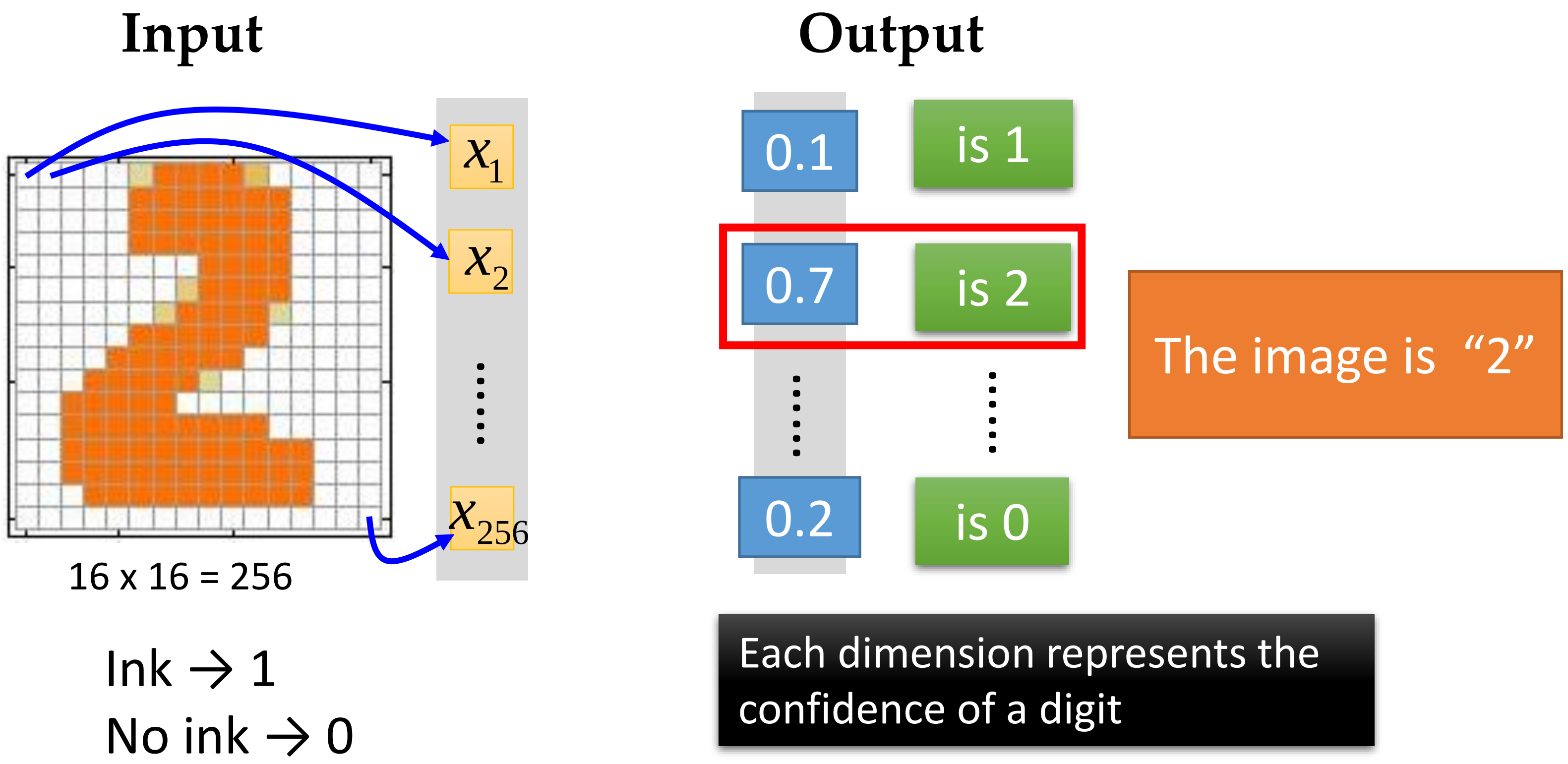
# Introduction to Neural Networks



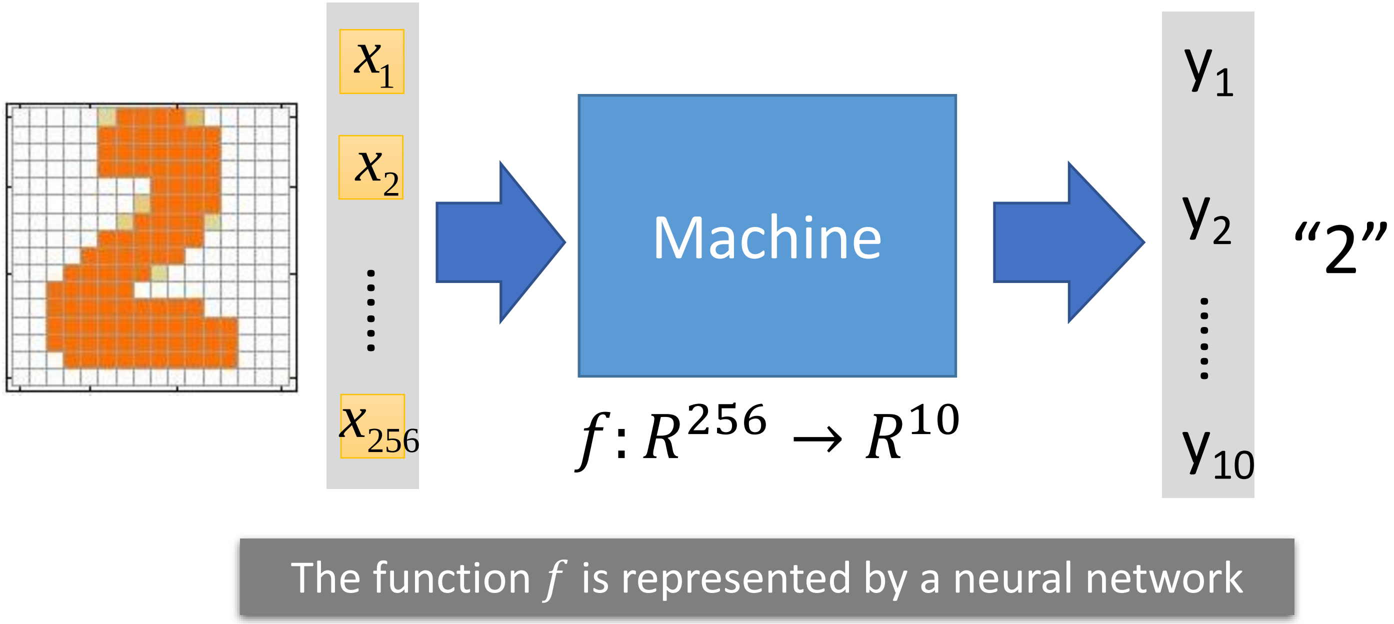
- Handwritten digit recognition (MNIST dataset)
  - The intensity of each pixel is considered an input element
  - Output is the class of the digit



# Introduction to Neural Networks

*Introduction to Neural Networks*

- Handwritten digit recognition



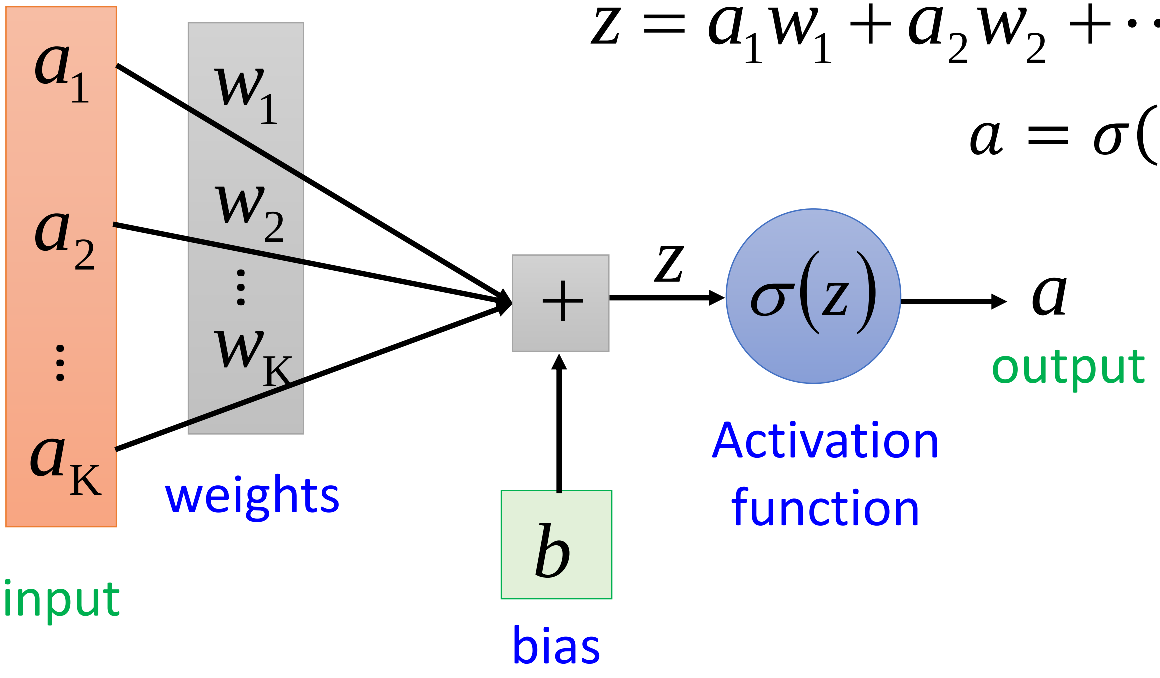
# Elements of Neural Networks

*Introduction to Neural Networks*

- NNs consist of hidden layers with neurons (i.e., computational units)
- A single neuron maps a set of inputs into an output number, or $f: R^K \rightarrow R$

$$z = a_1 w_1 + a_2 w_2 + \cdots + a_K w_K + b$$

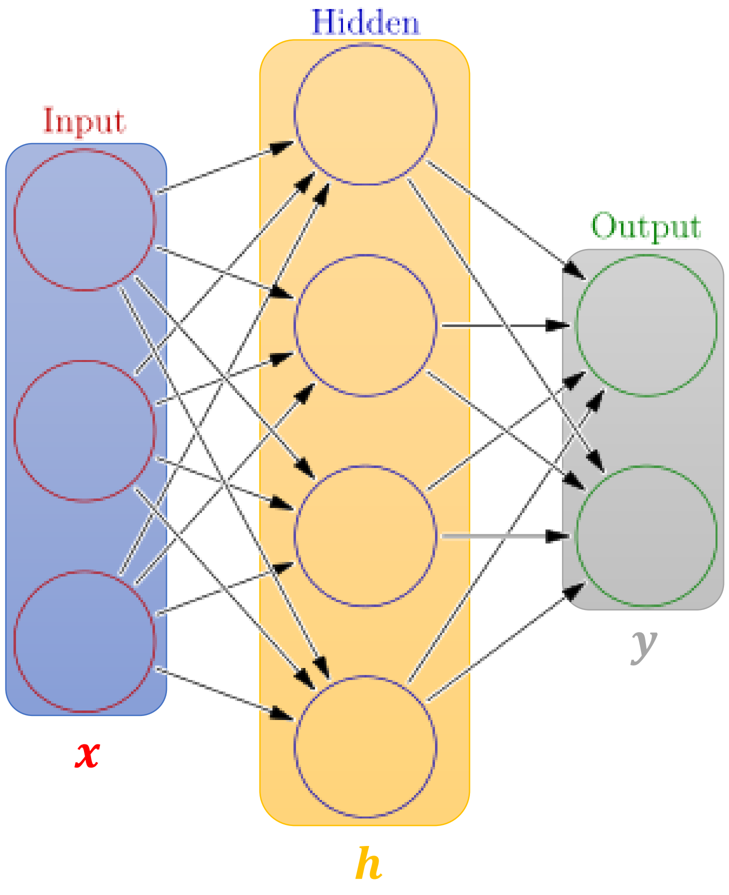
$$a = \sigma(z)$$



# Elements of Neural Networks

*Introduction to Neural Networks*

- A NN with one hidden layer and one output layer




Weights
Biases

$$\text{hidden layer } \boldsymbol{h} = \sigma(\mathbf{W}_1 \boldsymbol{x} + \boldsymbol{b}_1)$$

$$\text{output layer } \boldsymbol{y} = \sigma(\boldsymbol{W}_2 \boldsymbol{h} + \boldsymbol{b}_2)$$

Activation functions

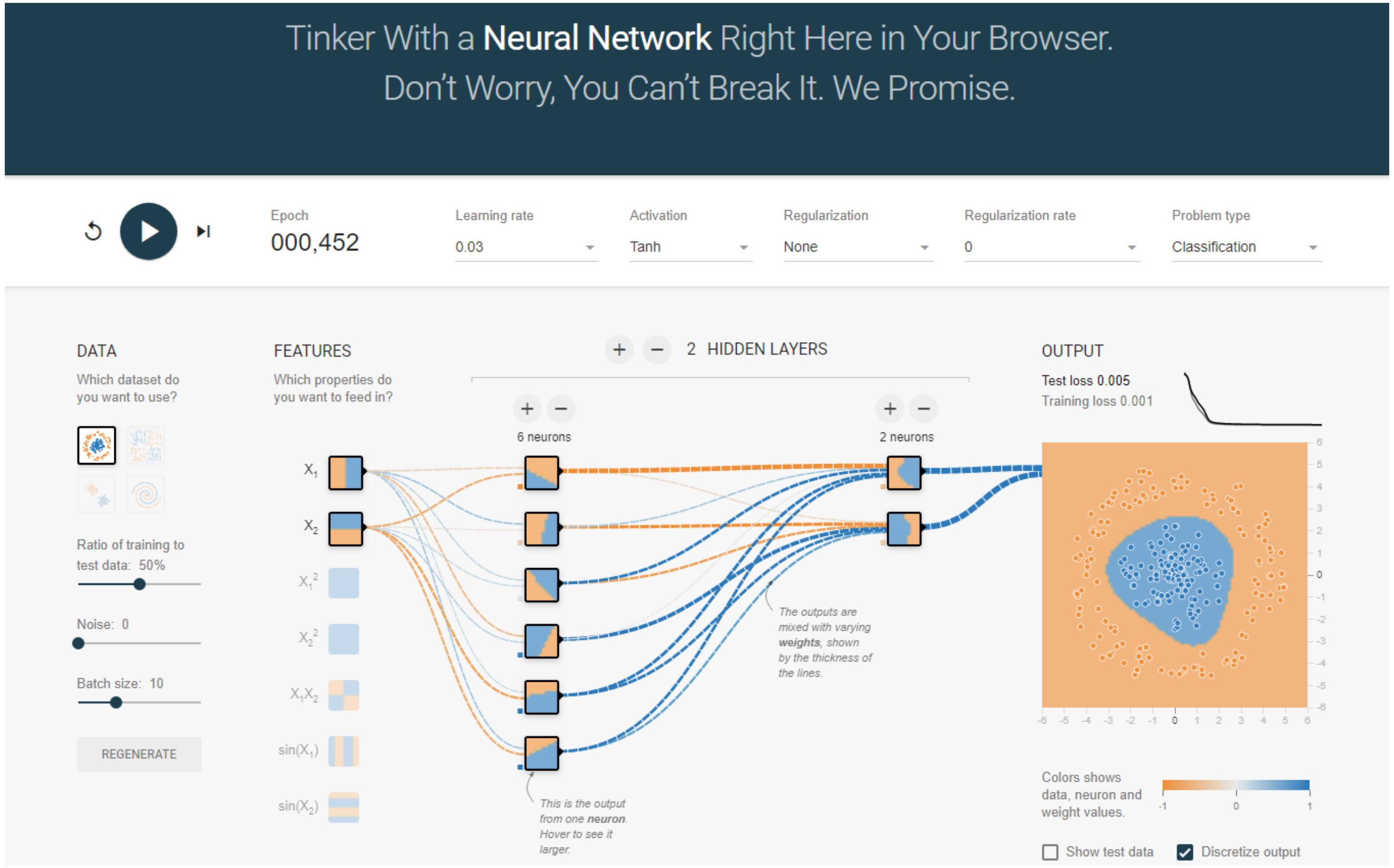
4 + 2 = 6 neurons (not counting inputs)
[3 × 4] + [4 × 2] = 20 weights
4 + 2 = 6 biases

26 learnable parameters

# Elements of Neural Networks

*Introduction to Neural Networks*

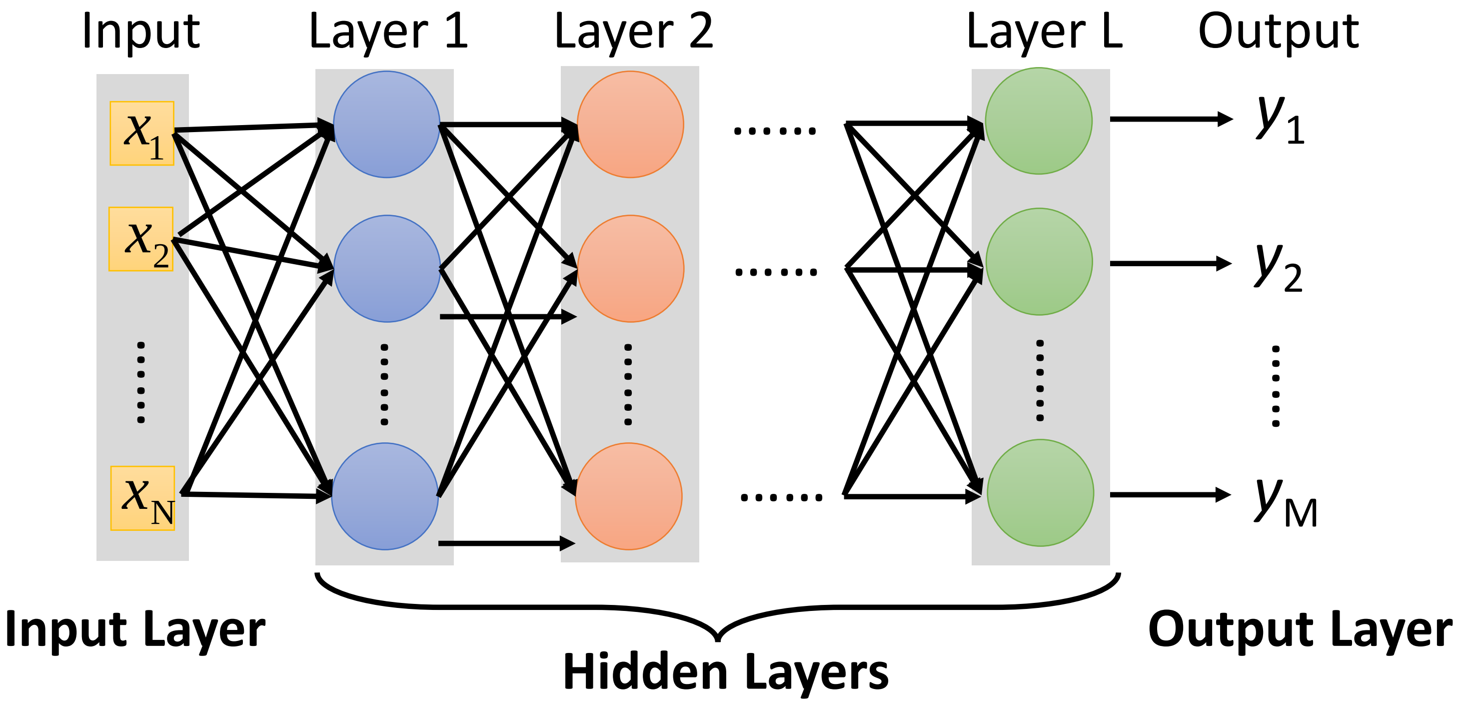
- A neural network playground [link]



# Elements of Neural Networks

*Introduction to Neural Networks*

- Deep NNs have many hidden layers
  - Fully-connected (dense) layers (a.k.a. Multi-Layer Perceptron or MLP)
  - Each neuron is connected to all neurons in the succeeding layer



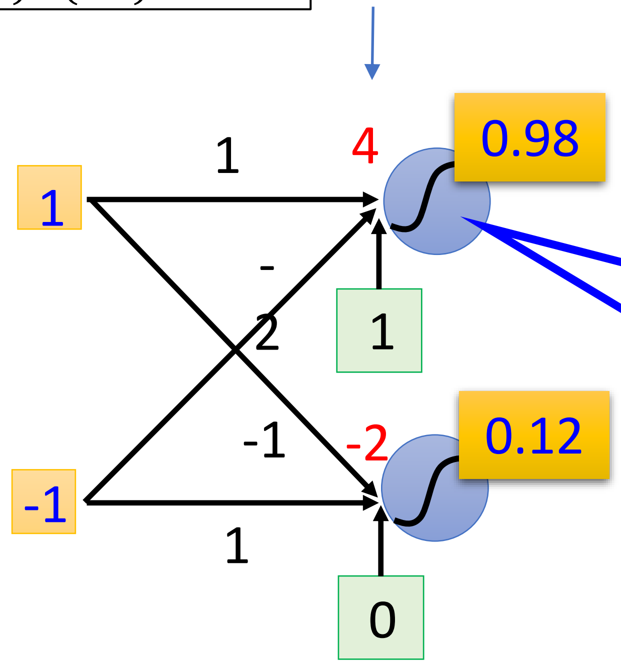

Slide credit: Hung-yi Lee – Deep Learning Tutorial

# Elements of Neural Networks

*Introduction to Neural Networks*

- A simple network, toy example

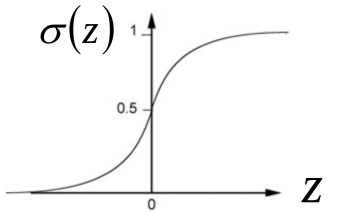
$$\boxed{(1 \cdot 1) + (-1) \cdot (-2) + 1 = 4}$$




$$\boxed{1 \cdot (-1) + (-1) \cdot 1 + 0 = -2}$$

Sigmoid Function

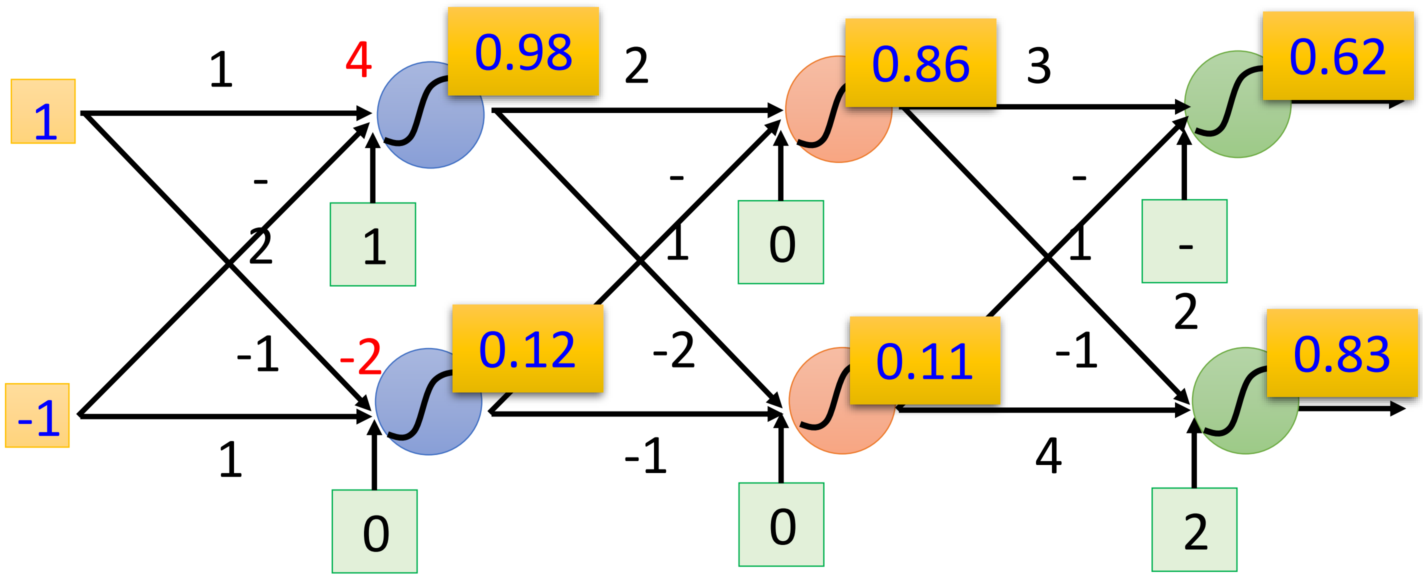
$$\sigma(z) = \frac{1}{1+e^{-z}}$$



# Elements of Neural Networks

*Introduction to Neural Networks*

- A simple network, toy example (cont'd)
  - For an input vector $[1 \quad -1]^T$, the output is $[0.62 \quad 0.83]^T$

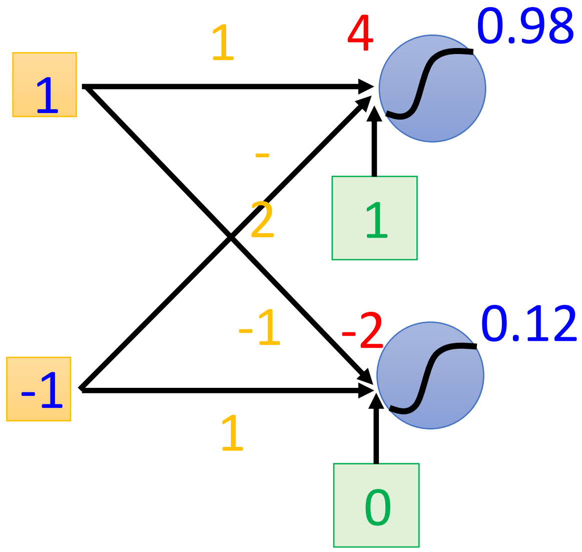



$$f: R^2 \rightarrow R^2 \qquad f\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix}\right) = \begin{bmatrix} 0.62 \\ 0.83 \end{bmatrix}$$

# Matrix Operation

*Introduction to Neural Networks*

- Matrix operations are helpful when working with multidimensional inputs and outputs




$$\sigma(\ W \ x + b\ ) = a$$

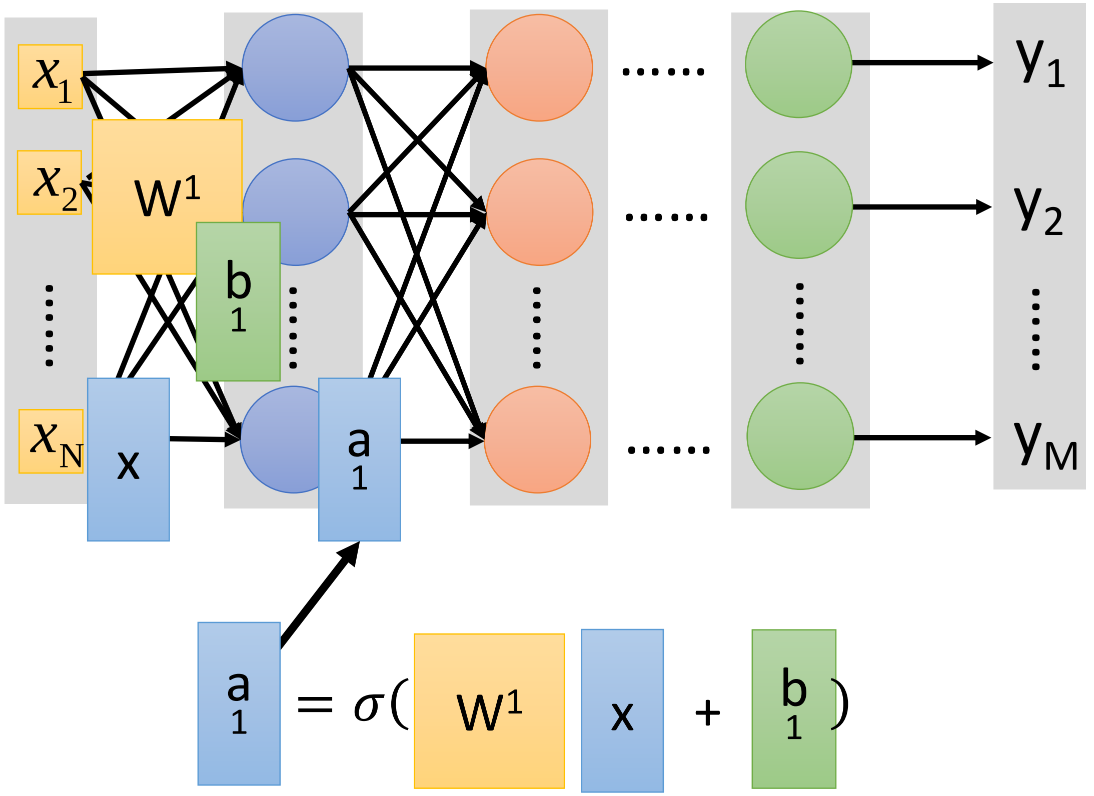
$$\sigma(\underbrace{\begin{bmatrix}1 & -2\\ -1 & 1\end{bmatrix}\begin{bmatrix}1\\ -1\end{bmatrix} + \begin{bmatrix}1\\ 0\end{bmatrix}}_{\begin{bmatrix}4\\ -2\end{bmatrix}}) = \begin{bmatrix}0.98\\ 0.12\end{bmatrix}$$

Slide credit: Hung-yi Lee – Deep Learning Tutorial

# Matrix Operation



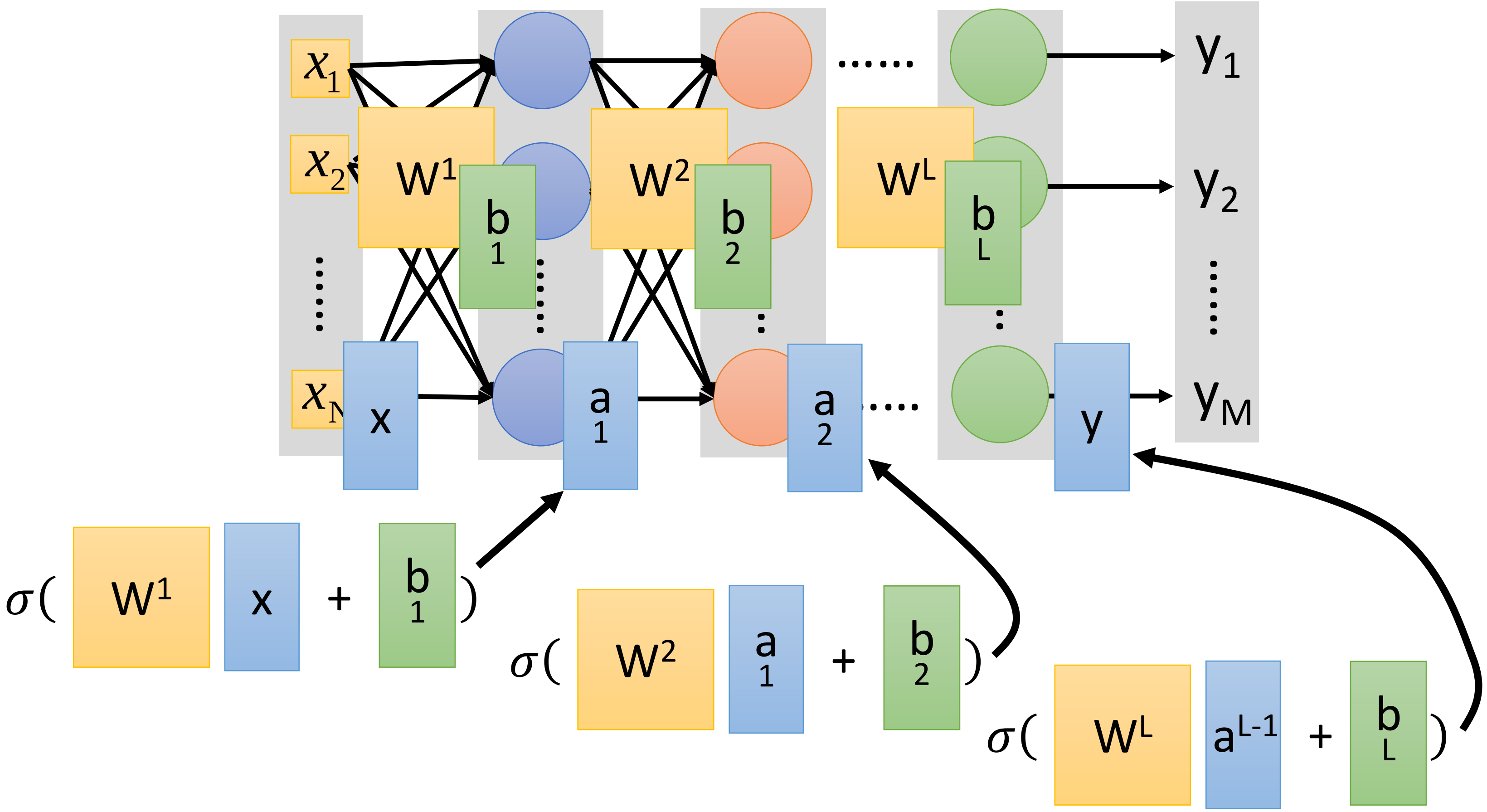
- Multilayer NN, matrix calculations for the first layer
  - Input vector $x$, weights matrix $W^1$, bias vector $b^1$, output vector $a^1$



# Matrix Operation

*Introduction to Neural Networks*

- Multilayer NN, matrix calculations for all layers

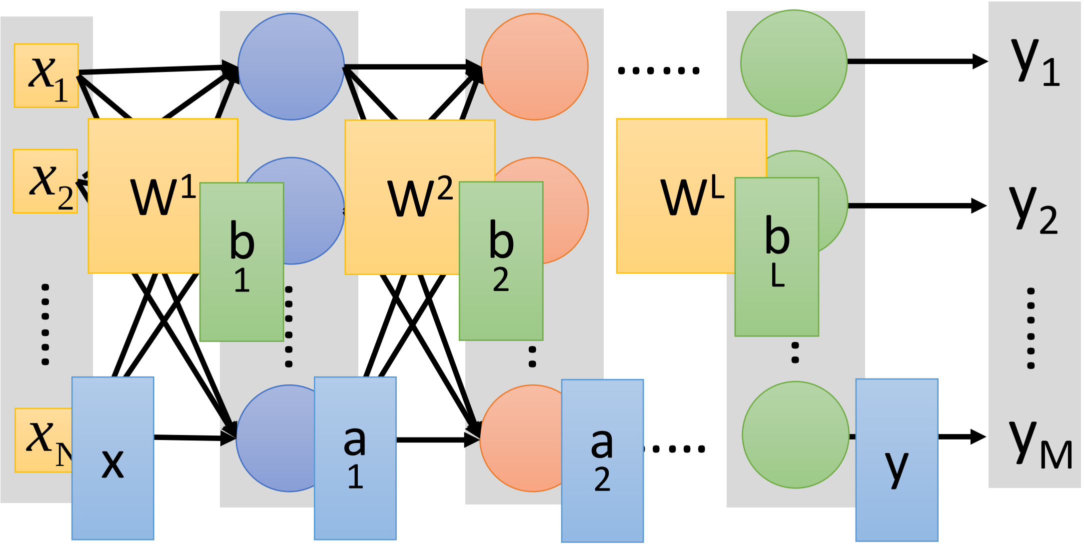



Slide credit: Hung-yi Lee – Deep Learning Tutorial

# Matrix Operation

*Introduction to Neural Networks*

- Multilayer NN, function $f$ maps inputs $x$ to outputs $y$, i.e., $y = f(x)$



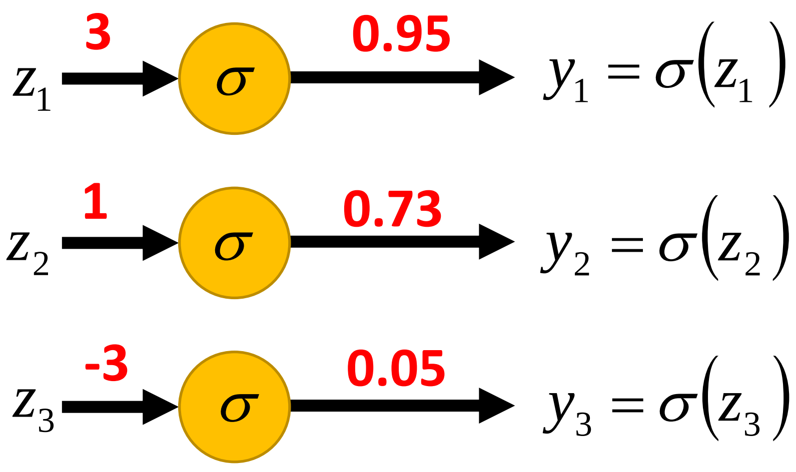

$$y = f(x) = \sigma(W^L \cdots \sigma(W^2\, \sigma(W^1 x + b_1) + b_2) \cdots + b_L)$$

# Softmax Layer

*Introduction to Neural Networks*

- In multi-class classification tasks, the output layer is typically a ***softmax layer***
  - I.e., it employs a ***softmax activation function***
  - If a layer with a sigmoid activation function is used as the output layer instead, the predictions by the NN may not be easy to interpret
    - Note that an output layer with sigmoid activations can still be used for binary classification

***A Layer with Sigmoid Activations***



# Softmax Layer

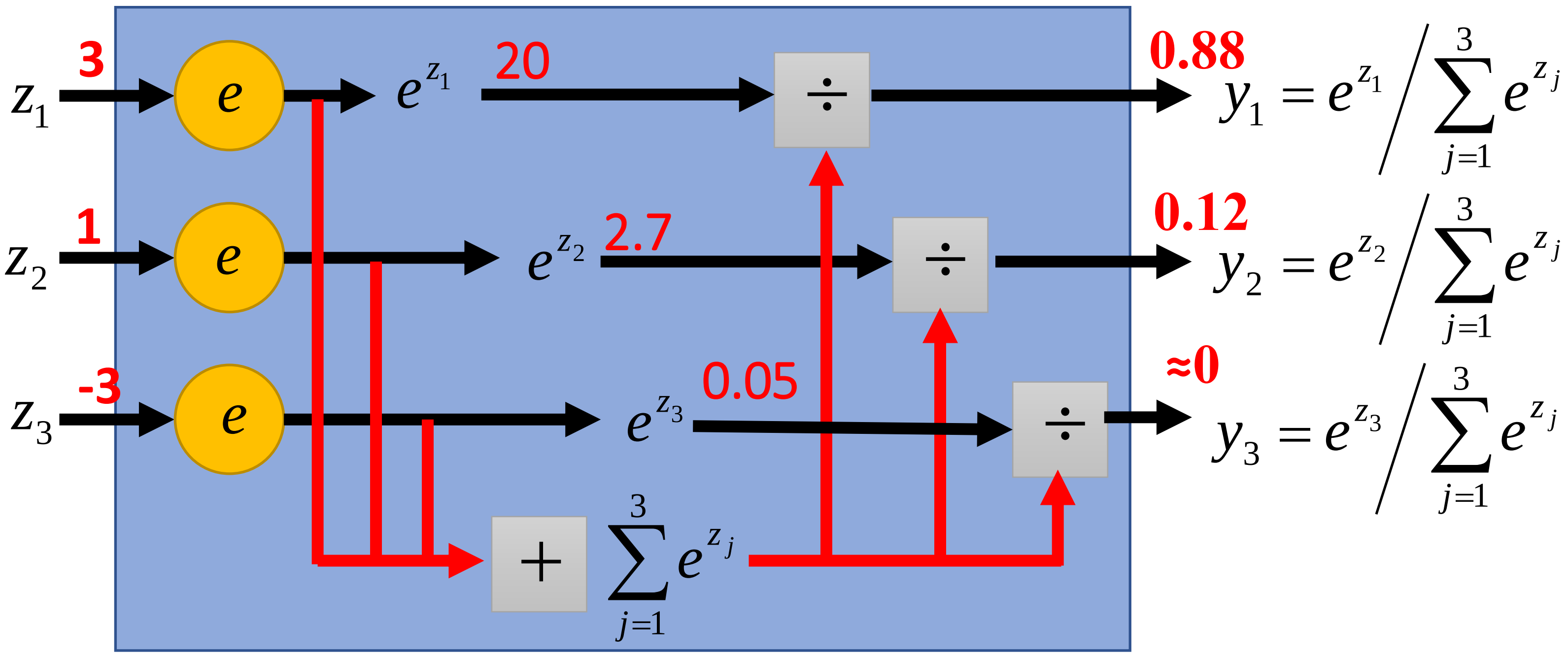
*Introduction to Neural Networks*

- The softmax layer applies softmax activations to output a probability value in the range [0, 1]
  - The values $z$ inputted to the softmax layer are referred to as ***logits***

***Probability***:

- $0 < y_i < 1$
- $\sum_i y_i = 1$

***A Softmax Layer***

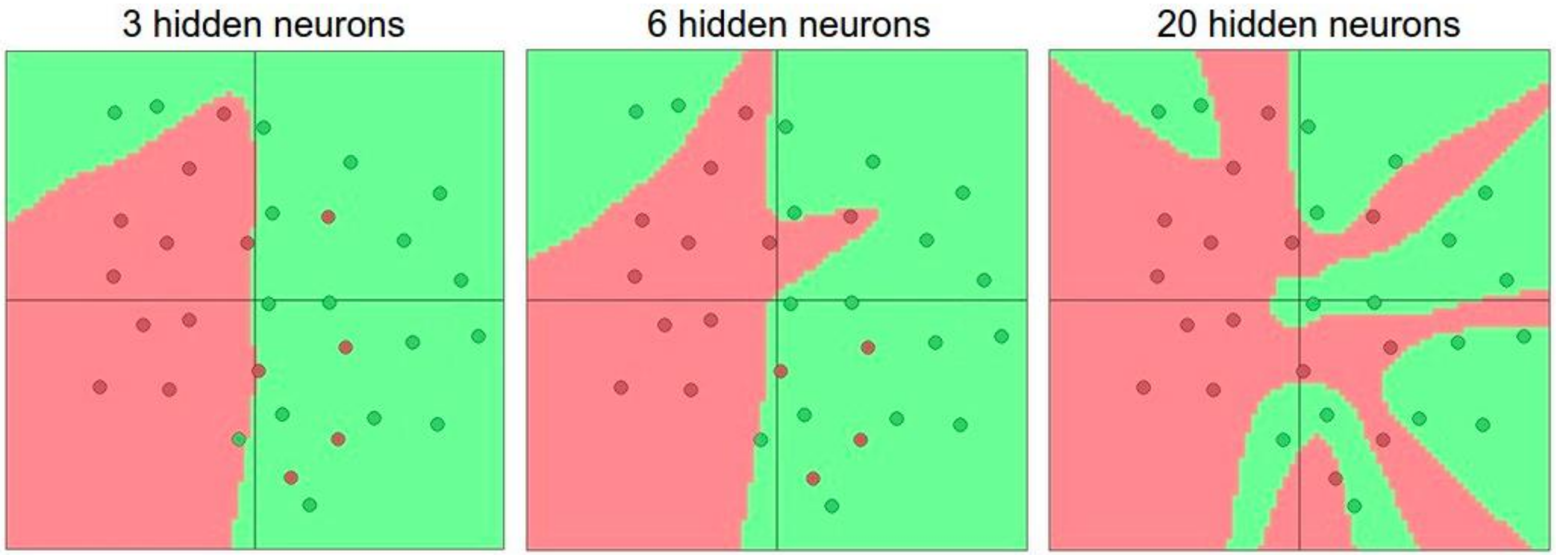



Slide credit: Hung-yi Lee – Deep Learning Tutorial

# Activation Functions

*Introduction to Neural Networks*

- Non-linear activations are needed to learn complex (non-linear) data representations
  - Otherwise, NNs would be just a linear function (such as $W_1 W_2 x = Wx$)
  - NNs with large number of layers (and neurons) can approximate more complex functions
    - Figure: more neurons improve representation (but, may overfit)

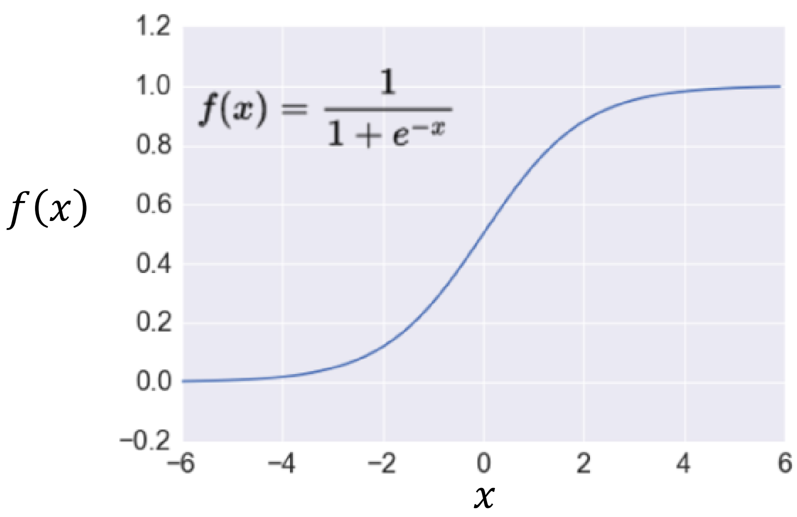



Picture from: http://cs231n.github.io/assets/nn1/layer_sizes.jpeg

# Activation: Sigmoid

*Introduction to Neural Networks*

- *Sigmoid function* σ: takes a real-valued number and "squashes" it into the range between 0 and 1
  - The output can be interpreted as the firing rate of a biological neuron
    - Not firing = 0; Fully firing = 1
  - When the neuron's activation are 0 or 1, sigmoid neurons saturate
    - Gradients at these regions are almost zero (almost no signal will flow)
  - Sigmoid activations are less common in modern NNs



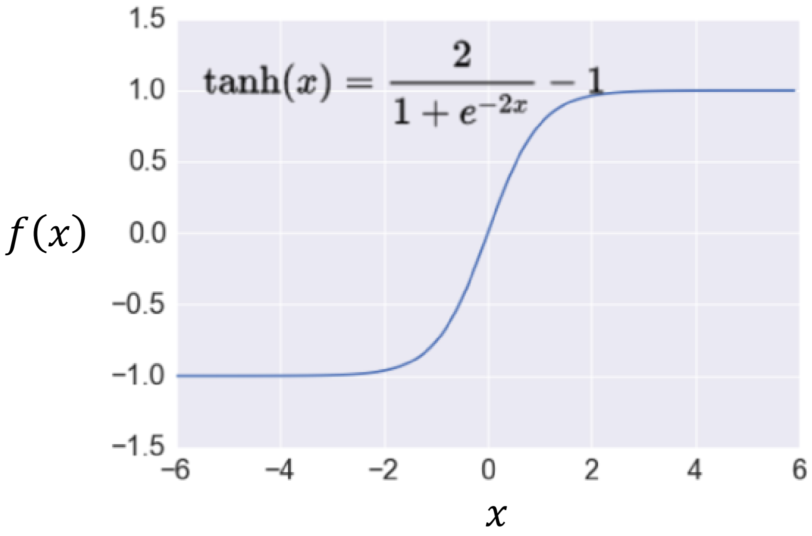

$\mathbb{R}^n \rightarrow [0,1]$

Slide credit: Ismini Lourentzou – Introduction to Deep Learning

# Activation: Tanh



- ***Tanh function***: takes a real-valued number and "squashes" it into range between -1 and 1
  - Like sigmoid, tanh neurons saturate
  - Unlike sigmoid, the output is zero-centered
    - It is therefore preferred than sigmoid
  - Tanh is a scaled sigmoid: $\tanh(x) = 2 \cdot \sigma(2x) - 1$




$\mathbb{R}^n \rightarrow [-1,1]$

Slide credit: Ismini Lourentzou – Introduction to Deep Learning

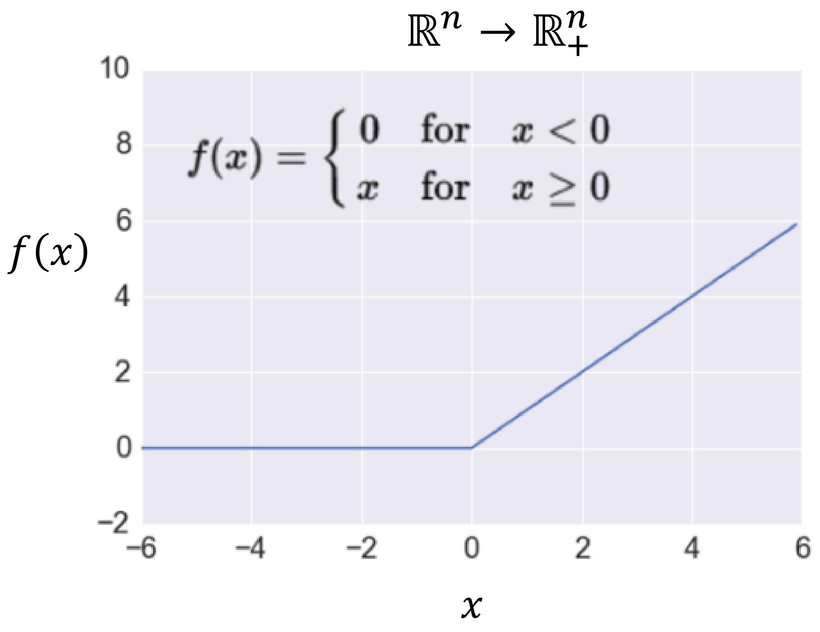
# Activation: ReLU



- ***ReLU*** (Rectified Linear Unit): takes a real-valued number and thresholds it at zero

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

  - Most modern deep NNs use ReLU activations
  - ReLU is fast to compute
    - Compared to sigmoid, tanh
    - Simply threshold a matrix at zero
  - Accelerates the convergence of gradient descent
    - Due to linear, non-saturating form
  - Prevents the gradient vanishing problem



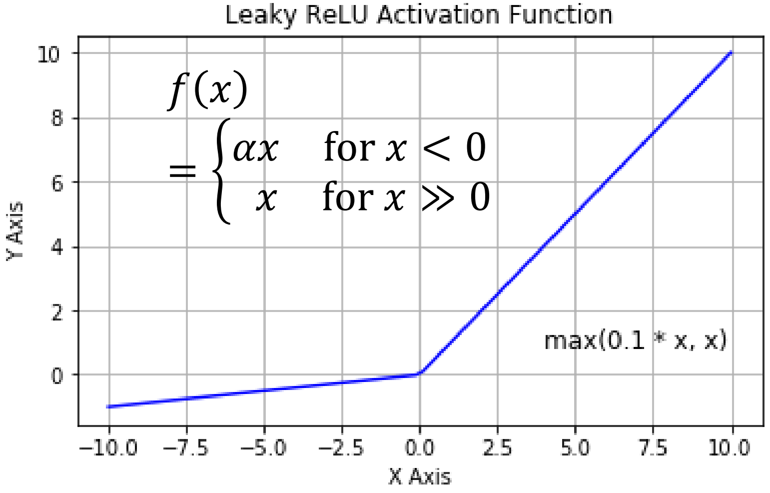
# Activation: Leaky ReLU



- The problem of ReLU activations: they can "die"
  - ReLU could cause weights to update  in a way that the gradients can become zero and the neuron will not activate again on any data
  - E.g., when a large learning rate is used

- ***Leaky ReLU*** activation function is a variant of ReLU
  - Instead of the function being 0 when $x < 0$, a leaky ReLU has a small negative slope (e.g., $\alpha = 0.01$, or similar)

  - This resolves the dying ReLU problem
  - Most current works still use ReLU
    - With a proper setting of the learning rate, the problem of dying ReLU can be avoided



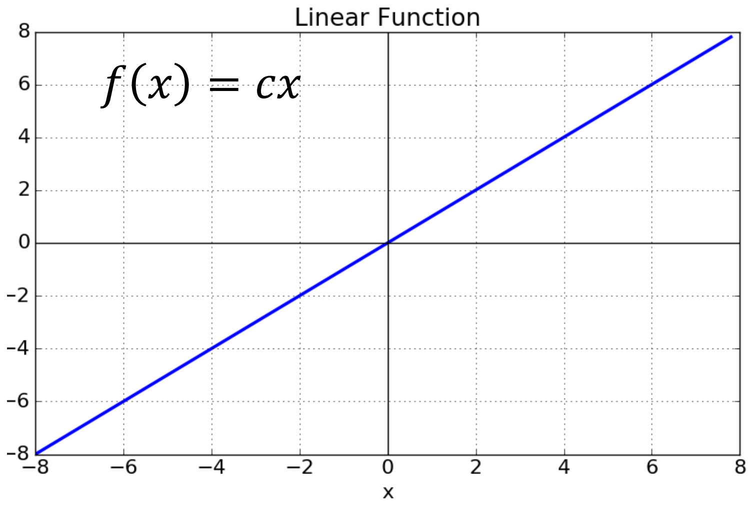
# Activation: Linear Function

*Introduction to Neural Networks*

- ***Linear function*** means that the output signal is proportional to the input signal to the neuron

$$\mathbb{R}^n \rightarrow \mathbb{R}^n$$

  - If the value of the constant $c$ is 1, it is also called identity activation function
  - This activation type is used in regression problems
    - E.g., the last layer can have linear activation function, in order to output a real number (and not a class membership)



# Training NNs

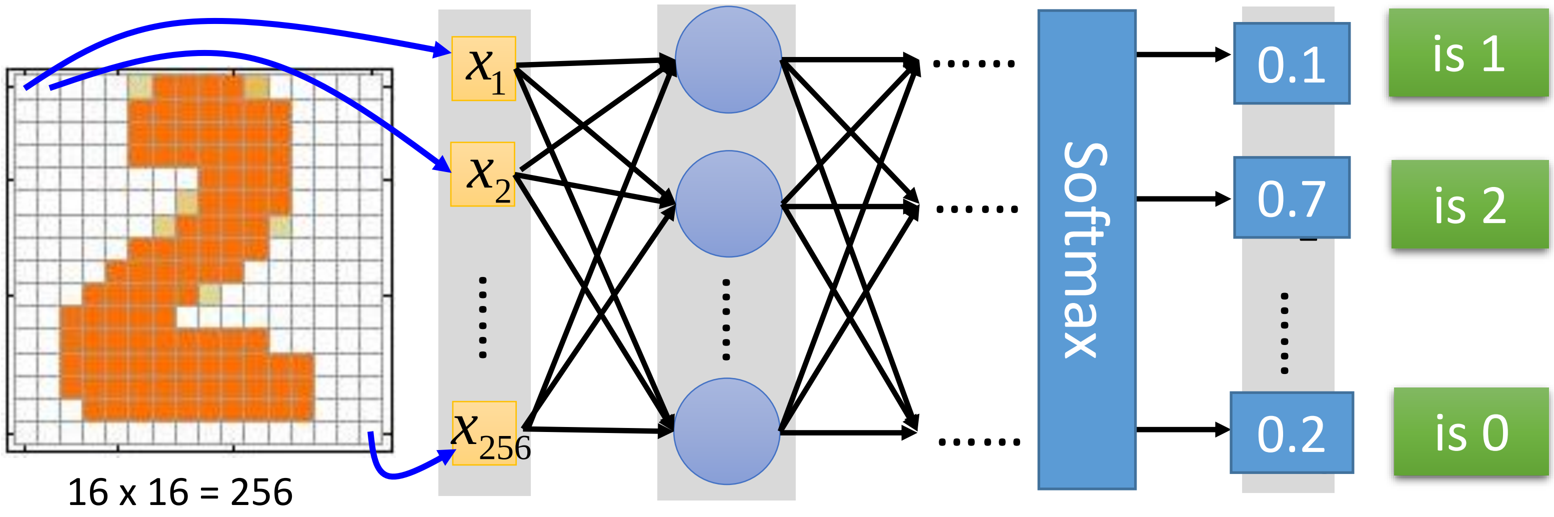
*Training Neural Networks*

- The network ***parameters*** $\theta$ include the weight matrices and bias vectors from all layers

$$\theta = \{W^1, b^1, W^2, b^2, \cdots W^L, b^L\}$$

  - Often, the model parameters $\theta$ are referred to as weights

- Training a model to learn a set of parameters $\theta$ that are optimal (according to a criterion) is one of the greatest challenges in ML

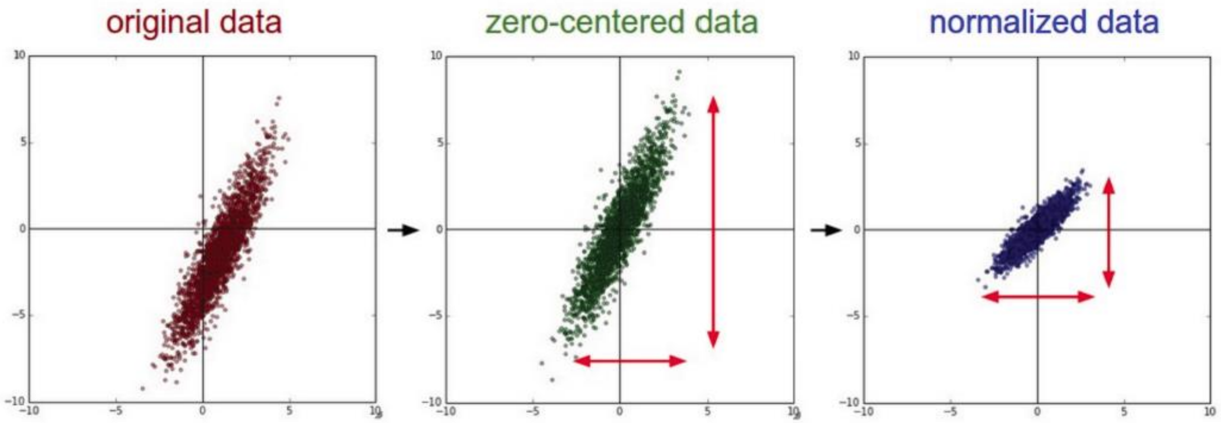



Slide credit: Hung-yi Lee – Deep Learning Tutorial

# Training NNs

*Training Neural Networks*

- *Data preprocessing* - helps convergence during training
  - Mean subtraction, to obtain zero-centered data
    - Subtract the mean for each individual data dimension (feature)
  - Normalization
    - Divide each feature by its standard deviation
      - To obtain standard deviation of 1 for each data dimension (feature)
    - Or, scale the data within the range [0,1] or [-1, 1]
      - E.g., image pixel intensities are divided by 255 to be scaled in the [0,1] range

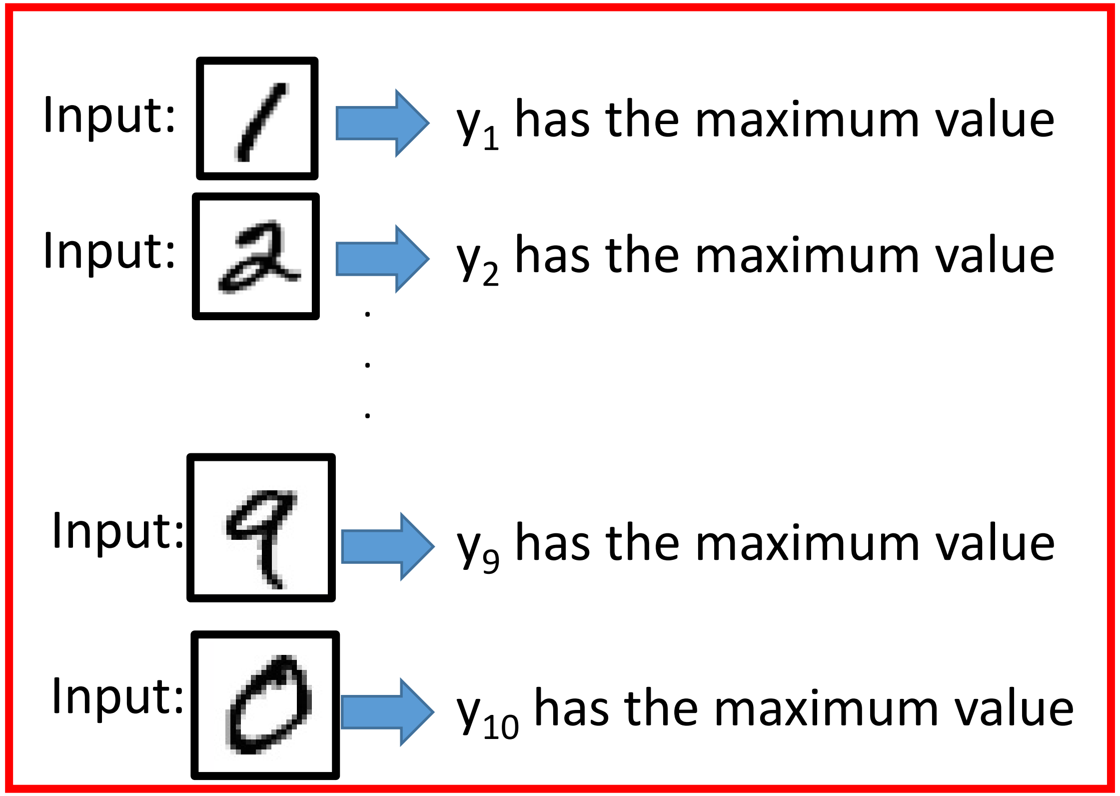



Picture from: [https://cs231n.github.io/neural-networks-2/](https://cs231n.github.io/neural-networks-2/)

# Training NNs

*Training Neural Networks*

- To train a NN, set the parameters $\theta$ such that for a training subset of images, the corresponding elements in the predicted output have maximum values

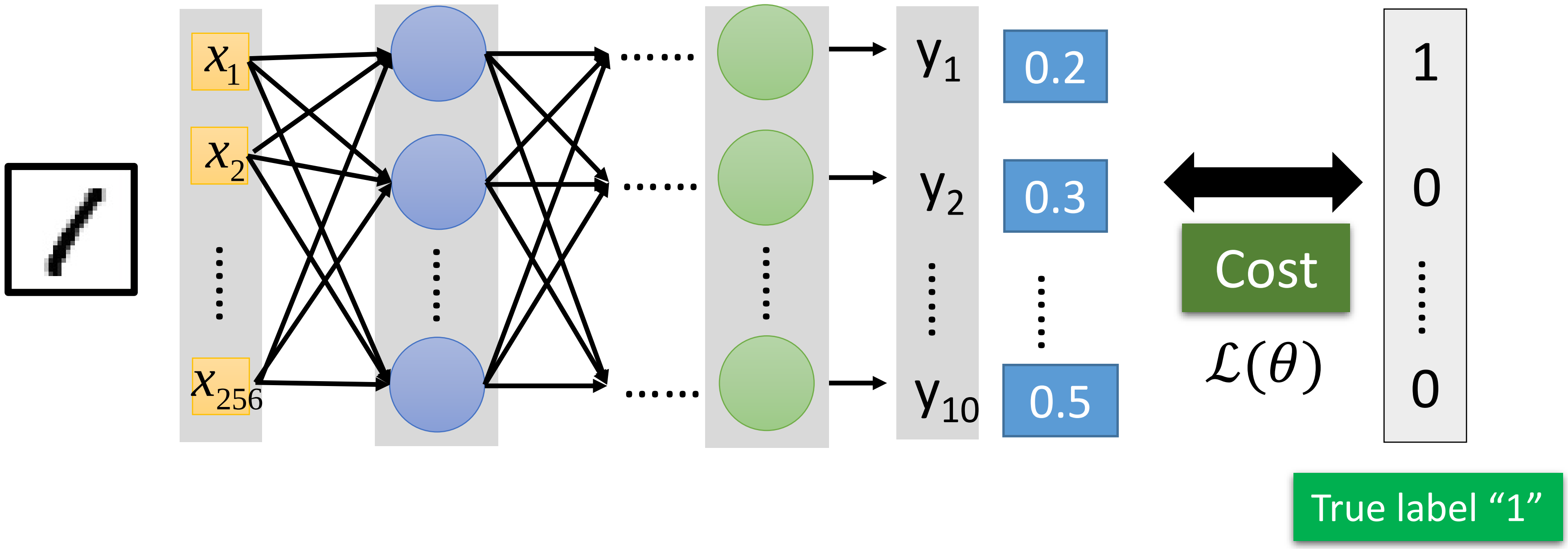



Slide credit: Hung-yi Lee – Deep Learning Tutorial

# Training NNs

*Training Neural Networks*

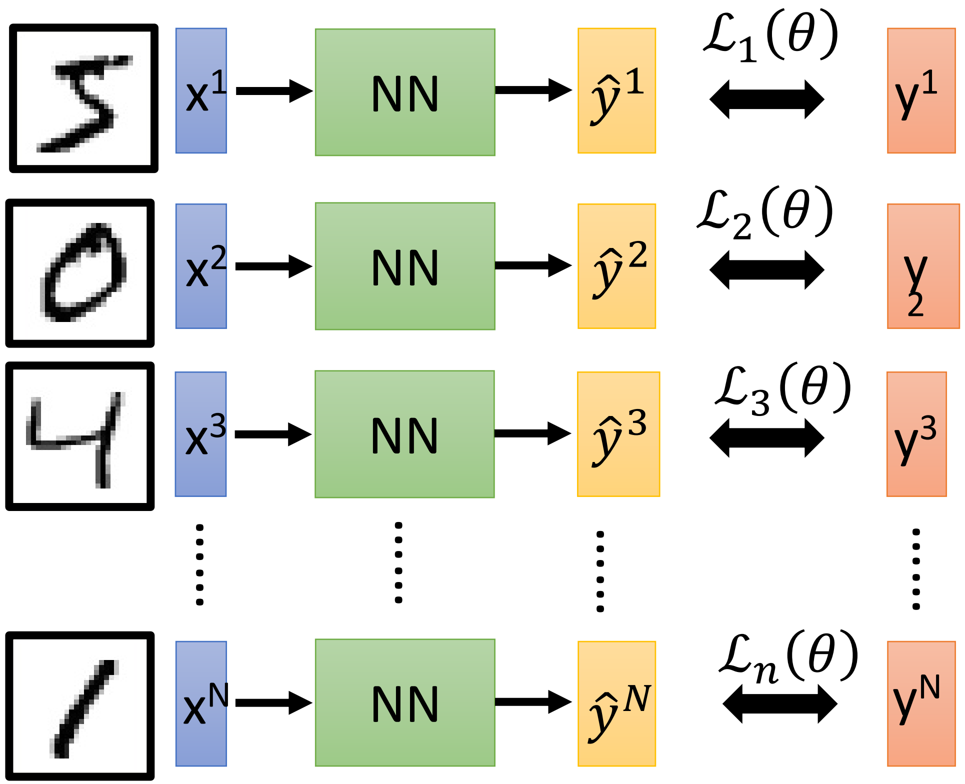
- Define a ***loss function***/objective function/cost function $\mathcal{L}(\theta)$ that calculates the difference (error) between the model prediction and the true label
  - E.g., $\mathcal{L}(\theta)$ can be mean-squared error, cross-entropy, etc.



# Training NNs

*Training Neural Networks*

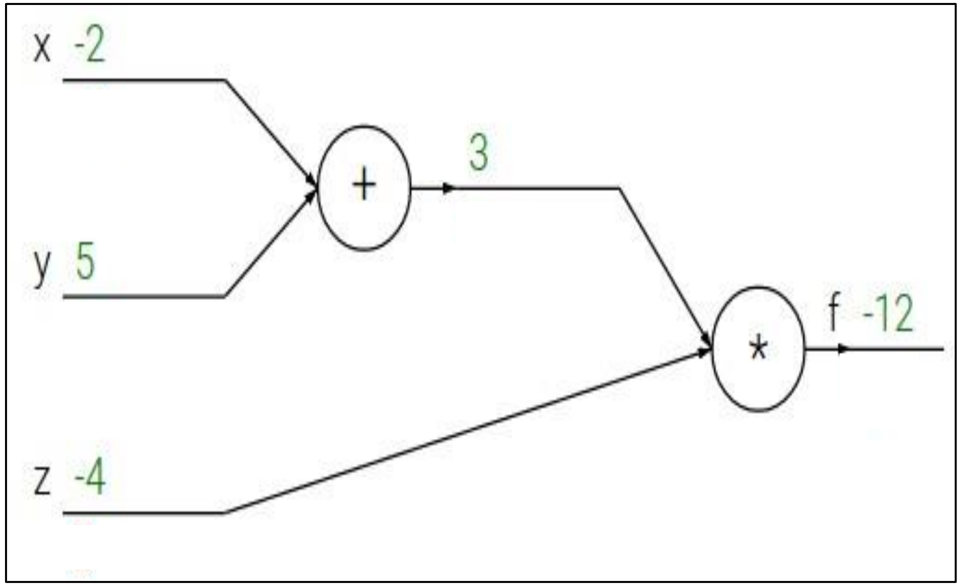
- For a training set of $N$ images, calculate the total loss overall all images: $\mathcal{L}(\theta) = \sum_{n=1}^{N} \mathcal{L}_n(\theta)$
- Find the optimal parameters $\theta^*$ that minimize the total loss $\mathcal{L}(\theta)$




Slide credit: Hung-yi Lee – Deep Learning Tutorial

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

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



We can write

$$\mathrm{f}(x, y, z) = g(h(x, y), z)$$

Where $h(x, y) = x + y$, and $g(a, b) = a * b$

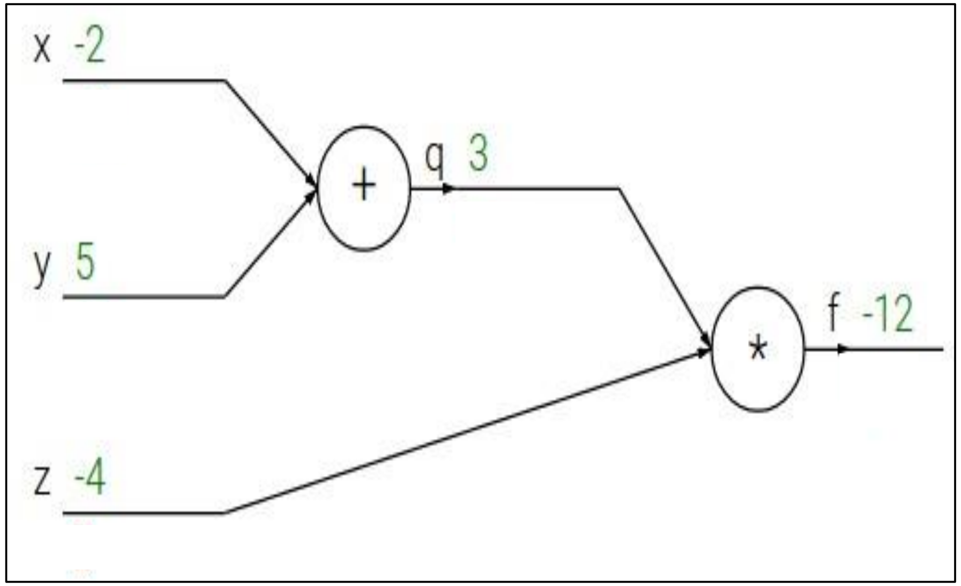
By the chain rule, $\frac{df}{dx} = \frac{dg}{dh}\frac{dh}{dx}$ and $\frac{df}{dy} = \frac{dg}{dh}\frac{dh}{dy}$

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

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

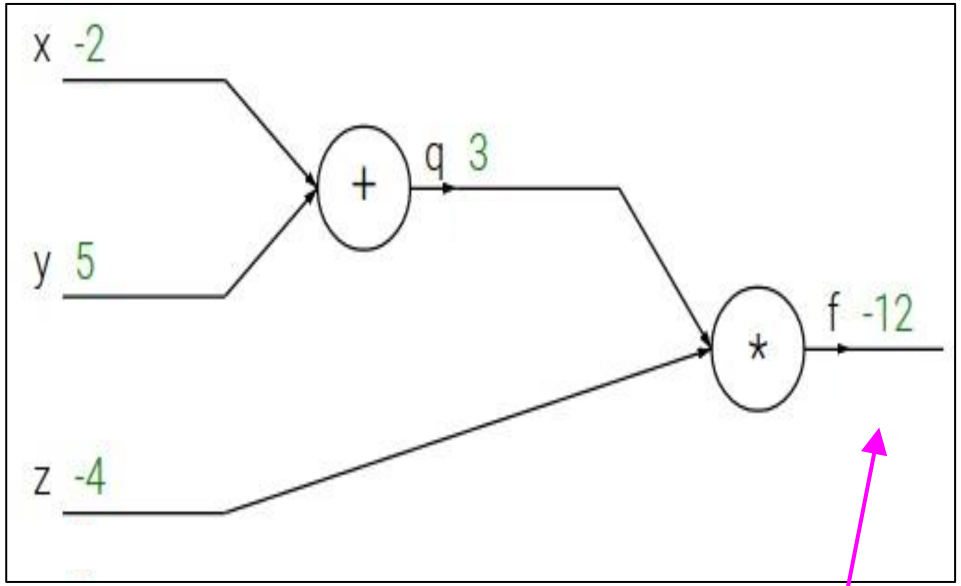
$$f(x, y, z) = (x + y)z$$

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

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

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

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



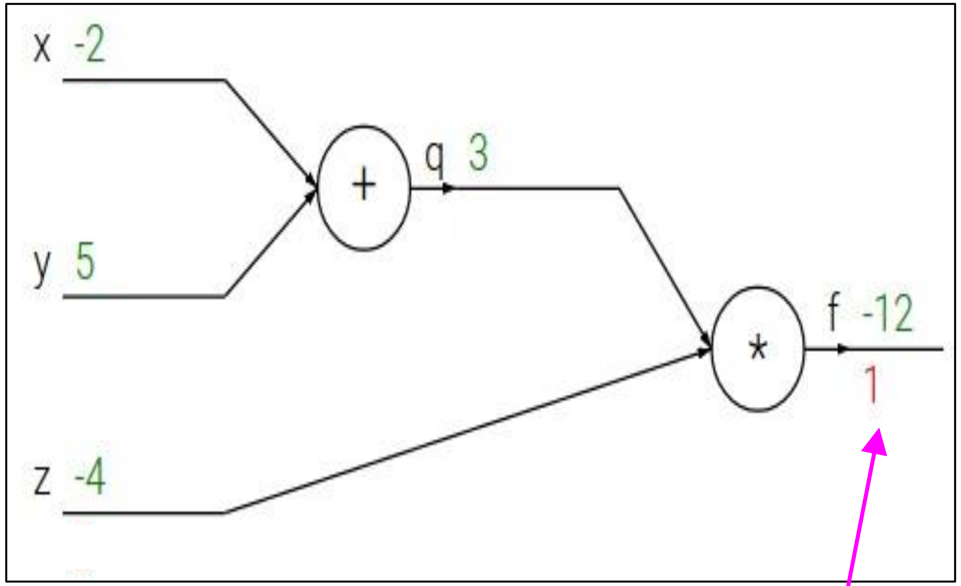

$$\frac{\partial f}{\partial f}$$

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

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

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

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




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

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

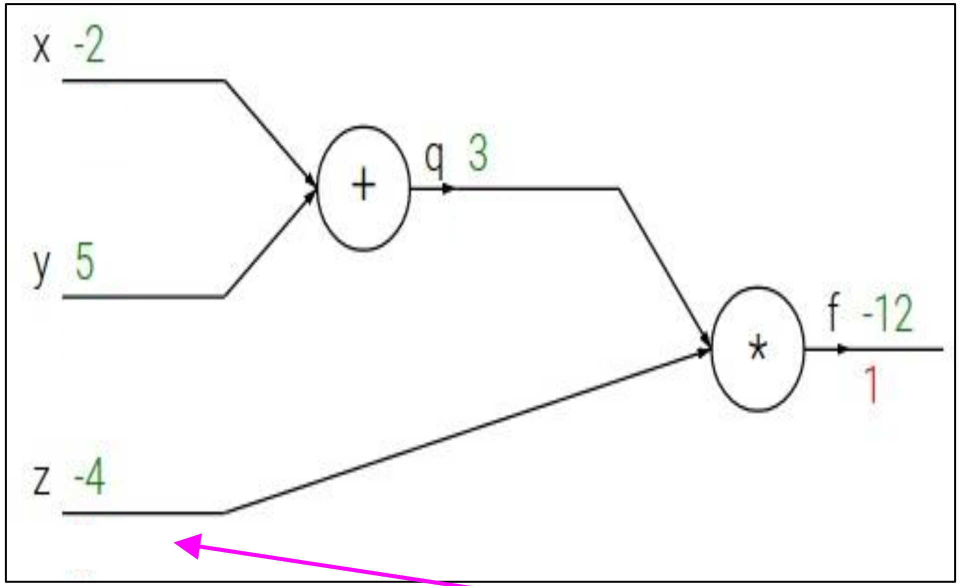
$$f(x,y,z) = (x+y)z$$

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

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

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

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




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

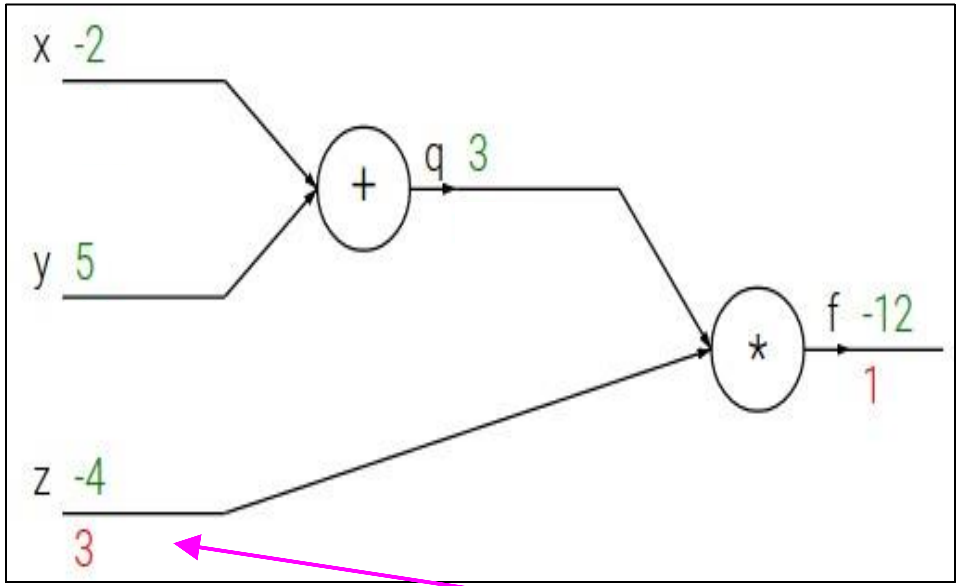
$$f(x,y,z) = (x+y)z$$

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

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

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

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




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

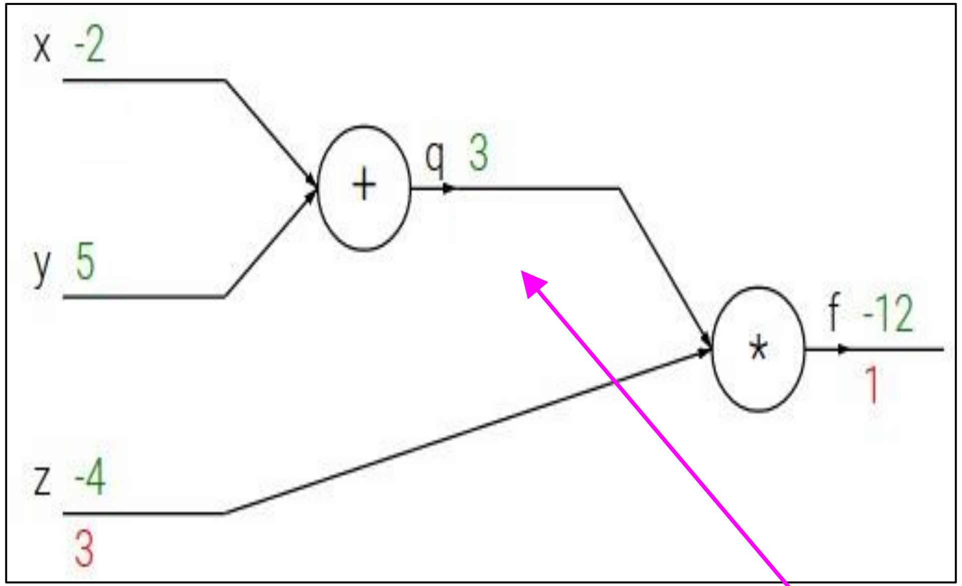
$$f(x,y,z) = (x+y)z$$

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

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

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

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




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

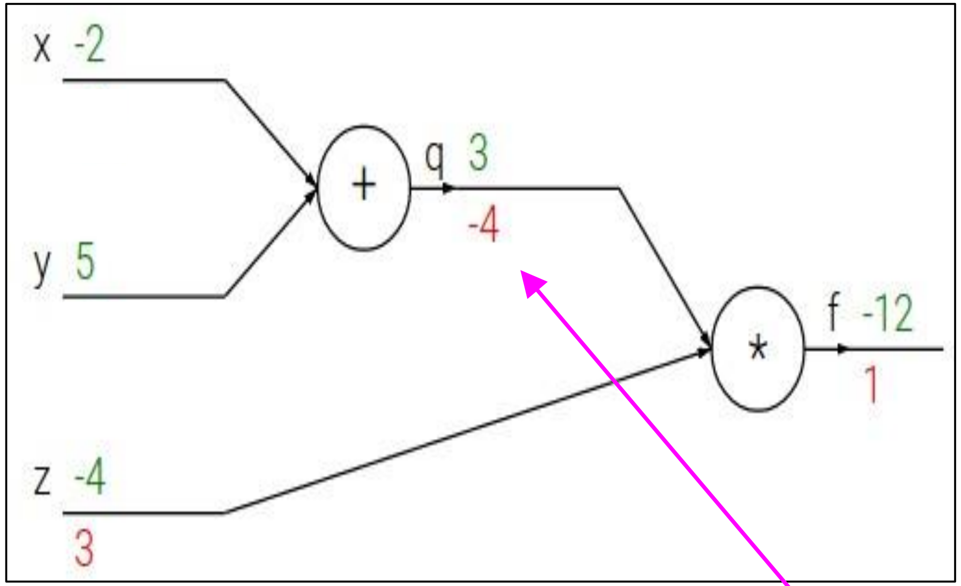
$$f(x,y,z) = (x+y)z$$

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

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

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

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



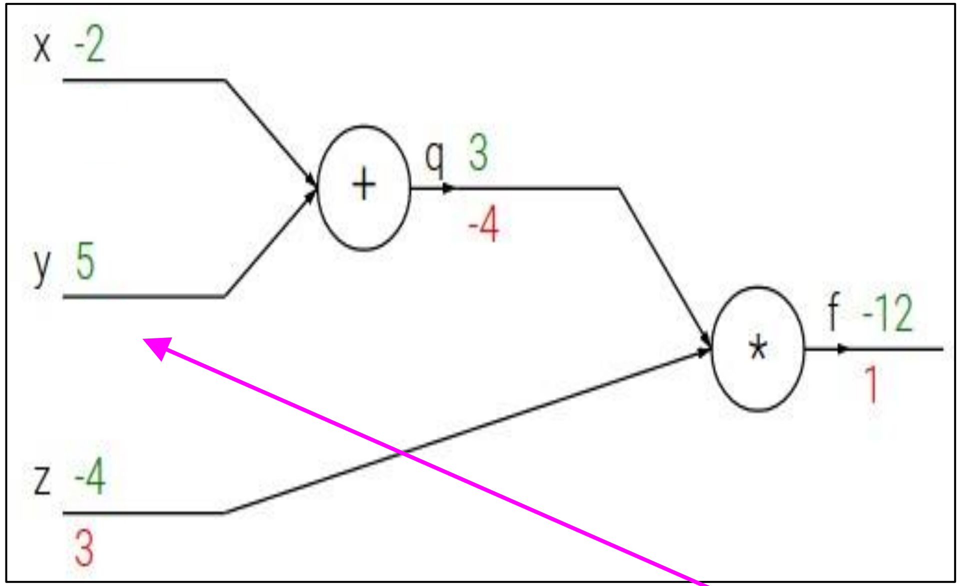

$$\frac{\partial f}{\partial q}$$

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

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

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

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




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

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

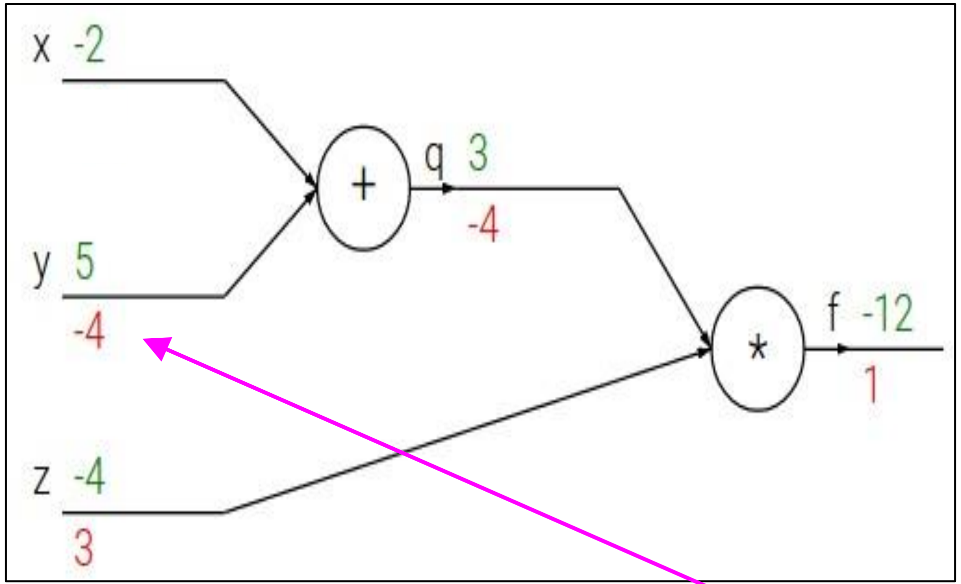
$$f(x,y,z) = (x+y)z$$

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

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

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

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




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

Chain rule:

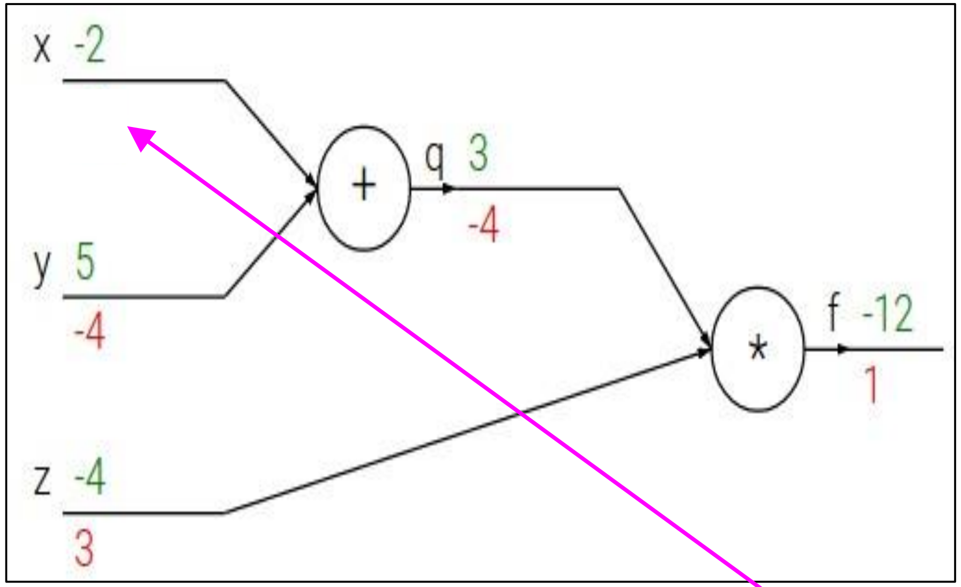
$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$$

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

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

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

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




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

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

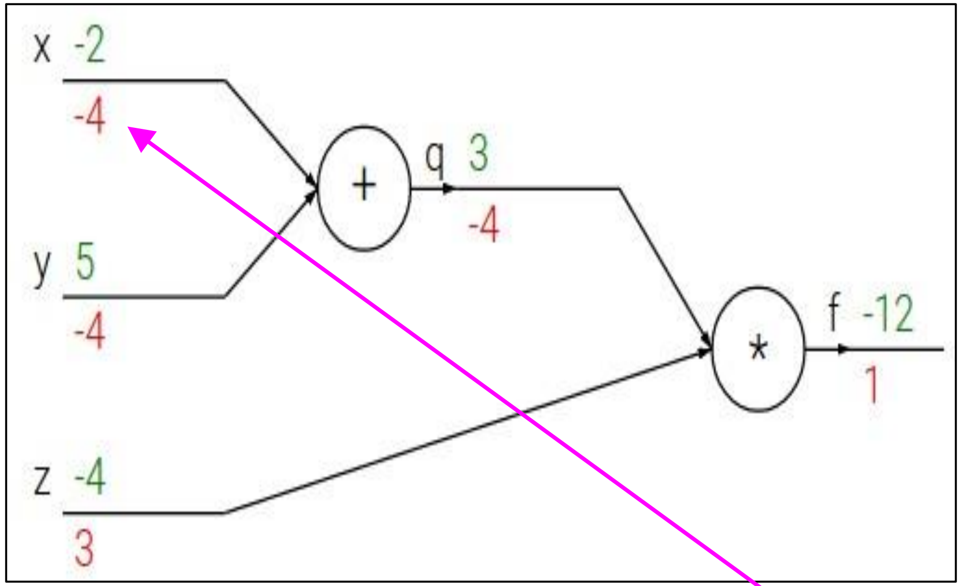
$$f(x, y, z) = (x + y)z$$

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

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

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

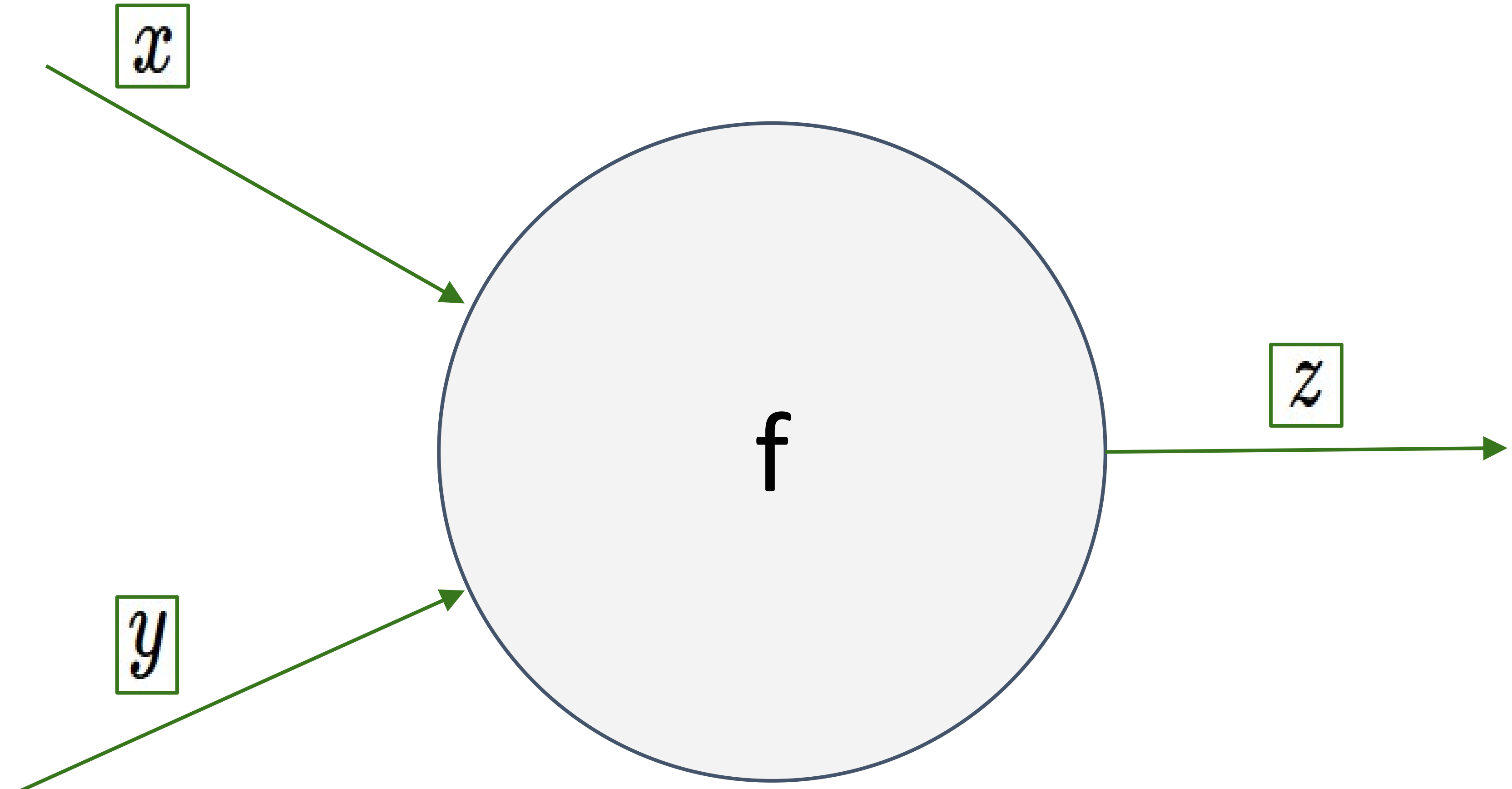
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

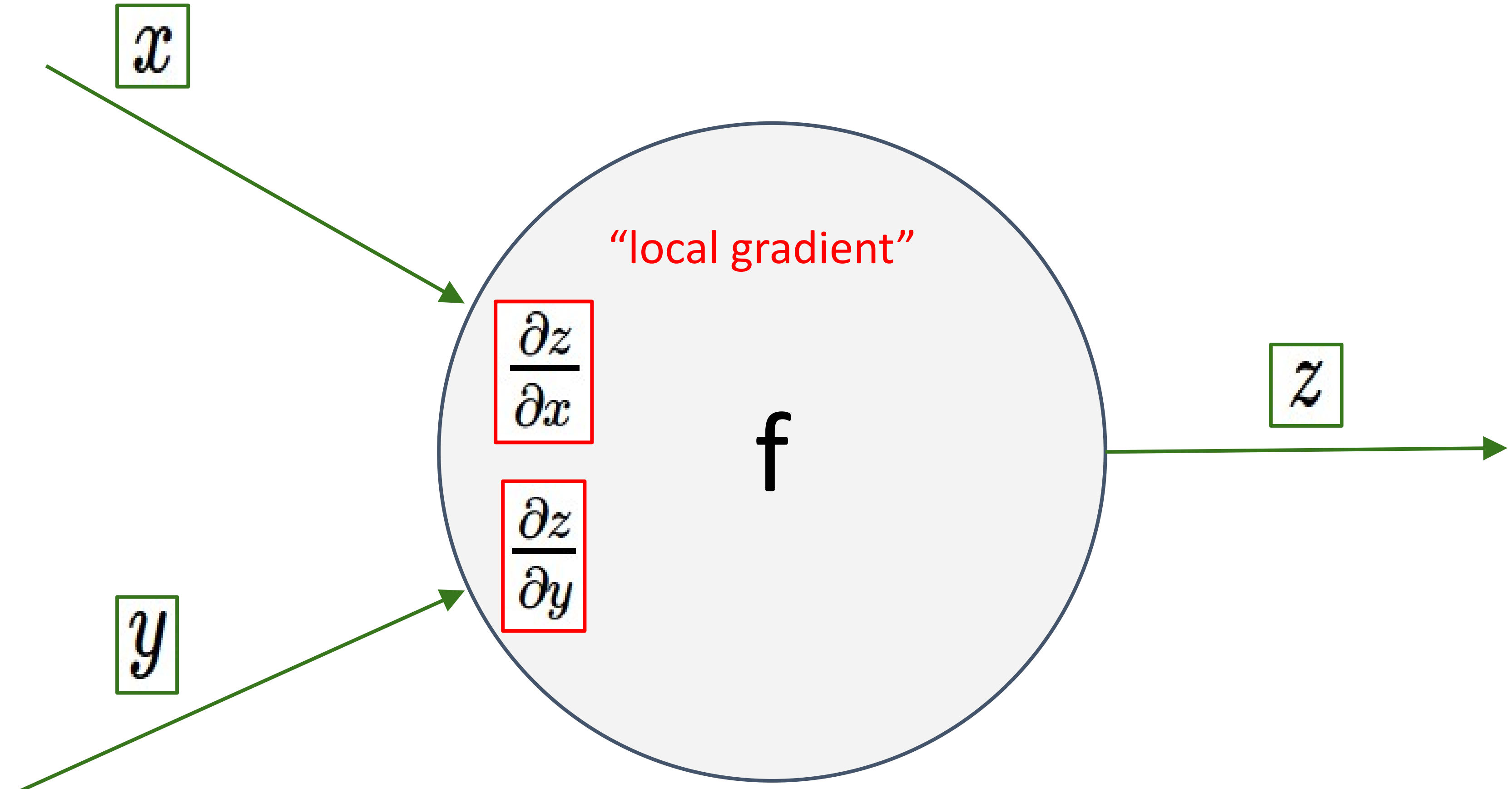


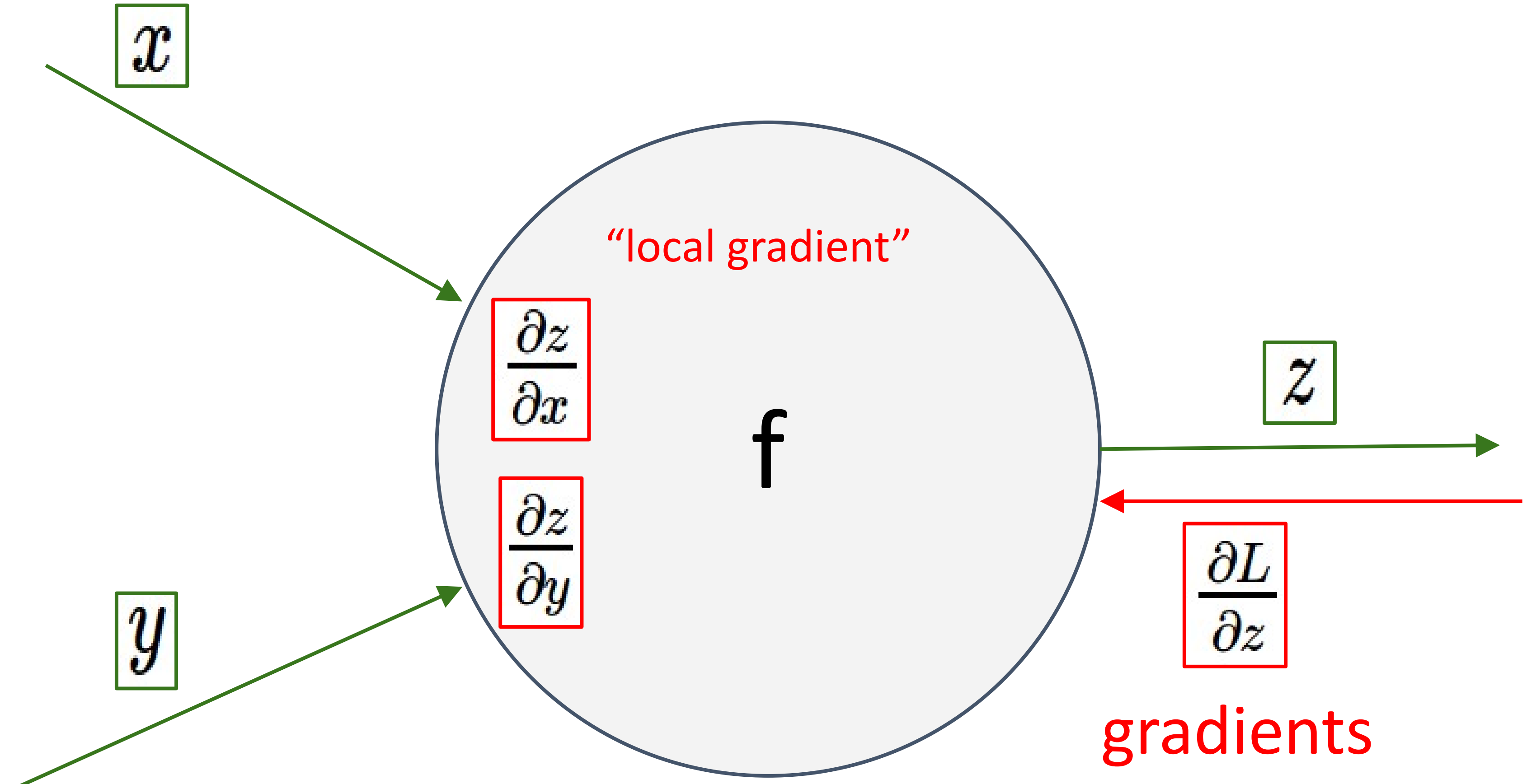

$\frac{\partial f}{\partial x}$

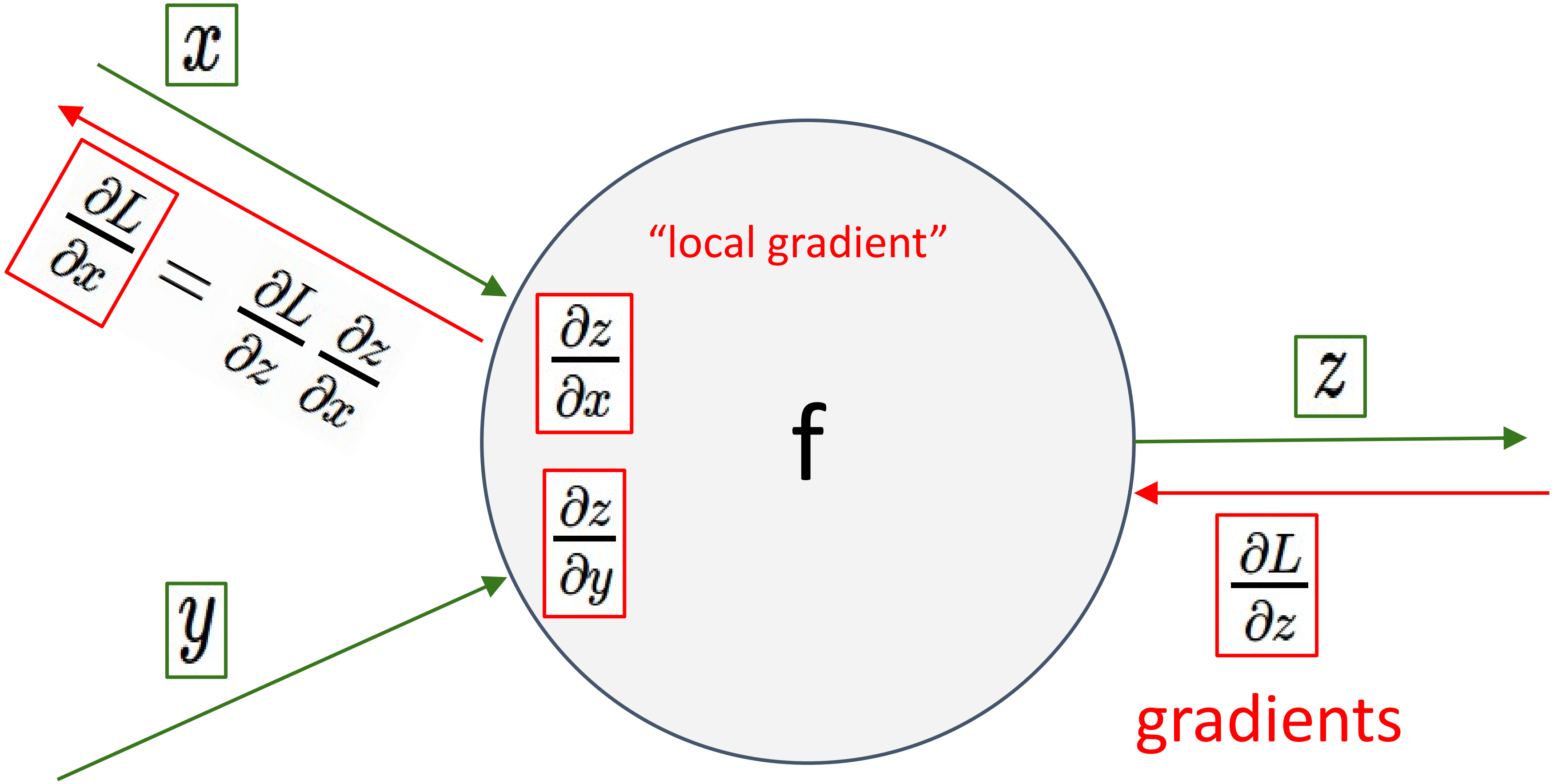
Chain rule:

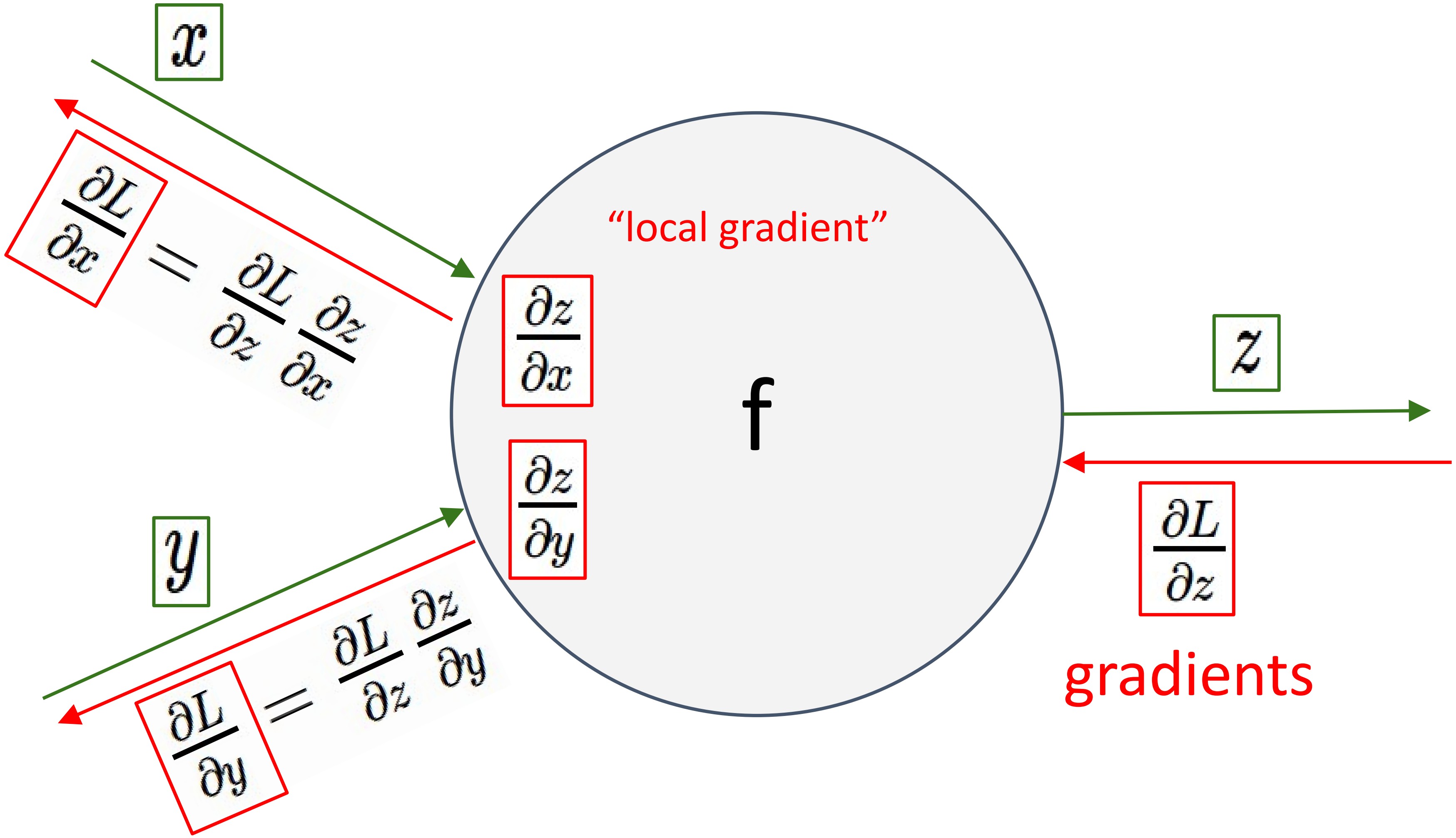
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

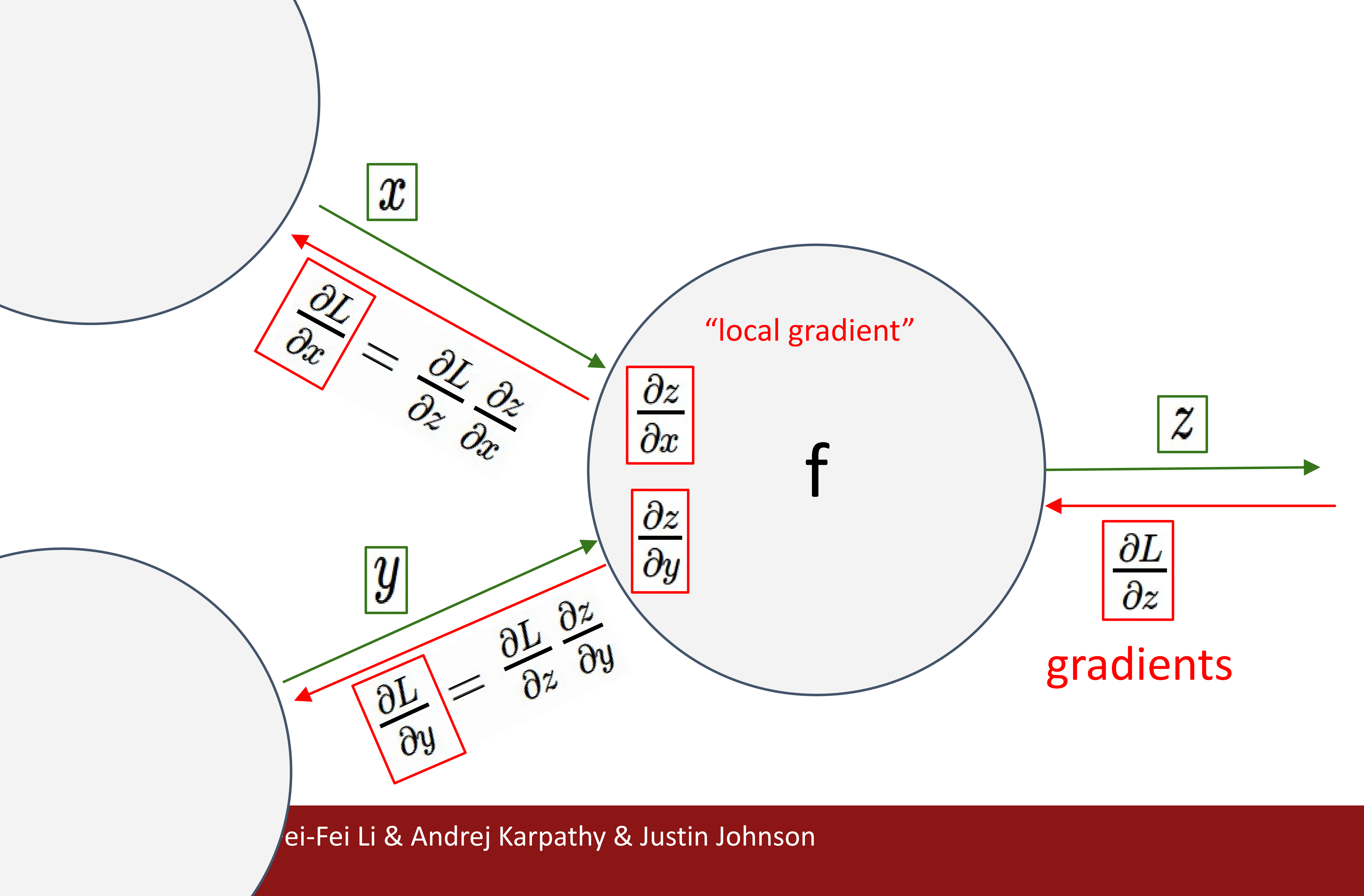

$x$
f
$z$
$y$


$x$
"local gradient"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
$z$
$y$


$x$
"local gradient"
$\frac{\partial z}{\partial x}$
f
$z$
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
$y$
gradients


$x$
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
$z$
$\frac{\partial L}{\partial z}$
$y$
gradients


$x$
$y$
$z$
f
"local gradient"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
gradients
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$

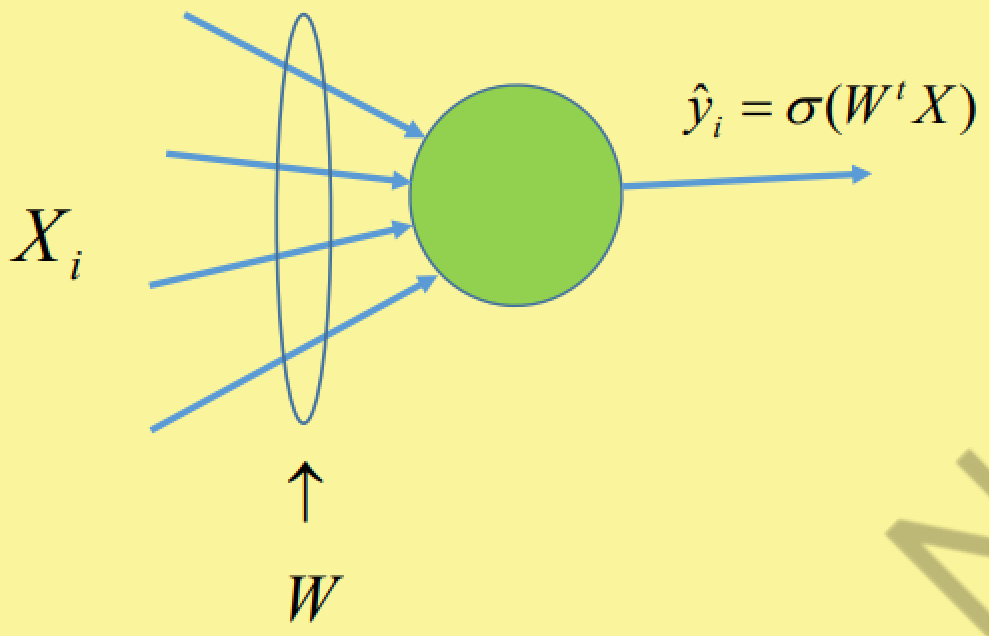

$x$
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
$z$
$\frac{\partial z}{\partial y}$
$y$
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
gradients

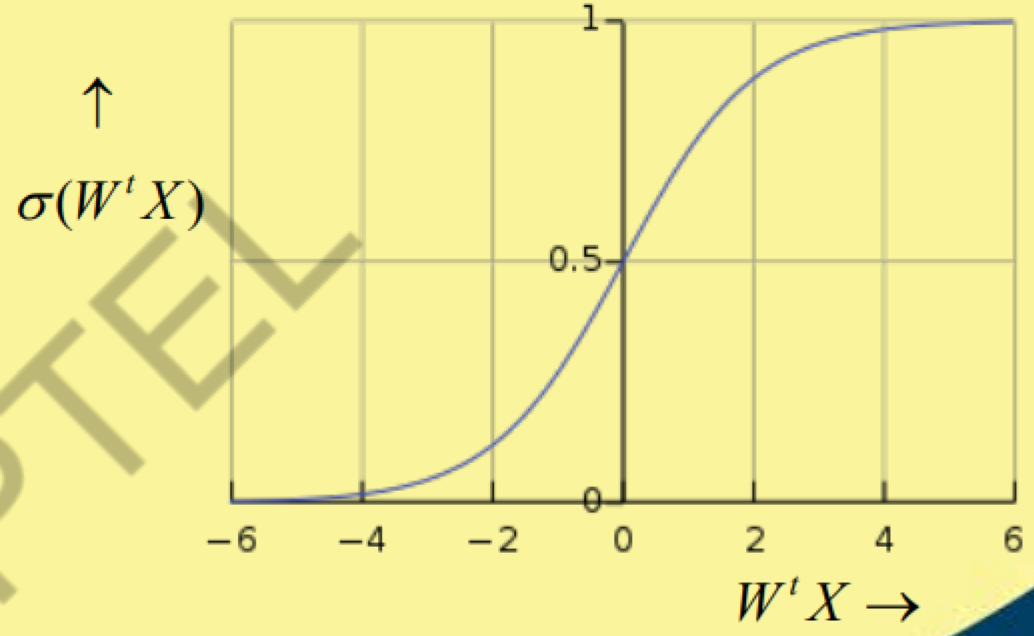
- We got approval to extend the course.
- Everyone who requested admission and has the prereqs, should be admitted soon. Please be patient.
- No new requests will be processed.
- Will try to get late add fees waived.

- VM image available for testing your assignment 1. Assignemnt 1 is due Weds night

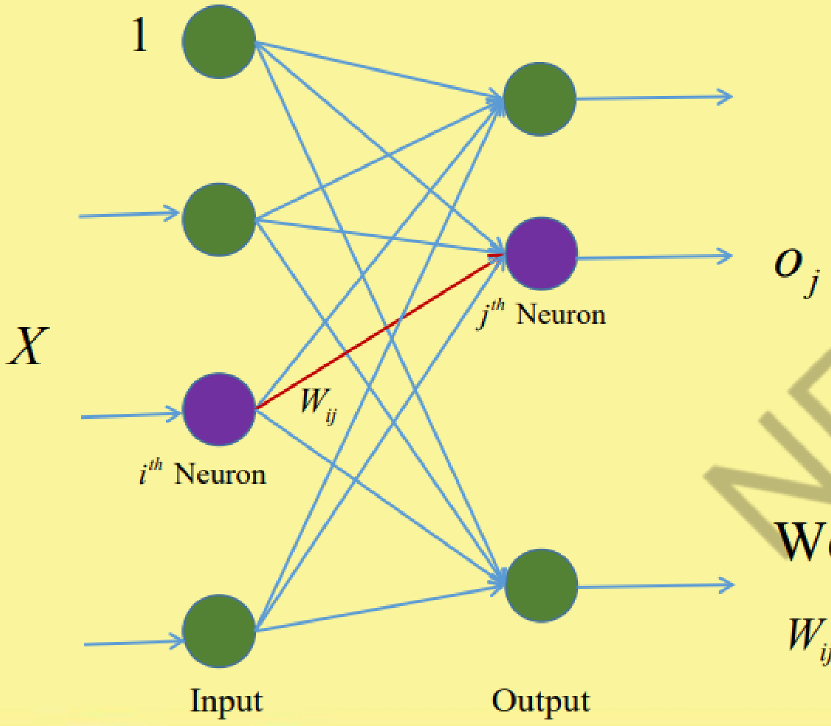
- Course project handout will be coming soon.
- Project can be:
  - A state-of-the-art deep network addressing and important challenge, ideally with Tensorflow.
  - A suggested project from a group on campus.
  - Your own project: see also ImageNet, Yelp and Kaggle challenges

# Single Layer Network- Single Output with nonlinearity



# Back Propagation Learning:- Single Layer Multiple Output




$$o_j = \frac{1}{1+e^{-\theta_j}} \qquad \theta_j = \sum_{i=1}^{D} W_{ij} x_i$$

$$E = \frac{1}{2}\sum_{j=1}^{M}\left(o_j - t_j\right)^2$$

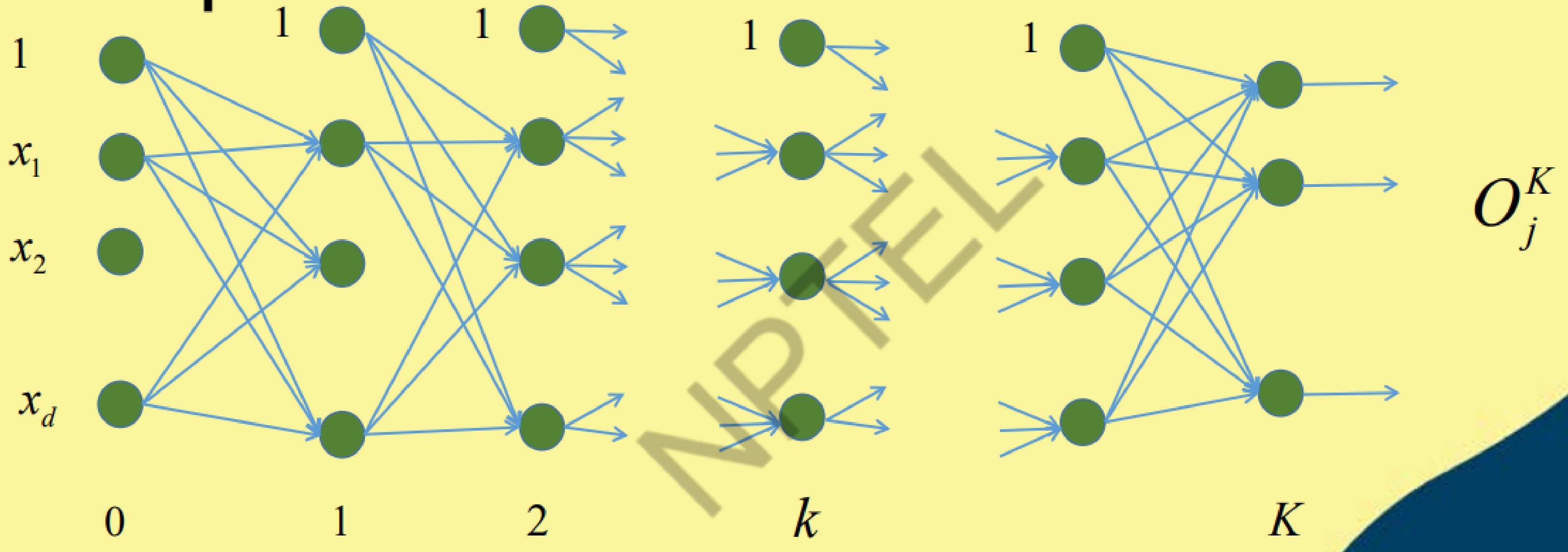
$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial o_j}\cdot\frac{\partial o_j}{\partial \theta_j}\cdot\frac{\partial \theta_j}{\partial W_{ij}}$$

$$= (o_j - t_j)o_j(1-o_j)x_i$$

Weight updation rule $\Rightarrow$

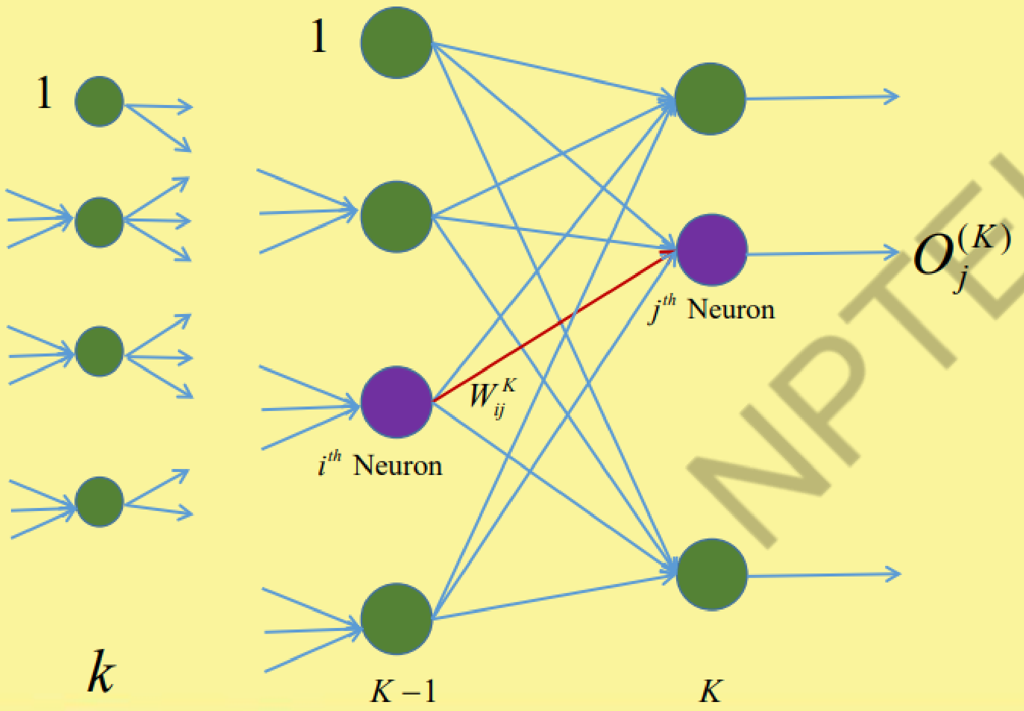
$$W_{ij} \leftarrow W_{ij} - \eta(o_j - t_j)o_j(1-o_j)x_i$$

# Multilayer Perceptron




$M_k \rightarrow$ No. of nodes in $k^{th}$ layer

# Back Propagation Learning:- Output Layer




$$O_j^K = \frac{1}{1+e^{-\theta_j^K}} \qquad \theta_j^K = \sum_{i=1}^{M_{K-1}} W_{ij}^K x_i^{K-1}$$

$$E = \frac{1}{2}\sum_{j=1}^{M_K}\left(O_j^K - t_j\right)^2$$

# Back Propagation Learning:- Output Layer

Find $W_{ij}^{K}$ that minimizes $E = \frac{1}{2}\sum_{j=1}^{M_K}\left(O_j^K - t_j\right)^2$

Gradient Descent $\dfrac{\partial E}{\partial W_{ij}^{K}}$

# Back Propagation Learning:- Output Layer

$$\frac{\partial E}{\partial W_{ij}^K} = \frac{\partial E}{\partial O_j^K} \cdot \frac{\partial O_j^K}{\partial \theta_j^K} \cdot \frac{\partial \theta_j^K}{\partial W_{ij}^K}$$

$$= (O_j^K - t_j) O_j^K (1 - O_j^K) O_i^{K-1}$$

Let $\delta_j^K = O_j^K (1 - O_j^K)(O_j^K - t_j)$

$$\Rightarrow \frac{\partial E}{\partial W_{ij}^K} = \delta_j^K O_i^{K-1}$$

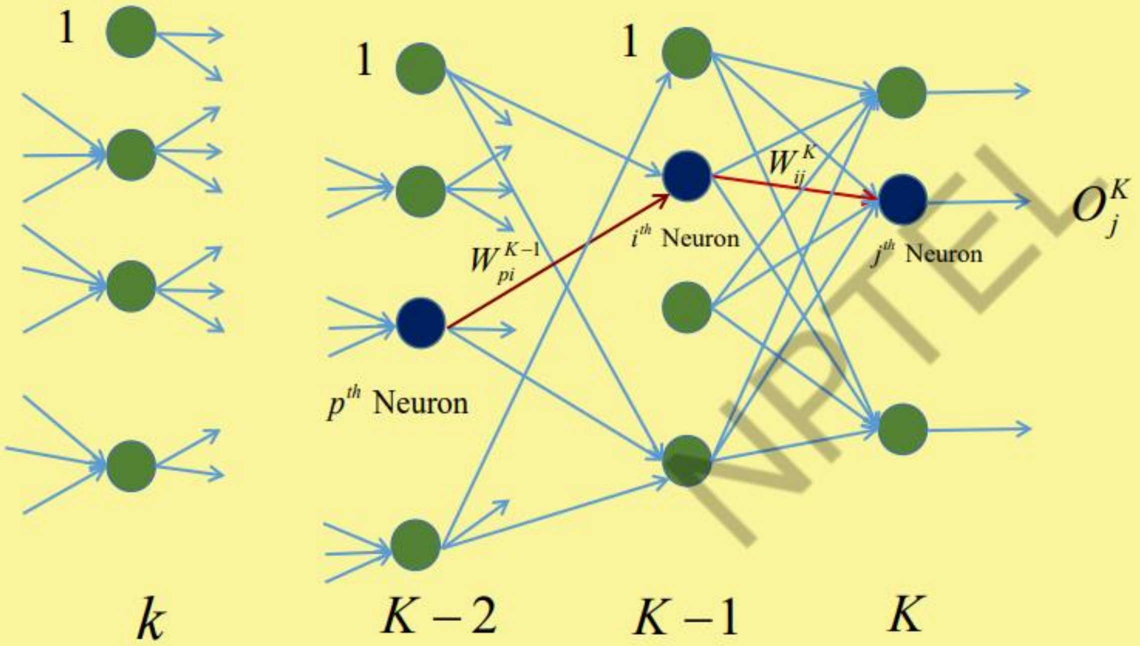
$$O_j^K = \frac{1}{1 + e^{-\theta_j^K}} \qquad \theta_j^K = \sum_{i=1}^{M_{K-1}} W_{ij}^K O_i^{K-1}$$

Weight updation rule

Output Layer

$$W_{ij}^K \leftarrow W_{ij}^K - \eta \delta_j^K O_i^{K-1}$$

# Back Propagation Learning:- Hidden Layer



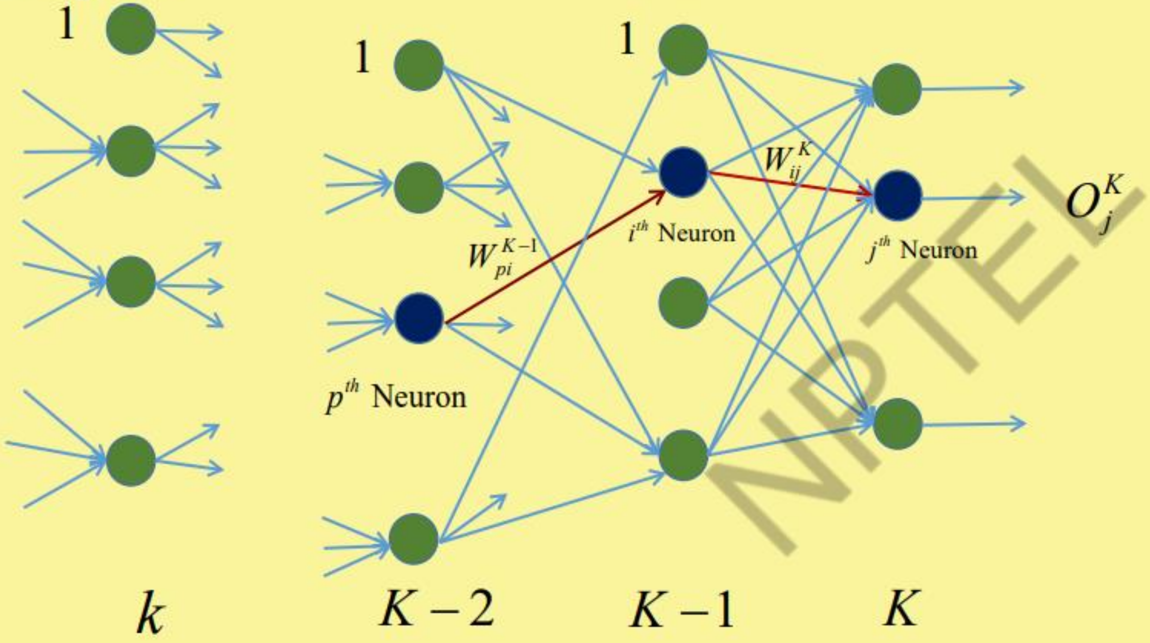

$$E = \frac{1}{2}\sum_{j=1}^{M_K}\left(O_j^K - t_j\right)^2$$

# Back Propagation Learning:- Hidden Layer

Find $W_{pi}^{K-1}$ that minimizes $E = \frac{1}{2}\sum_{j=1}^{M_K}\left(O_j^K - t_j\right)^2$

Gradient Descent $\Rightarrow \frac{\partial E}{\partial W_{pi}^{K-1}}$

# Back Propagation Learning:- Hidden Layer




$$O_i^{K-1} = \frac{1}{1+e^{-\theta_i^{K-1}}}$$

$$\theta_i^{K-1} = \sum_{p=1}^{M_{K-2}} W_{pi}^{K-1} O_p^{K-2}$$

# Back Propagation Learning:- Hidden Layer

$$\frac{\partial E}{\partial W_{pi}^{K-1}} = \frac{\partial E}{\partial O_i^{K-1}} \cdot \frac{\partial O_i^{K-1}}{\partial W_{pi}^{K-1}}$$

$$= \frac{\partial E}{\partial O_i^{K-1}} \cdot \frac{\partial O_i^{K-1}}{\partial \theta_i^{K-1}} \cdot \frac{\partial \theta_i^{K-1}}{\partial W_{pi}^{K-1}}$$

$$= \frac{\partial E}{\partial O_i^{K-1}} . O_i^{K-1}(1 - O_i^{K-1}) . O_p^{K-2}$$

$$O_i^{K-1} = \frac{1}{1 + e^{-\theta_i^{K-1}}}$$

$$\theta_i^{K-1} = \sum_{p=1}^{M_{K-2}} W_{pi}^{K-1} O_p^{K-2}$$

# Back Propagation Learning:- Hidden Layer

$$\frac{\partial E}{\partial O_i^{K-1}} = \frac{\partial E}{\partial O_j^K} \cdot \frac{\partial O_j^K}{\partial \theta_j^K} \cdot \frac{\partial \theta_j^K}{\partial O_i^{K-1}}$$

$$= \sum_{j=1}^{M_K} (O_j^k - t_j) O_j^K (1 - O_j^K) W_{ij}^K$$

$$= \sum_{j=1}^{M_K} \partial_j^K W_{ij}^K$$

$$E = \frac{1}{2} \sum_{j=1}^{M_K} \left(O_j^K - t_j\right)^2$$

$$O_j^K = \frac{1}{1 + e^{-\theta_j^K}} \qquad \theta_j^K = \sum_{i=1}^{M_{K-1}} W_{ij}^K O_i^{K-1}$$

$$\delta_j^K = O_j^K (1 - O_j^K)(O_j^K - t_j)$$

# Back Propagation Learning:- Hidden Layer

$$\frac{\partial E}{\partial W_{pi}^{K-1}} = \frac{\partial E}{\partial x_i^{K-1}} .O_i^{K-1}(1-O_i^{K-1}).O_p^{K-2} = O_i^{K-1}(1-O_i^{K-1}).O_p^{K-2}\sum_{j=1}^{M_K}\partial_j^K W_{ij}^K$$

Putting $\delta_i^{K-1} = O_i^{K-1}(1-O_i^{K-1})\sum_{j=1}^{M_K}\partial_j^K W_{ij}^K$

Weight updation rule

Last but Output Layer

$$W_{pi}^{K-1} \leftarrow W_{pi}^{K-1} - \eta\delta_i^{K-1}O_p^{K-2}$$

# Back Propagation Learning:- any Hidden Layer

For any hidden layer weight $W_{ij}^{k}$

Putting $\delta_i^k = O_i^k(1 - O_i^k)\sum_{j=1}^{M_{k+1}} \partial_j^{k+1} W_{ij}^{k+1}$

Weight updation rule

$$W_{ij}^{k} \leftarrow W_{ij}^{k} - \eta \delta_j^k O_i^{k-1}$$