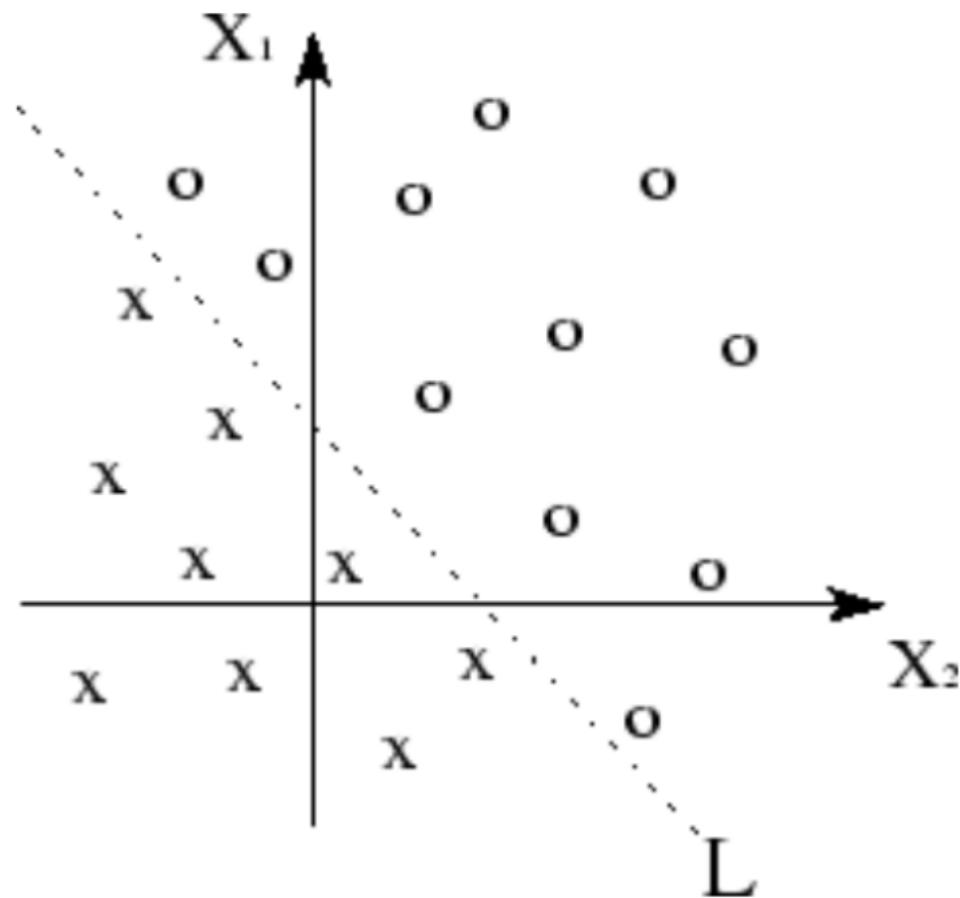
$$X = \begin{bmatrix} 5 \\ \vdots \\ 16 \\ \vdots \\ 7 \\ \vdots \end{bmatrix}$$

X Dim is  $n_x \times m \times 3$   
 $N_x$  is num features

Y dim is 1  
 $X \rightarrow Y$

Credit: Nick Barnes

# Linear Classifier



# Logistic Regression

- Logistic regression (LR) is actually a classification method.
- LR introduces an extra non-linearity over a linear classifier,

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$

by using a non-linear function  $\sigma(\cdot)$

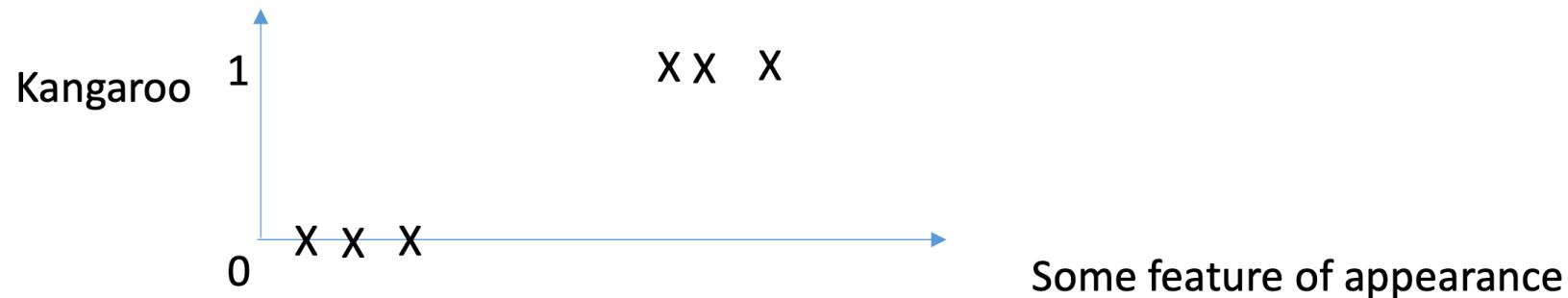
- The LR classifier is defined as

$$\sigma(f(\mathbf{x}_i)) \begin{cases} \geq 0.5 & y_i = +1 \\ < 0.5 & y_i = -1 \end{cases}$$

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

# Linear Regression Vs Logistic Regression

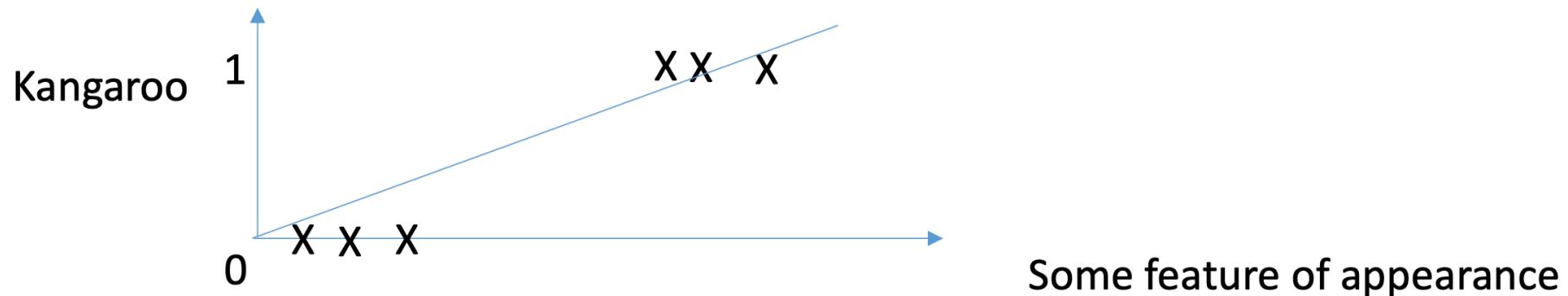
- In 1 dimension



Threshold of classification is 0.5

# Linear Regression Vs Logistic Regression

- In 1 dimension

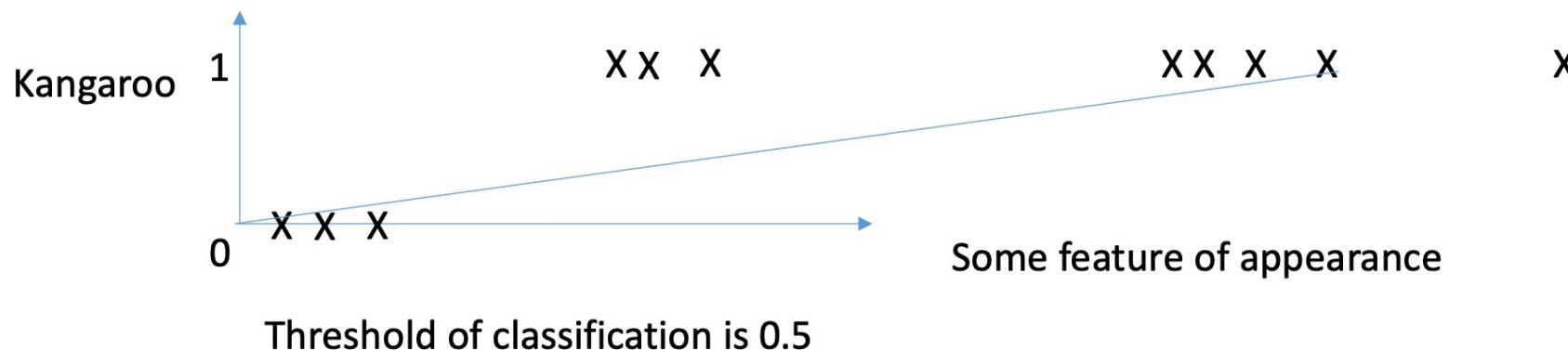


Threshold of classification is 0.5

Credits: Nick Barnes

# Linear Regression Vs Logistic Regression

- But what about

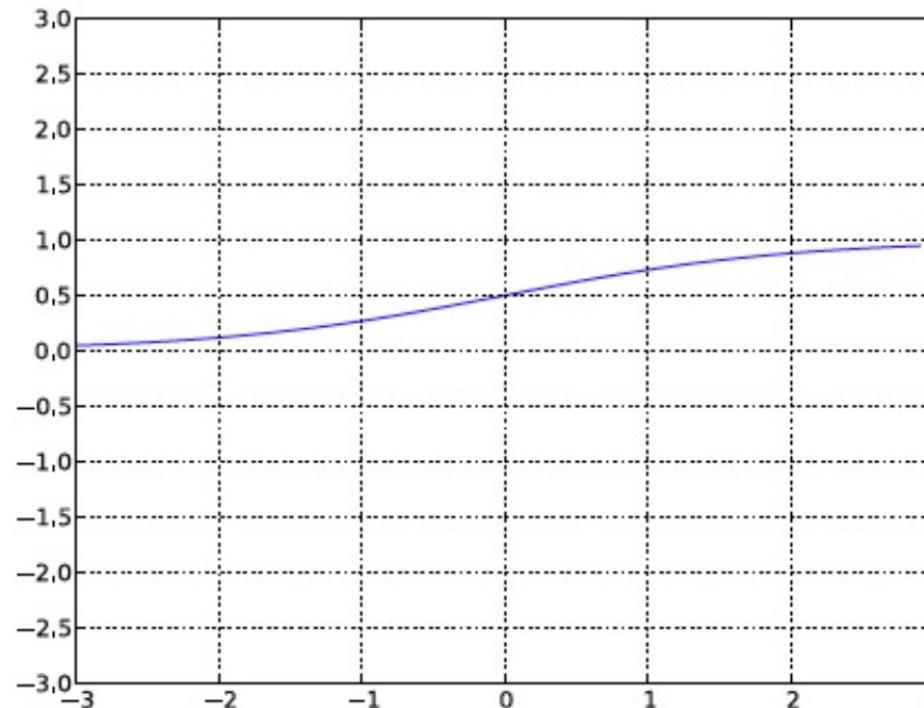


- Its not really what we want
- The decision boundary is being dominated by distant points.
- We want a function that is mostly zero below 0.5 and mostly 1 above 0.5.

# Logistic Regression

- Sigmoid function
  - Squashes the curve between 0 and 1
  - Always positive
  - Bounded
  - Strictly increasing
  - Continuous
  - Differentiable

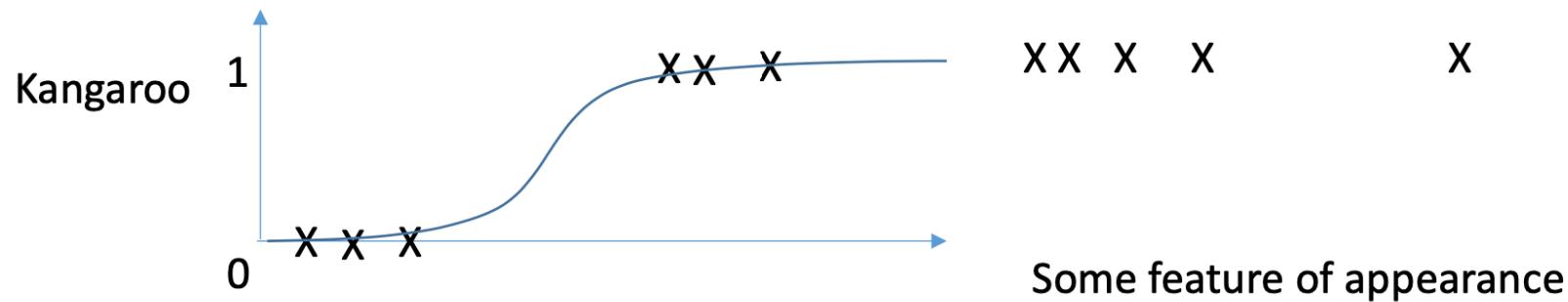
$$g(a) = \text{sigm}(a) = \frac{1}{1+\exp(-a)}$$



Credits: Nick Barnes

# Logistic Regression

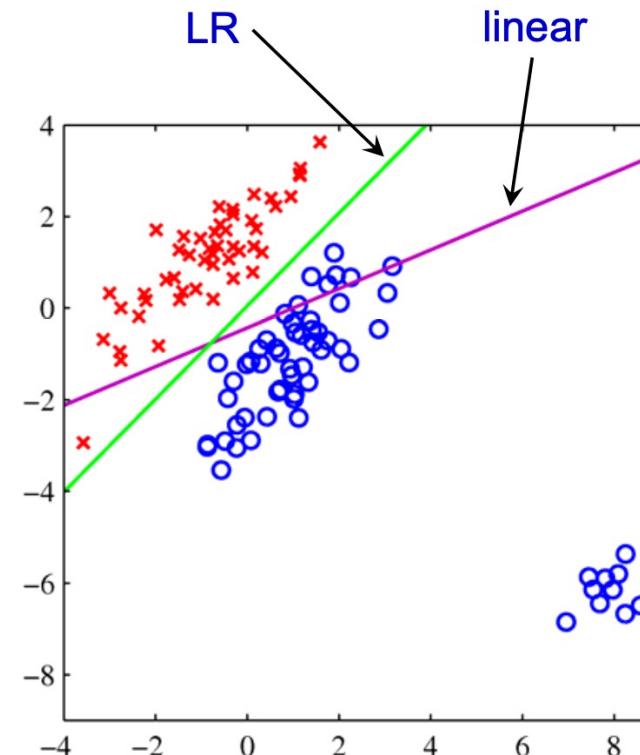
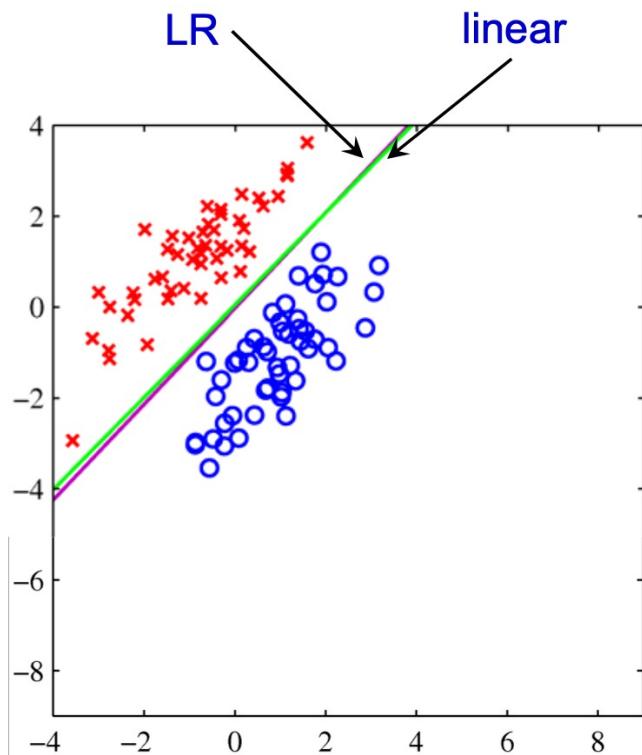
- Predict  $y = 1$  if  $f(x) \geq 0.5$ , otherwise predict  $y=0$  if  $f(x)<0.5$



$$g(a) = \text{sigm}(a) = \frac{1}{1+\exp(-a)}$$

Model is a sigmoid function (or a logistic function)

# Logistic Regression



$$\sigma(w_1x_1 + w_2x_2 + b) \text{ fit, vs } w_1x_1 + w_2x_2 + b$$

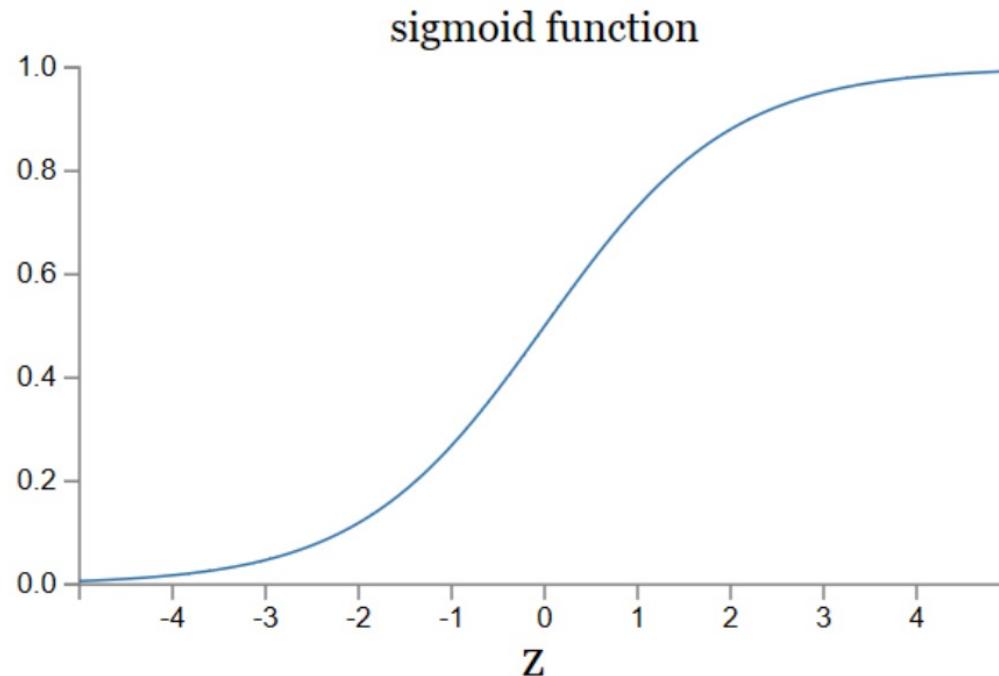
Credit: A. Zisserman

# Sigmoid non-linear activation

Sigmoid, non-linear function, is used to transform features.

$$\sigma(w \cdot x + b)$$

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



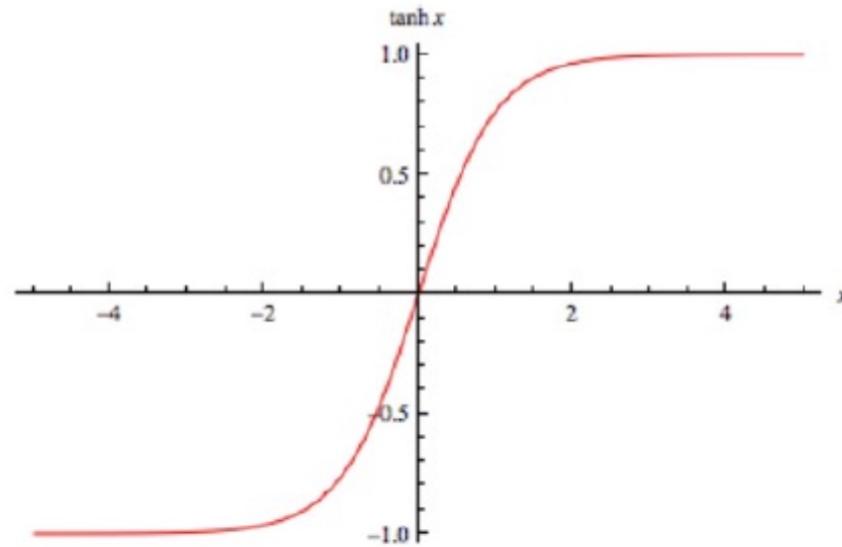
# Activation Functions

1. Sigmoid function.

2. Hyperbolic tangent.

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

- Squash between -1 and 1
- Positive or negative
- Bounded
- Strictly increasing

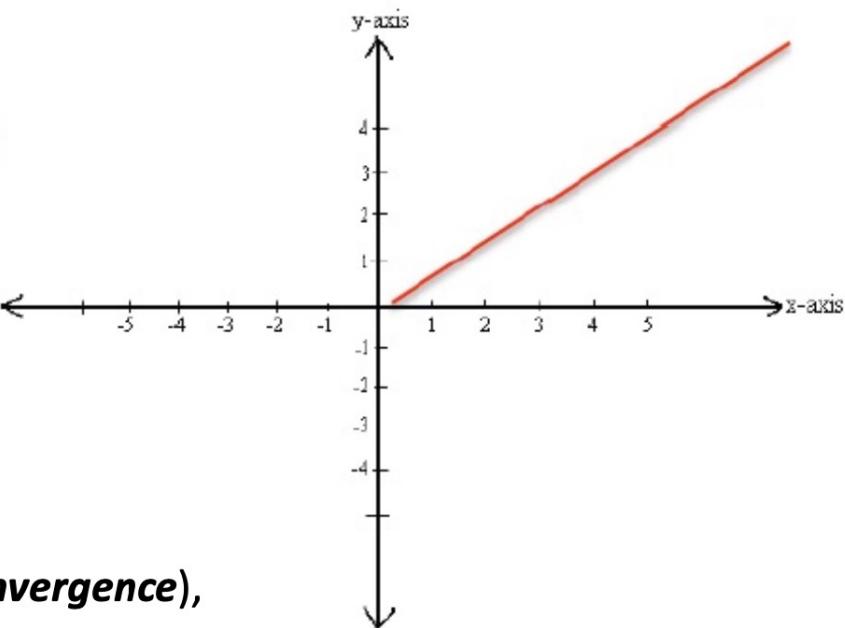


Credit: Nick Barnes

# Activation Functions

1. Sigmoid function.
2. Hyperbolic tangent.
3. Linear Rectifier (ReLU)
  - Bounded below by 0
  - Not upper bounded
  - Strictly increasing
  - Sparse neurons

$$\text{rectifier}(x) = \max(0, x)$$



Does not saturate ( $>0$ )

Very computationally efficient (no exp)

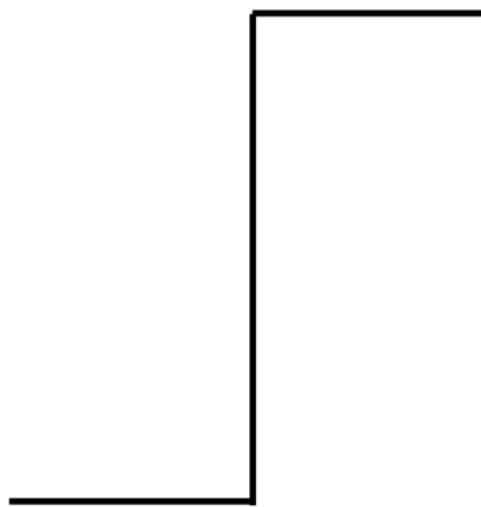
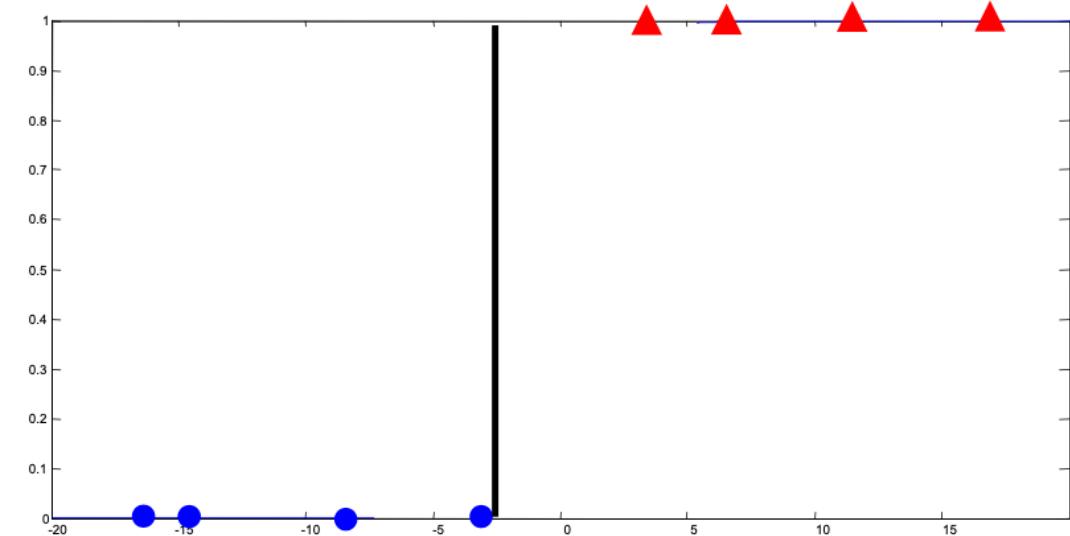
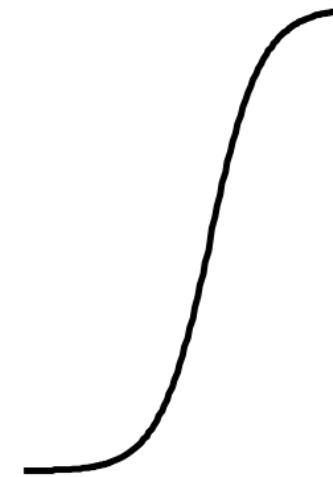
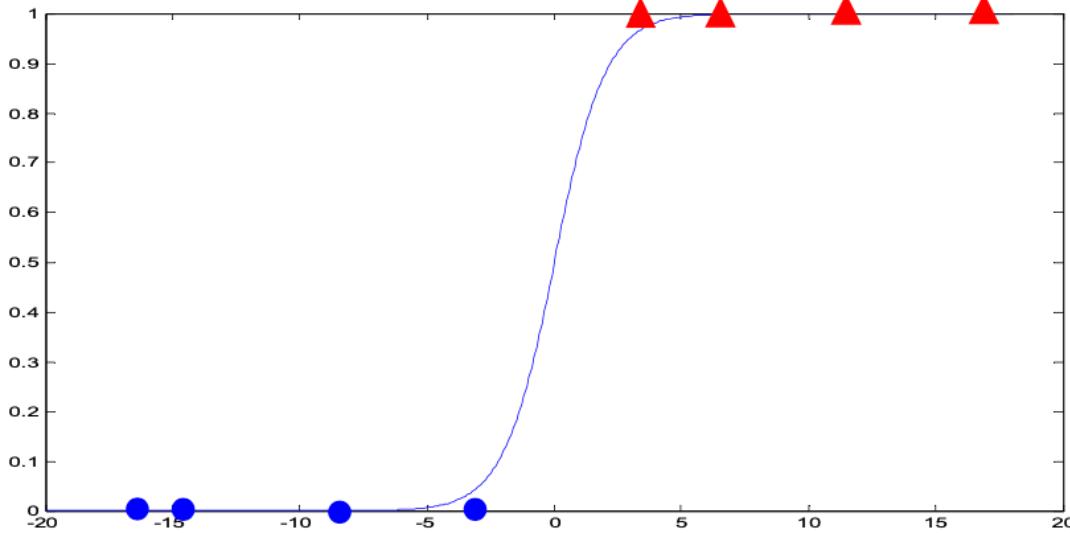
Converges much faster (**e.g. 6x faster convergence**),

AlexNet [Krizhevsky et al., 2012]

Used it.

Glorot, 2011 introduced to deep nets

Deep Sparse Rectifier Neural Network



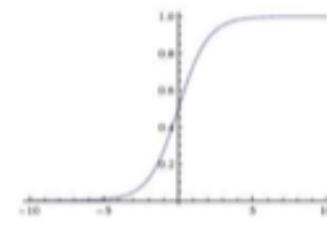
- A sigmoid favours a larger margin compared with a step classifier.

# Types of activation function

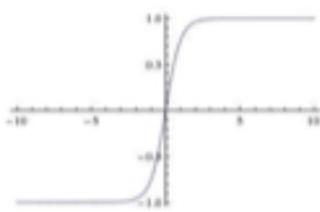
## Activation Functions

### Sigmoid

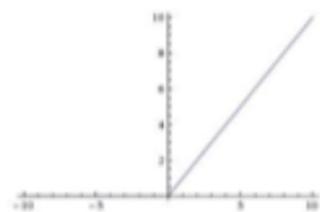
$$\sigma(x) = 1/(1 + e^{-x})$$



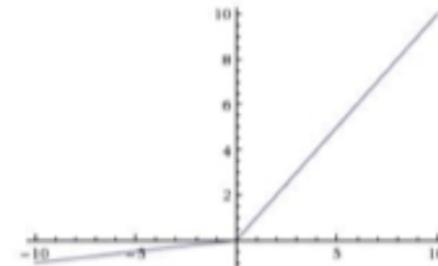
### tanh tanh(x)



### ReLU max(0,x)



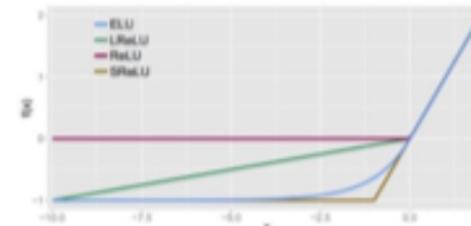
### Leaky ReLU $\max(0.1x, x)$



### Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

### ELU

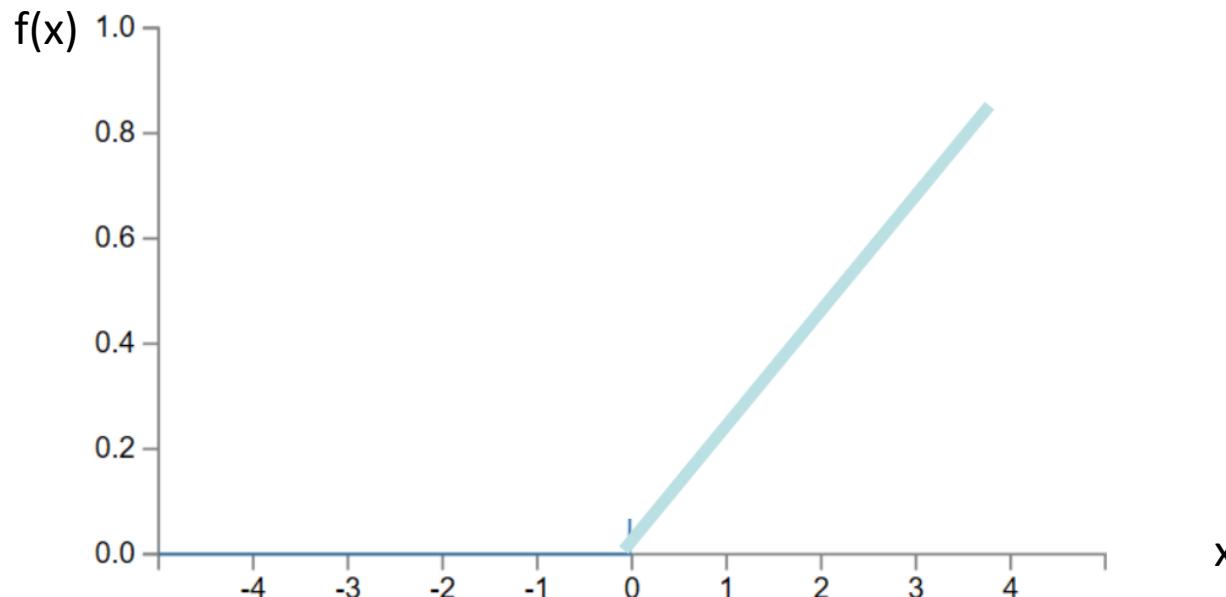
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$



# Another non-linear activation function: Rectified Linear Unit (ReLU)

- ReLU

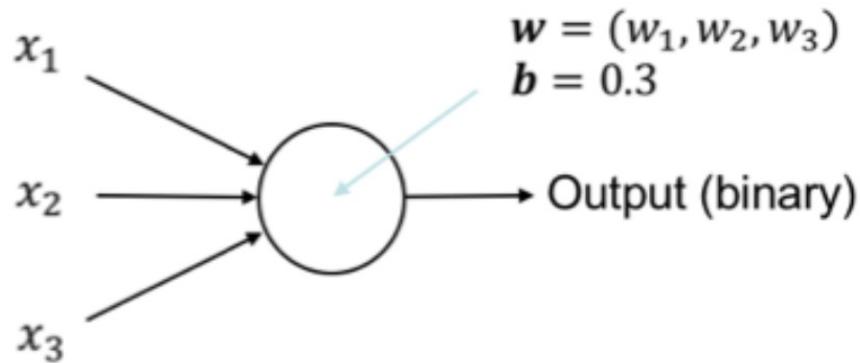
$$f(x) = \max(0, x)$$



## Recall: Binary Classifying an image

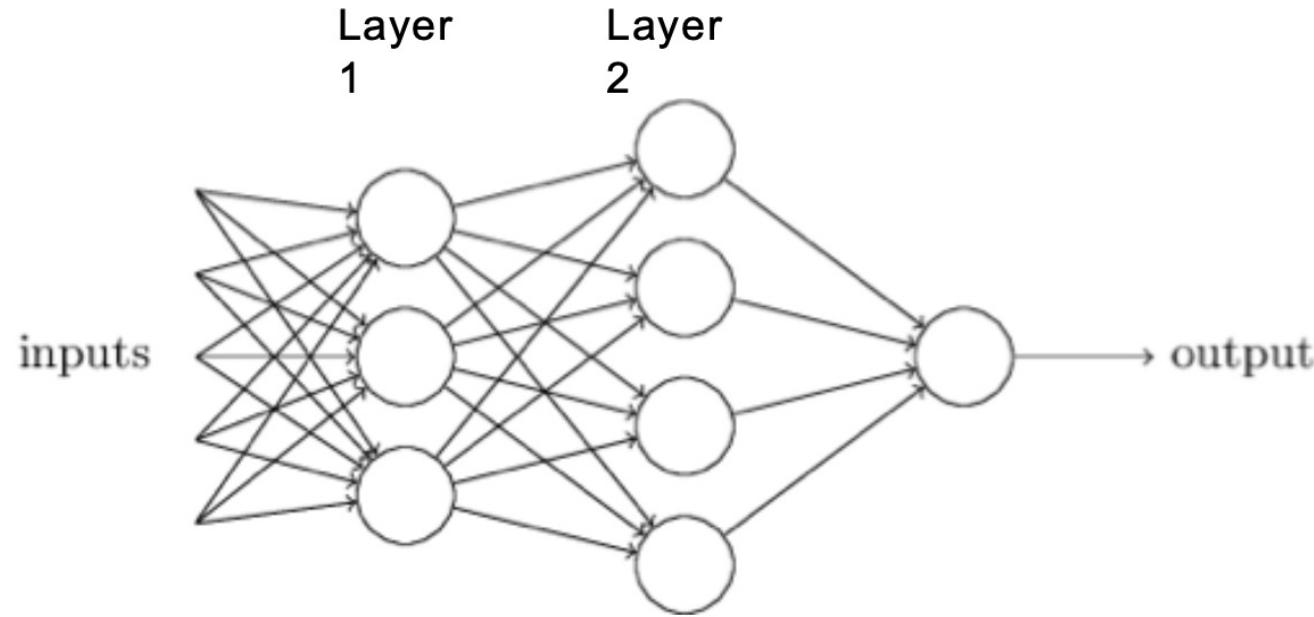
- Each pixel of the image would be an input.
- So, for a  $28 \times 28$  image, we vectorize.
- $\mathbf{x} = 1 \times 784$
- $\mathbf{w}$  is a vector of weights for each pixel,  $784 \times 1$
- $b$  is a scalar bias per perceptron
- $\text{result} = \mathbf{xw} + b \rightarrow (1 \times 784) \times (784 \times 1) + b = (1 \times 1) + b$

# linear classifier



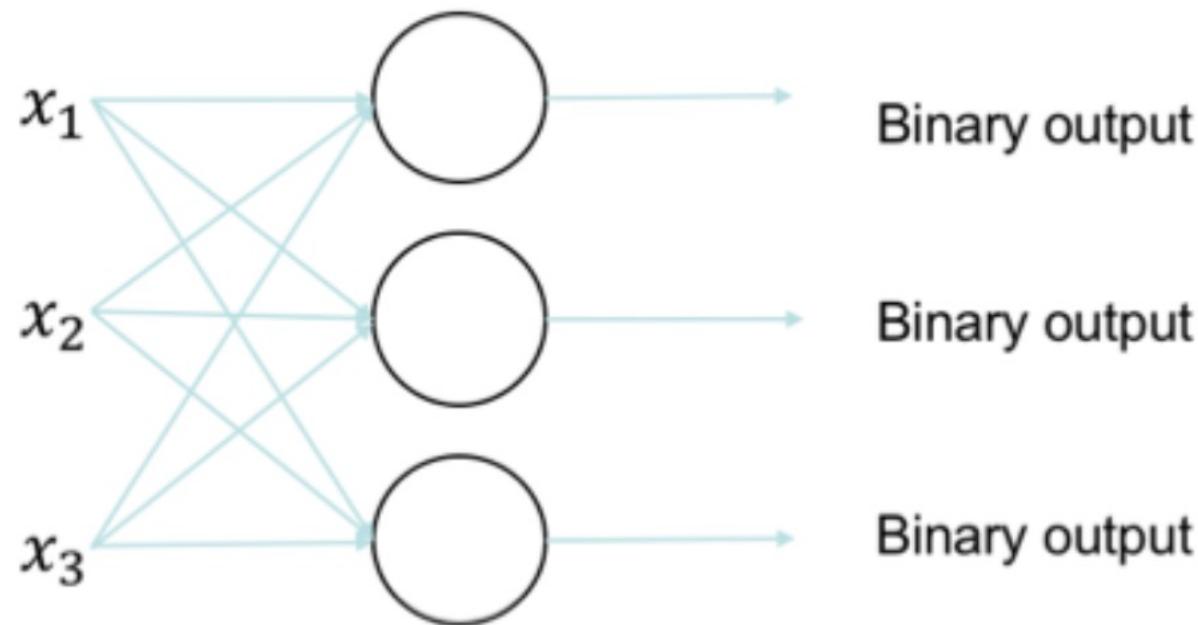
$$\text{output} = \begin{cases} 0 & \text{if } \mathbf{w} \cdot \mathbf{x} + b \leq 0 \\ 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \end{cases} \quad \mathbf{w} \cdot \mathbf{x} \equiv \sum_j w_j x_j$$

# Composition



Sets of layers and the connections (weights) between them define the *network architecture*.

Add more neurons -> Multiclass neural networks

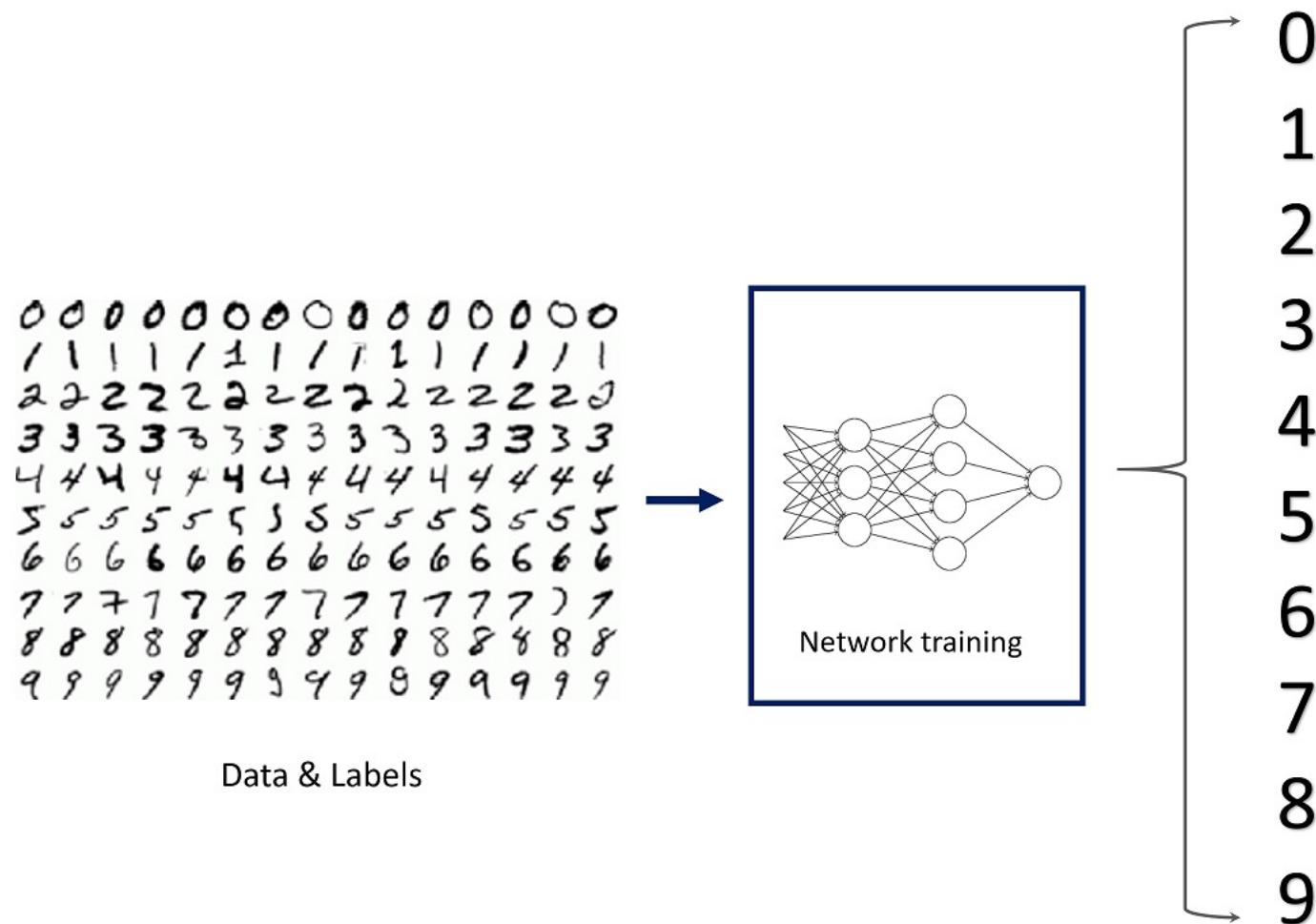


# Recall: Multi-class classification

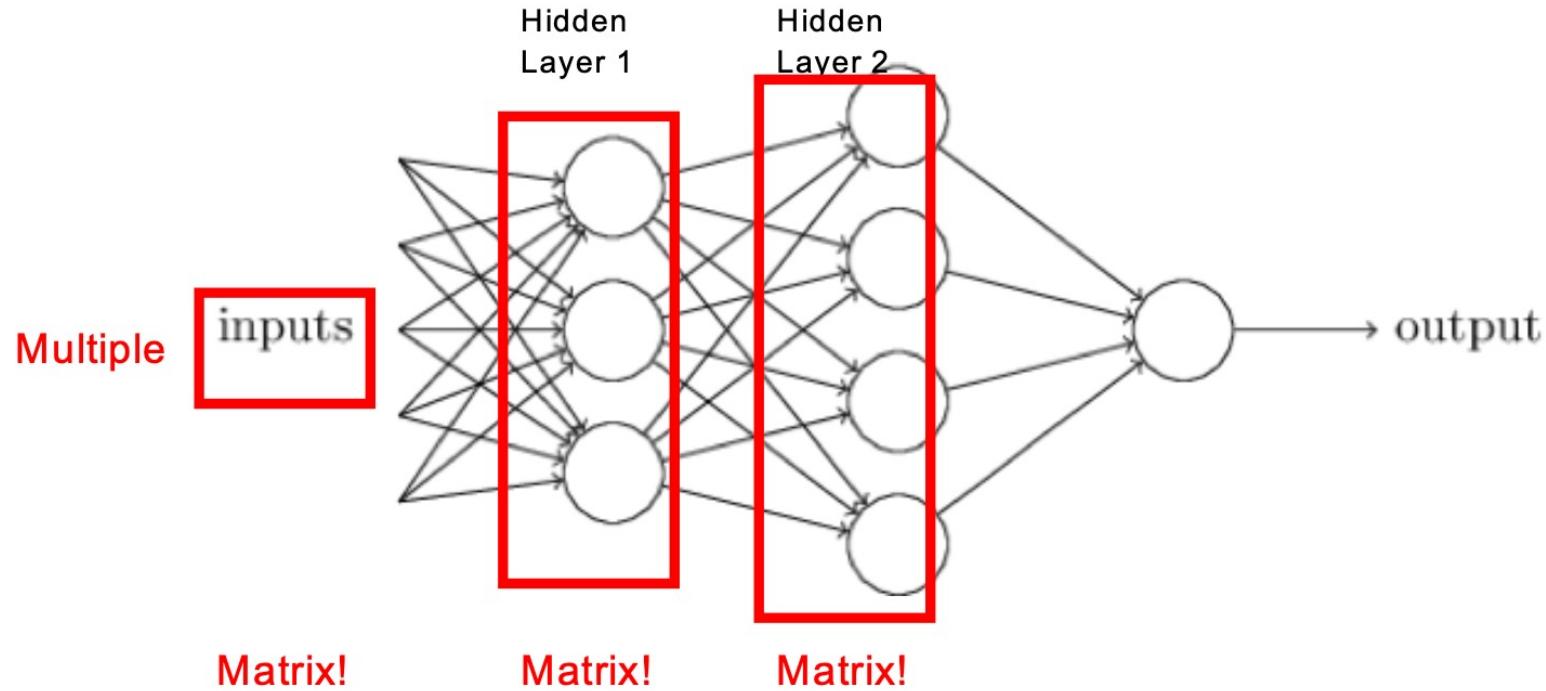
- Each pixel of the image would be an input node.
- So, for a 28x28 image, we vectorize the input image as a  $\mathbf{x} = 1 \times 784$  for  $\#\text{classes} = \text{e.g., } 10$ , in MNIST digit recognition
- $\mathbf{W}$  is a matrix of weights for each pixel/each perceptron –  $\mathbf{W} = 784 \times 10$  (10-class classification)
- $\mathbf{b}$  is a bias *per perceptron* (vector of biases);  $(1 \times 10)$
- result  $\mathbf{Z} = \mathbf{x}\mathbf{W} + \mathbf{b}$   $\rightarrow (1 \times 784) \times (784 \times 10) + \mathbf{b} \rightarrow (1 \times 10) + (1 \times 10) = 10\text{-D}$  output vector
- Activation function: softmax:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} \quad \text{for } i = 1, \dots, K \text{ and } \mathbf{z} = (z_1, \dots, z_K) \in \mathbb{R}^K$$

## Application: Multi-class classification



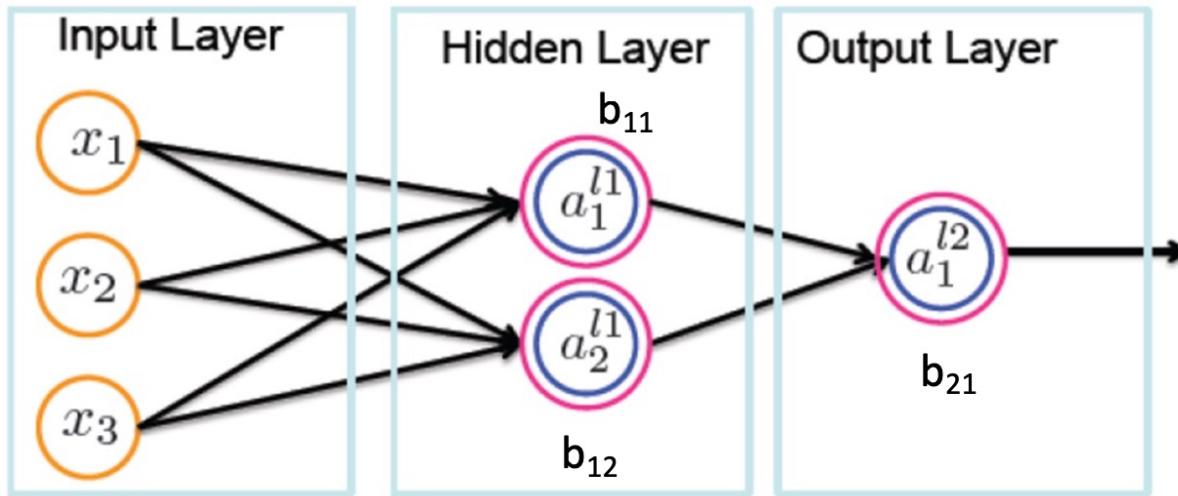
# Parallel computation



It's all just matrix-vector multiplications !  
*GPUs -> special hardware for fast/large matrix multiplication.*

Nielsen

# Single Hidden Layer Example



$$a_1^{l1} = w_{11}^{l0} x_1 + w_{21}^{l0} x_2 + w_{31}^{l0} x_3 + b_{11}$$

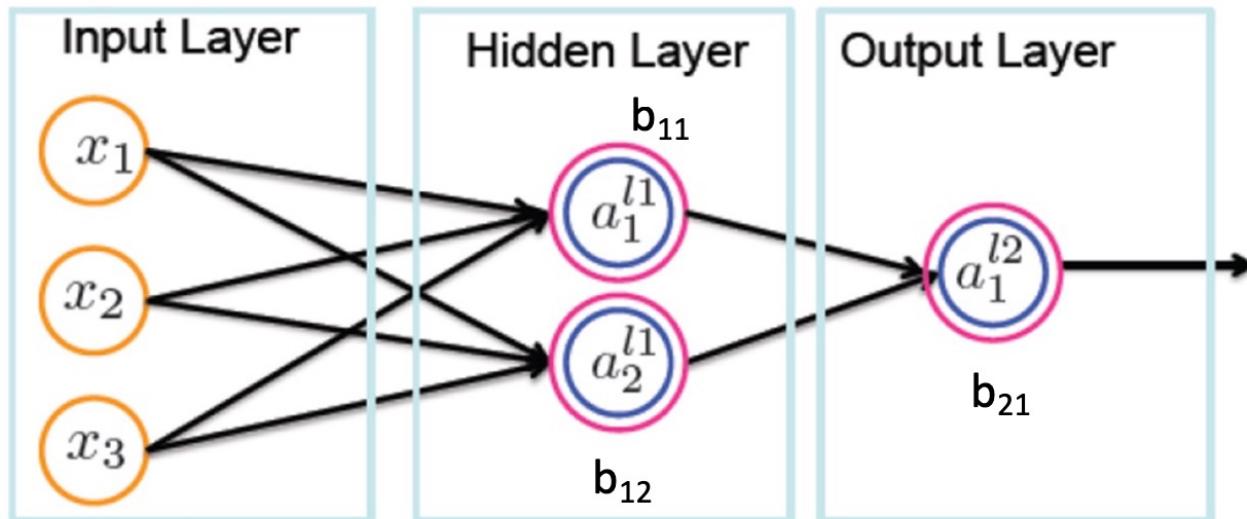
$$o_1^{l1} = g(a_1^{l1})$$

$$o_1^{l1} = g(w_{11}^{l0} x_1 + w_{21}^{l0} x_2 + w_{31}^{l0} x_3 + b_{11}) \quad \rightarrow \quad o_1^{l2} = g(w_{11}^{l1} o_1^{l1} + w_{12}^{l1} o_2^{l1} + b_{21})$$

$$o_2^{l1} = g(w_{12}^{l0} x_1 + w_{22}^{l0} x_2 + w_{32}^{l0} x_3 + b_{12})$$

**Output Layer:** 
$$o_1^{l2} = g(w^{l1} g(w^{l0} x))$$

# Single Hidden Layer Example



Example, suppose:

$$X = \{0.4, 0.2, 0.2\}$$

$$w_1^{l0} = \{1.0, 0.5\}$$

$$w_2^{l0} = \{0.2, 1.2\}$$

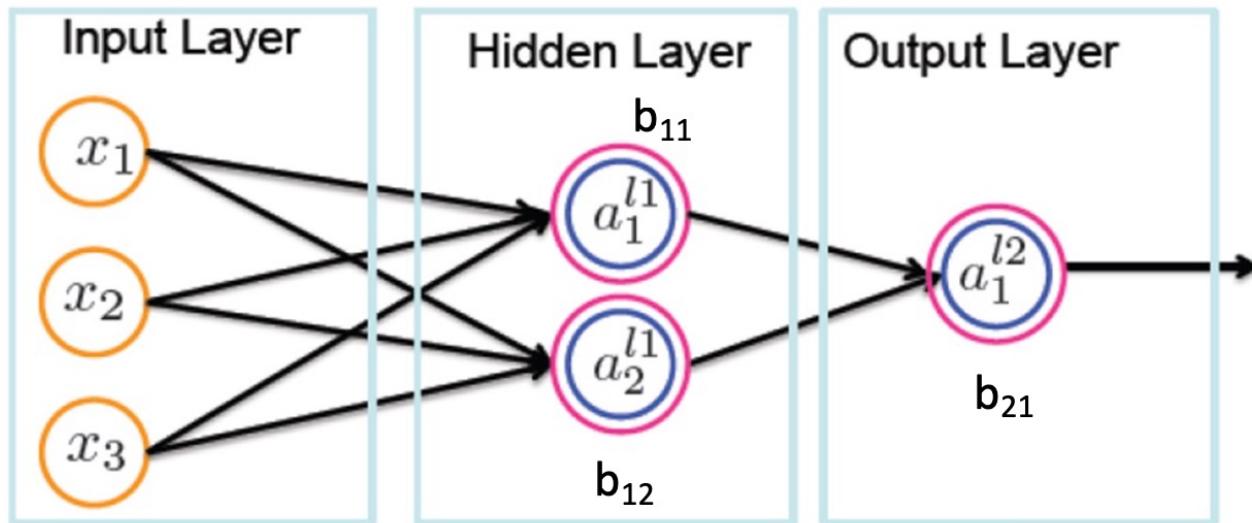
$$w_3^{l0} = \{1.2, 0.1\}$$

$$w_1^{l1} = \{0.3, 0.9\}$$

$$B_{11} = 0.1$$

$$B_{12} = 0.2$$

$$B_{21} = 0.3$$



Example, suppose:

$$X = \{0.4, 0.2, 0.2\}$$

$$w_1^{l0} = \{1.0, 0.5\}$$

$$w_2^{l0} = \{0.2, 1.2\}$$

$$w_3^{l0} = \{1.2, 0.1\}$$

$$w_1^{l1} = \{0.3, 0.9\}$$

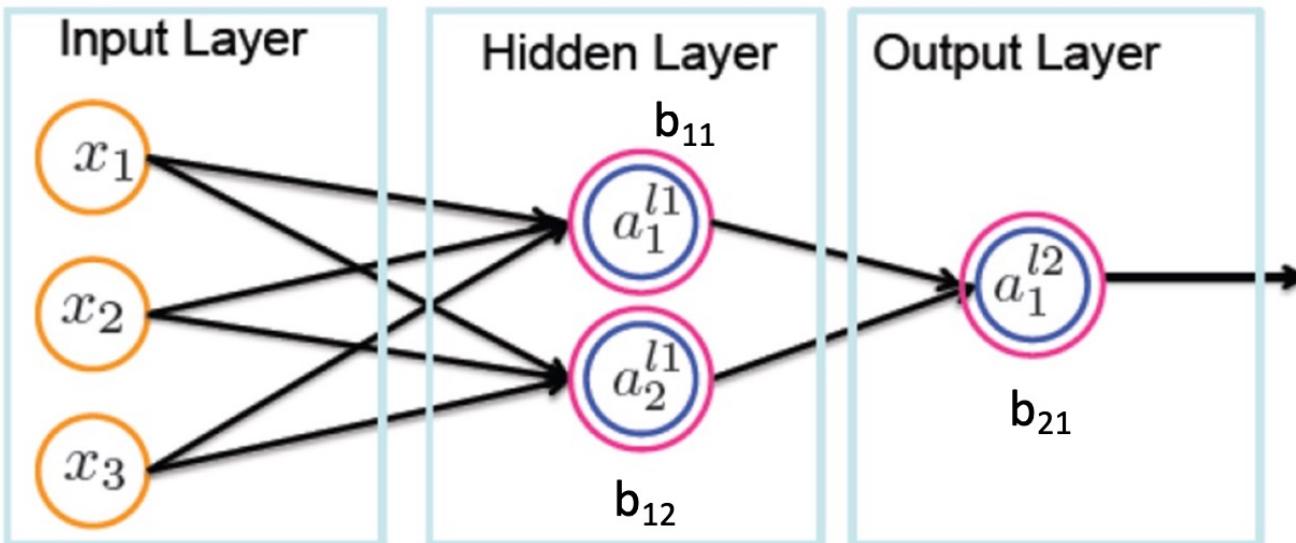
$$B_{11} = 0.1$$

$$B_{12} = 0.2$$

$$B_{21} = 0.3$$

$$\begin{aligned} o_1^{l1} &= g(w_{11}^{l0} x_1 + w_{21}^{l0} x_2 + w_{31}^{l0} x_3 + b_{11}) \\ &= g(1.0 * 0.4 + 0.2 * 0.2 + 1.2 * 0.2 + 0.1) \end{aligned}$$

# Single Hidden Layer Example



Example, suppose:

$$X = \{0.4, 0.2, 0.2\}$$

$$w_{1.}^{l0} = \{1.0, 0.5\}$$

$$w_{2.}^{l0} = \{0.2, 1.2\}$$

$$w_{3.}^{l0} = \{1.2, 0.1\}$$

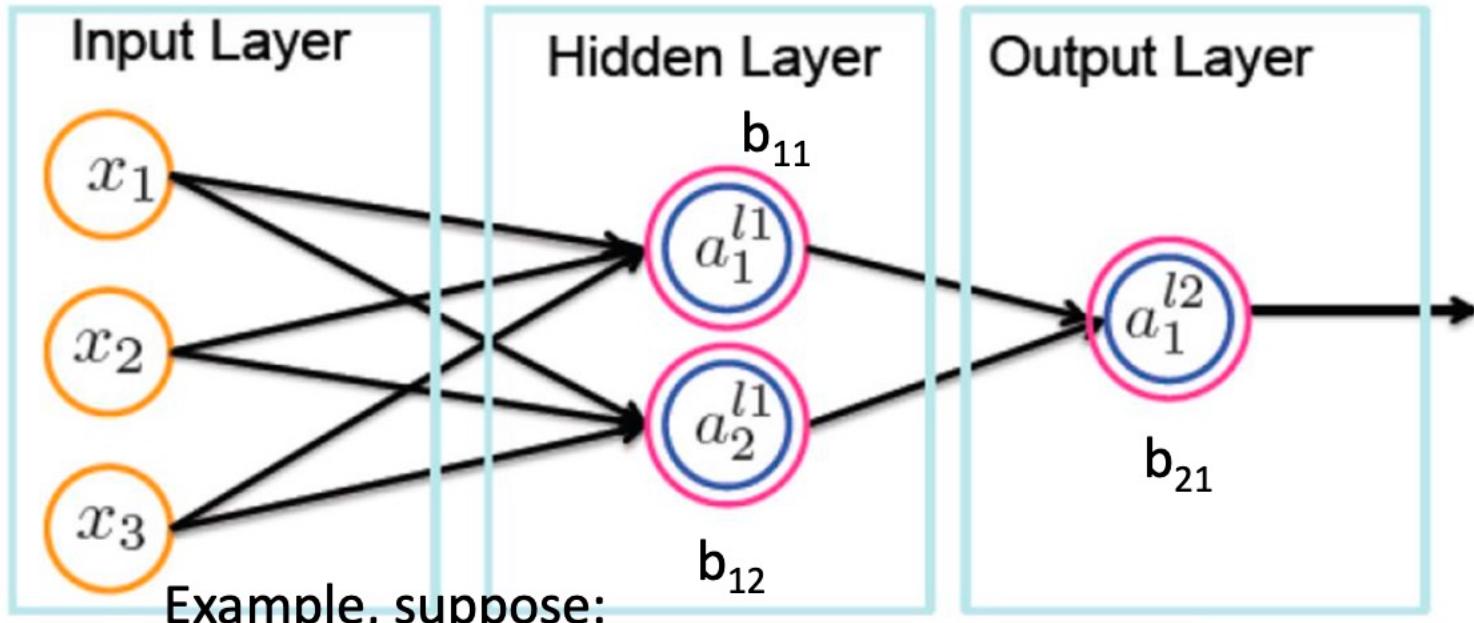
$$w_{1.}^{l1} = \{0.3, 0.9\}$$

$$B_{11} = 0.1$$

$$B_{12} = 0.2$$

$$B_{21} = 0.3$$

$$\begin{aligned} o_1^{l1} &= g(w_{11}^{l0} x_1 + w_{21}^{l0} x_2 + w_{31}^{l0} x_3 + b_{11}) \\ &= g(1.0 * 0.4 + 0.2 * 0.2 + 1.2 * 0.2 + 0.1) \\ &= g(0.4 + 0.04 + 0.24 + 0.1) \\ &= g(0.78) \\ &= 1/(1+\exp(0.78)) \end{aligned}$$



$$X = \{0.4, 0.2, 0.2\}$$

$$w_1^{l0} = \{1.0, 0.5\}$$

$$w_2^{l0} = \{0.2, 1.2\}$$

$$w_3^{l0} = \{1.2, 0.1\}$$

$$w_1^{l1} = \{0.3, 0.9\}$$

$$B_{11} = 0.1$$

$$B_{12} = 0.2$$

$$B_{21} = 0.3$$

$$o_2^{l1} = g(w_{12}^{l0} x_1 + w_{22}^{l0} x_2 + w_{32}^{l0} x_3 + b_{12})$$

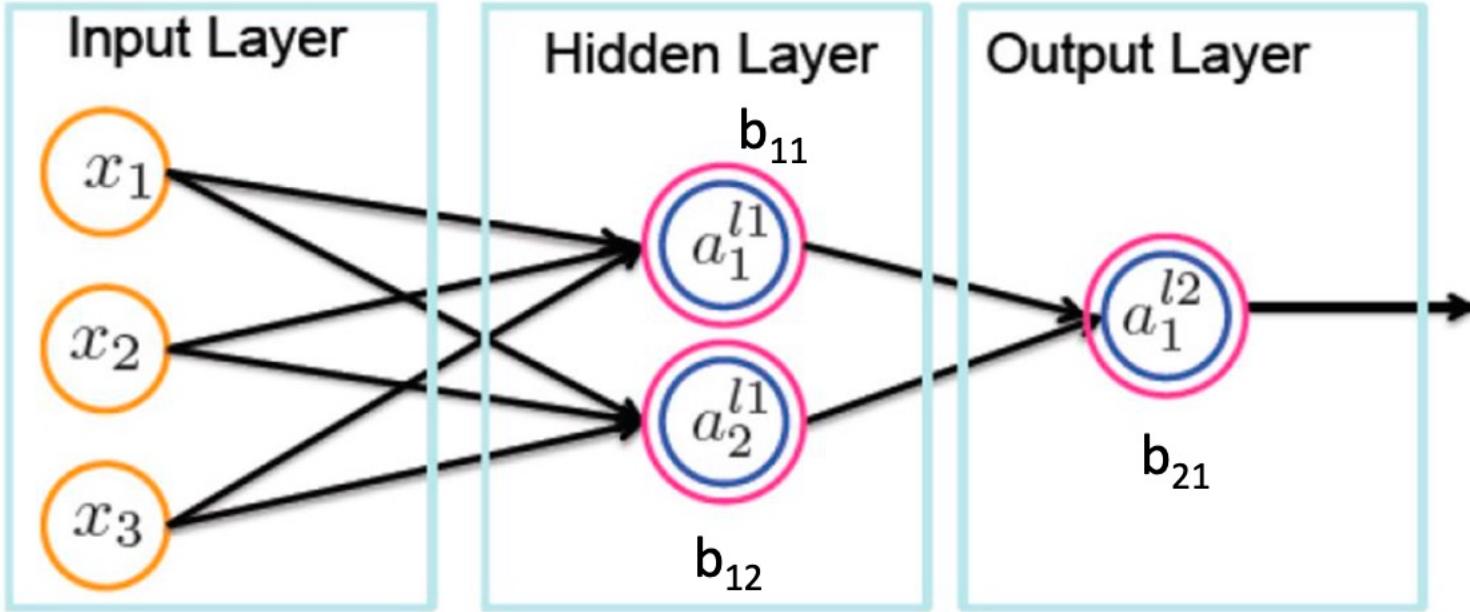
$$= g(0.5 * 0.4 + 1.2 * 0.2 + 0.1 * 0.2 + 0.2)$$

$$= g(0.2 + 0.24 + 0.03 + 0.2)$$

$$= g(0.49)$$

$$= 1 / (1 + \exp(-0.49))$$

$$= 0.62..$$

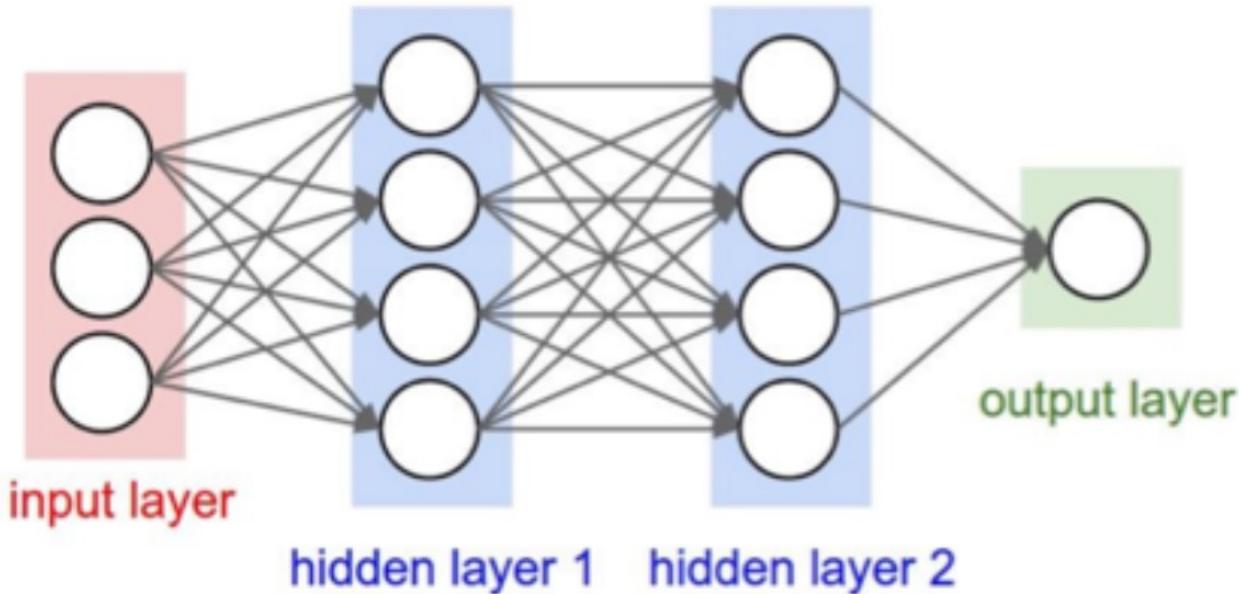


Note that the computation of the activations at layer 1 are vector computations, and so can be parallelized in a GPU!  
 Many computations are like this so GPU's greatly decrease computation time (but not order).

Example, suppose:

$$\begin{aligned}
 X &= \{0.4, 0.2, 0.2\} & o_1^{l2} &= g(w_{11}^{l1}o_1^{l1} + w_{12}^{l1}o_2^{l1} + b_{21}) \\
 w_1^{l1} &= \{1.0, 0.5\} & &= g(0.3 * 0.68 + 0.9 * 0.62 + 0.3) \\
 w_2^{l1} &= \{0.2, 1.2\} & &= g(0.204 + 0.558 + 0.3) \\
 w_3^{l1} &= \{1.2, 0.1\} & &= g(1.062) \\
 w_1^{l2} &= \{0.3, 0.9\} & &= 1 / (1 + \exp(-1.062)) \\
 B_{11} &= 0.1 & &= 0.74.. \\
 B_{12} &= 0.2 & & \\
 B_{21} &= 0.3 & &
 \end{aligned}$$

# Parallel Computation



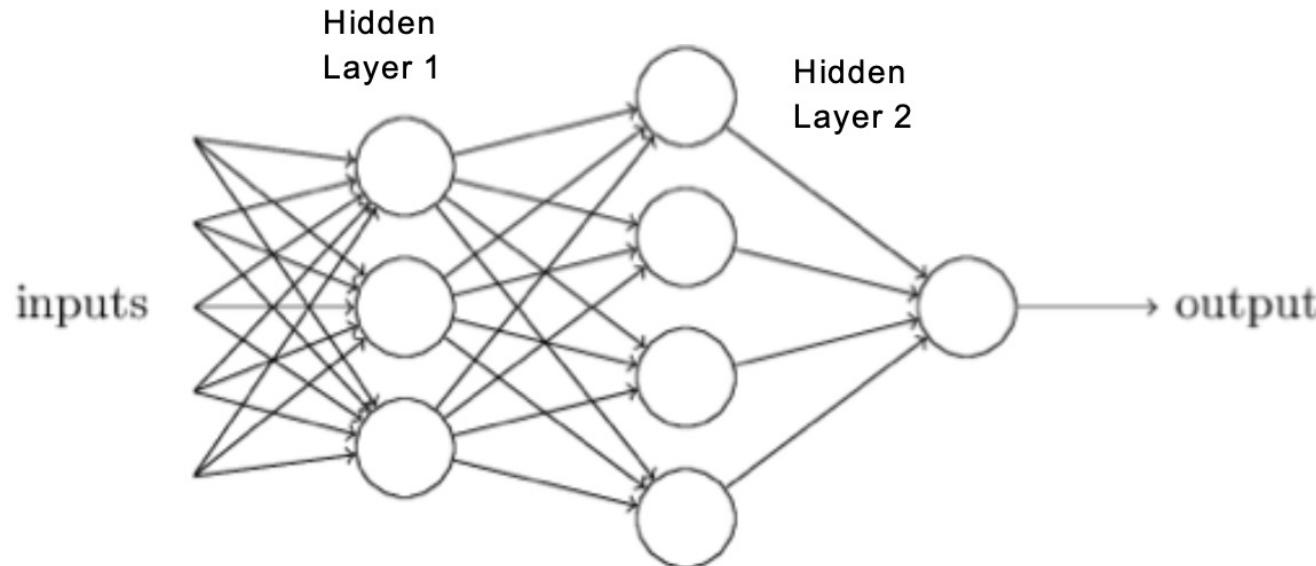
```
# forward-pass of a 3-layer neural network:  
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)  
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)  
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)  
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)  
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

# Problems with using all-linear functions in the hidden layers

- We have formed multiple layers or chains of linear functions.
- We know that linear function of linear functions is a linear function, i.e. they can be reduced:
  - $g = f(h(x))$
  - Eg.  $O_1 = xw_1 + b_1$ ;  $O_2 = o_1w_2 + b_2$ ;  $\Rightarrow O_2 = (xw_1 + b_1)w_2 + b_2 = xw_1w_2 + (b_1w_2 + b_2)$

*Our multi-layer composition of functions is really just a single linear function!!!!*

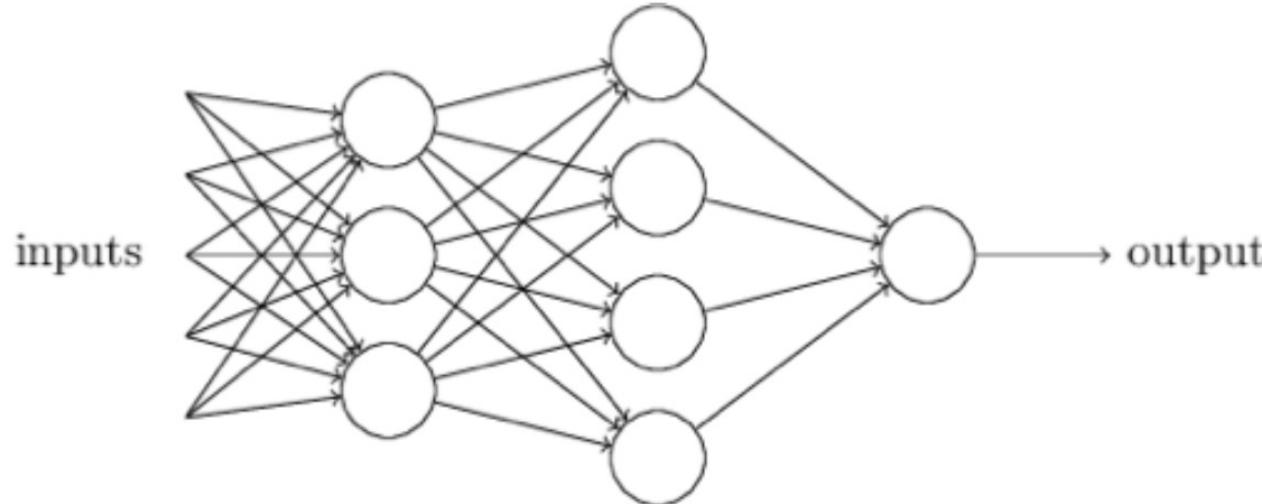
# Multi-Layer Perceptron (MLP)



- Attempt to represent complex functions as compositions of smaller functions.
- Outputs from one perception are fed into inputs of another perceptron.
- Layers that are in between the input and the output are called *hidden layers*, because we are going to *learn* their weights via an optimization process.

# Multi-Layer Perceptron (MLP)

- ...is a '*fully connected*' neural network with non-linear activation functions.



- '*Feed-forward*' neural network

# Mathematical Theorem for MLP

- The application of MLP is grounded in theory.
  - **Universal Approximation Theorem.**

With three (or more) layers and enough parameters, MLP can approximate any function.

Proof(Michael Nielson):

<http://neuralnetworksanddeeplearning.com/chap4.html>

*If a 3-layer network can approximate any function  
...given enough parameters ... then why should we  
need to go deeper ?*

- Intuitively, composition is efficient because it allows the *reuse* of layer of neurons.
- Empirically, deeper networks do a better job than shallow networks at learning certain hierarchies of knowledge.

# Why go deeper ?

- *A three-layer network is already a “universal approximator”,*
  - *Only if the number of neurons in the hidden layer is sufficiently many.*
  - *The identification of the weights is troublesome (possible convergence to wrong local minima)*
- Deep neural networks
  - Have many levels of non-linearity
  - Compactly represent highly non-linear functions
  - Layered structure facilitates convergence to a good optimum
  - **There are many local optima of approximately the same (and good) quality**

# Interpretation

**Question:** What does a hidden unit do?

**Answer:** It can be thought of as a classifier or feature detector.

**Question:** How many layers? How many hidden units?

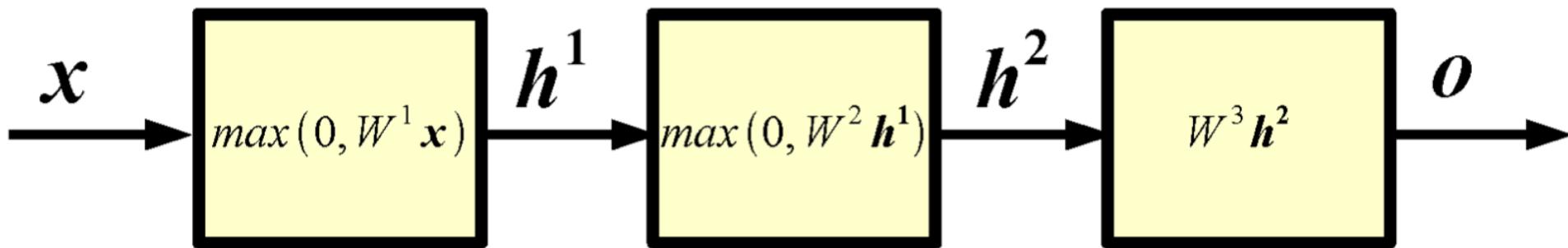
**Answer:** Cross-validation or hyper-parameter search methods are the answer. In general, the wider and the deeper the network the more complicated the mapping.

**Question:** How do I set the weight matrices?

**Answer:** Weight matrices and biases are learned.

First, we need to define a measure of quality of the current mapping.  
Then, we need to define a procedure to adjust the parameters.

# Neural Networks: example



$x$  input

$h^1$  1-st layer hidden units

$h^2$  2-nd layer hidden units

$o$  output

Example of a 2 hidden layer neural network (or 4 layer network,  
counting also input and output).

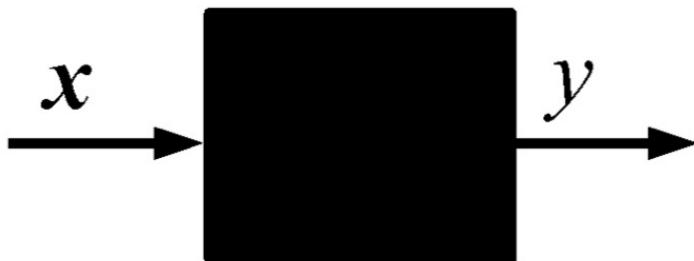
# Supervised Learning

$\{(\mathbf{x}^i, y^i), i=1 \dots P\}$  training dataset

$\mathbf{x}^i$  i-th input training example

$y^i$  i-th target label

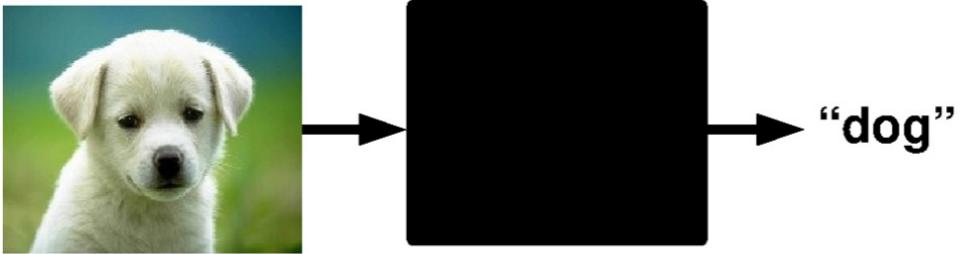
$P$  number of training examples



Goal: predict the target label of unseen inputs.

# Supervised Learning Example

## Classification



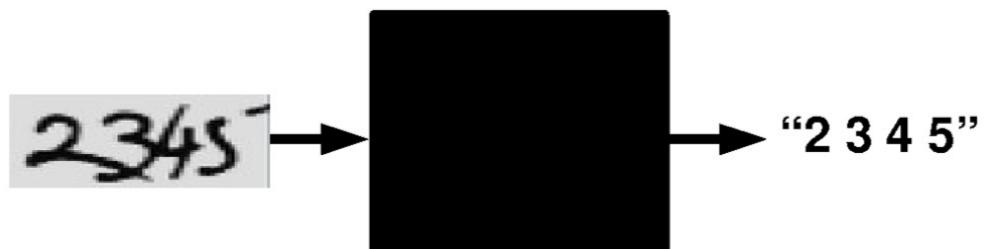
classification

## Denoising



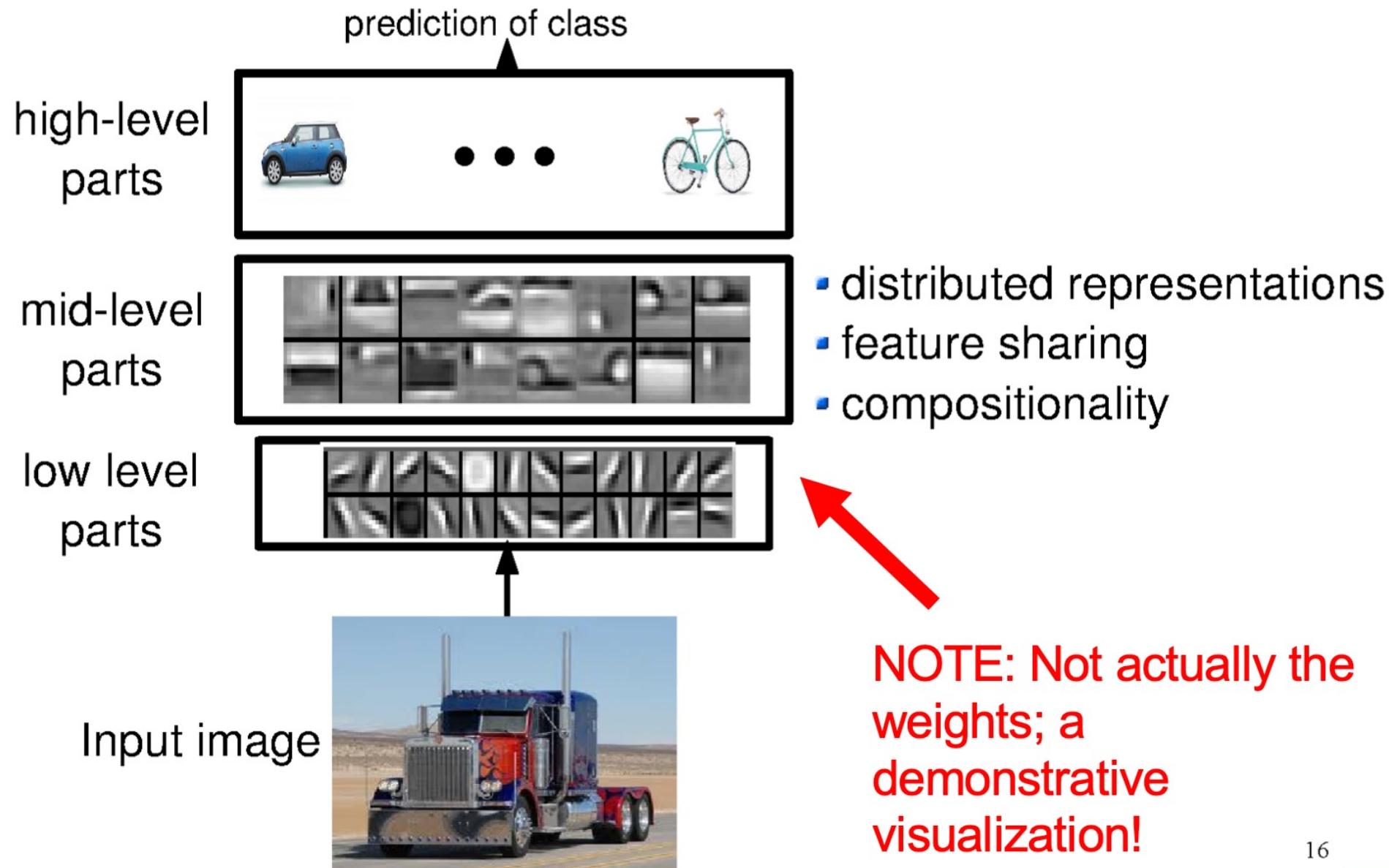
regression

## OCR

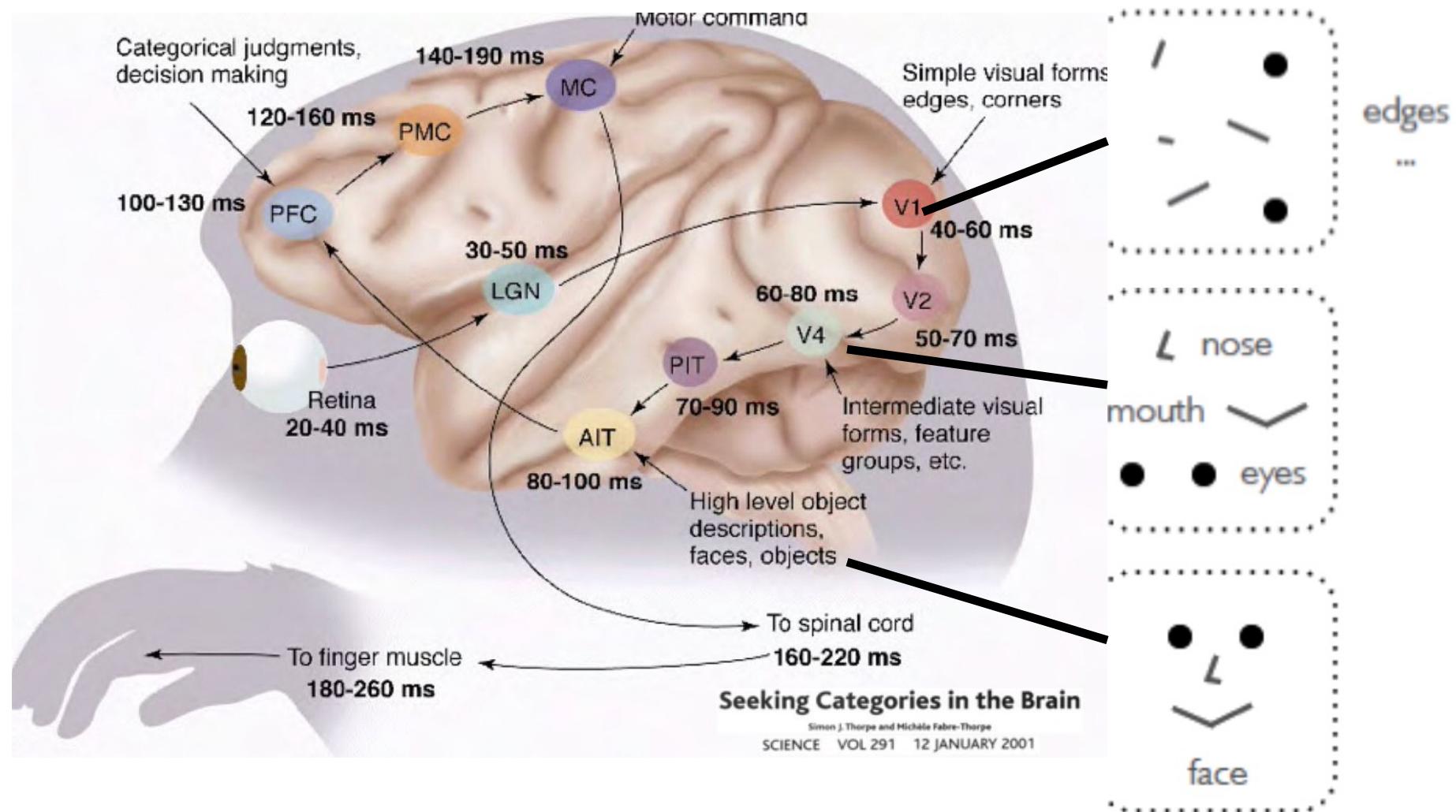


structured  
prediction

# Interpretation



# Biological inspiration: Human Neural Systems



# Multilayer Neural Networks

*Forward Computation*

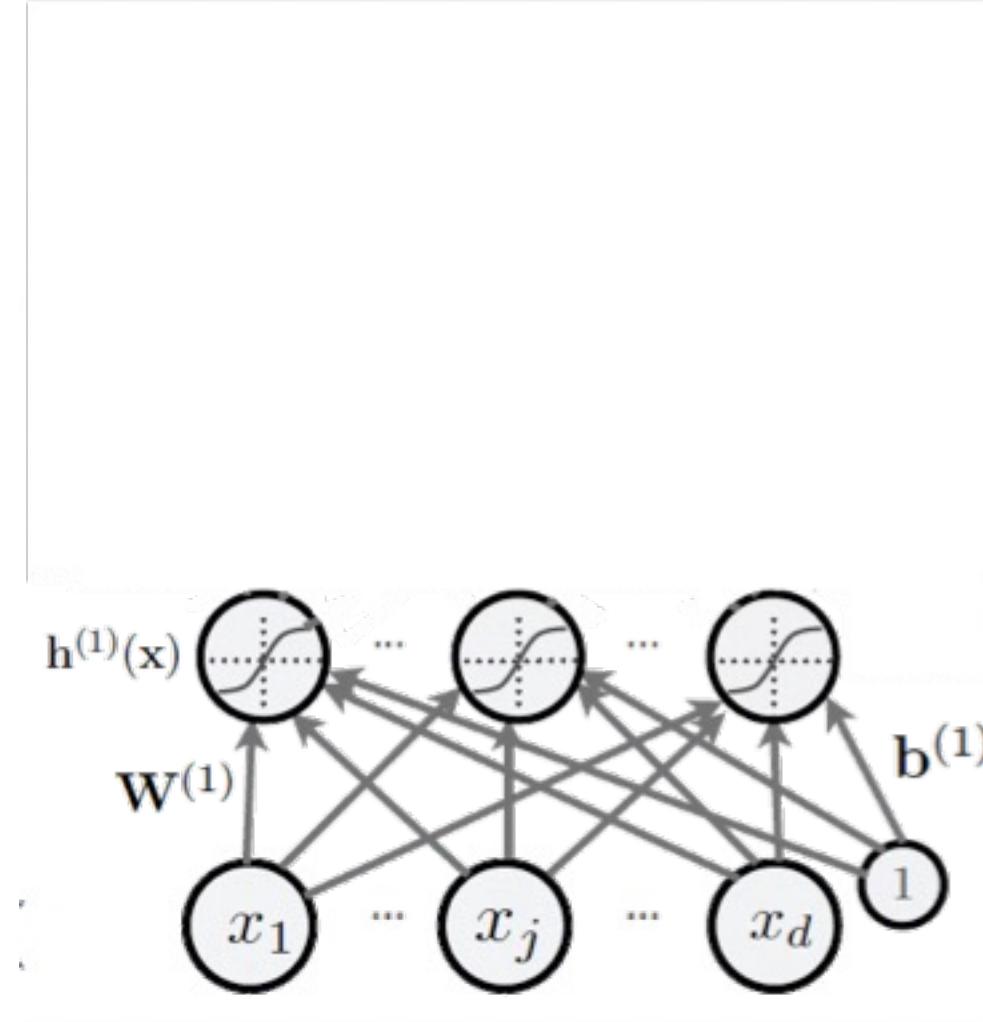
# Multilayer Neural Network

- Forward



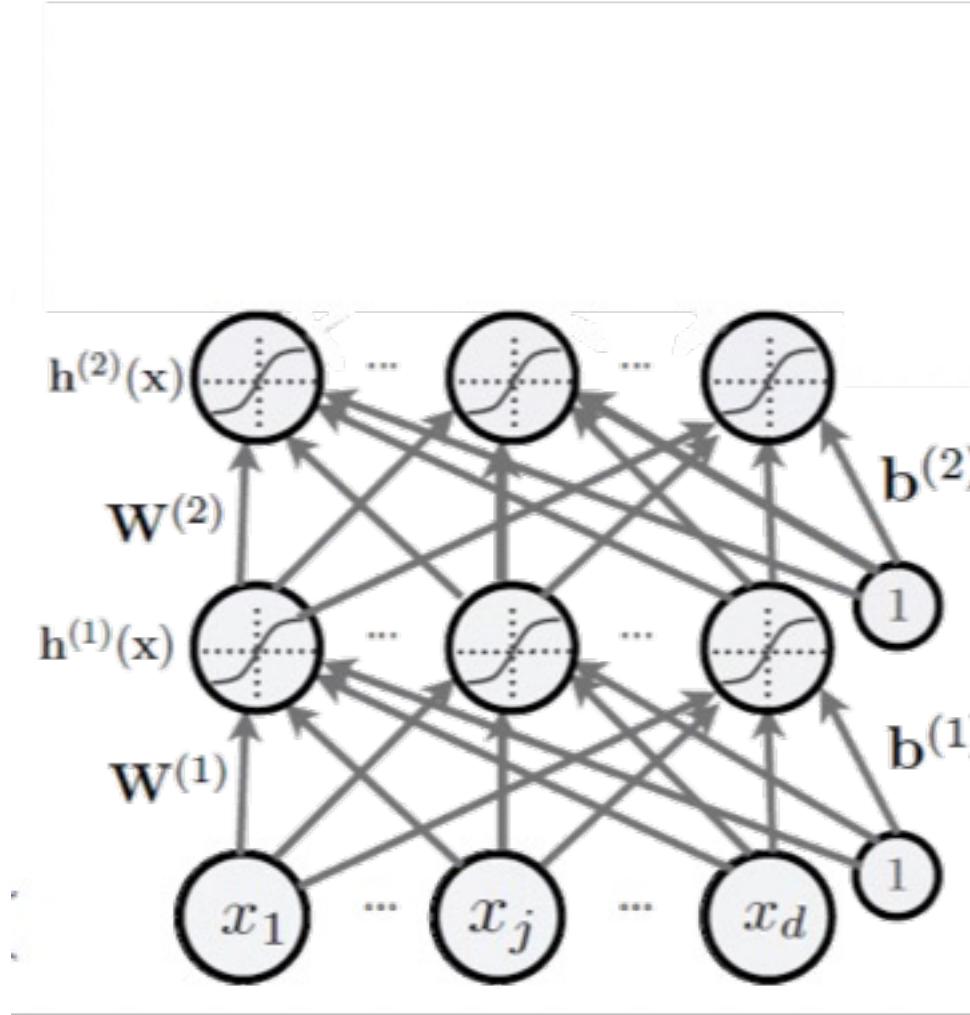
# Multilayer Neural Network

- Forward



# Multilayer Neural Network

- Forward



# Multilayer Neural Network

- Could have  $L$  hidden layers:

- Layer pre-activation for  $k > 0$ :

$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

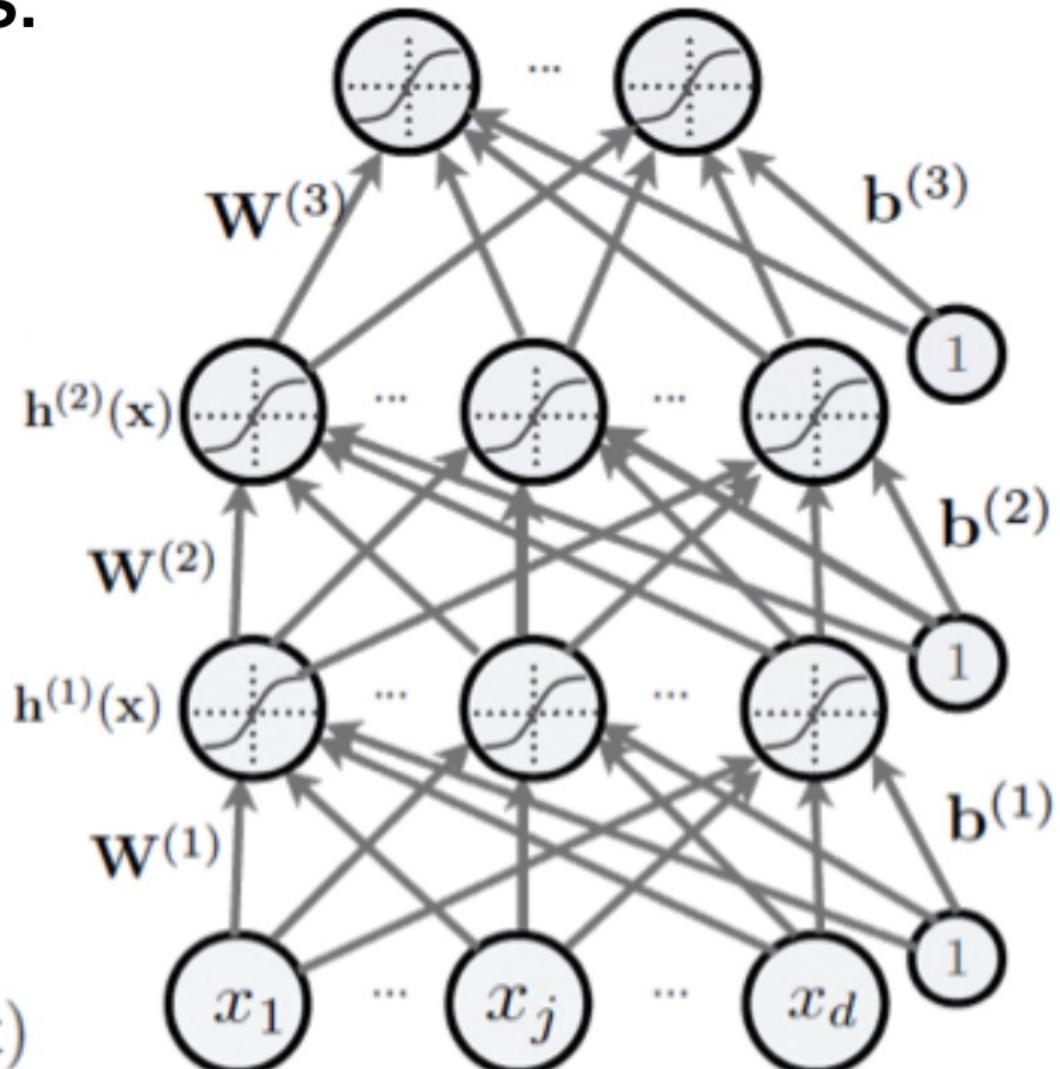
$$(\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x})$$

- Hidden layer activation ( $k$  from 1 to  $L$ ):

$$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x}))$$

- Output layer activation ( $k=L+1$ ):

$$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$



# Multilayer Neural Network

- Multilayer neural network models a function  $f(x, \{w_i, b_i\})$  by multiple layers of neurons.
- *The goal of using neural network is to model the data by solving the weight parameters,  $\{w_i, b_i\}$ , of the neural network through an optimization process which is defined via the backpropagation.*

# Machine Learning?

- In our case, this means:
    - Need to find a way to identify good values for our network's weights
    - Need to do this by relying on training examples (past experiences)
    - Need to do this by employing some sort of performance measure
- Loss function:  
mathematical minimization!

# Empirical risk minimization

- Framework to design learning algorithms

$$\arg \min_{\theta} \frac{1}{T} \sum_t l(f(\mathbf{x}^{(t)}; \theta), y^{(t)}) + \lambda \Omega(\theta)$$

- $l(f(\mathbf{x}^{(t)}; \theta), y^{(t)})$  = loss function
- $\Omega(\theta)$  = regularizer (penalizes certain values of  $\theta$ )

- Learning is cast as mathematical optimization
  - Ideally, we optimize classification errors, but it's not smooth
  - Loss function is a surrogate for what we truly should optimize (e.g. upper bound)

Classification error is non-smooth and non-convex

$$\hat{f}^{\text{erm}} = \operatorname*{argmin}_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n \mathbb{I}(Y_i \neq f(X_i)).$$

The following is an example of convex surrogates of loss functions.

- Hinge loss:  $\ell(y) = \max(0, 1 - t \cdot y)$

# Recipe for design a Machine Learning system

1. Given training data:

$$\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$$

2. Choose each of these:

- Decision function

$$\hat{\mathbf{y}} = f_{\boldsymbol{\theta}}(\mathbf{x}_i)$$

- Loss function

$$\ell(\hat{\mathbf{y}}, \mathbf{y}_i) \in \mathbb{R}$$

3. Define Goal state:

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta}} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)$$

4. Train with BP in SGD:  
(take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)$$

# Loss function for classification

- Neural network estimates  $f(\mathbf{x})_c = p(y = c|\mathbf{x})$ 
  - We could maximize the probabilities of  $y^{(t)}$  given  $\mathbf{x}^{(t)}$  in the training set

- To frame as minimization, we minimize the negative log-likelihood

$$l(f(\mathbf{x}), y) = - \sum_c 1_{(y=c)} \log f(\mathbf{x})_c = - \log f(\mathbf{x})_y$$

Natural log ( $\ln$ )

- We take the log to simplify for numerical stability and math simplicity
- Sometimes referred to as cross-entropy

# Regularization

- L2-regularization  $\Omega(\theta) = \sum_k \sum_i \sum_j \left( W_{i,j}^{(k)} \right)^2 = \sum_k \|\mathbf{W}^{(k)}\|_F^2$ 
  - Only applied on weights, not on biases (weight decay)
  - Can be interpreted as having a Gaussian prior on weights
- L1-regularization

$$\Omega(\theta) = \sum_k \sum_i \sum_j |W_{i,j}^{(k)}|$$

- Again, only applied on weights
- Unlike L2, L1 will push certain weights to be exactly 0
- Can be interpreted as having a Laplacian prior on weights

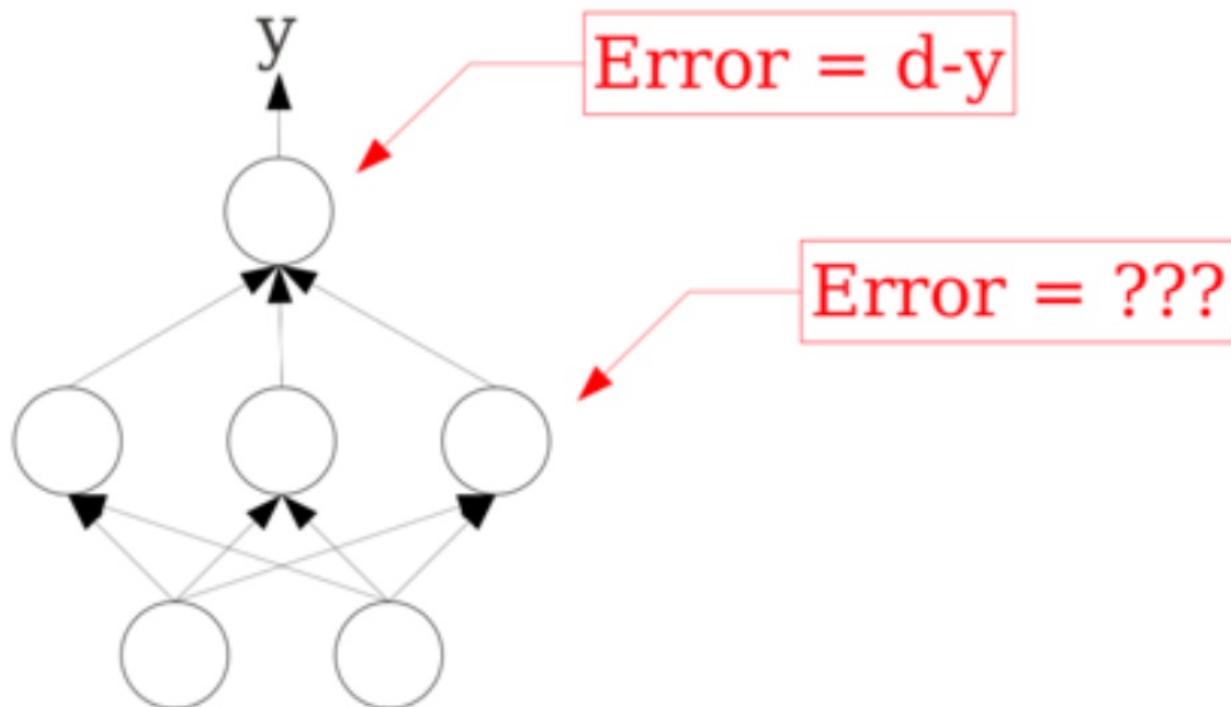
# Learning Algorithm

- Algorithm that performs updates after input each data:

$$\theta \quad (\theta \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \dots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\})$$

- Initialize  $(x^{(t)}, y^{(t)})$
  - For N iterations:  $\theta \leftarrow \theta + \alpha \Delta$
- iterate through each training example
- 
- Training epoch  
=  
Iteration over all examples

# How Do We Train A Multi-Layer Network?



$$\arg \min_{\theta} \frac{1}{T} \sum_t l(f(\mathbf{x}^{(t)}; \theta), y^{(t)}) + \lambda \Omega(\theta)$$

# Review Gradient Descent

# Gradient descent

- Based on observation that
  - If multi-variate function  $f(\boldsymbol{\theta})$  defined and differentiable in neighborhood of a point  $\boldsymbol{\theta}_0$
  - Then  $f(\boldsymbol{\theta})$  decreases fastest if one goes from  $\boldsymbol{\theta}_0$  in the direction of the negative gradient of  $f(\boldsymbol{\theta})$  at  $\boldsymbol{\theta}_0$
  - The gradient of  $f(\boldsymbol{\theta})$  in  $\boldsymbol{\theta}_0$  is denoted  $\nabla f(\boldsymbol{\theta}_0)$
  - It follows that, for small enough  $\gamma$

$$\boldsymbol{\theta}_1 = \boldsymbol{\theta}_0 - \gamma \cdot \nabla f(\boldsymbol{\theta}_0)$$

we have  $f(\boldsymbol{\theta}_1) < f(\boldsymbol{\theta}_0)$

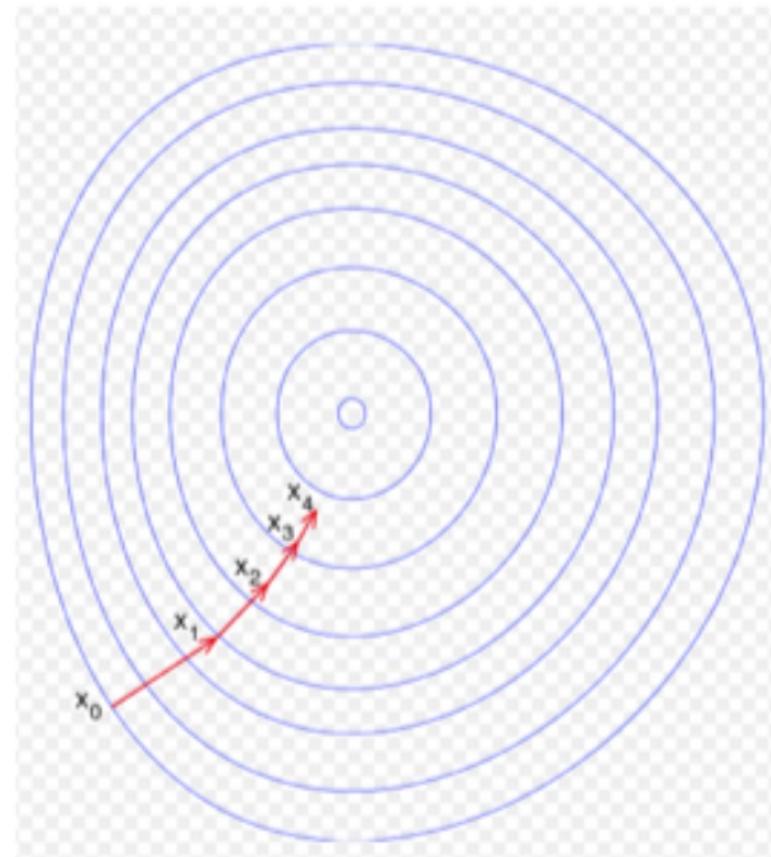
- If executing  $\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \gamma \cdot \nabla f(\boldsymbol{\theta}_n)$

we obtain  $f(\boldsymbol{\theta}_n) < f(\boldsymbol{\theta}_{n-1}) < \dots f(\boldsymbol{\theta}_2) < f(\boldsymbol{\theta}_1) < f(\boldsymbol{\theta}_0)$

Note: here the notation  $f(\cdot)$  is a general objective function to be minimised.

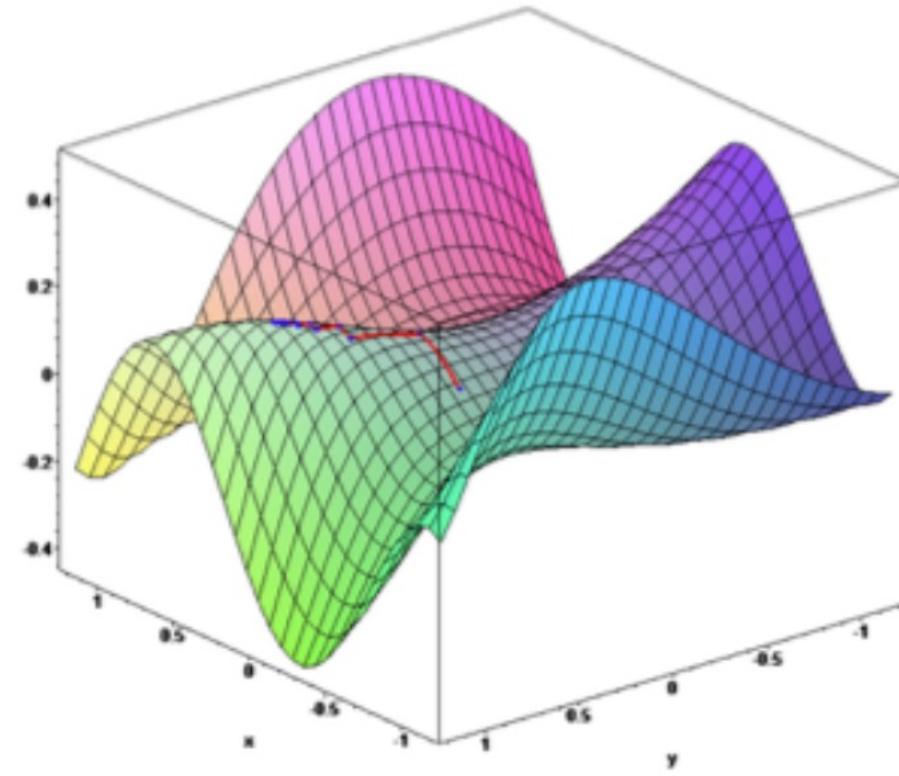
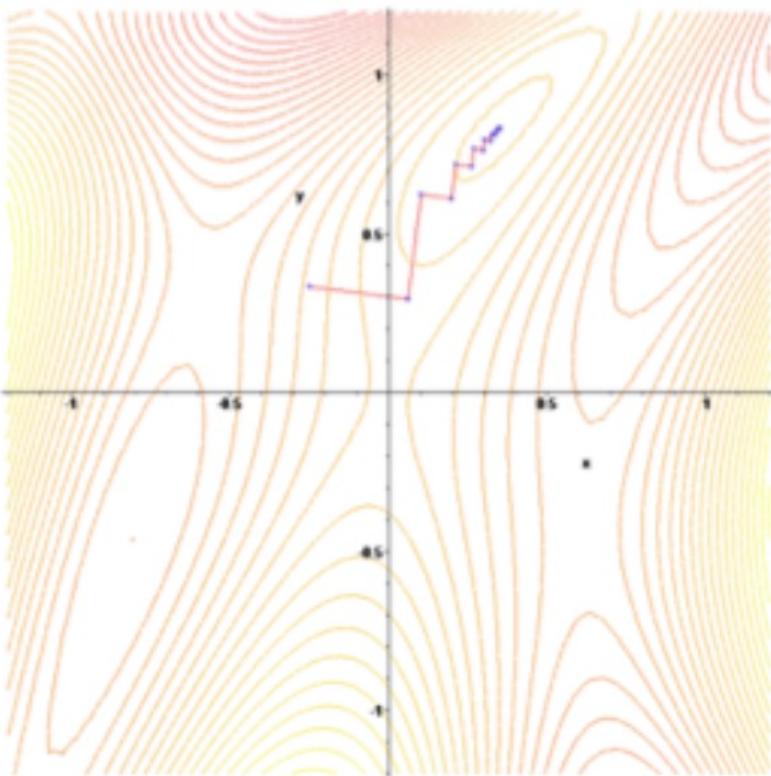
# Gradient descent

- With properly chosen  $\gamma$  in each iteration
  - Convergence to local minimum!
  - $\gamma$  can be adapted between iterations
    - or**
  - We can iterate over  $\gamma$  inside individual iterations
    - Example:  
line-search

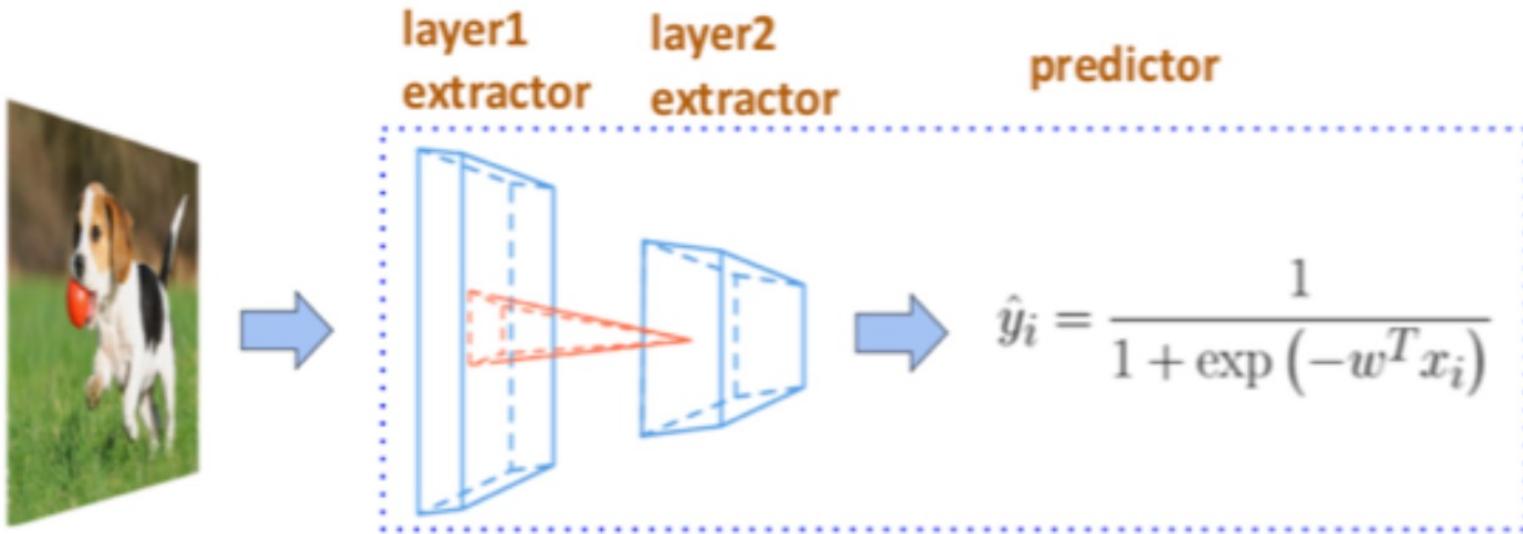


# Gradient descent

- Works in arbitrary dimensions



# Model Training Overview



**Objective**

$$L(w) = \sum_{i=1}^n l(y_i, \hat{y}_i) + \lambda \|w\|^2$$

**Training**

$$w \leftarrow w - \eta \nabla_w L(w)$$

# Learning Algorithm

- Initialize  $\theta$  ( $\theta \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \dots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$ )
  - For N iterations:
    - For each training example  $(\mathbf{x}^{(t)}, y^{(t)})$
    - $\Delta = -\nabla_{\theta} l(f(\mathbf{x}^{(t)}; \theta), y^{(t)}) - \lambda \nabla_{\theta} \Omega(\theta)$
    - $\theta \leftarrow \theta + \alpha \Delta$
  - To apply this algorithm to neural network training, we need
    - The loss function  $l(f(\mathbf{x}^{(t)}; \theta), y^{(t)})$
    - Procedure to compute parameter gradients  $\nabla_{\theta} l(f(\mathbf{x}^{(t)}; \theta), y^{(t)})$
    - The regularizer  $\Omega(\theta)$  (and the gradient  $\nabla_{\theta} \Omega(\theta)$ )
    - Initialization method
- Training epoch  
=  
Iteration over all examples

# Back-propagation learning

# Training deep neural networks

- Outlook:
  - Computing gradients is hard (many parameters)  
→ Back-propagation!
- Network represents a chain of function calls (one per layer):

$$\begin{aligned} \mathbf{f}(\mathbf{x}) &= \mathbf{f}_{L+1} \circ \mathbf{f}_L \circ \cdots \circ \mathbf{f}_1(\mathbf{x}) \\ &= \mathbf{f}_{L+1}(\mathbf{f}_L(\dots \mathbf{f}_1(\mathbf{x}))) \end{aligned}$$

- Gradients: Chain rule can be applied!

$$\frac{\partial \mathbf{f}(\mathbf{x})}{\partial \boldsymbol{\theta}} = \frac{\partial \mathbf{f}_{L+1}}{\partial \mathbf{f}_L} \cdot \frac{\partial \mathbf{f}_L}{\partial \mathbf{f}_{L-1}} \cdot \dots \cdot \frac{\partial \mathbf{f}_1}{\partial \boldsymbol{\theta}}$$

# How to compute gradient?

## Symbolic Differentiation

- Input formulae is a symbolic expression tree (computation graph).
- Implement differentiation rules, e.g., sum rule, product rule, chain rule

$$\frac{d(f + g)}{dx} = \frac{df}{dx} + \frac{dg}{dx} \quad \frac{d(fg)}{dx} = \frac{df}{dx}g + f\frac{dg}{dx} \quad \frac{d(h(x))}{dx} = \frac{df(g(x))}{dx} \cdot \frac{dg(x)}{x}$$

- ✖ For complicated functions, the resultant expression can be exponentially large.
- ✖ Wasteful to keep around intermediate symbolic expressions if we only need a numeric value of the gradient in the end
- ✖ Prone to error

# How to compute gradient?

## Numerical Differentiation

- We can approximate the gradient using

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \lim_{h \rightarrow 0} \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x})}{h}$$

$$f(W, x) = W \cdot x$$
$$[-0.8 \quad 0.3] \cdot \begin{bmatrix} 0.5 \\ -0.2 \end{bmatrix}$$

# How to compute gradient?

## Numerical Differentiation

- We can approximate the gradient using

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \lim_{h \rightarrow 0} \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x})}{h}$$

$$f(W, x) = W \cdot x$$

$$[-0.8 \quad 0.3] \cdot \begin{bmatrix} 0.5 \\ -0.2 \end{bmatrix}$$

$$f(W, x) = W \cdot x$$

$$[-0.8 + \varepsilon \quad 0.3] \cdot \begin{bmatrix} 0.5 \\ -0.2 \end{bmatrix}$$

# How to compute gradient?

## Numerical Differentiation

- We can approximate the gradient using

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \lim_{h \rightarrow 0} \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x})}{h}$$

- Reduce the truncation error by using center difference

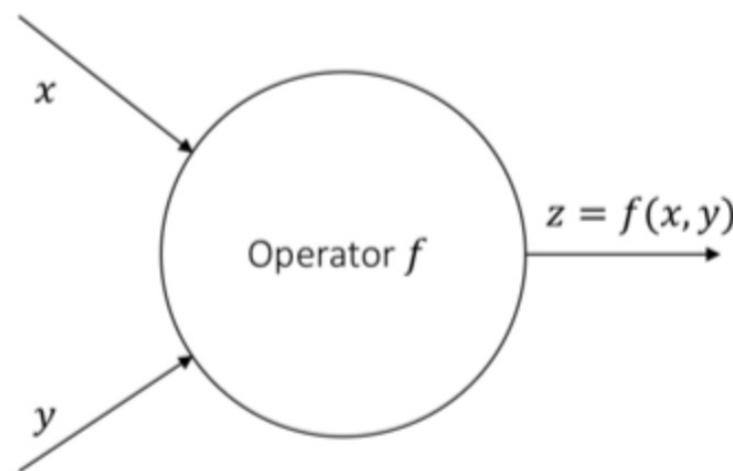
$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \lim_{h \rightarrow 0} \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x} - h\mathbf{e}_i)}{2h}$$

✗ Bad: rounding error, and slow to compute

✓ A powerful tool to check the correctness of implementation, usually use  $h = 1e-6$ .

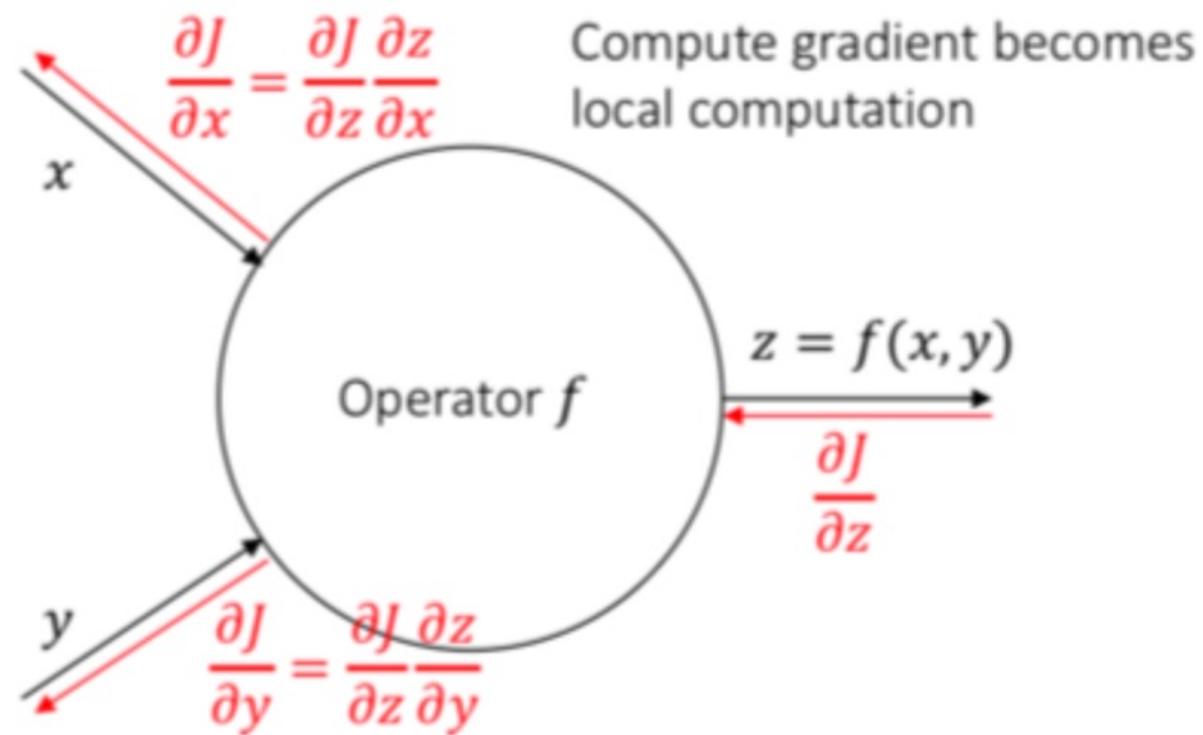
# Back Propagation for computing differentiation

Backpropagation



# Back Propagation for computing differentiation

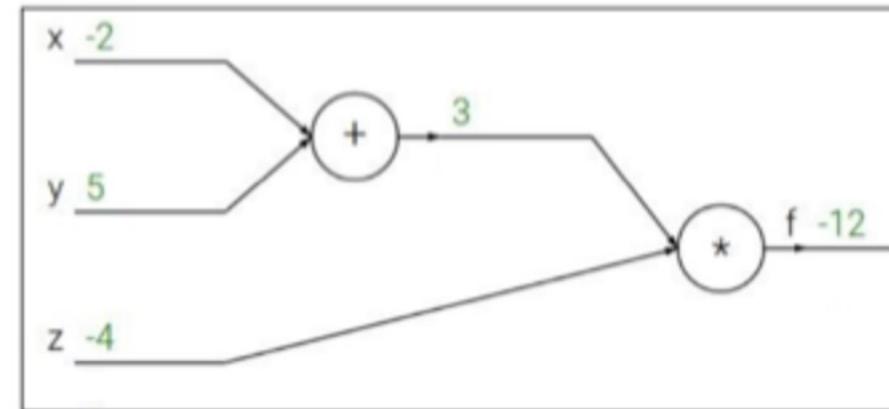
## Backpropagation



# Example: Back-Prop

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

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



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

- slide source: Stanford CS231n lecture note.

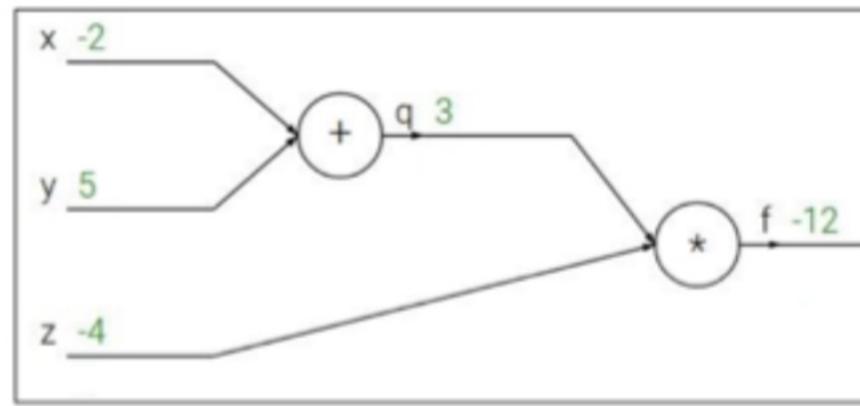
# Example: Back-Prop

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

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

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

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



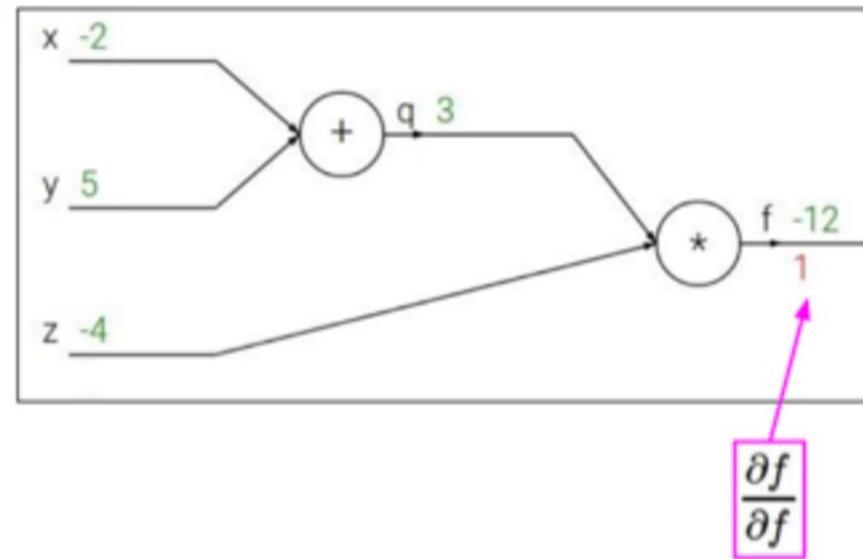
# Example: Back-Prop

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

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

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

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



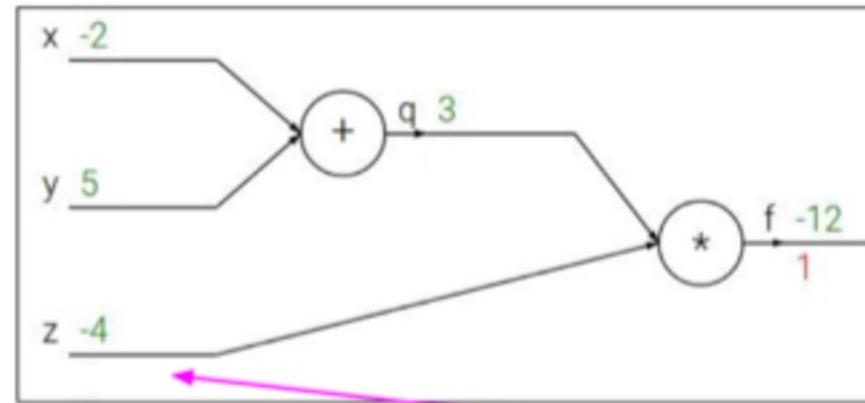
# Example: Back-Prop

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

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

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

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



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

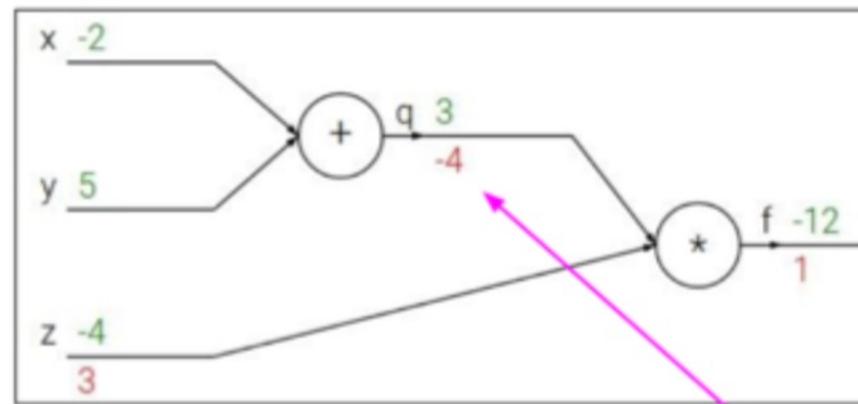
# Example: Back-Prop

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

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

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

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



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

# Example: Back-Prop

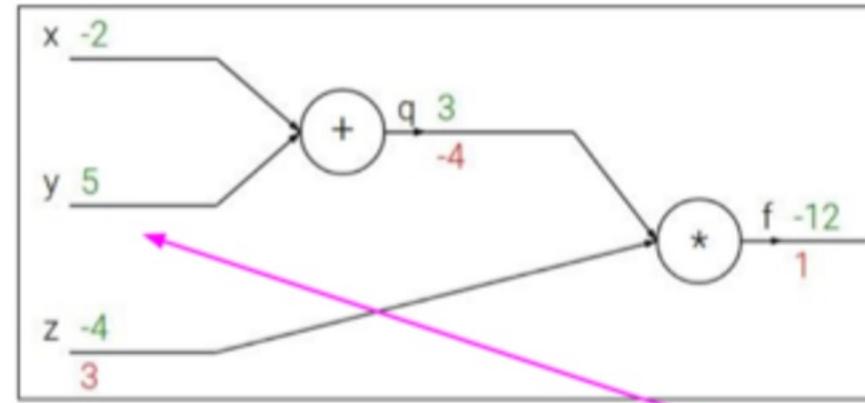
$$f(x, y, z) = (x + y)z$$

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

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

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

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



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

# Example: Back-Prop

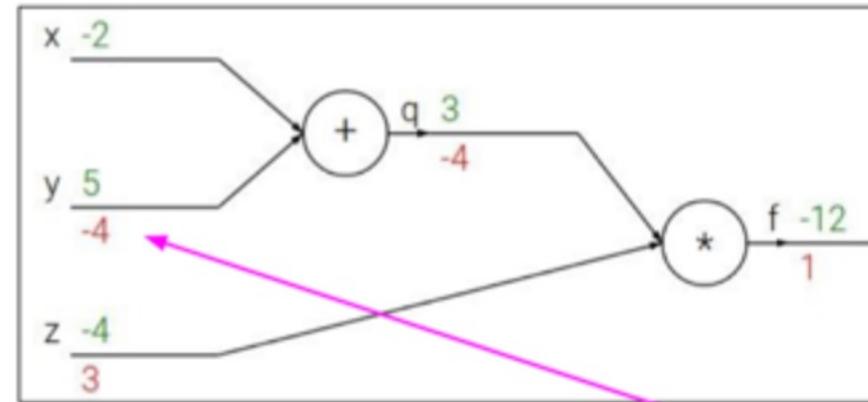
$$f(x, y, z) = (x + y)z$$

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

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

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

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



Chain rule:

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

Upstream  
Gradient

Local  
Gradient

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

# Example: Back-Prop

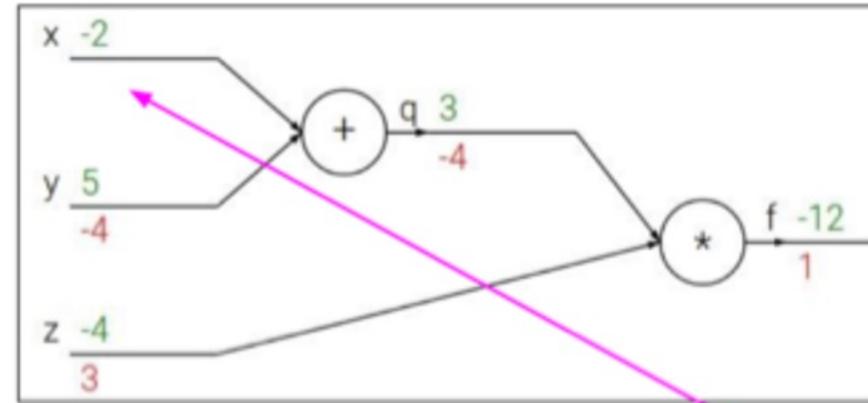
$$f(x, y, z) = (x + y)z$$

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

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

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

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



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

# Example: Back-Prop

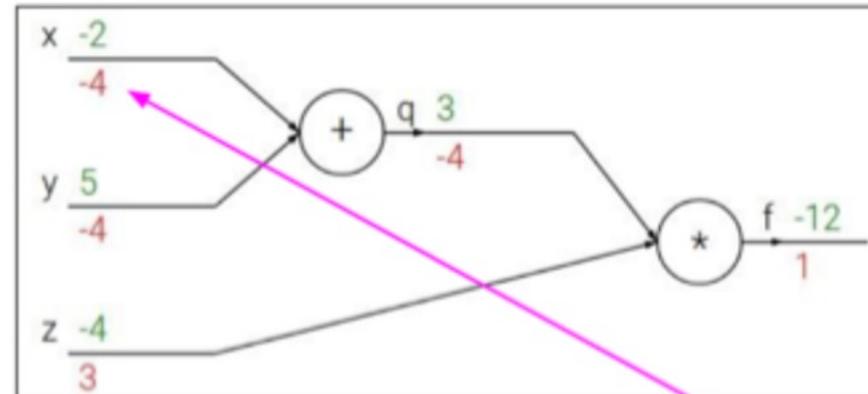
$$f(x, y, z) = (x + y)z$$

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

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

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

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



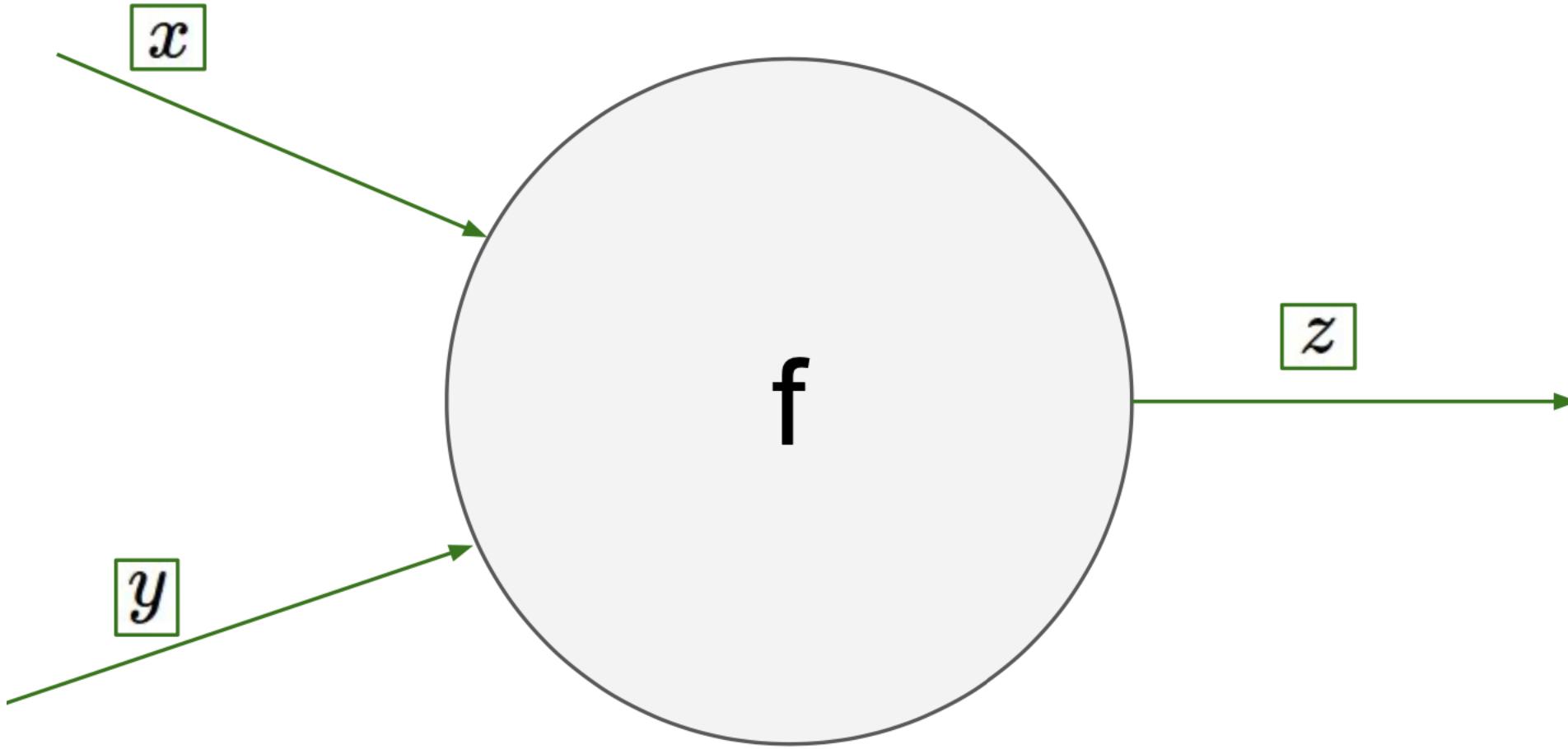
$$\frac{\partial f}{\partial x}$$

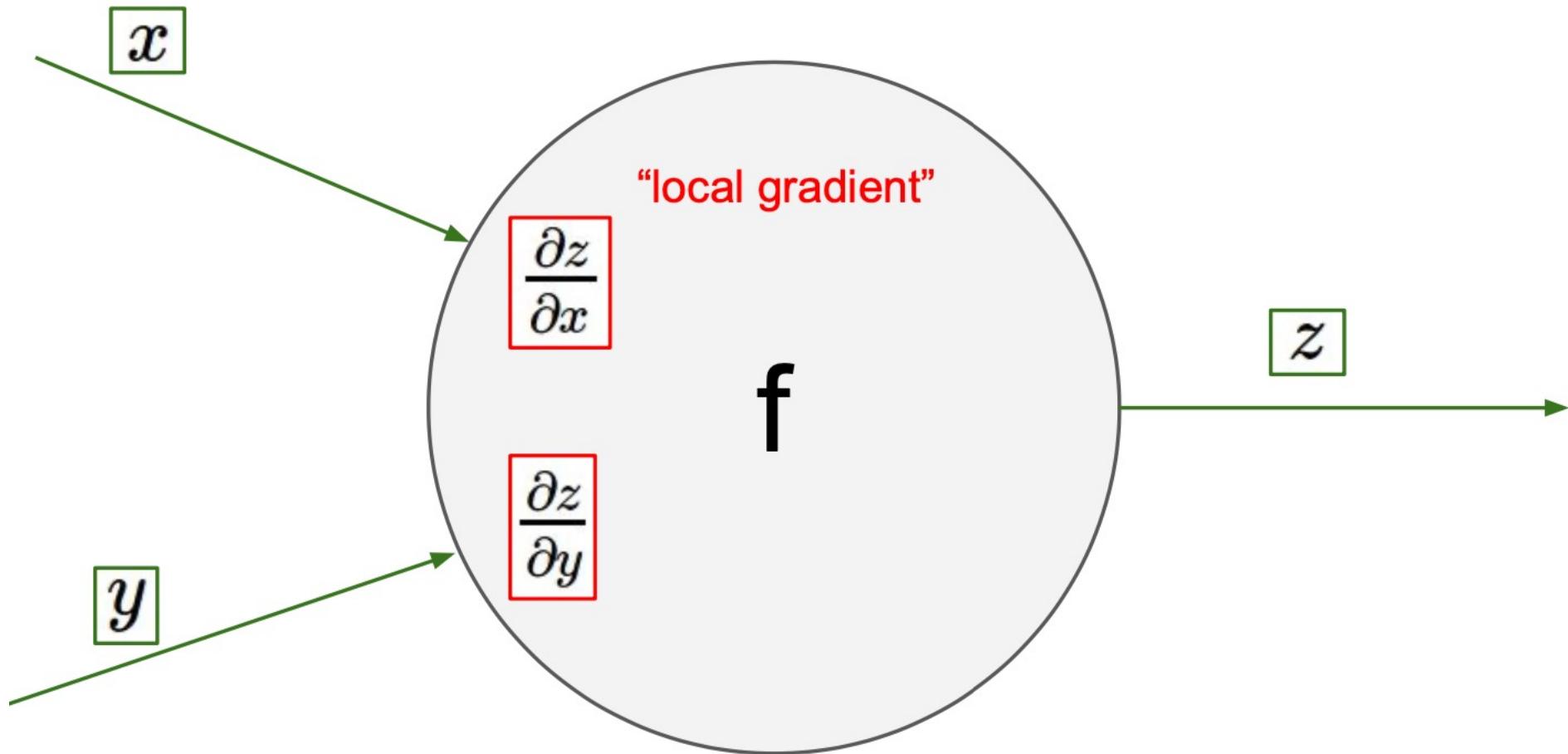
Chain rule:

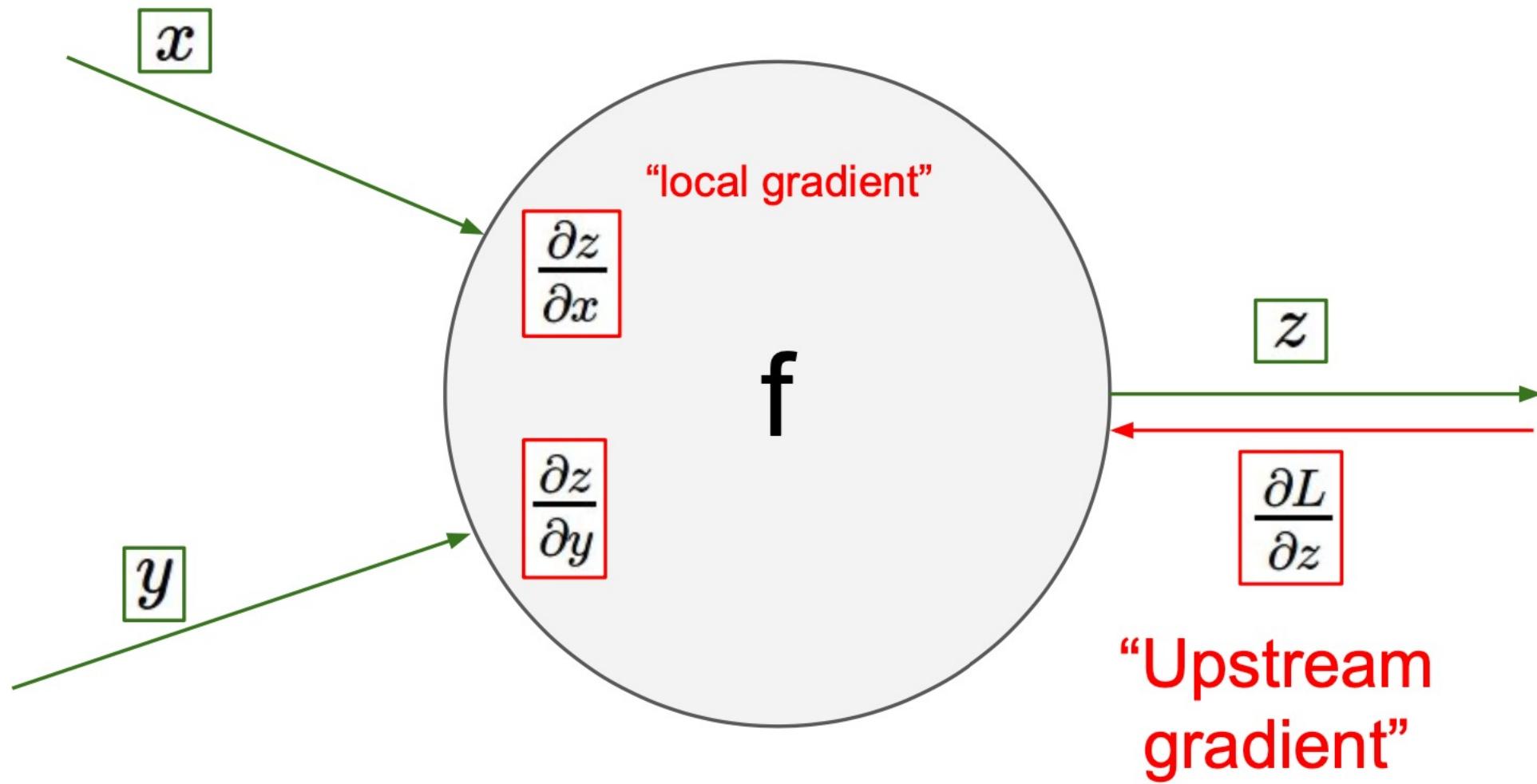
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

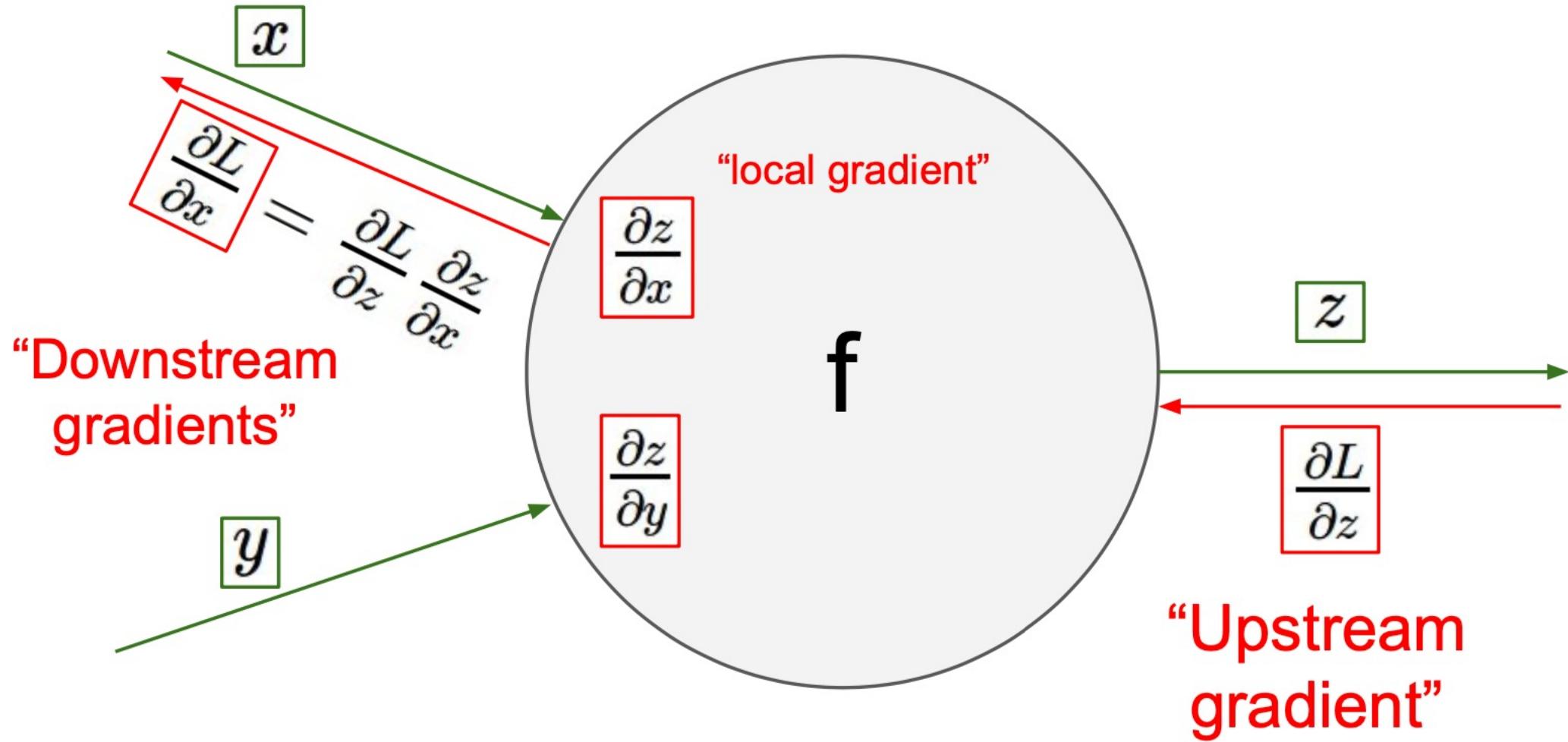
Upstream  
Gradient

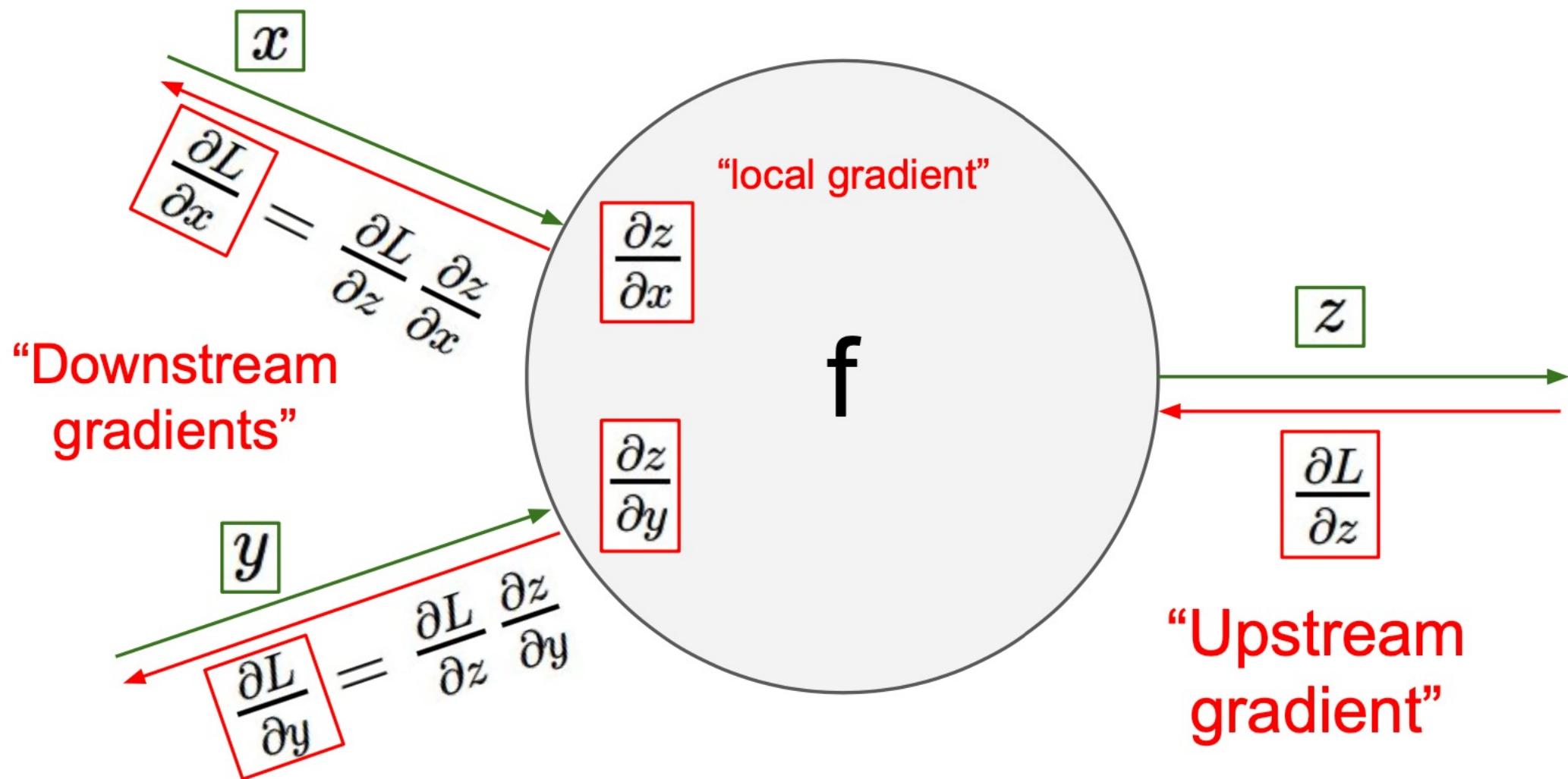
Local  
Gradient

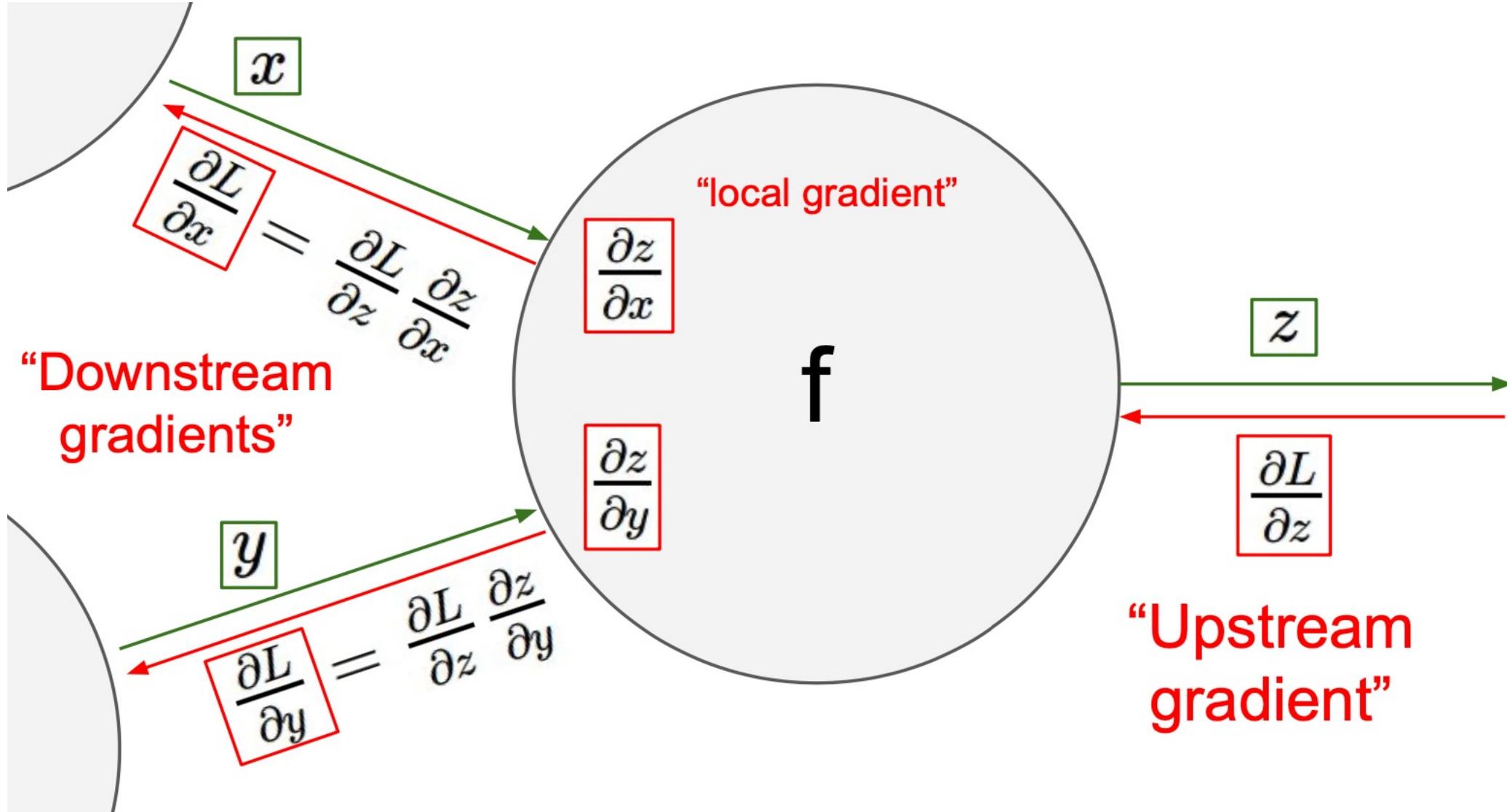






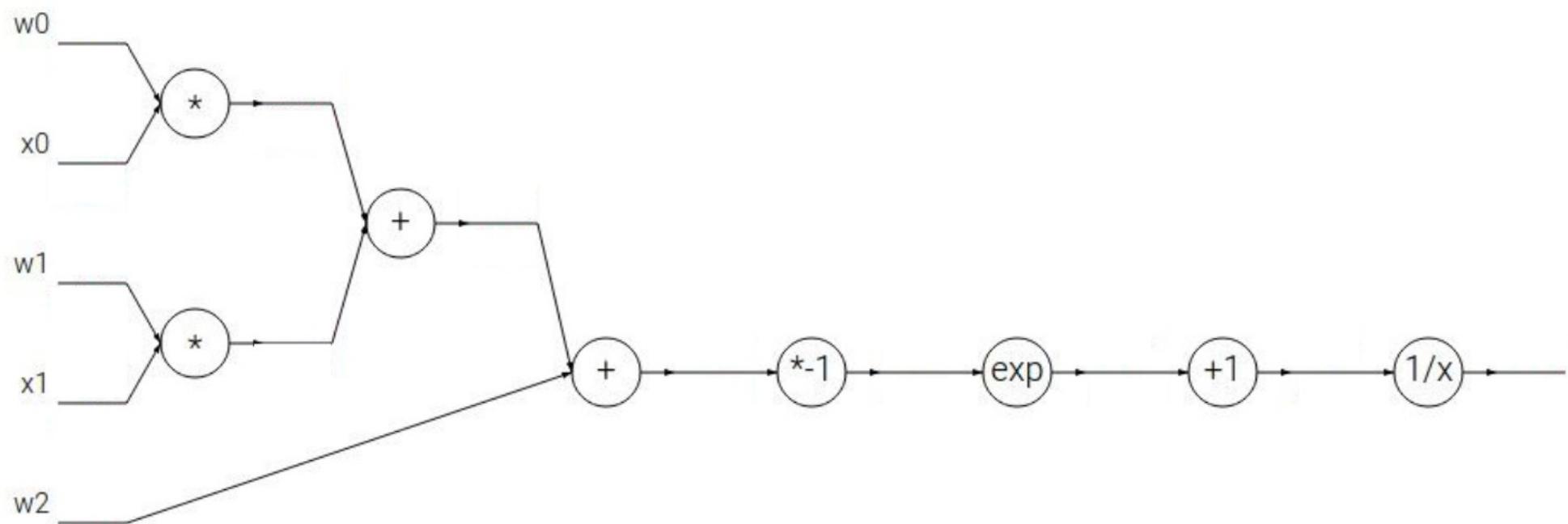






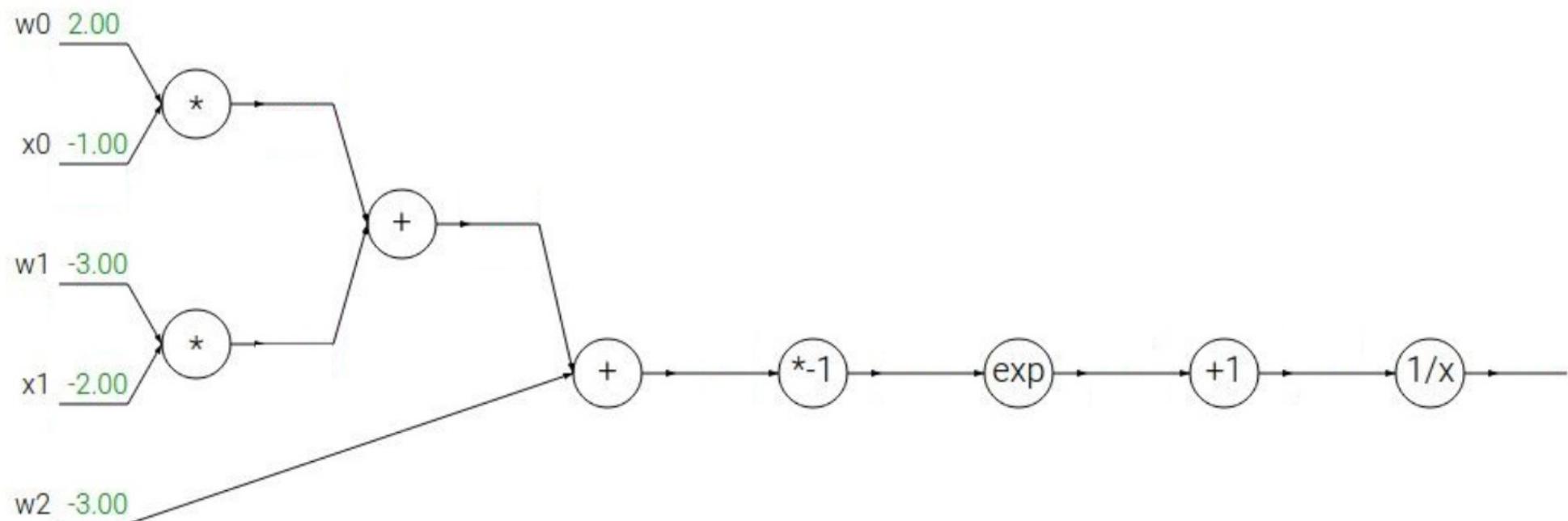
**Another example:**

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



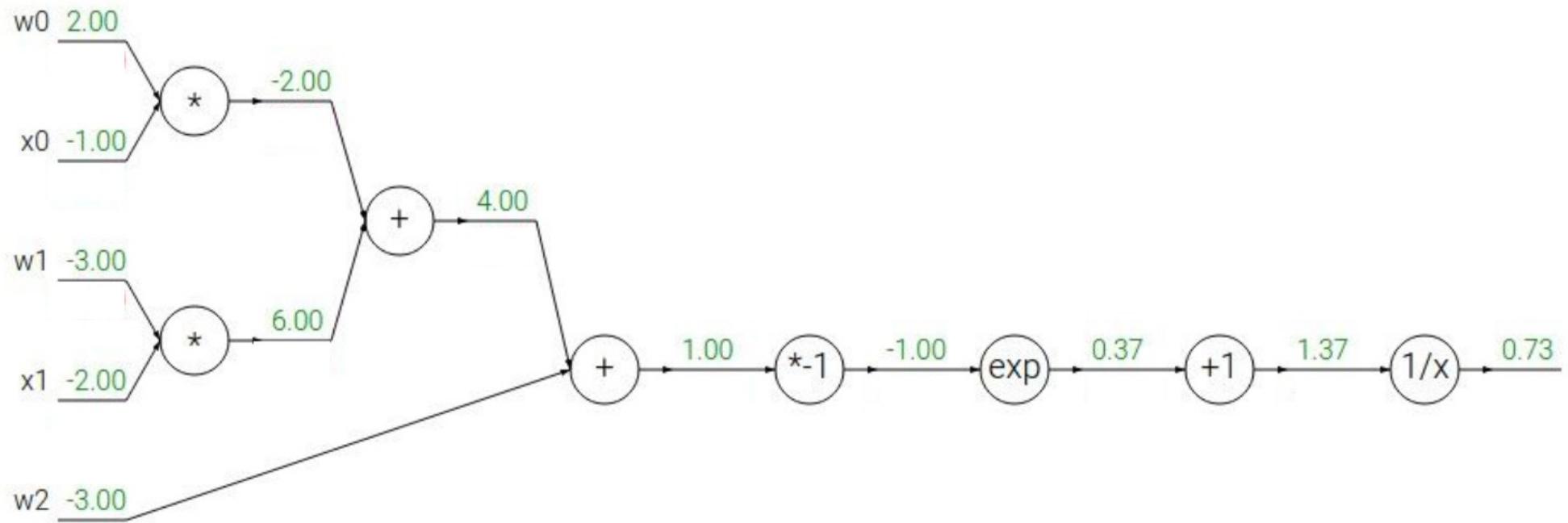
Another example:

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



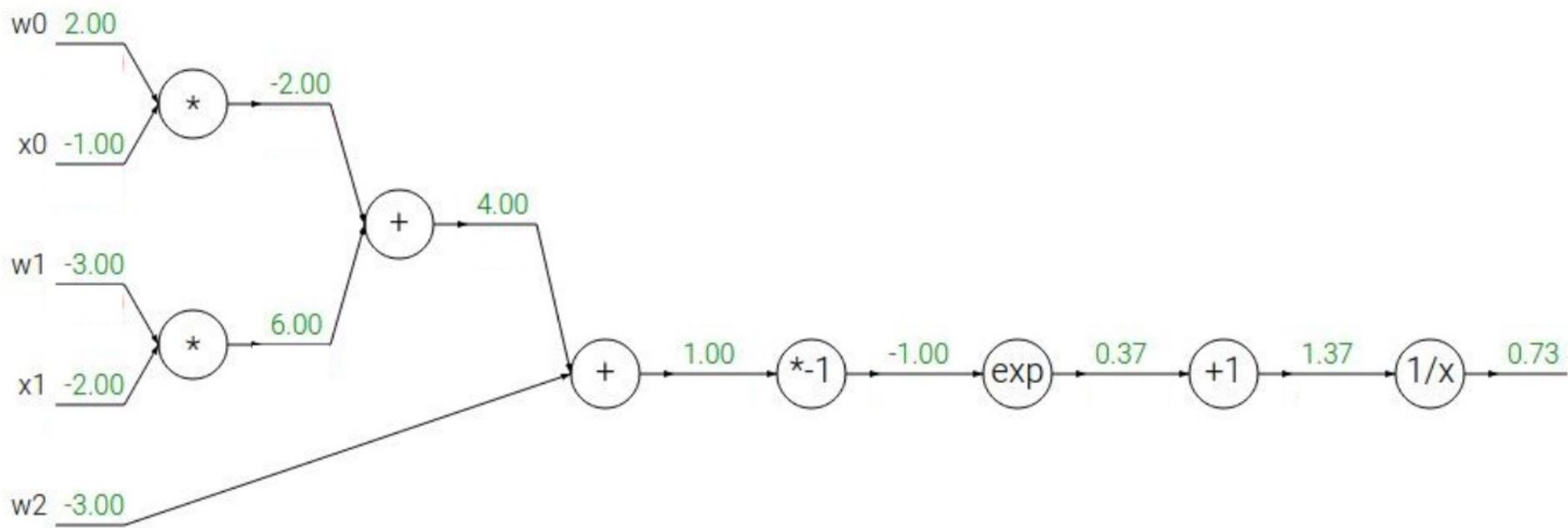
Another example:

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



Another example:

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



$$f(x) = e^x$$

→

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

$$f_a(x) = ax$$

→

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$$f_c(x) = c + x$$

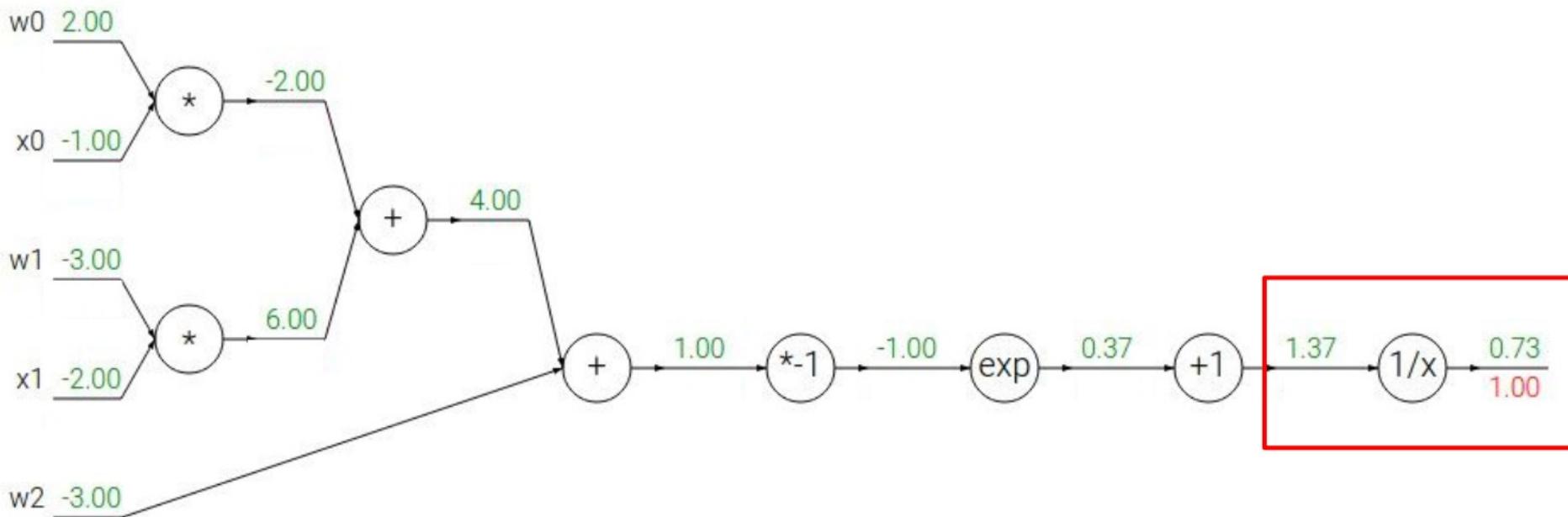
→

$$\frac{df}{dx} = -1/x^2$$

$$\frac{df}{dx} = 1$$

Another example:

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



$$f(x) = e^x$$

→

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

$$f_a(x) = ax$$

→

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

→

$$\frac{df}{dx} = -1/x^2$$

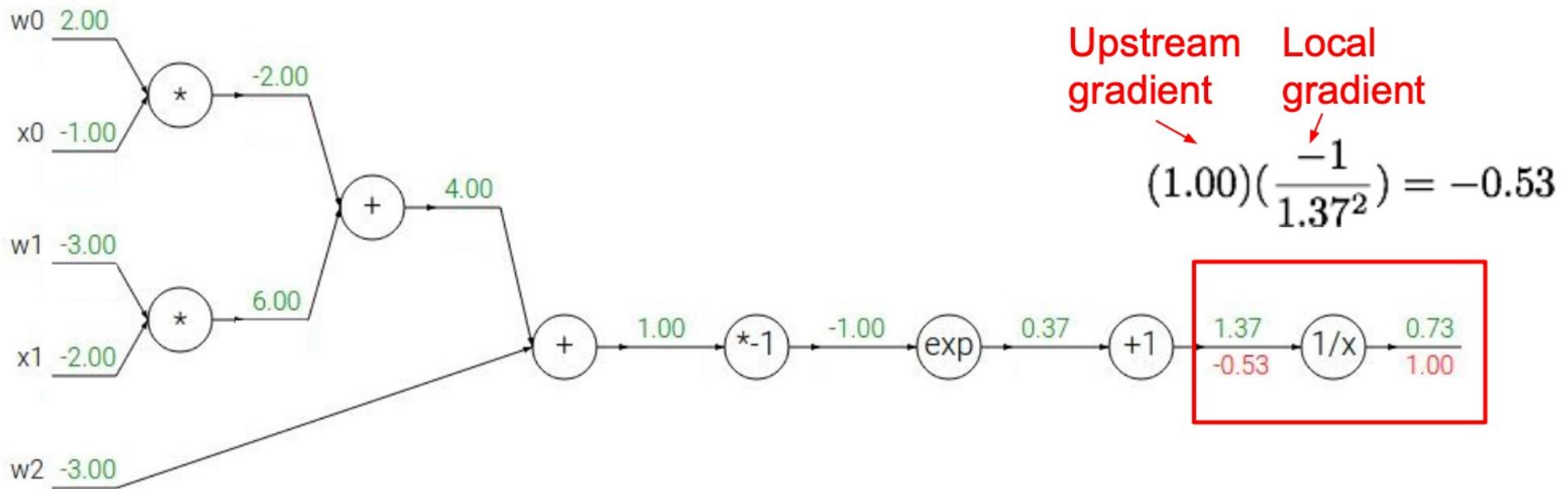
$$f_c(x) = c + x$$

→

$$\frac{df}{dx} = 1$$

Another example:

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



$$f(x) = e^x$$

→

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

$$f_a(x) = ax$$

→

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

→

$$\frac{df}{dx} = -1/x^2$$

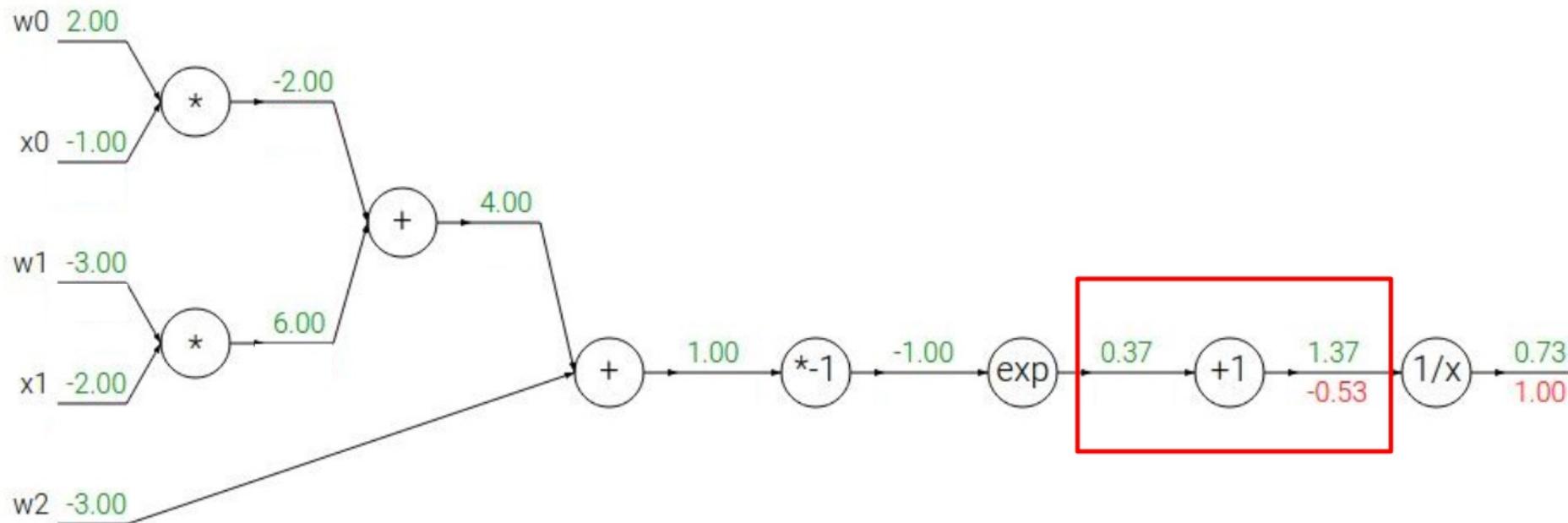
$$f_c(x) = c + x$$

→

$$\frac{df}{dx} = 1$$

Another example:

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



$$f(x) = e^x$$

→

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

$$f_a(x) = ax$$

→

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

→

$$\frac{df}{dx} = -1/x^2$$

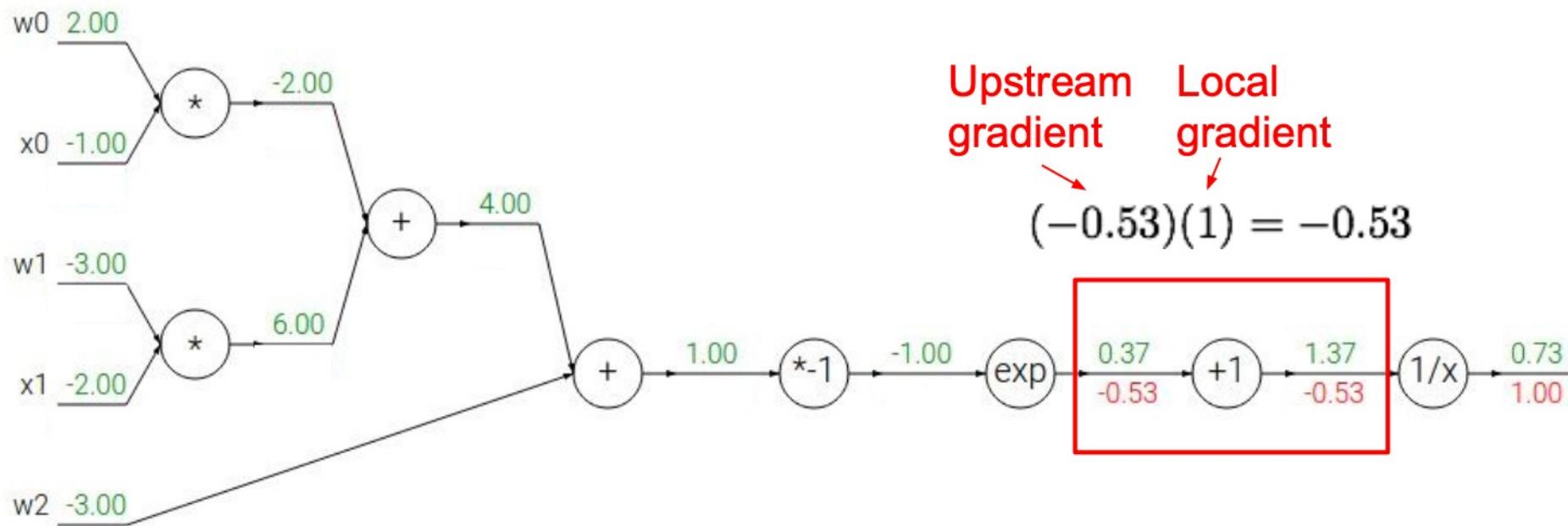
$$f_c(x) = c + x$$

→

$$\frac{df}{dx} = 1$$

Another example:

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



$$f(x) = e^x$$

→

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

$$f_a(x) = ax$$

→

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

→

$$\frac{df}{dx} = -1/x^2$$

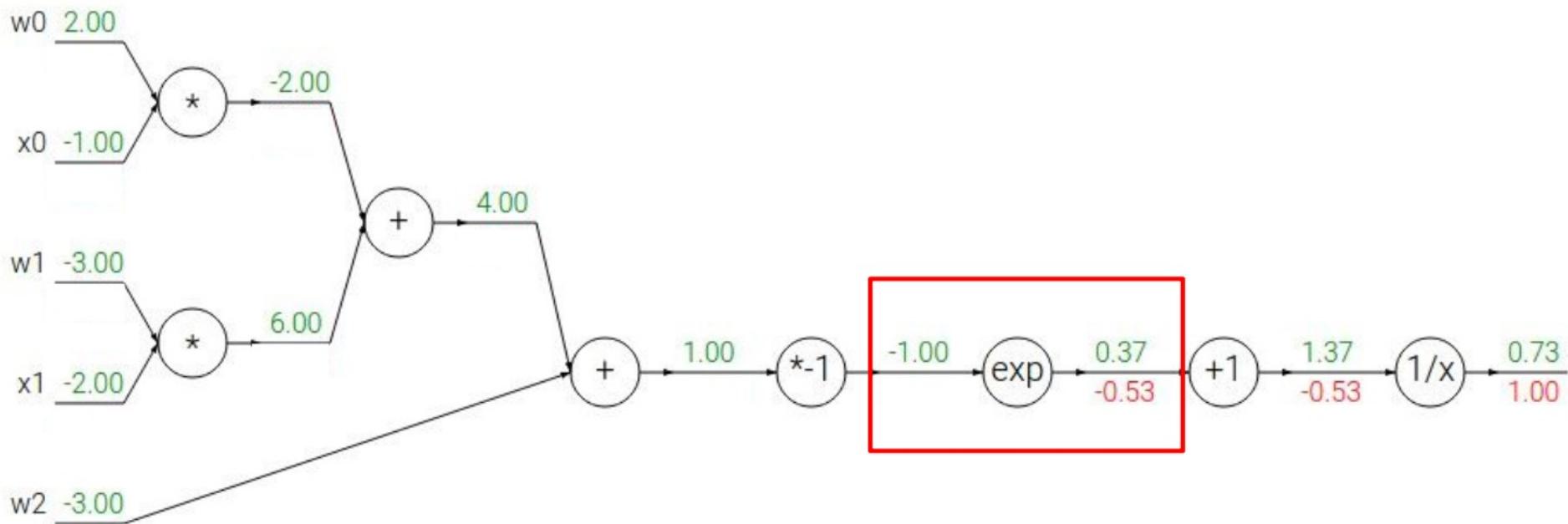
$$f_c(x) = c + x$$

→

$$\frac{df}{dx} = 1$$

Another example:

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



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

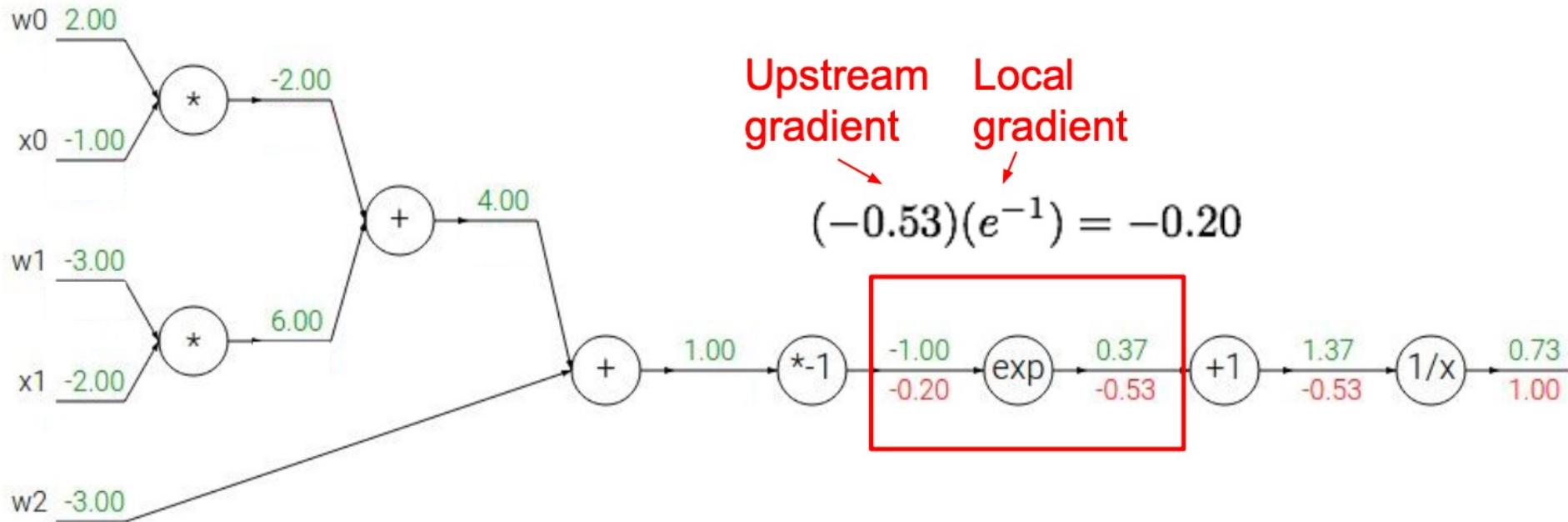
$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

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

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

Another example:

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



$$f(x) = e^x$$

$\rightarrow$

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

$$f_a(x) = ax$$

$\rightarrow$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$$f_c(x) = c + x$$

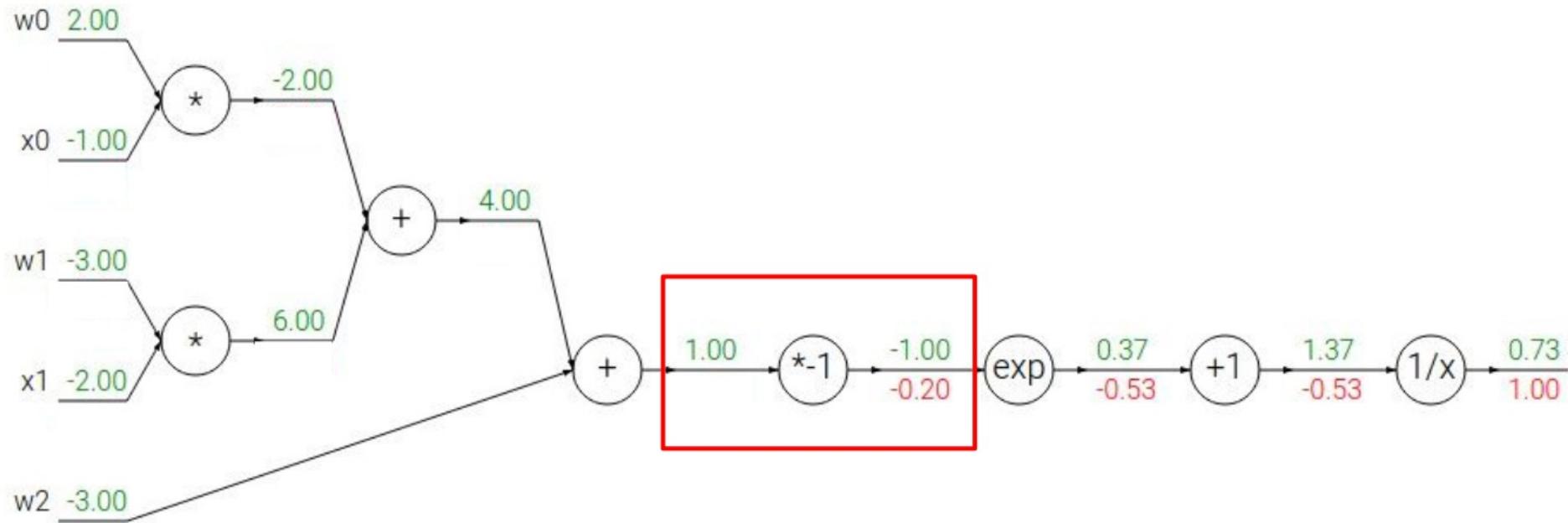
$\rightarrow$

$$\frac{df}{dx} = -1/x^2$$

$$\frac{df}{dx} = 1$$

Another example:

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



$$f(x) = e^x$$

$\rightarrow$

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

$$f_a(x) = ax$$

$\rightarrow$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$$f_c(x) = c + x$$

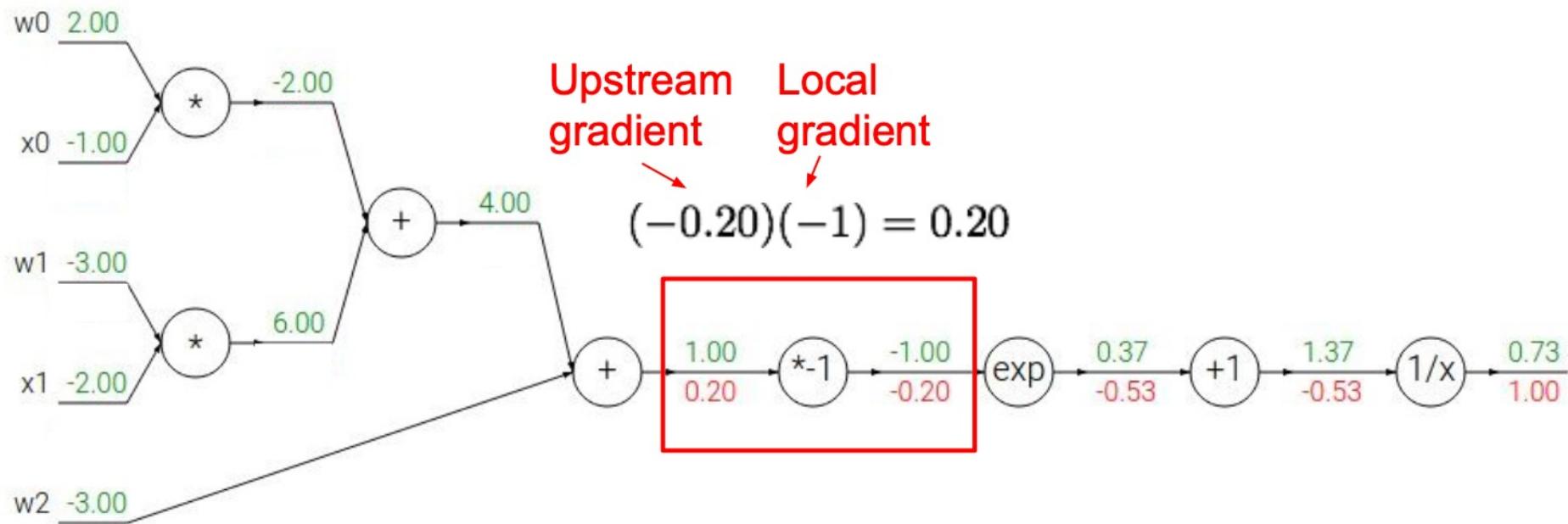
$\rightarrow$

$$\frac{df}{dx} = -1/x^2$$

$$\frac{df}{dx} = 1$$

Another example:

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



$$f(x) = e^x$$

$\rightarrow$

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

$$f_a(x) = ax$$

$\rightarrow$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$\rightarrow$

$$\frac{df}{dx} = -1/x^2$$

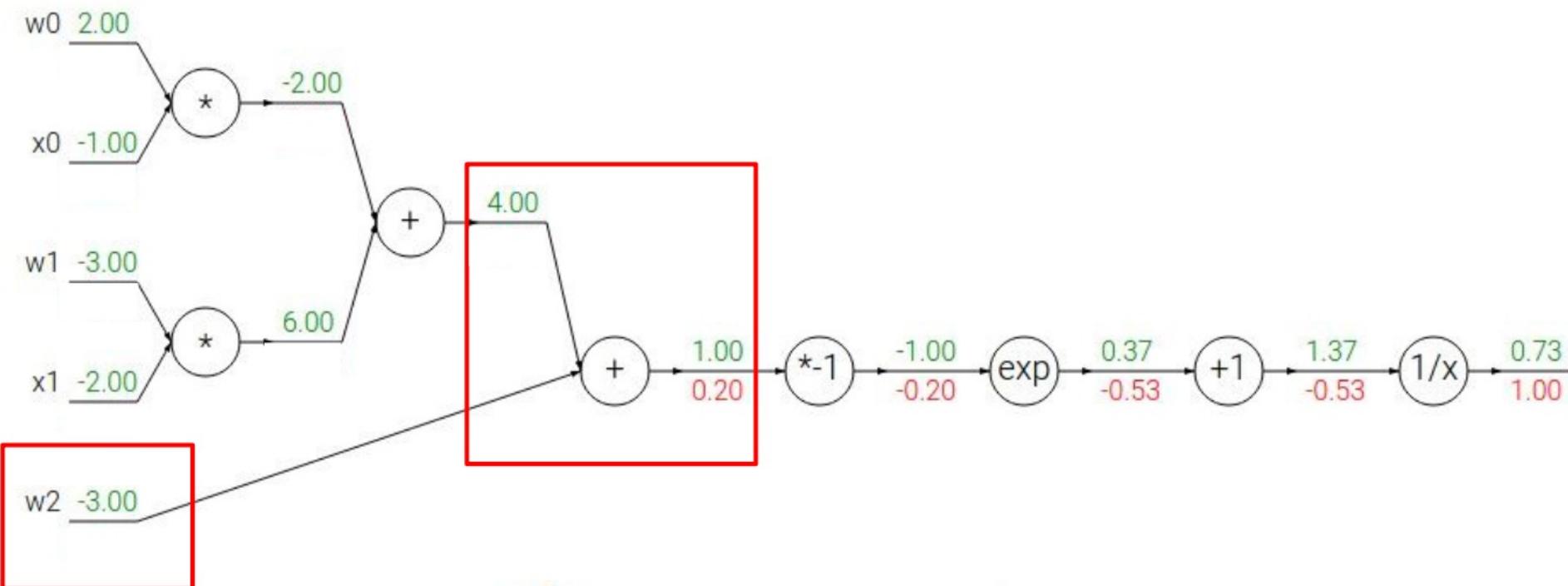
$$f_c(x) = c + x$$

$\rightarrow$

$$\frac{df}{dx} = 1$$

Another example:

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



$$f(x) = e^x \rightarrow$$

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

$$f_a(x) = ax \rightarrow$$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow$$

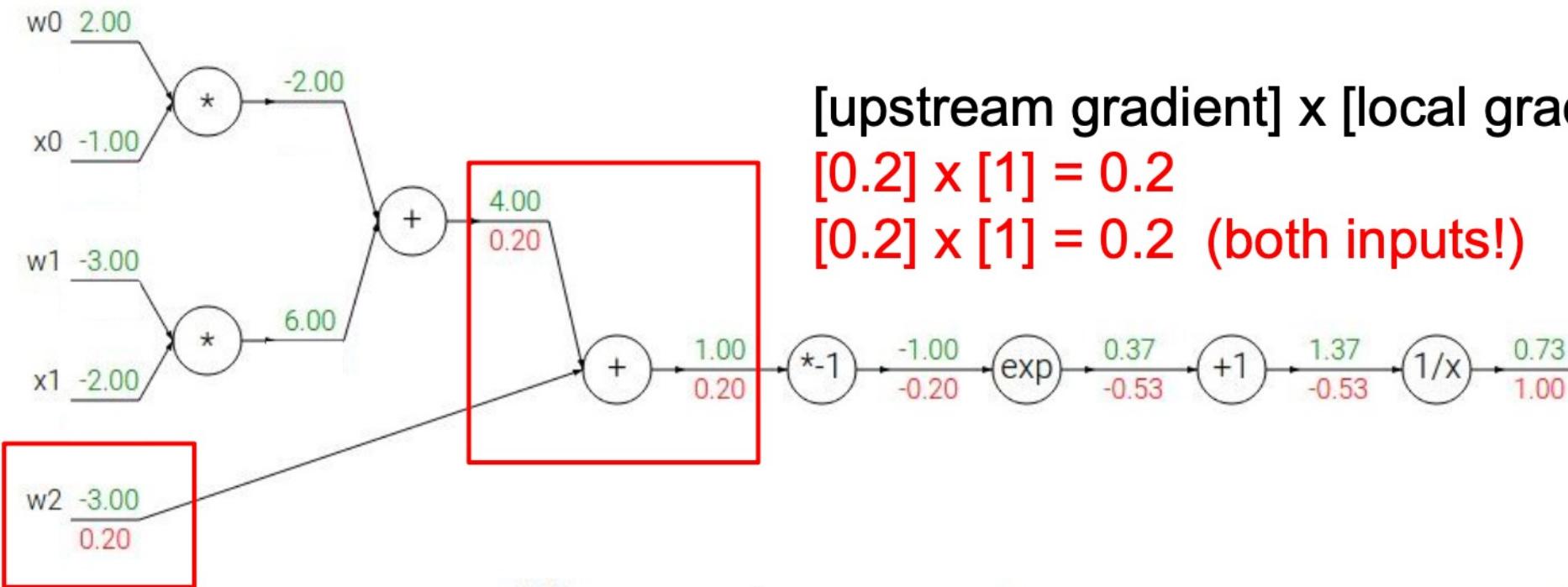
$$f_c(x) = c + x$$

$$\frac{df}{dx} = -1/x^2$$

$$\frac{df}{dx} = 1$$

Another example:

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



$$f(x) = e^x$$

→

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

$$f_a(x) = ax$$

→

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$$f_c(x) = c + x$$

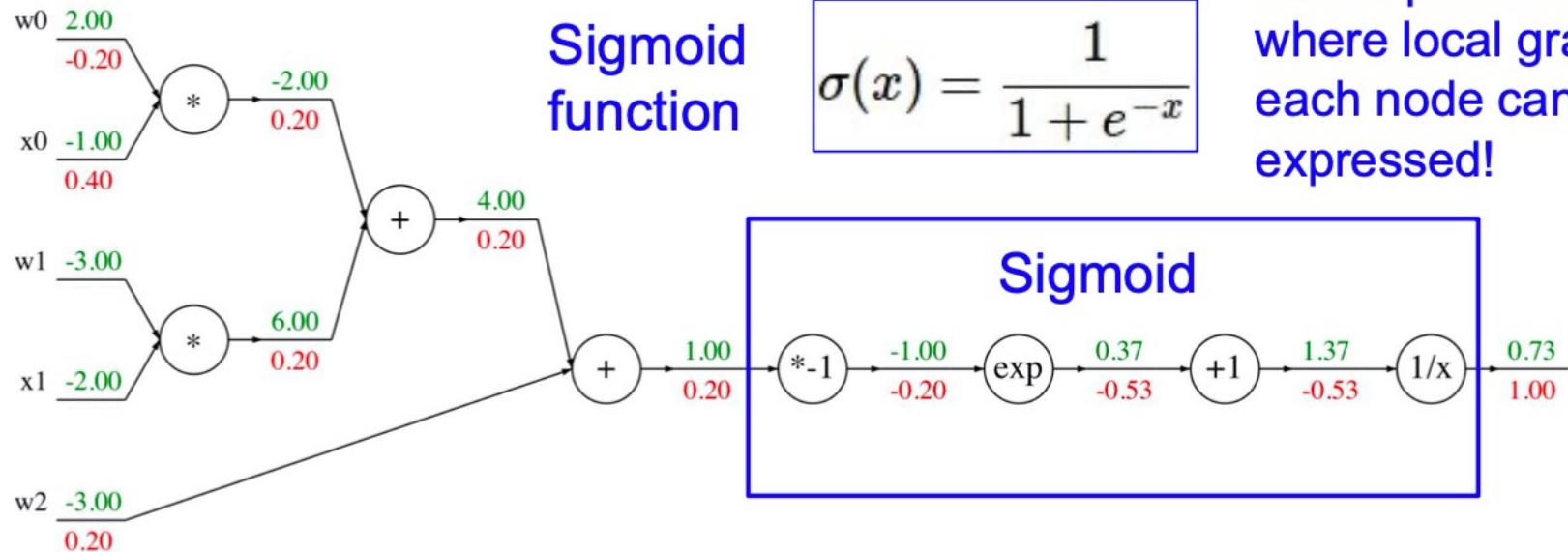
→

$$\frac{df}{dx} = -1/x^2$$

$$\frac{df}{dx} = 1$$

## Another example:

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



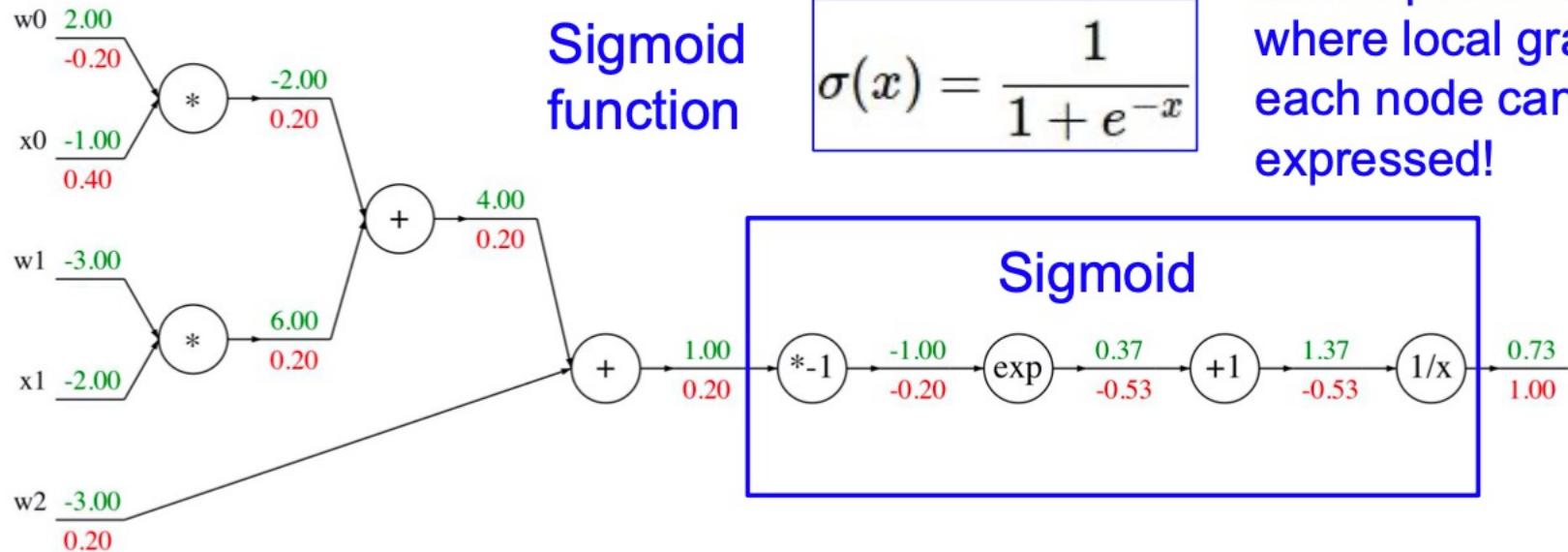
Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

## Another example:

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

Sigmoid  
function

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



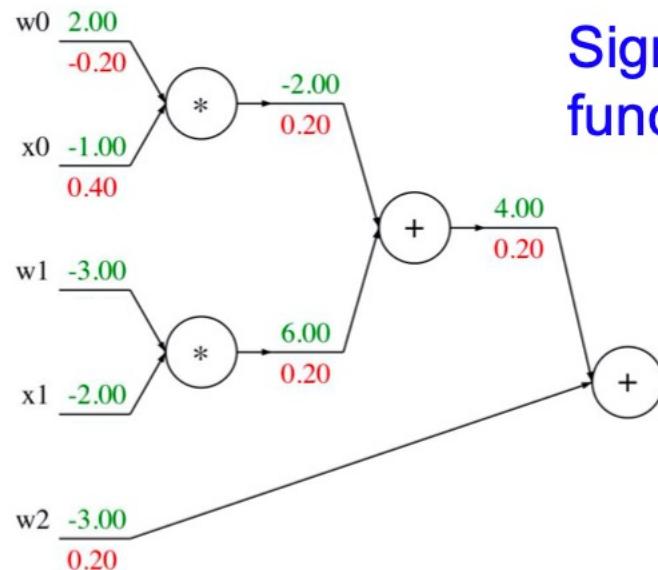
Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

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:

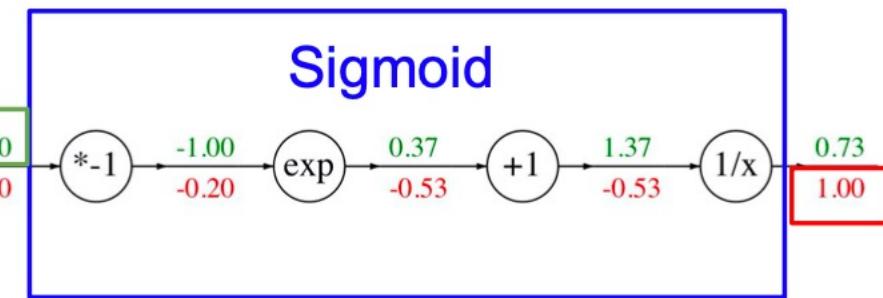
$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



Sigmoid  
function

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

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



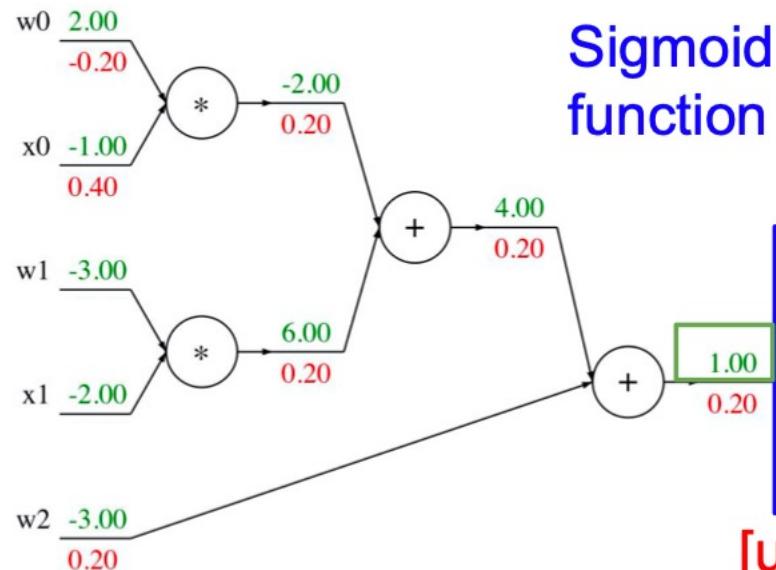
[upstream gradient]  $\times$  [local gradient]  
 $[1.00] \times [(1 - 1/(1+e^1)) (1/(1+e^1))] = 0.2$

Sigmoid local  
gradient:

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

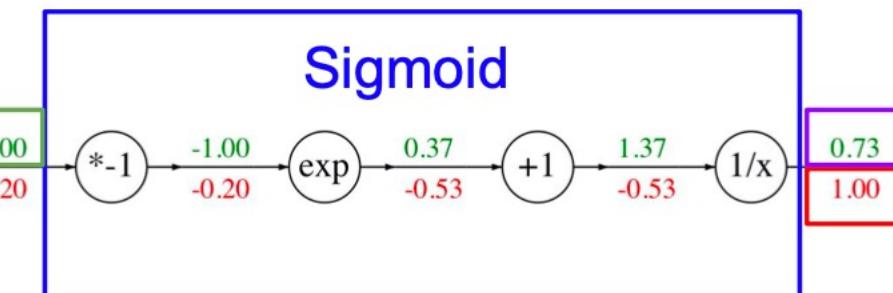
## Another example:

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



Sigmoid function

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



[upstream gradient]  $\times$  [local gradient]  
 $[1.00] \times [(1 - 0.73)(0.73)] = 0.2$

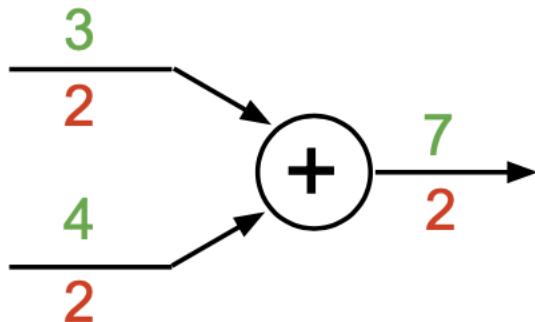
Sigmoid local gradient:

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

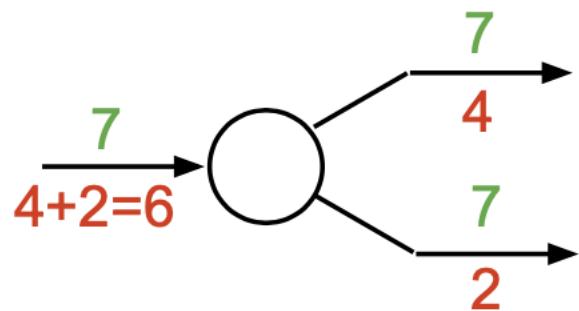
Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

# Patterns in gradient flow

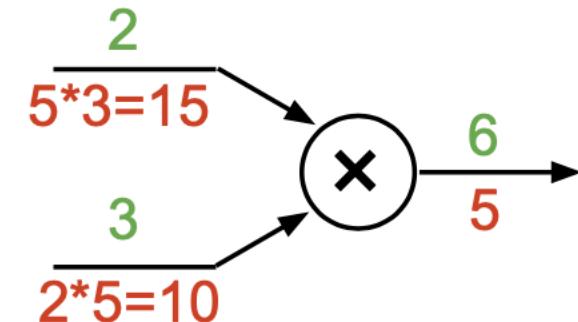
**add gate:** gradient distributor



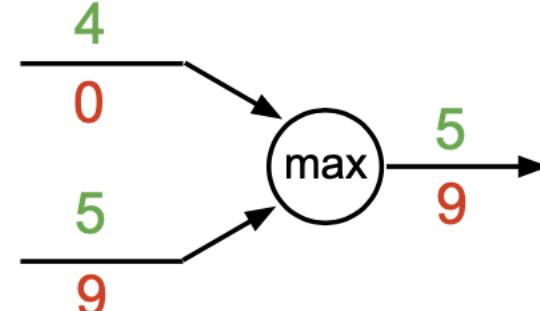
**copy gate:** gradient adder



**mul gate:** “swap multiplier”



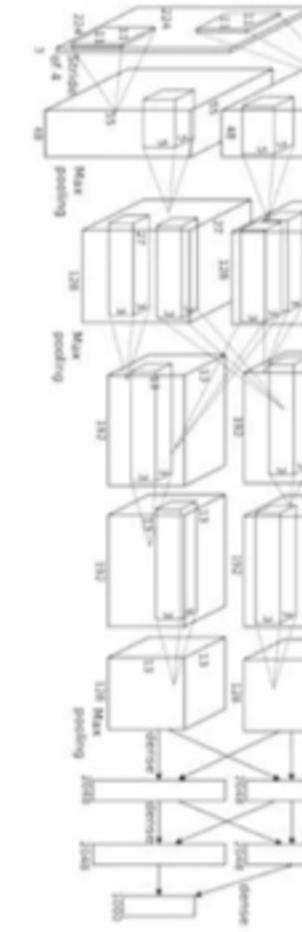
**max gate:** gradient router



Can be applied to arbitrarily complex deep networks

Convolutional Network  
(AlexNet)

input image  
weights  
loss



# Overall Steps of BP

After choosing the weights of the network randomly, the backpropagation algorithm is used to compute the necessary corrections. The algorithm can be decomposed in the following four steps:

- i) Feed-forward computation
- ii) Backpropagation to the output layer
- iii) Backpropagation to the hidden layer
- iv) Weight updates

The algorithm is stopped when the value of the error function has become sufficiently small.

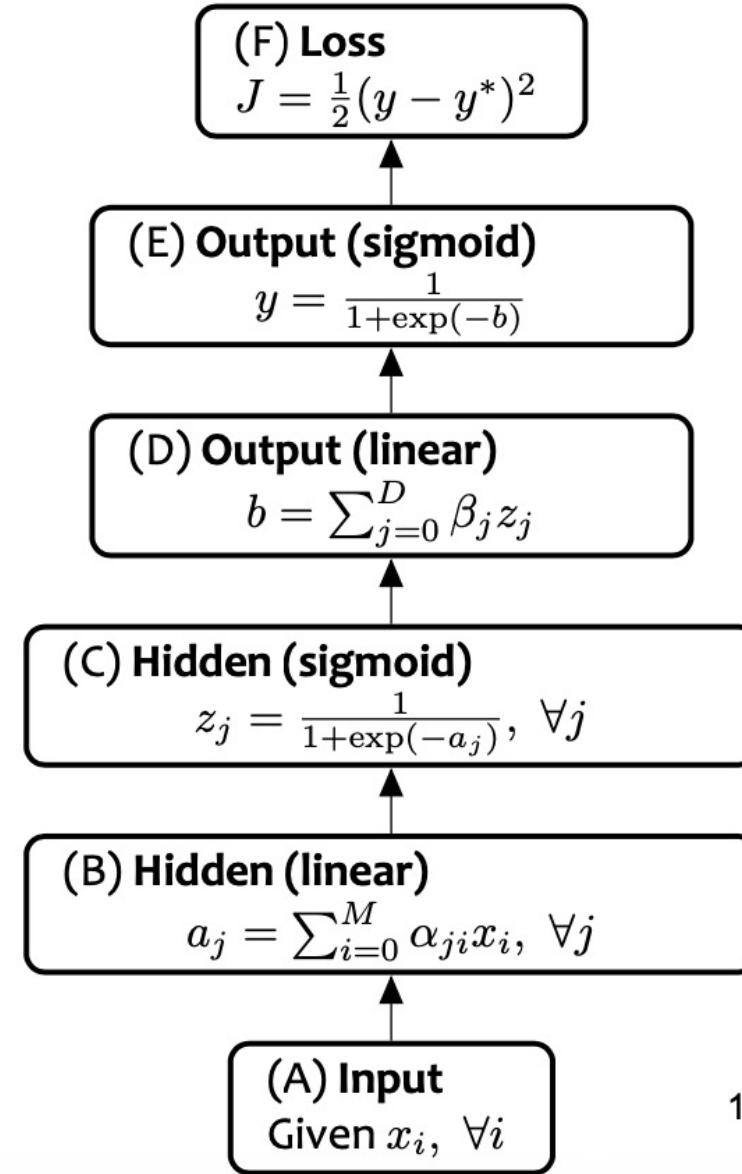
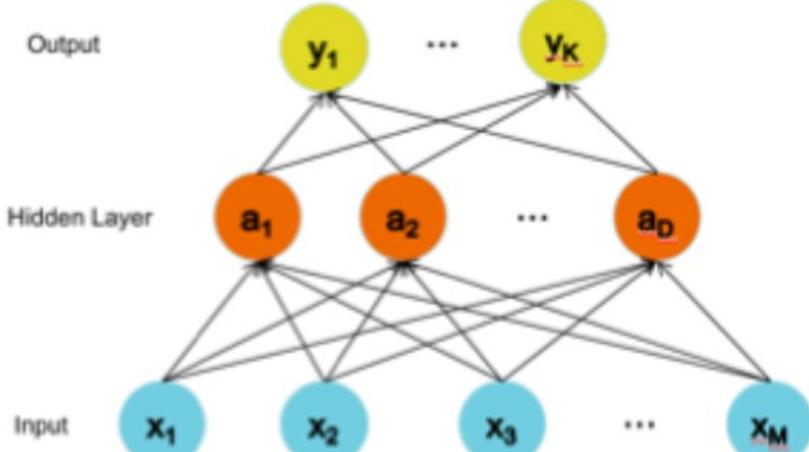
# Multilayer Neural Networks

## Network Design

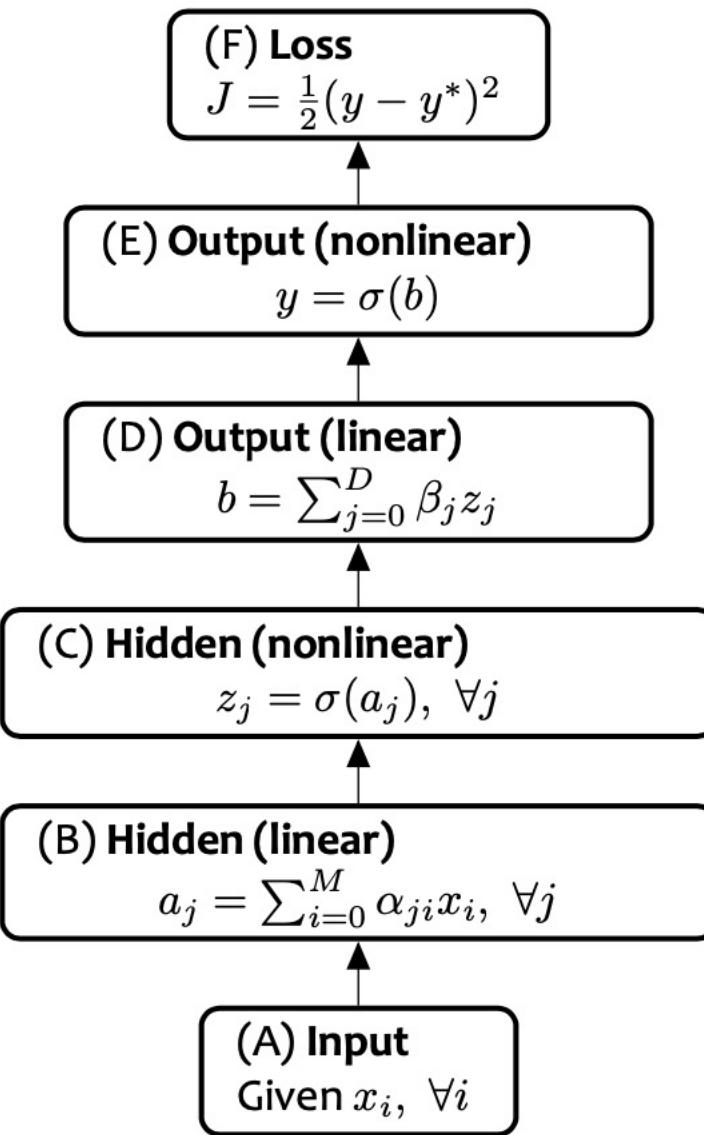
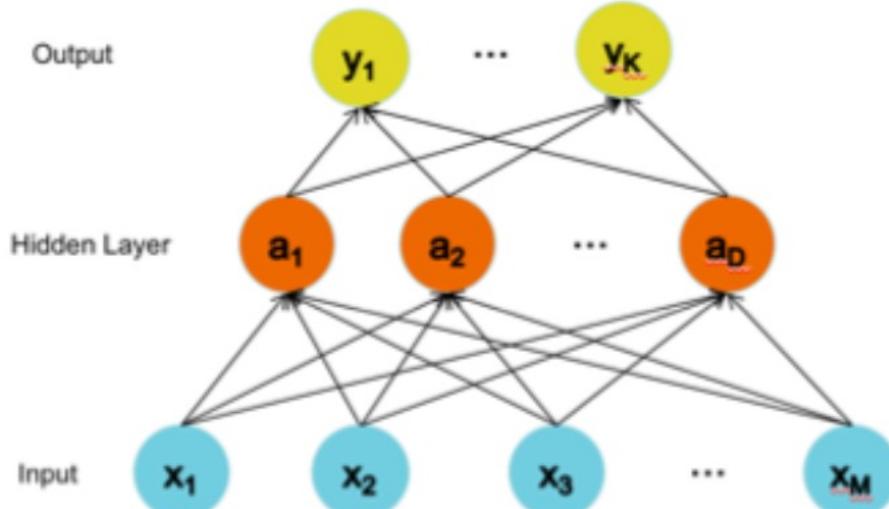
# Neural Network Architectures

- Even for a basic Multilayer Neural Network, there are many design decisions to make:
  - 1) Number of hidden layers (depth of the network)
  - 2) Number of neurons per hidden layer (width)
  - 3) Type of activation functions
  - 4) Form of objective functions

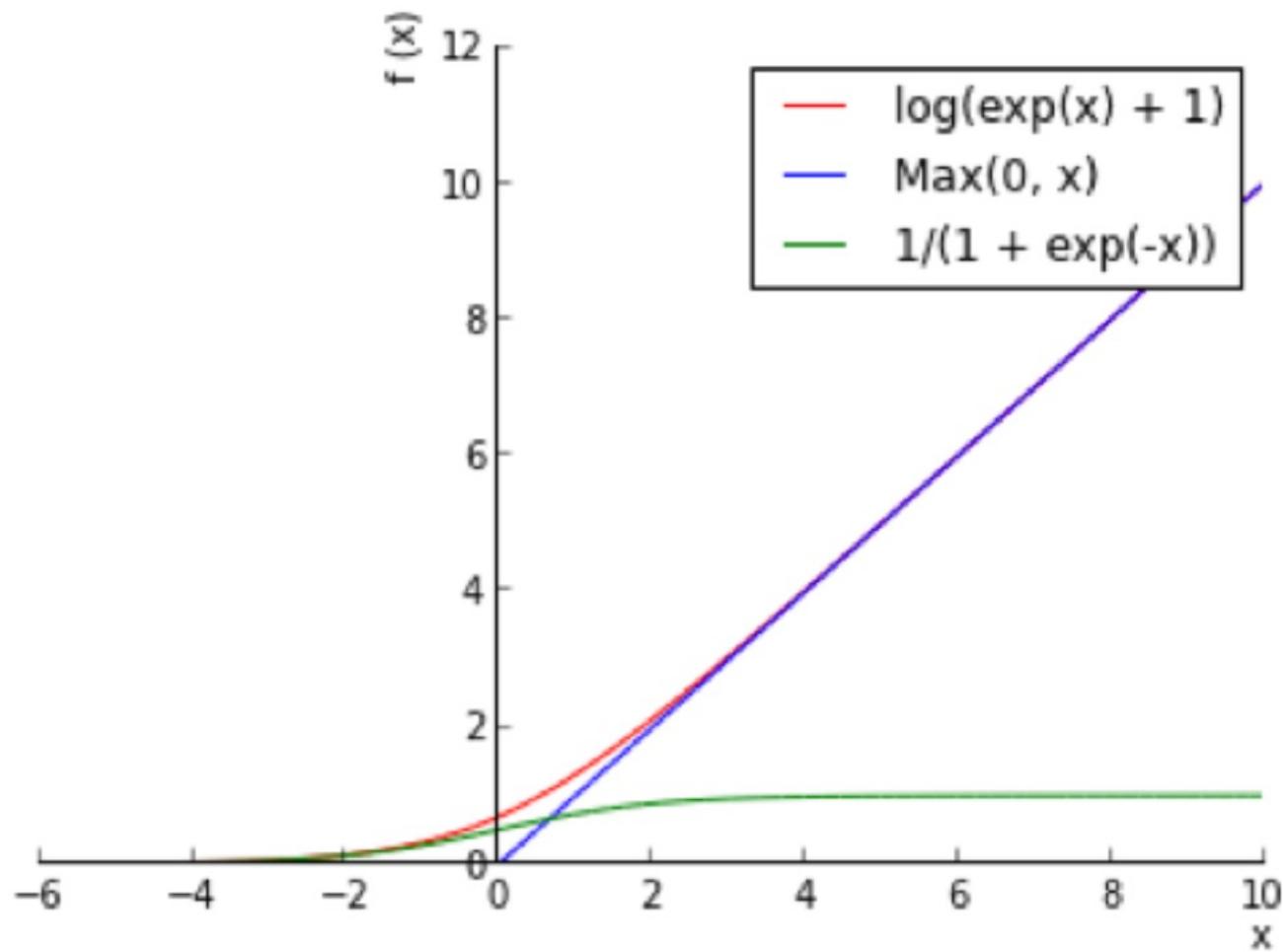
## Example: Neural Network with sigmoid activation functions



## Neural Network with arbitrary nonlinear activation functions



# ReLU often used in computer vision tasks

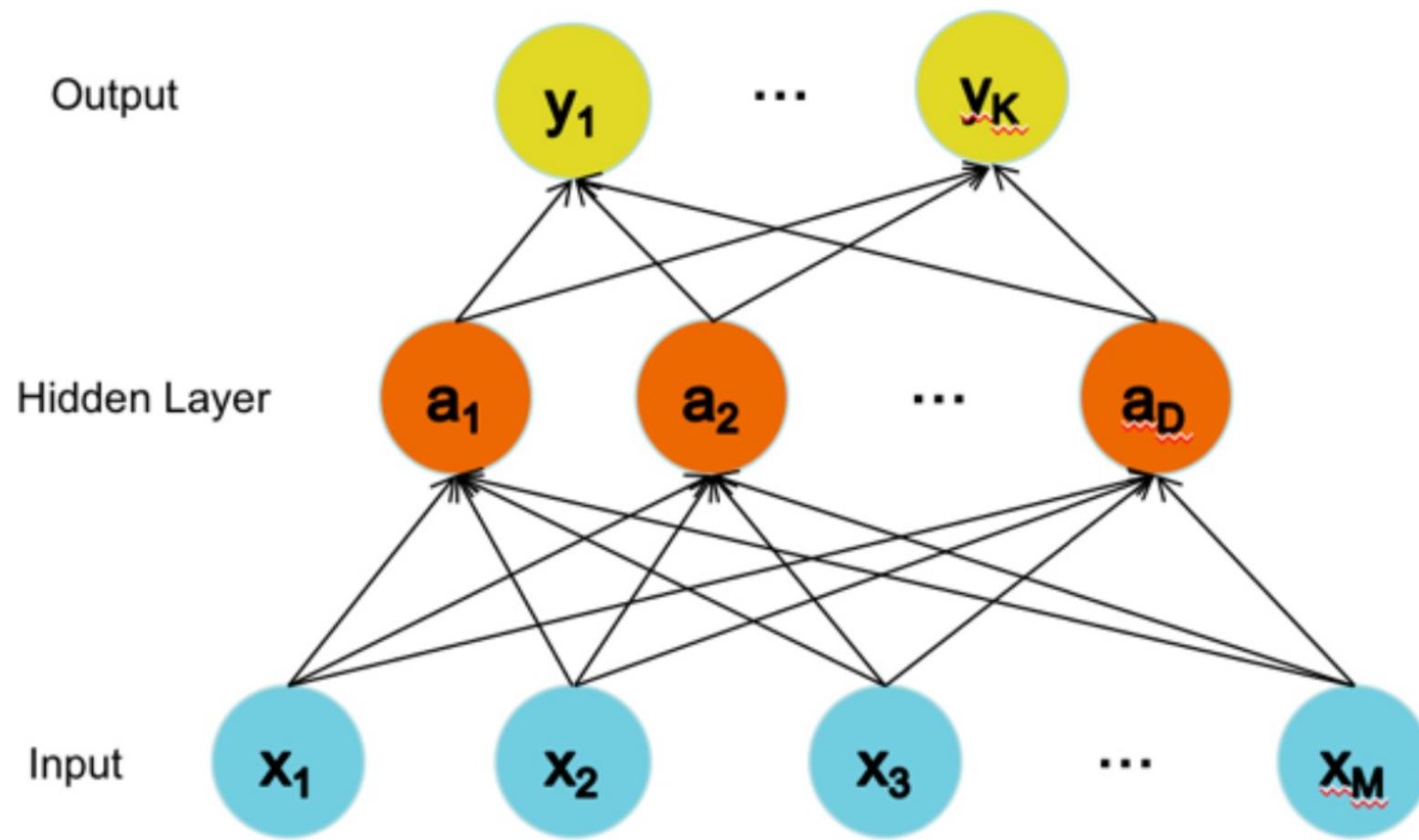


# Loss Functions for NN

- **Regression:**
  - Use the same objective as Linear Regression
  - Quadratic loss (i.e. mean squared error)
- **Classification:**
  - Use the same objective as Logistic Regression
  - Cross-entropy (i.e. negative log likelihood)
  - This requires probabilities, so we add an additional “softmax” layer at the end of our network

	Forward	Backward
Quadratic	$J = \frac{1}{2}(y - y^*)^2$	$\frac{dJ}{dy} = y - y^*$
Cross Entropy	$J = y^* \log(y) + (1 - y^*) \log(1 - y)$	$\frac{dJ}{dy} = y^* \frac{1}{y} + (1 - y^*) \frac{1}{1-y}$

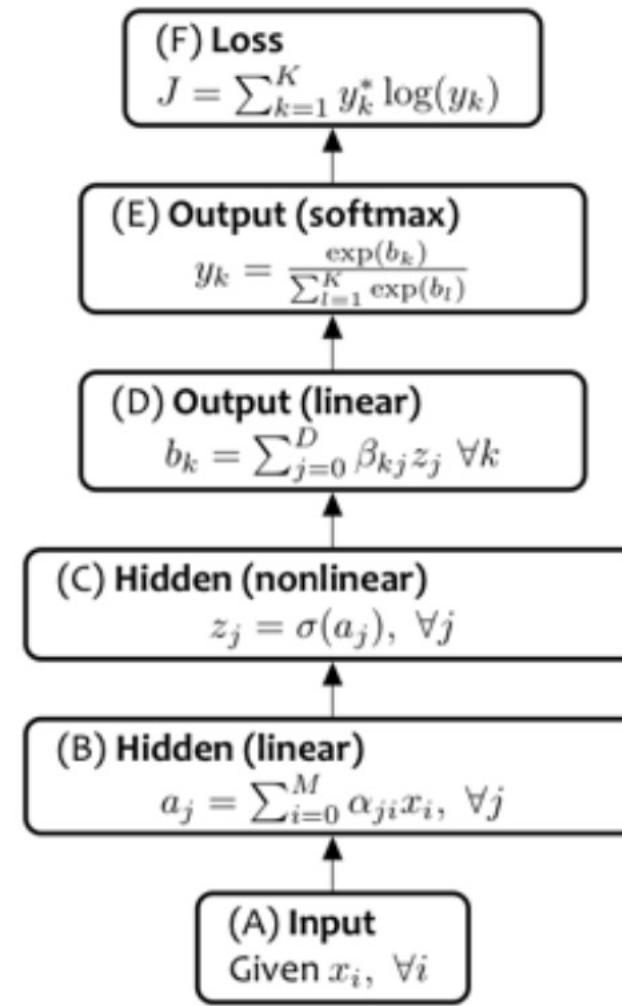
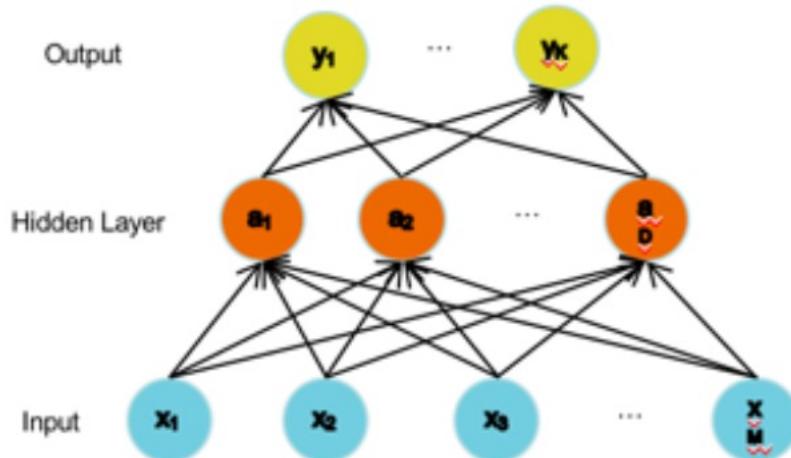
# Multi-class Output



# Multi-class Output

Softmax:

$$y_k = \frac{\exp(b_k)}{\sum_{l=1}^K \exp(b_l)}$$



# Reference

- Chapter 6 Deep feedforward networks, Deep Learning,  
<https://www.deeplearningbook.org/>