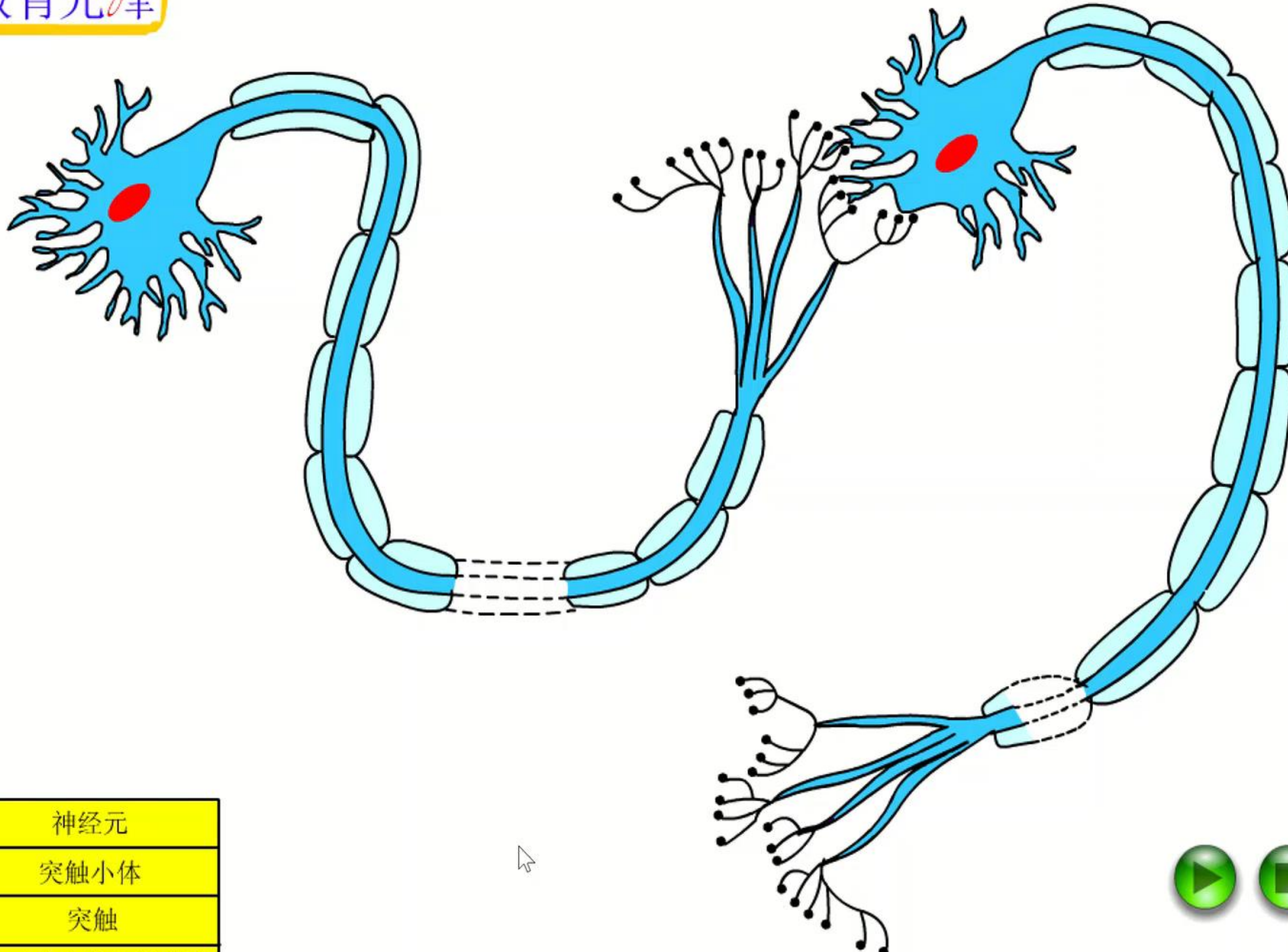
Neural Network

Outline

- Modeling one neuron
- Activation functions
- Fully connected feed-forward network
- How to train a multi-layer network
- Representational power of NN

Biology

- Neurons respond slowly
  - 10^{-3} s compared to 10^{-9} s for electrical circuits
- The brain uses massively parallel computation
  - $\approx 10^{11}$ neurons in the brain
  - $\approx 10^{4}$ connections per neuron



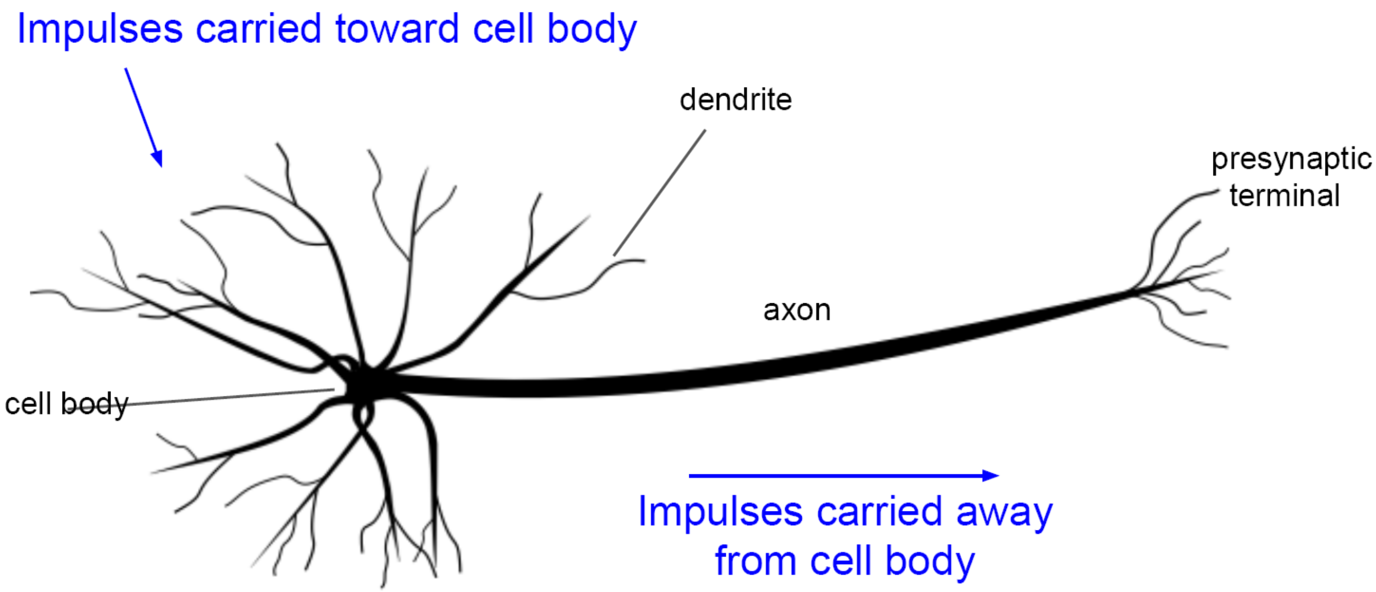
神经元

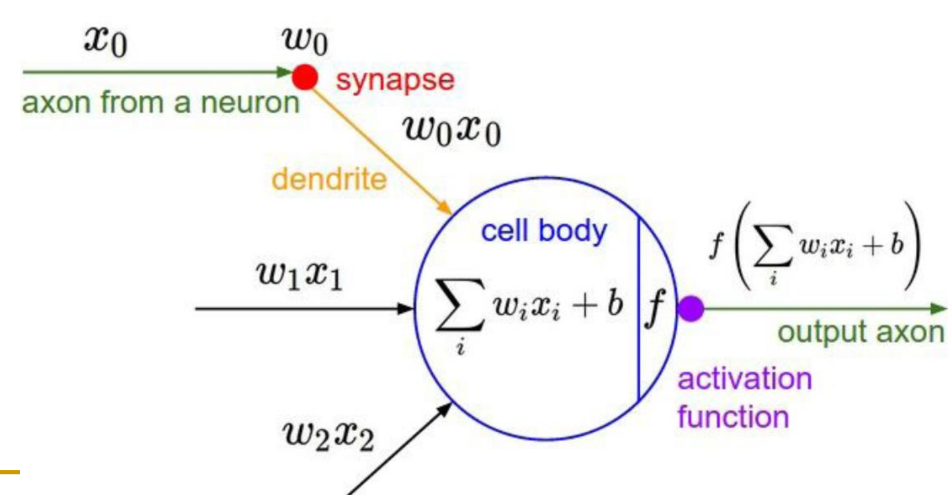
突触小体

突触

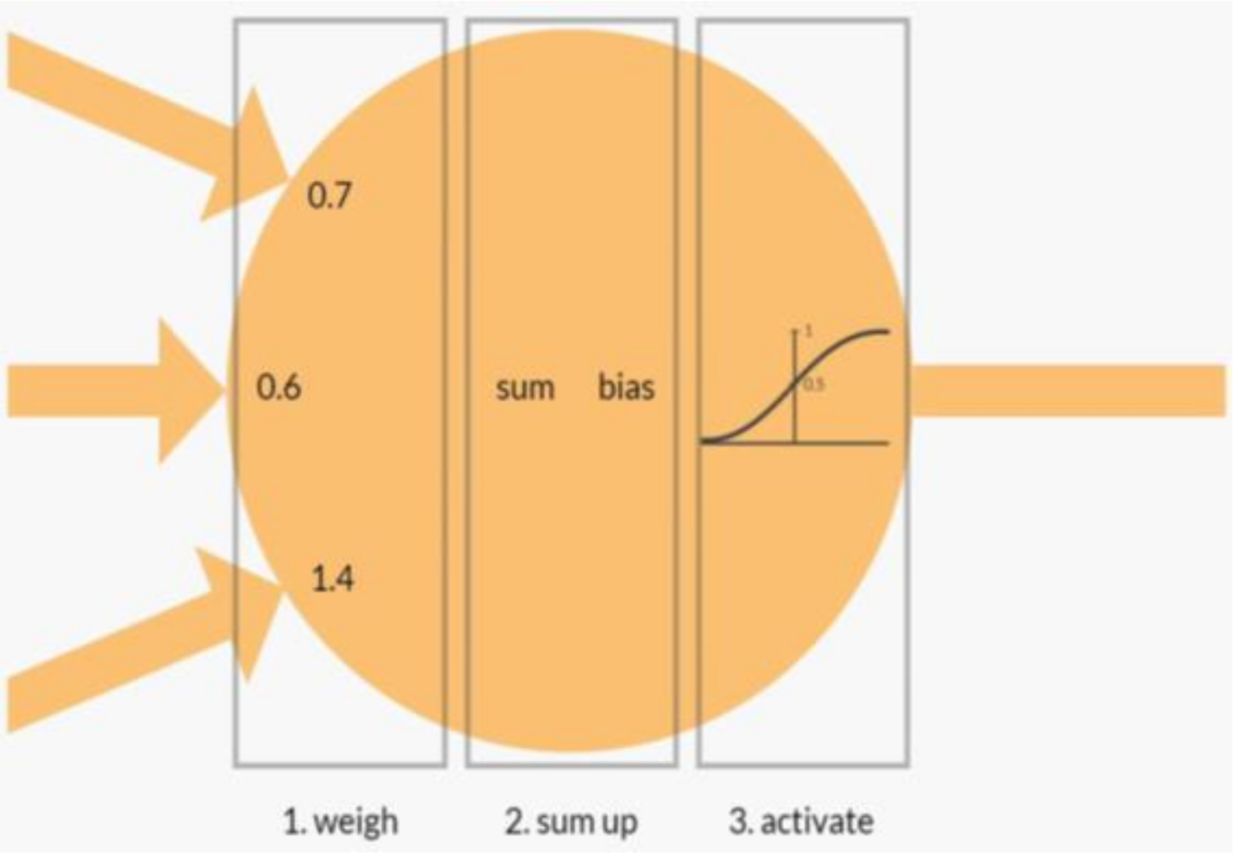
冲动在突触上的传递

Modeling one neuron





Neuron



Single neuron as a linear classifier

- With an appropriate loss function on the neuron's output, a single neuron can be turned into a linear classifier.
- A single neuron can be used to implement a binary classifier (e.g. binary Softmax or binary SVM classifiers).

Single neuron as a linear classifier

- **Binary Softmax classifier**
  - Interpreting:
    - $\sigma(\sum_i w_i x_i + b)$ to be the probability of one of the classes

      $P(y_i = 1 \mid x_i; w)$
    - The probability of the other class to be

      $P(y_i = 0 \mid x_i; w) = 1 - P(y_i = 1 \mid x_i; w)$
  - With this interpretation, we can formulate the cross-entropy loss , and optimizing it would lead to a **binary Softmax classifier** (also known as logistic regression)
- Alternatively, we could attach a max-margin hinge loss to the output of the neuron and train it to become a **binary Support Vector Machine**.

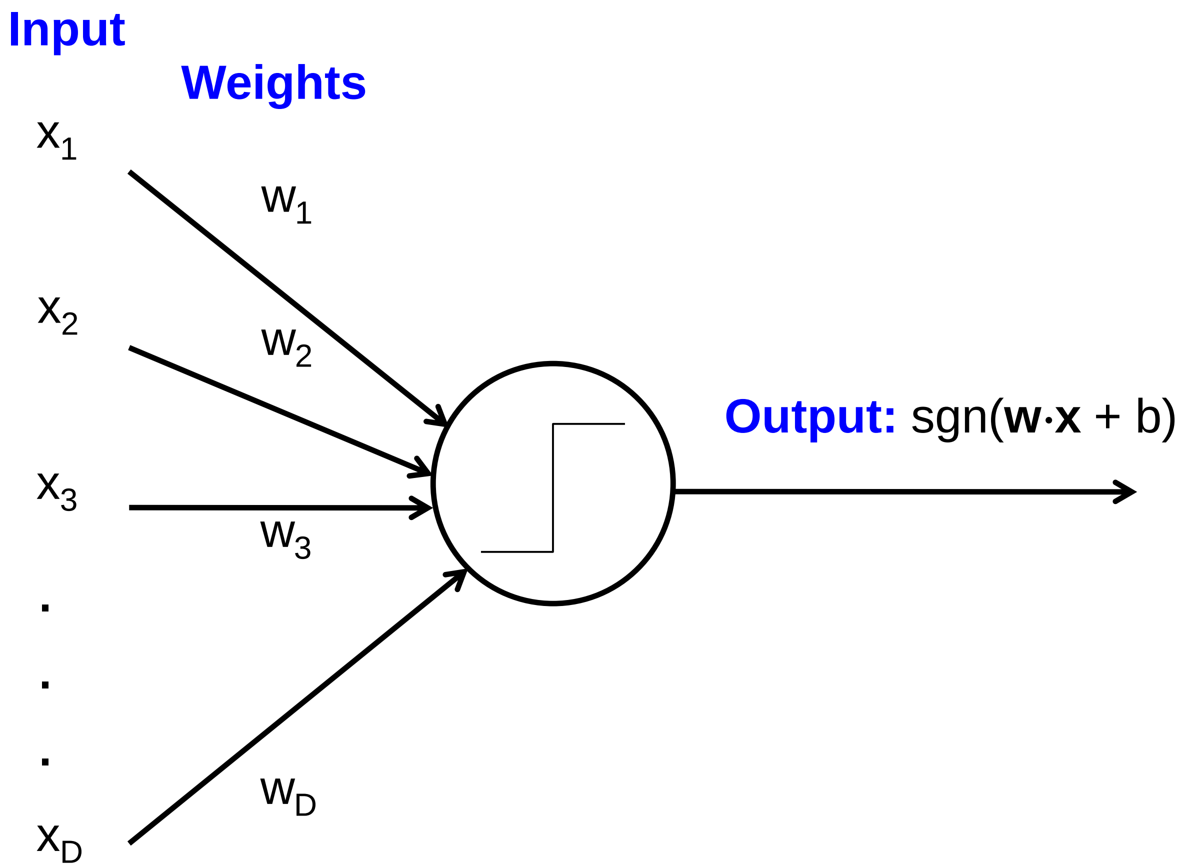
Outline

- Modeling one neuron
- Activation functions
- Fully connected feed-forward network
- How to train a multi-layer network
- Representational power of NN

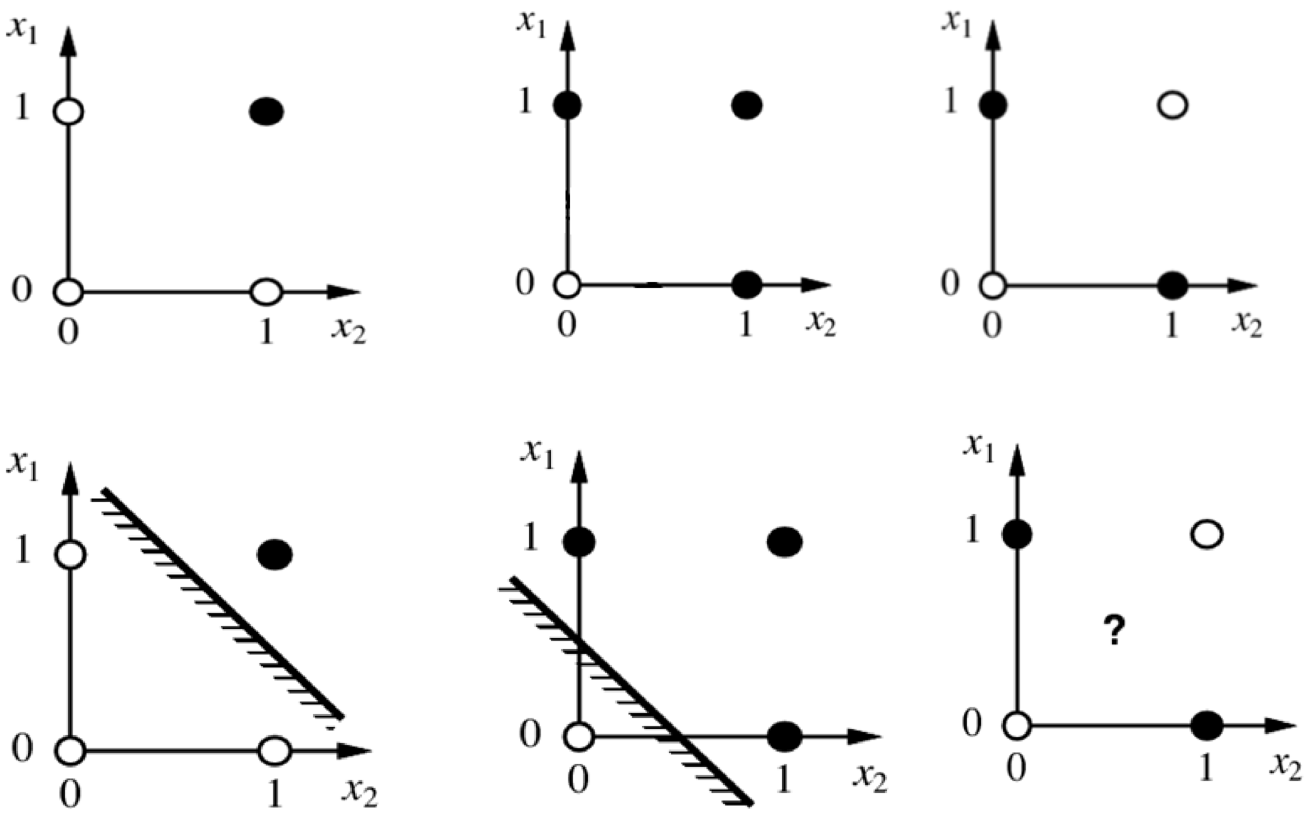
Activation Functions

- Neural networks are used to implement complex functions, and non-linear Activation Functions enable them to approximate arbitrarily complex functions.
- Without the non-linearity introduced by the Activation Function：
  - a neuron is equivalent to a linear classifier
  - a multi-layer neural network is equivalent to a single layer neural network

Perceptron



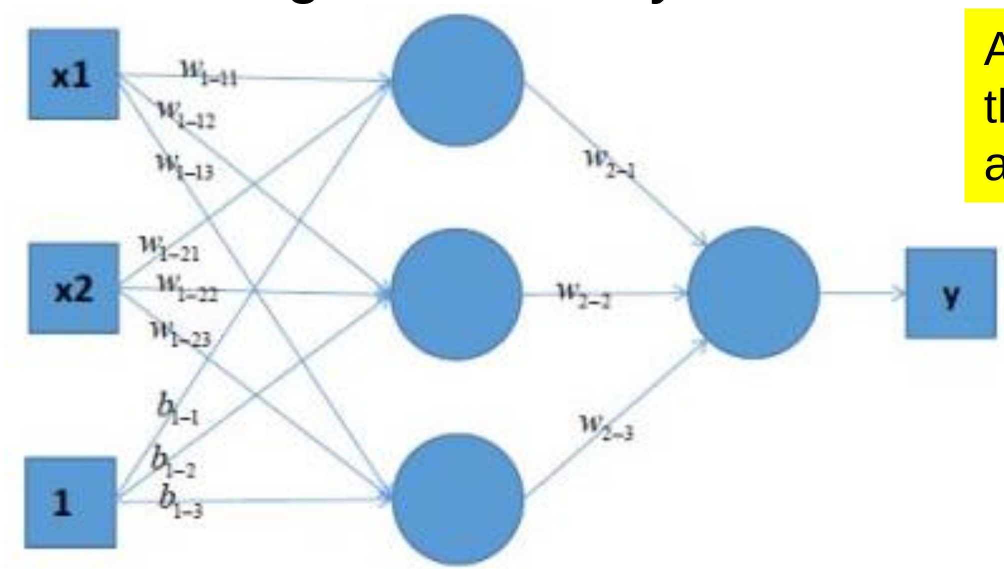
Linear separability



x_1 **and** x_2 x_1 **or** x_2 x_1 **xor** x_2

Multilayer Perceptron（MLP）

- A MLP with a single hidden layer



Add nonlinearity though activation functions!

$$y = w_{2-1}(w_{1-11}x_1 + w_{1-21}x_2 + b_{1-1}) + w_{2-2}(w_{1-12}x_1 + w_{1-22}x_2 + b_{1-2}) + w_{2-3}(w_{1-13}x_1 + w_{1-23}x_2 + b_{1-3})$$

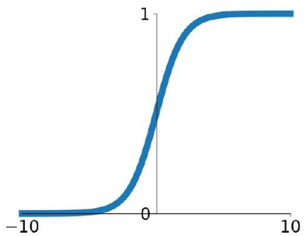
$$y = x_1(w_{2-1}w_{1-11} + w_{2-2}w_{1-12} + w_{2-3}w_{1-13}) + x_2(w_{2-1}w_{1-21} + w_{2-2}w_{1-22} + w_{2-3}w_{1-23}) + w_{2-1}b_{1-1} + w_{2-2}b_{1-2} + w_{2-3}b_{1-3}$$

Still a linear classifier

Some Activation Functions

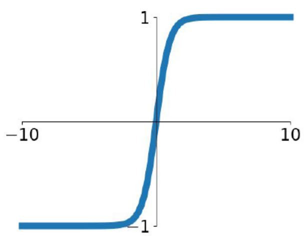
Sigmoid

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



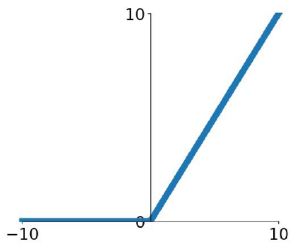
tanh

$\tanh(x)$



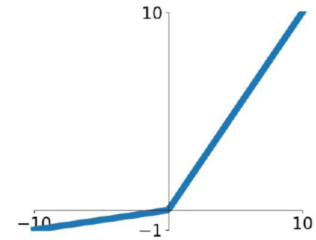
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

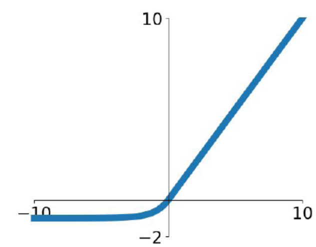


Maxout

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

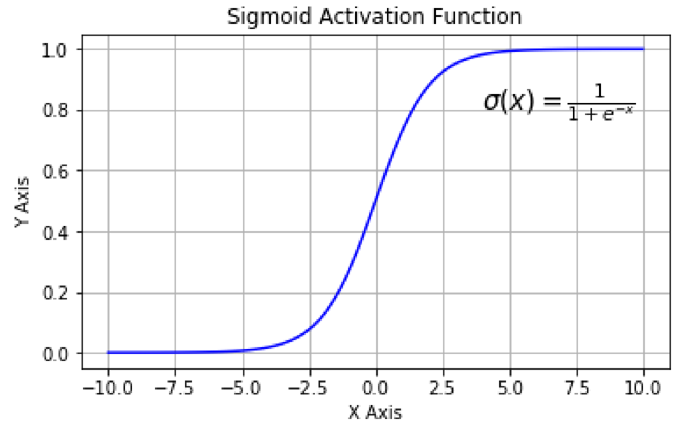
ELU

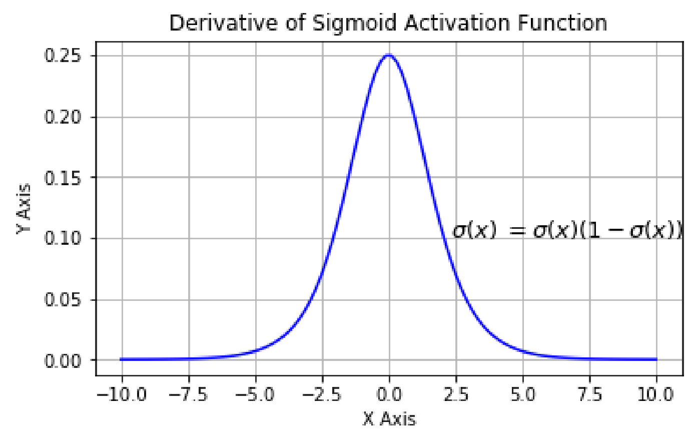
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Activation Functions

- **Sigmoid** $\sigma(x) = 1/(1+e^{-x})$
  - Squashes numbers to range [0,1]
  - Historically popular since they have nice interpretation as a saturating "firing rate" of a neuron
  - Problems:
    - Saturated neurons "kill" the gradients
    - Sigmoid outputs are not zero-centered
    - exp() is a bit compute expensive





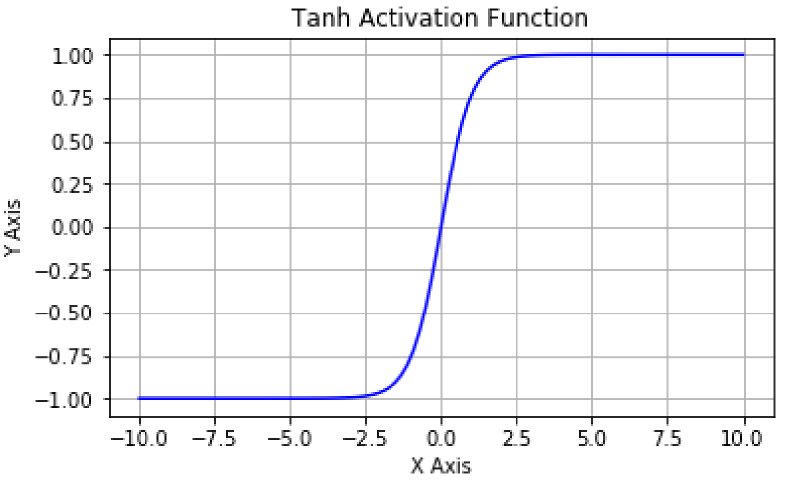
Activation Functions

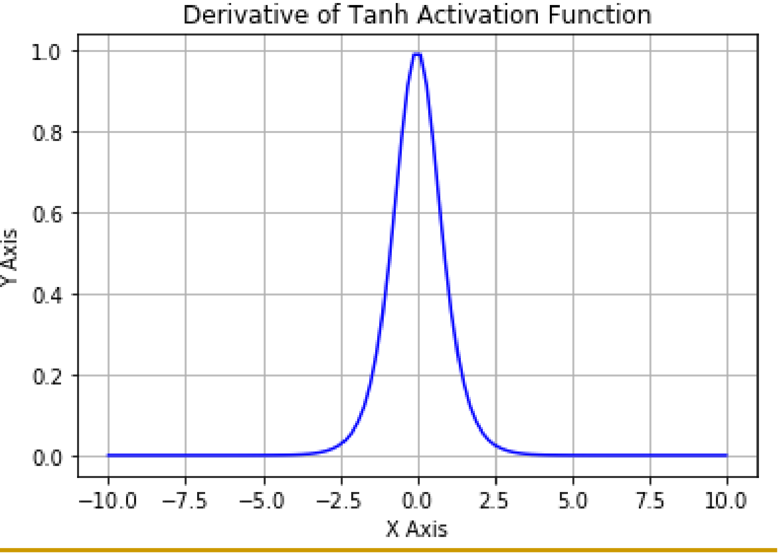
$$tanhx = \frac{sinhx}{coshx} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- **Tanh:**

  $\tanh(x) = 2\sigma(2x) - 1$

  - Squashes numbers to range [-1,1]
  - Zero centered
  - Problem:
    - kills gradients when saturated



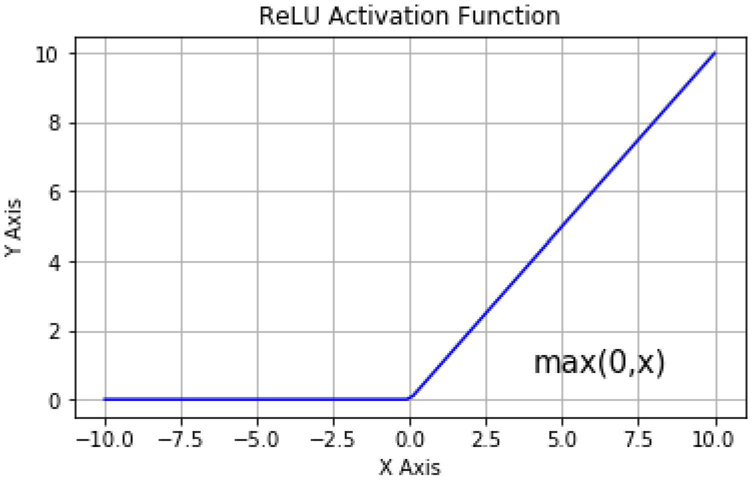


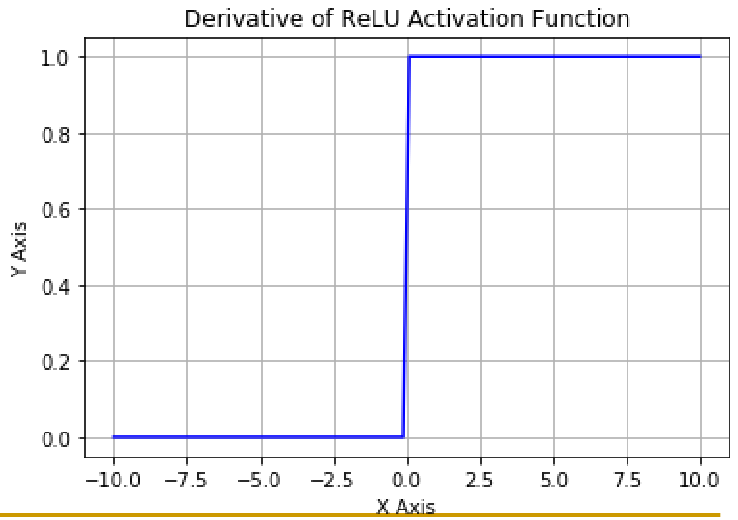
Activation Functions

- **Relu： f(x) = max(0,x)**

  （Rectified Linear Unit）

  - Does not saturate (in +region)
  - Very computationally efficient
  - Converges much faster than sigmoid/tanh in practice (e.g. 6x)
  - Actually more biologically plausible than sigmoid
  - Problem:
    - Not zero-centered output

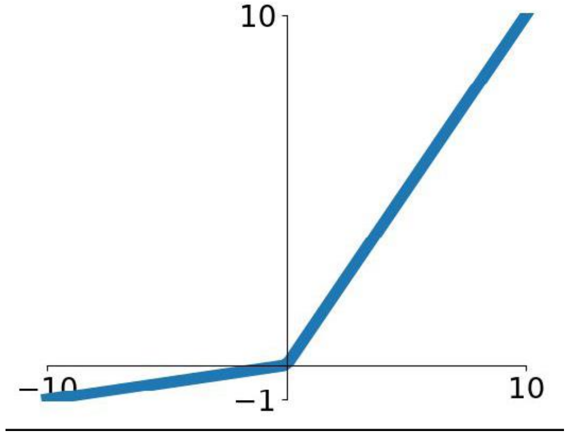




Activation Functions:

Leaky ReLU

- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/ tanh in practice! (e.g. 6x)
- Will not “die”.



Leaky ReLU

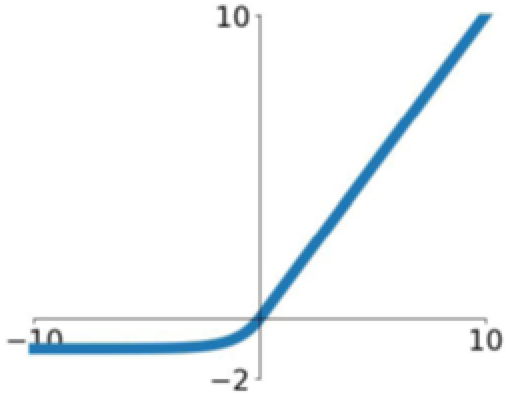
$$f(x) = \max(0.01x, x)$$

Parametric Rectifier (PReLU)

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

Activation Functions

- **Exponential Linear Units (ELU)**
  - All benefits of ReLU
  - Closer to zero mean outputs
  - Negative saturation regime compared with Leaky ReLU
  - Adds some robustness to noise



Exponential Linear Units (ELU)

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

Activation Functions:

- **Maxout :** $\max(w_1^T x + b_1, w_2^T x + b_2)$
  - Does not have the basic form of dot product -> nonlinearity
  - Generalizes ReLU and Leaky ReLU
  - Linear Regime! Does not saturate! Does not die!
  - Problem:
    - doubles the number of parameters/neuron

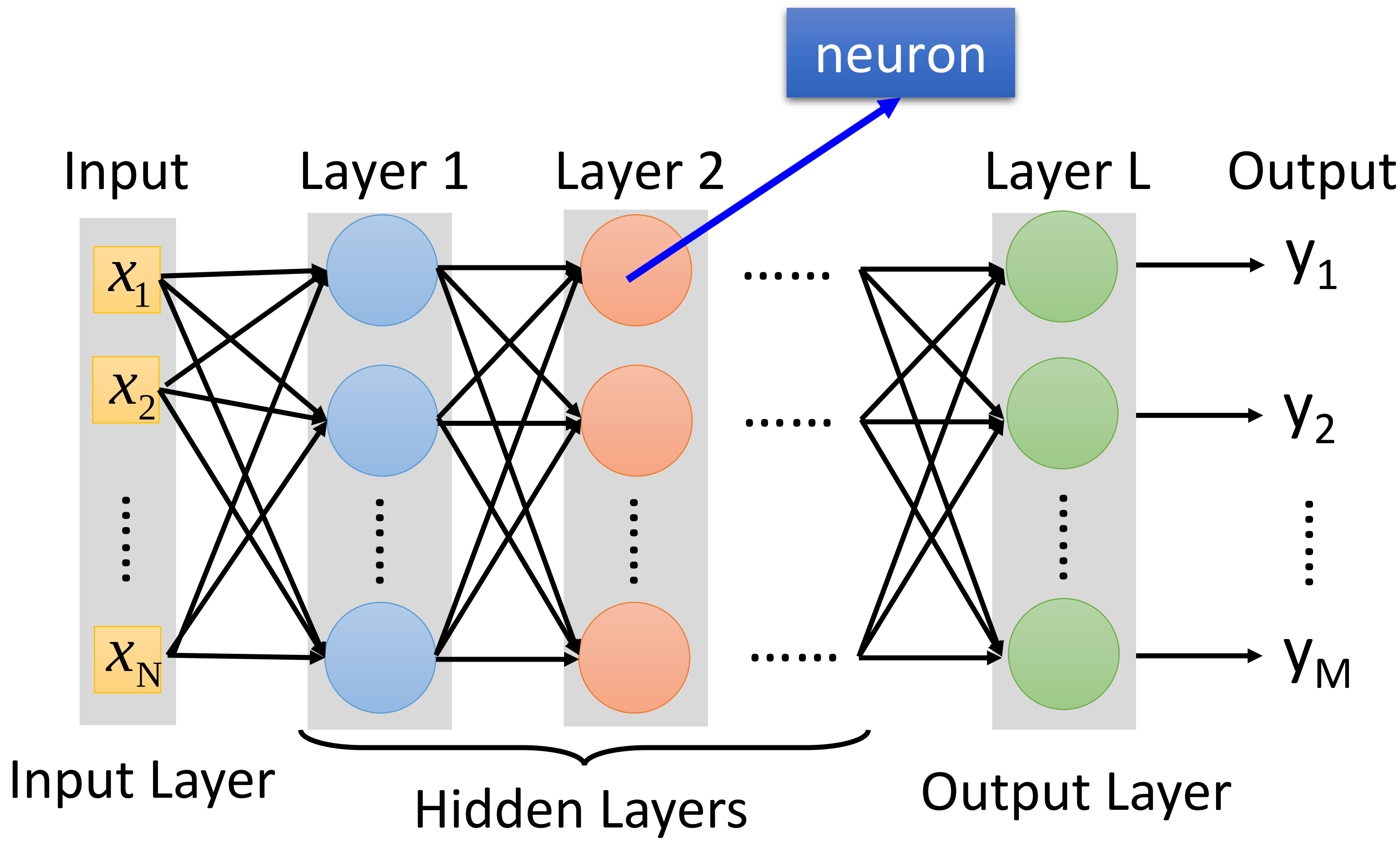
Choosing the right Activation Function

- Good or bad – there is no rule of thumb
- Depending upon the properties of the problem, we might be able to make a better choice for **easy and quicker** convergence of the network
  - Sigmoid functions and their combinations generally work better **in the case of classifiers**
  - Sigmoid and tanh functions: sometimes to be avoided due to the vanishing gradient problem
  - ReLU function : a general activation function, used in most cases these days, but only be used in the hidden layers. Be careful with the learning rates
  - In the case of dead neurons in networks, try out the Leaky ReLU / Maxout / ELU
- Usually can begin with using ReLU function and then move over to other activation functions in case ReLU doesn’t provide with optimum results

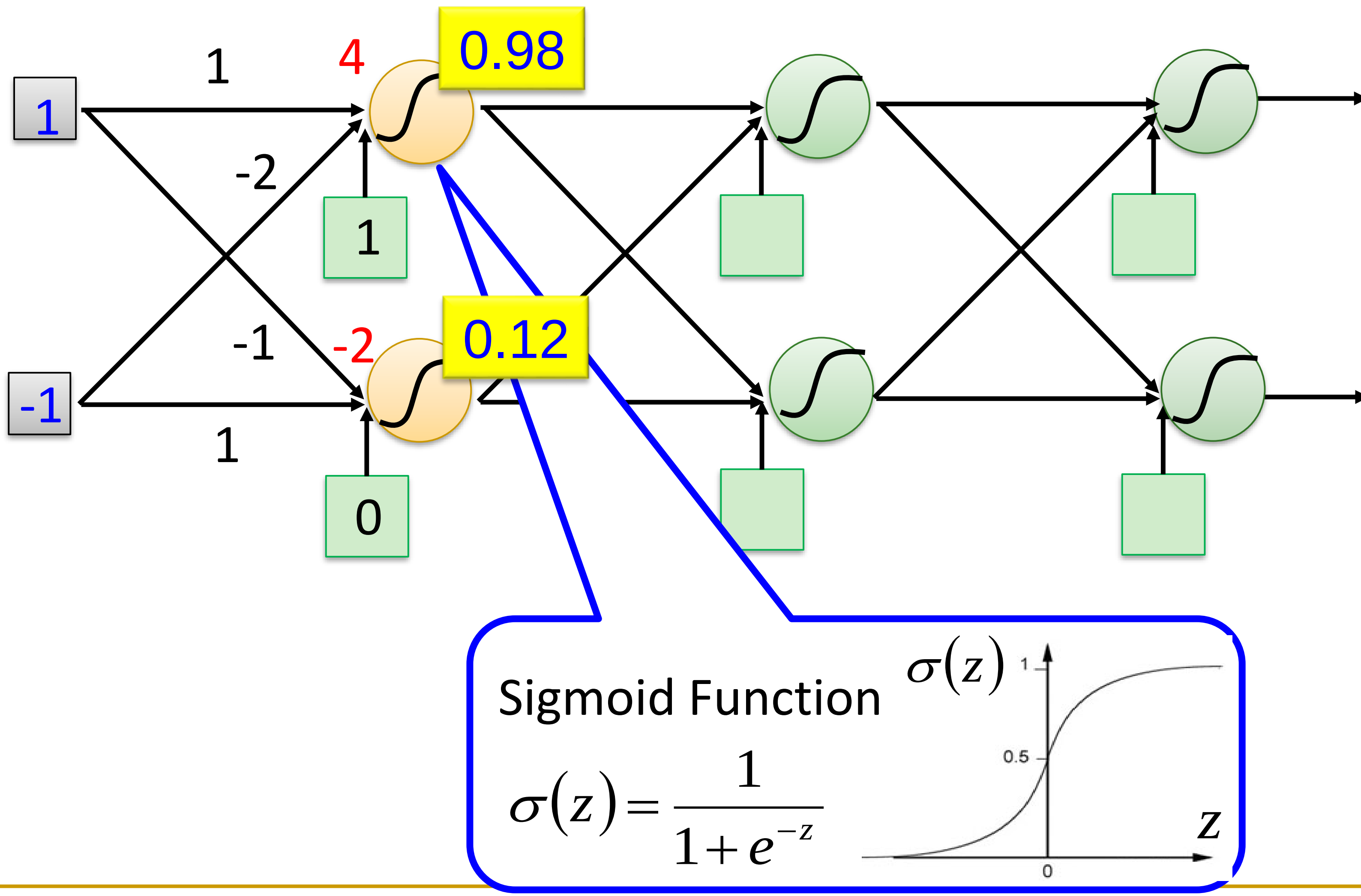
Outline

- Modeling one neuron
- Activation functions
- Fully connected feed-forward network
- How to train a multi-layer network
- Representational power of NN

Fully connected Feedforward Network

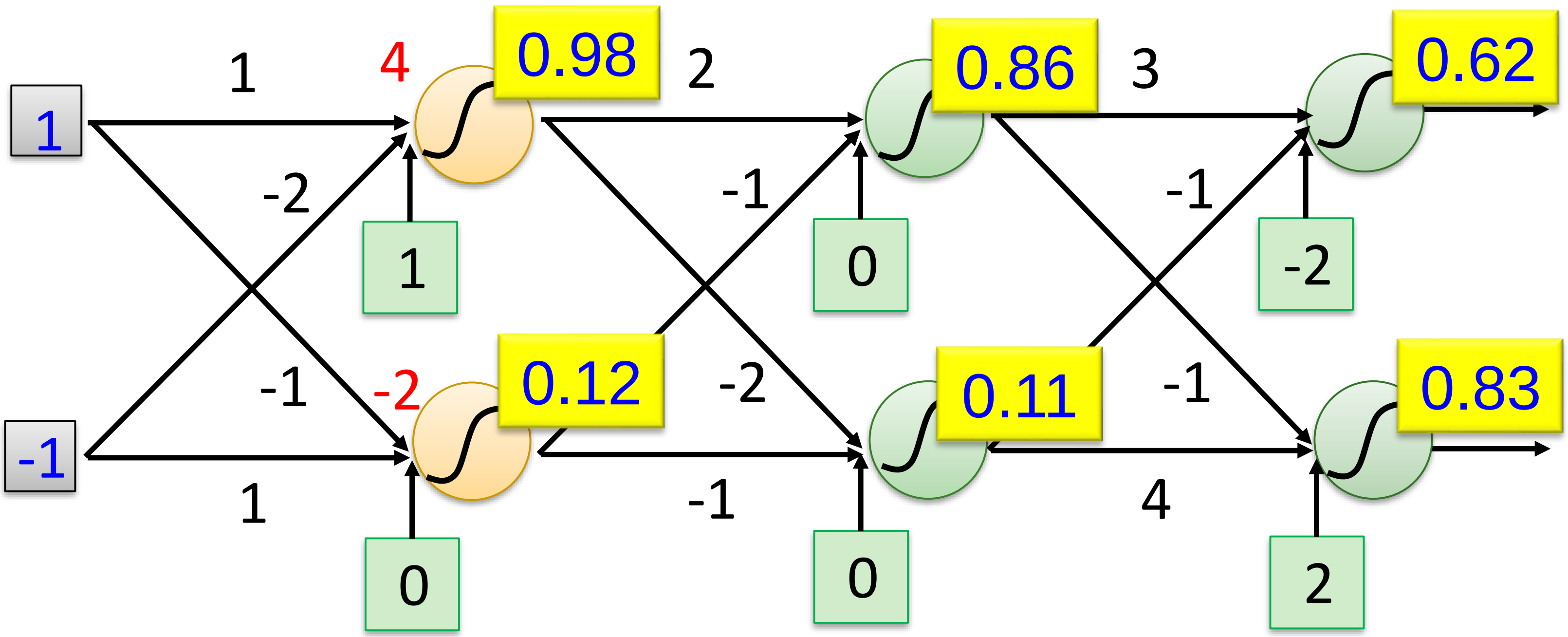


Fully connected Feedforward Network

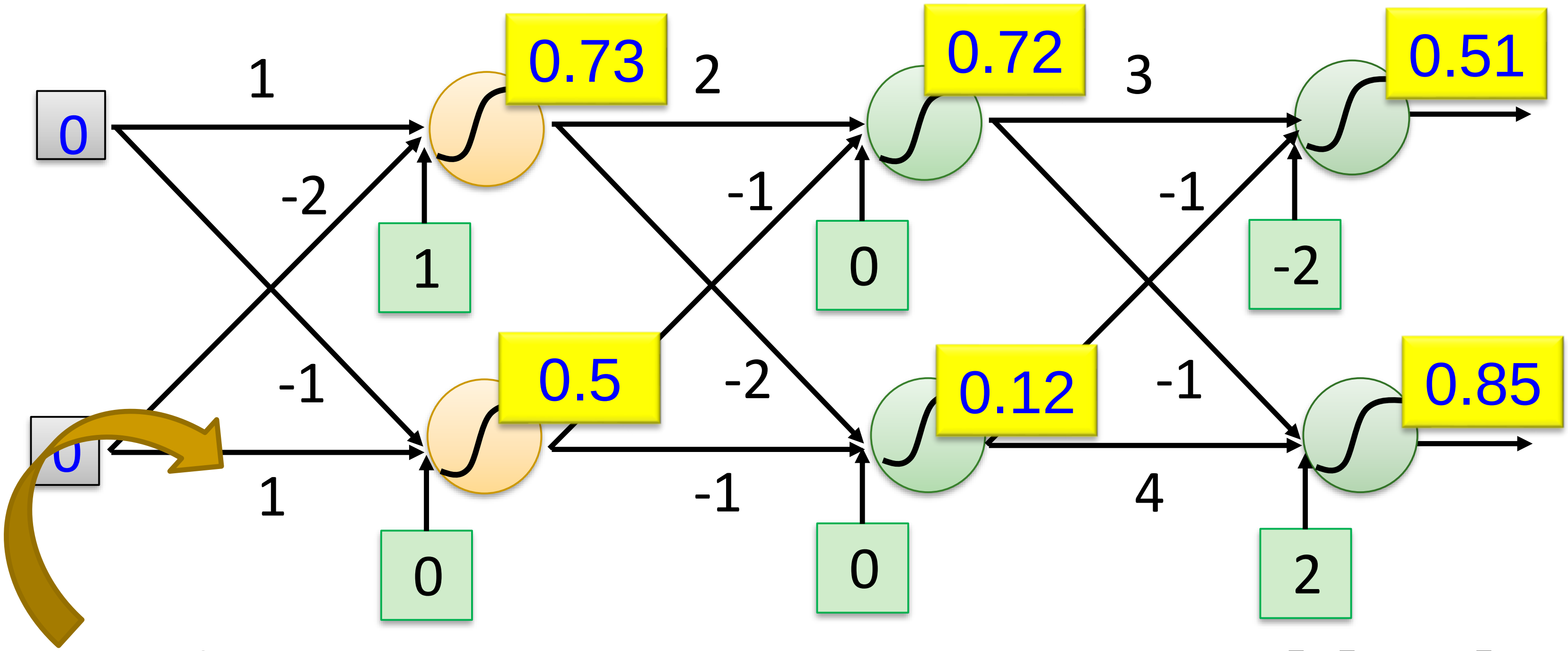


$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Fully connected Feedforward Network



Fully connected Feedforward Network



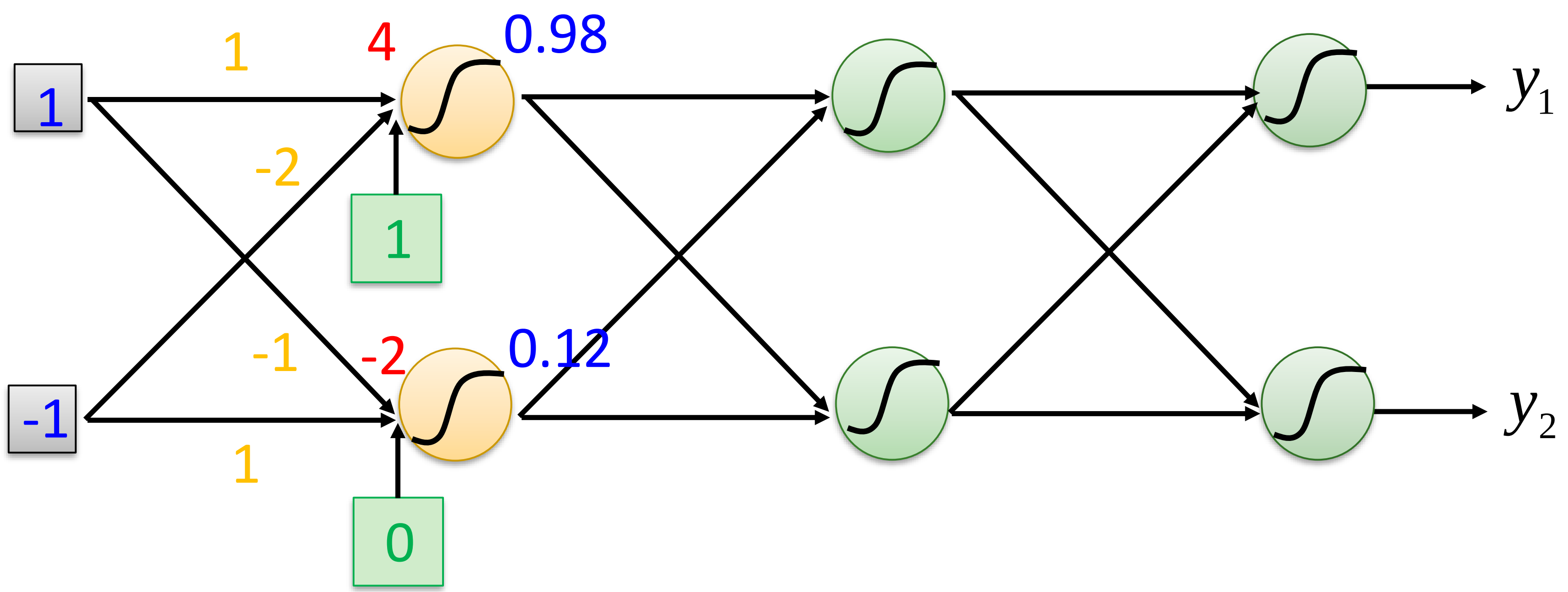
This is a function.

Input vector, output vector

$$f\left(\begin{bmatrix}1\\-1\end{bmatrix}\right)=\begin{bmatrix}0.62\\0.83\end{bmatrix} \quad f\left(\begin{bmatrix}0\\0\end{bmatrix}\right)=\begin{bmatrix}0.51\\0.85\end{bmatrix}$$

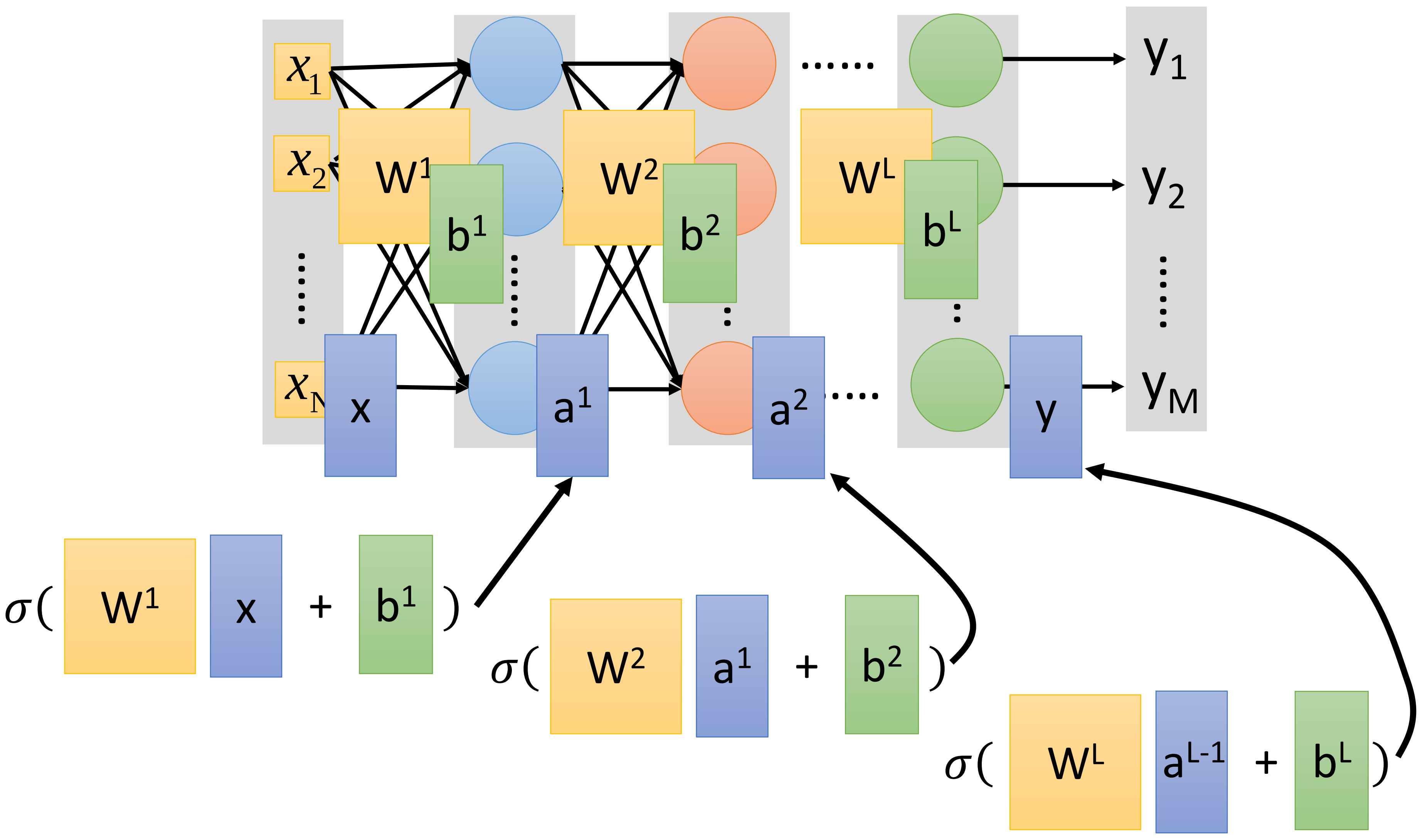
Given network structure, we define ***a function set***

Matrix Operation

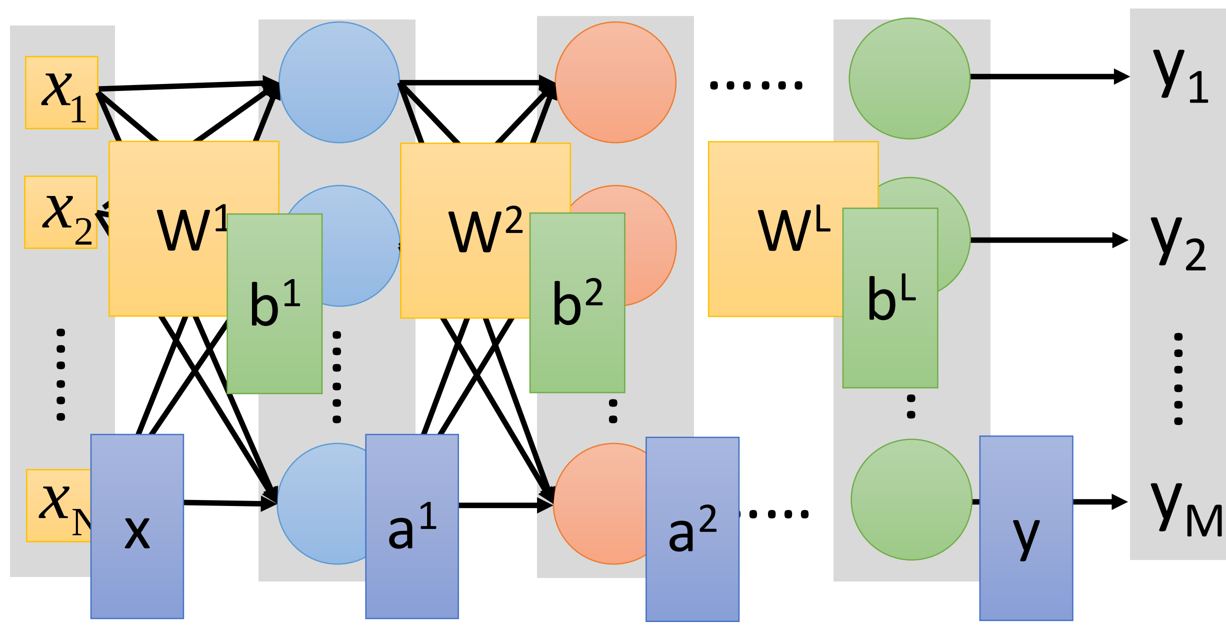


$$\sigma\left(\underbrace{\begin{bmatrix} 1 & -2 \\ -1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}}_{\begin{bmatrix} 4 \\ -2 \end{bmatrix}}\right) = \begin{bmatrix} 0.98 \\ 0.12 \end{bmatrix}$$

Neural Network



Neural Network

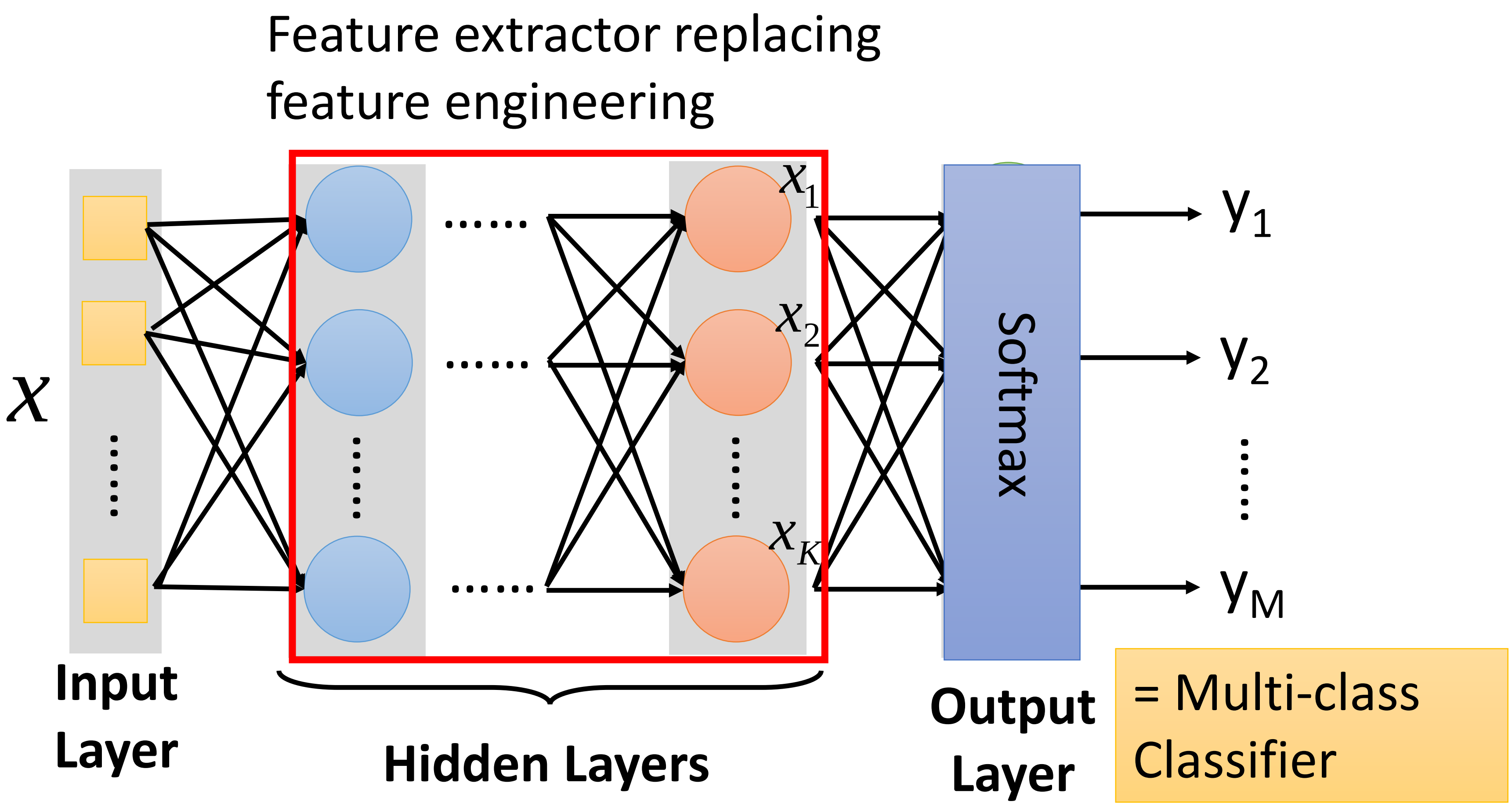


$$\mathrm{y} = f(\mathrm{x})$$

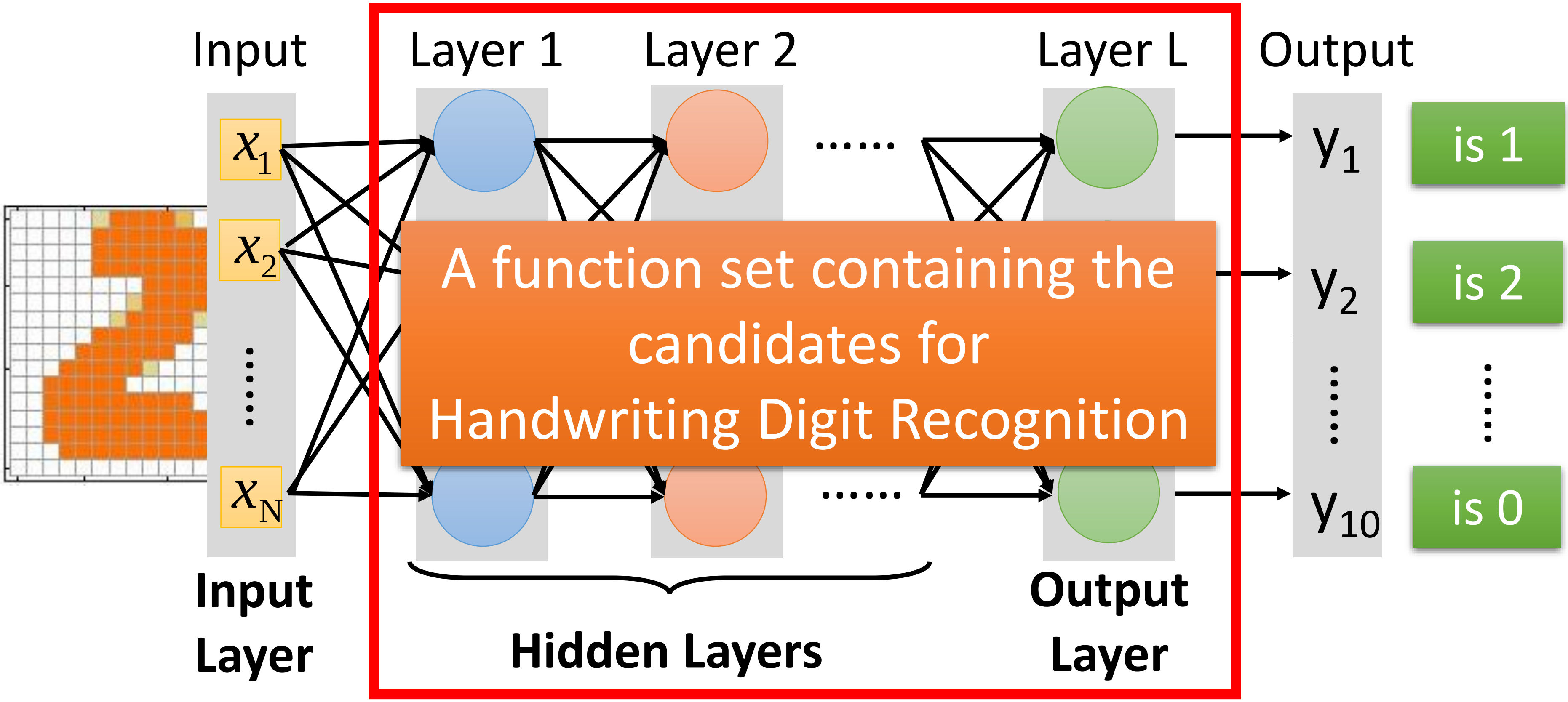
Using parallel computing techniques to speed up matrix operation

$$= \sigma(W^L \cdots \sigma(W^2 \, \sigma(W^1 \mathrm{x} + b^1) + b^2) \cdots + b^L)$$

Output Layer as a Multi-Class Classifier



Example Application



Need to decide the network structure to work well on your dataset.

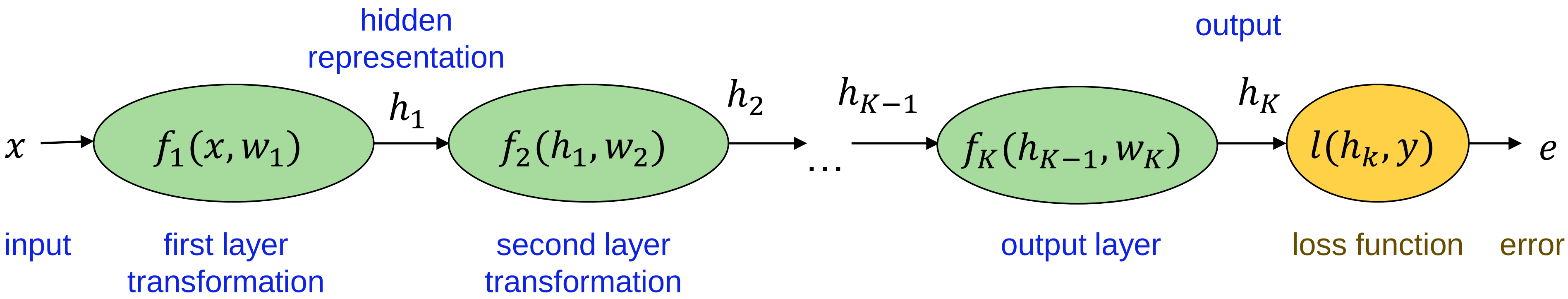
“Deep” pipeline

- Learn a *feature hierarchy*
- Each layer extracts features from the output of previous layer
- All layers are trained jointly

Outline

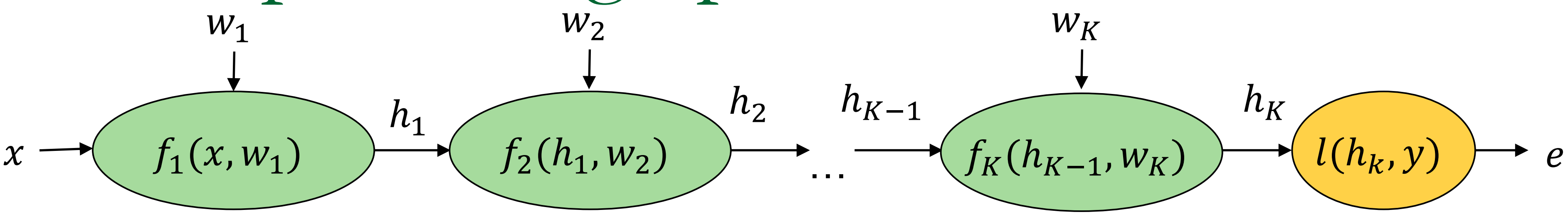
- Modeling one neuron
- Activation functions
- Fully connected feed-forward network
- How to train a multi-layer network
- Representational power of NN

How to train a multi-layer network?



- We need to find the gradient of the error w. r.t. the parameters of each layer, $\frac{\partial e}{\partial w_k}$, to perform updates $w_k \leftarrow w_k - \eta \frac{\partial e}{\partial w_k}$

Computation graph



Chain Rule

Case 1

$$y = g(x) \qquad z = h(y)$$

$$\Delta x \rightarrow \Delta y \rightarrow \Delta z \qquad\qquad \frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

Case 2

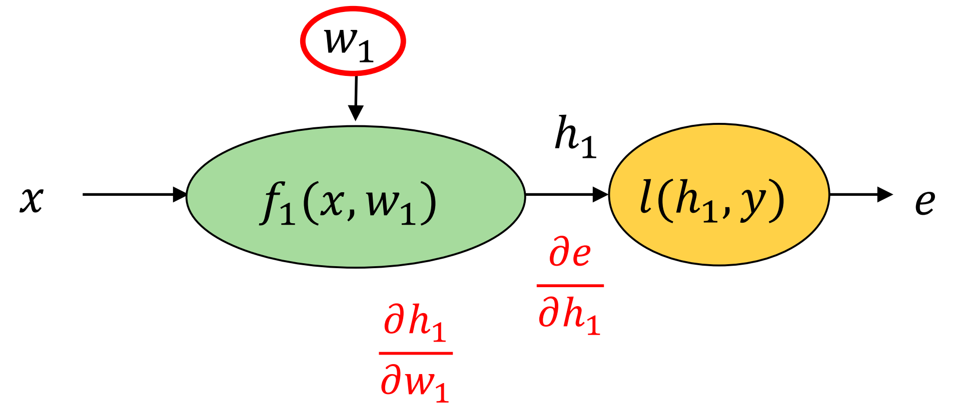
$$x = g(s) \qquad y = h(s) \qquad z = k(x, y)$$

$$\begin{tikzcd}
& \Delta x \arrow[dr] & \\
\Delta s \arrow[ur] \arrow[dr] & & \Delta z \\
& \Delta y \arrow[ur] &
\end{tikzcd}$$

$$\frac{dz}{ds} = \frac{\partial z}{\partial x}\frac{dx}{ds} + \frac{\partial z}{\partial y}\frac{dy}{ds}$$

Chain rule

Let's start with $k = 1$



- $e = l(f_1(x, w_1), y)$
- $\frac{\partial}{\partial w_1} l(f_1(x, w_1), y) = \frac{\partial e}{\partial h_1} \frac{\partial h_1}{\partial w_1}$

- Example: $e = (y - w_1^T x)^2$
- $h_1 = f_1(x, w_1) = w_1^T x$
- $e = l(h_1, y) = (y - h_1)^2$

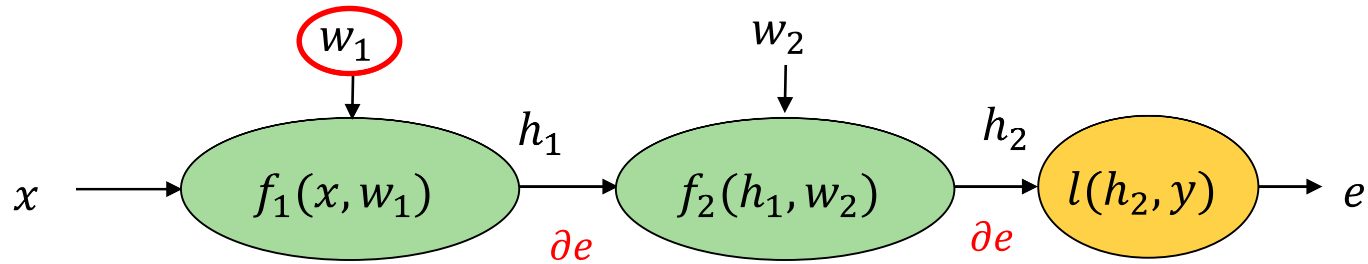
$$\frac{\partial h_1}{\partial w_1} = x$$

$$\frac{\partial e}{\partial h_1} = -2(y - h_1) = -2(y - w_1^T x)$$

- $\frac{\partial e}{\partial w_1} = \frac{\partial e}{\partial h_1} \frac{\partial h_1}{\partial w_1} = -2x(y - w_1^T x)$

Chain rule

$k = 2$



$\frac{\partial h_1}{\partial w_1}$ $\quad \frac{\partial e}{\partial h_1}$ $\quad \frac{\partial h_2}{\partial h_1}$ $\quad \frac{\partial e}{\partial h_2}$

- $e = l(f_2(f_1(x, w_1), w_2))$

$$\frac{\partial e}{\partial w_2} = \frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial w_2} \qquad \frac{\partial e}{\partial w_1} = \frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial w_1}$$

- Example: $e = -\log\left(\sigma\left(w_1^T x\right)\right)$ (assume $y = 1$)

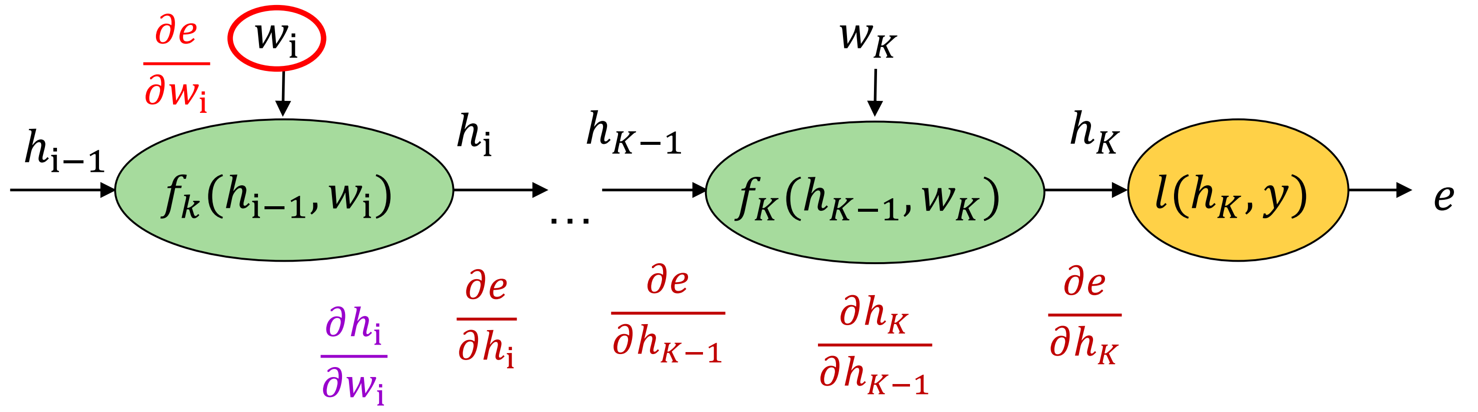
$h_1 = f_1(x, w_1) = w_1^T x$ $\qquad \frac{\partial h_1}{\partial w_1} = x$

$h_2 = f_2(h_1, w_2) = \sigma(h_1)$ $\qquad \frac{\partial h_2}{\partial h_1} = \sigma'(h_1) = \sigma(h_1)(1 - \sigma(h_1))$

$e = l(h_2, 1) = -\log(h_2)$ $\qquad \frac{\partial e}{\partial h_2} = -\frac{1}{h_2}$

$$\frac{\partial e}{\partial w_1} = \frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial w_1} = -\frac{1}{\sigma(w_1^T x)}\sigma(w_1^T x)\left(1 - \sigma\left(w_1^T x\right)\right)x = \sigma(-w_1^T x)x - x$$

Chain rule

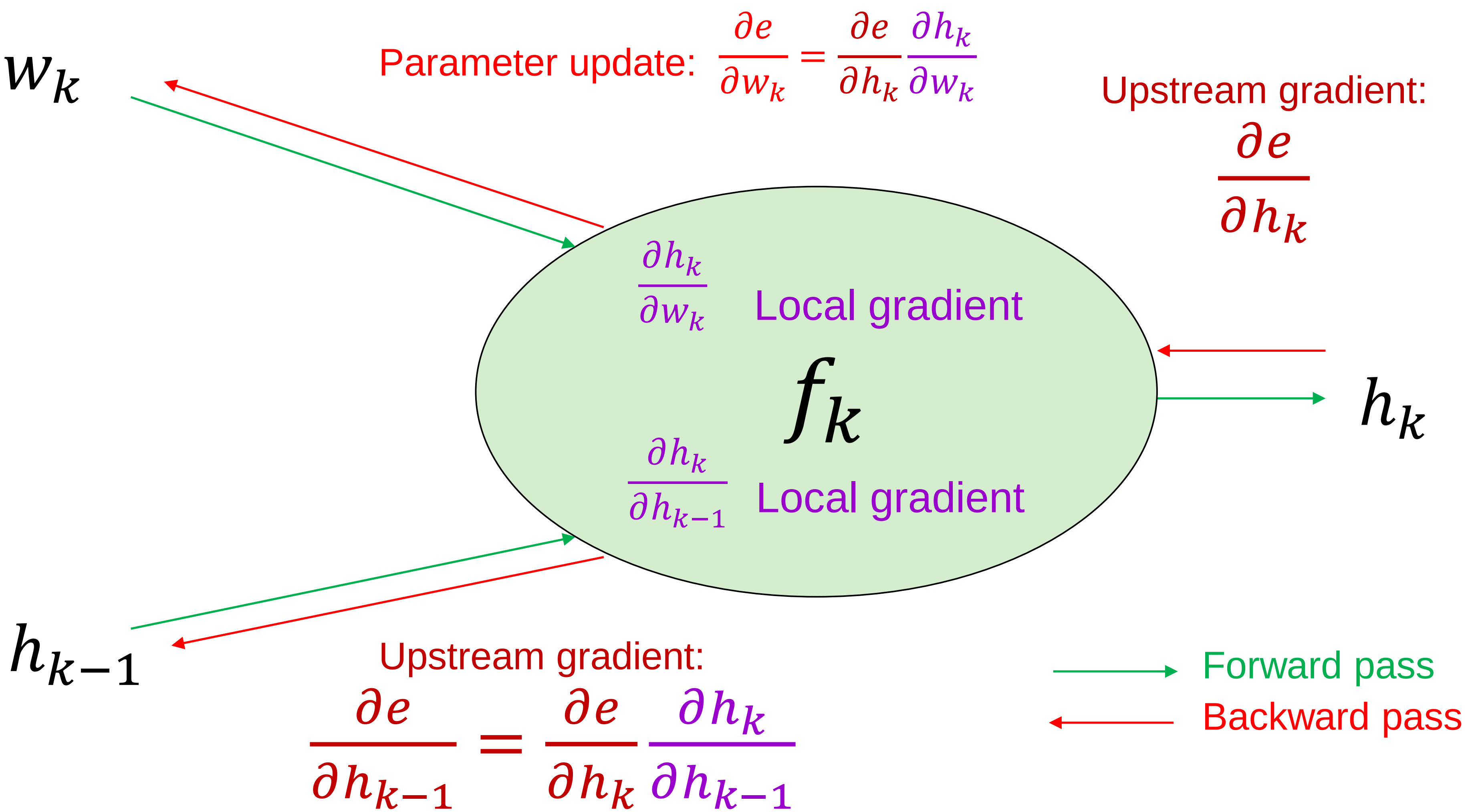


- General case:
- $$\frac{\partial e}{\partial w_i} = \boxed{\frac{\partial e}{\partial h_K} \frac{\partial h_K}{\partial h_{K-1}} \dots \frac{\partial h_{i+1}}{\partial h_i}} \frac{\partial h_i}{\partial w_i}$$

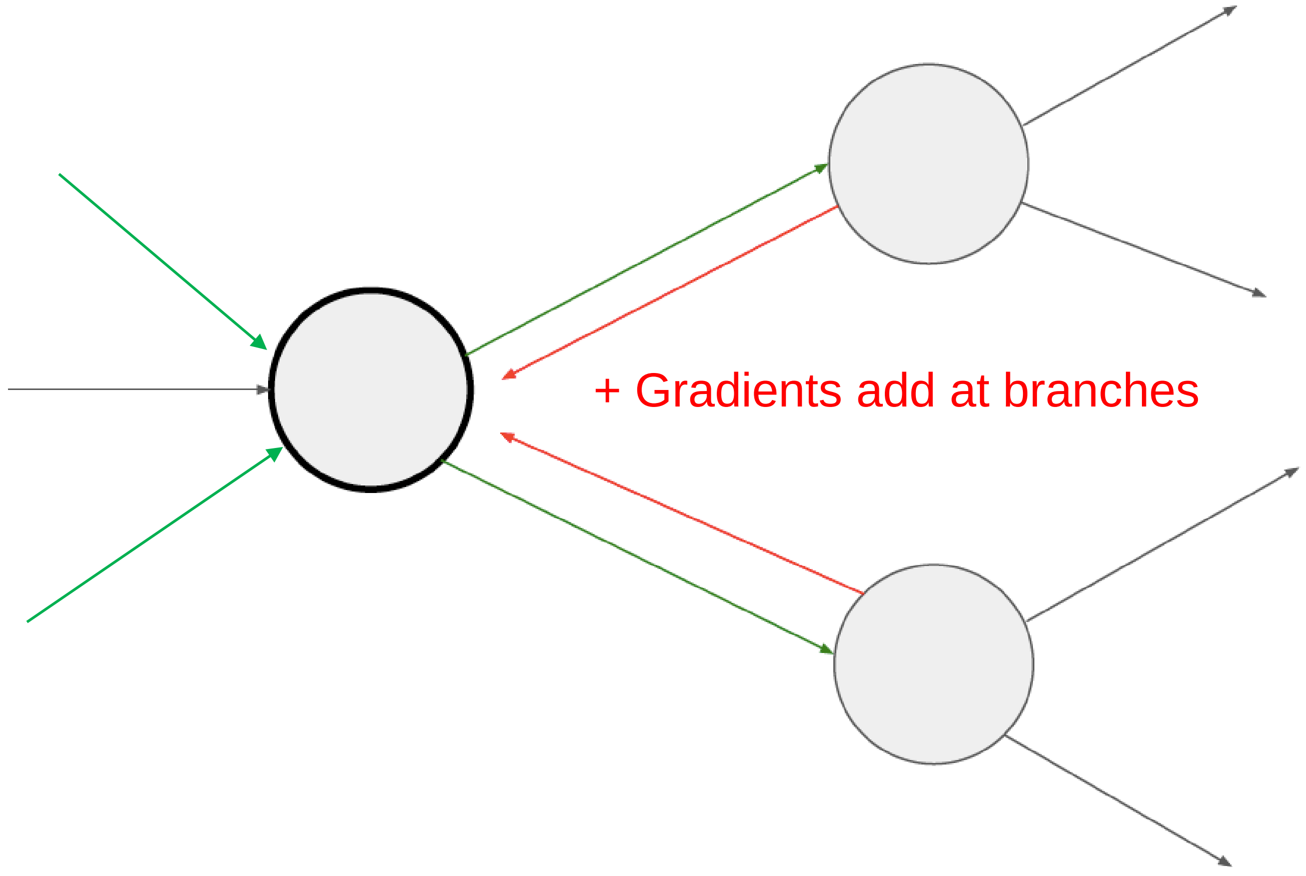
Upstream gradient $\frac{\partial e}{\partial h_i}$

Local gradient

Backpropagation summary

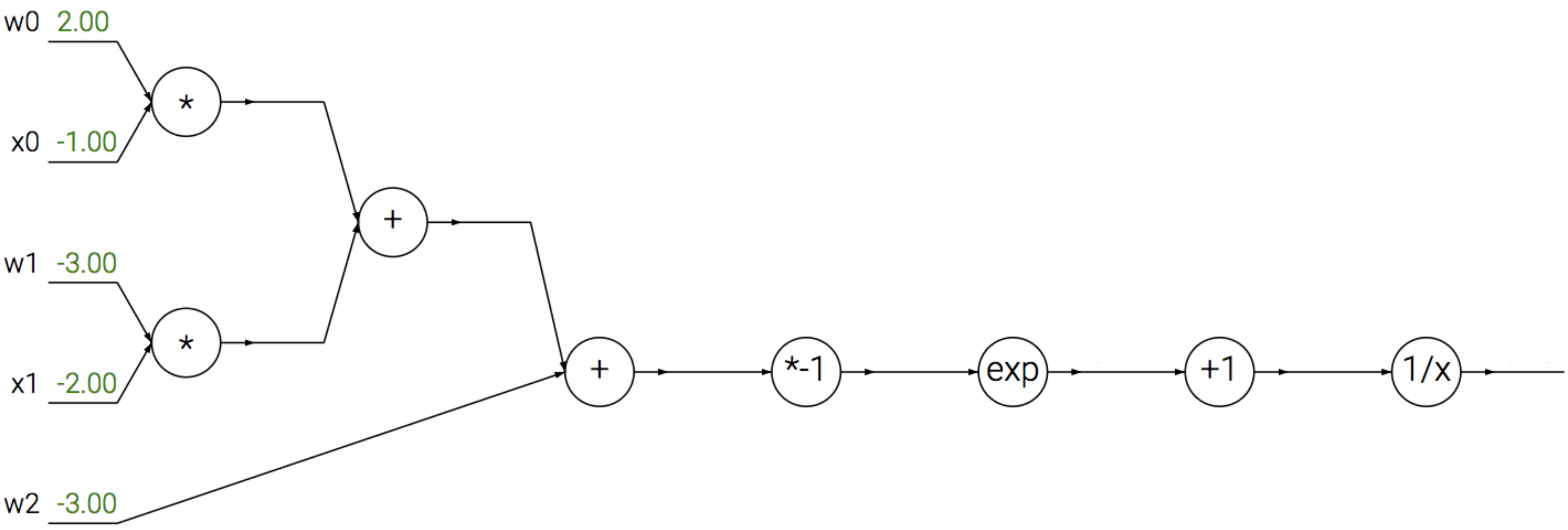


What about more general computation graphs?



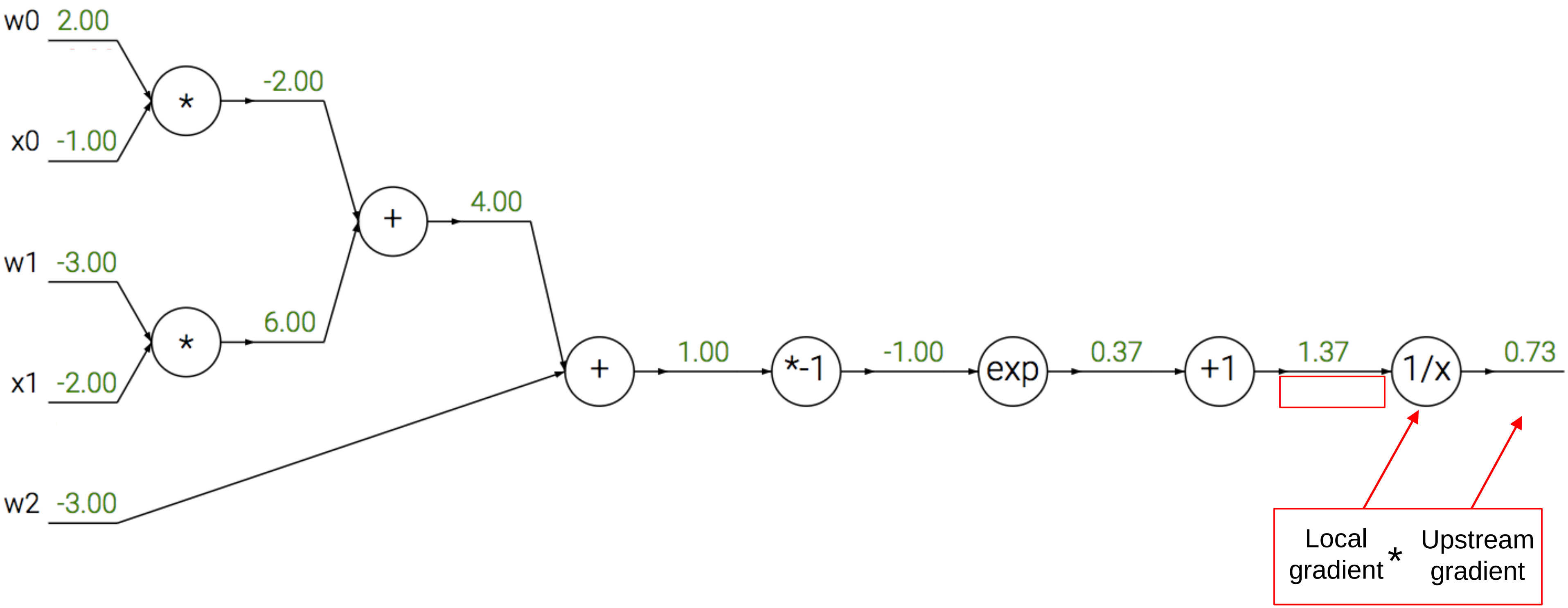
A detailed example

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



A detailed example

$$f(x,w) = \frac{1}{1+\exp[-(w_0x_0+w_1x_1+w_2)]}$$

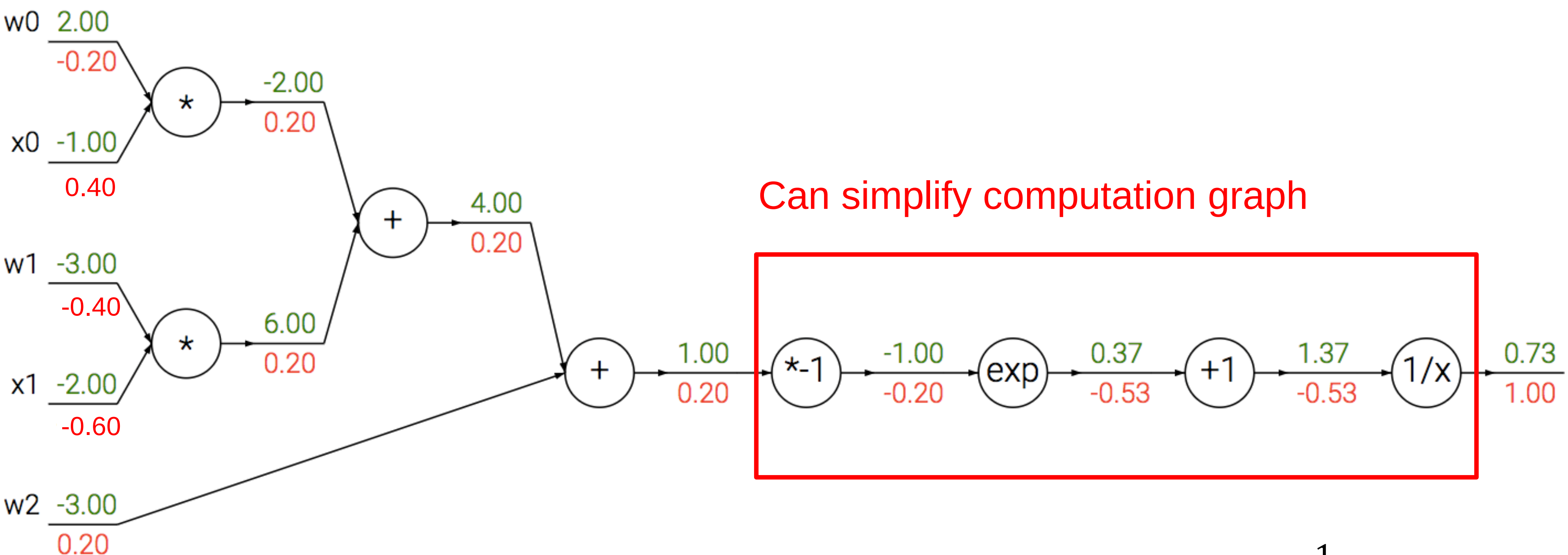


$$(1/x)' = -1/x^2$$

$$-\frac{1}{1.37^2} * 1 = -0.53$$

A detailed example

$$f(x,w) = \frac{1}{1+\exp[-(w_0x_0 + w_1x_1 + w_2)]}$$



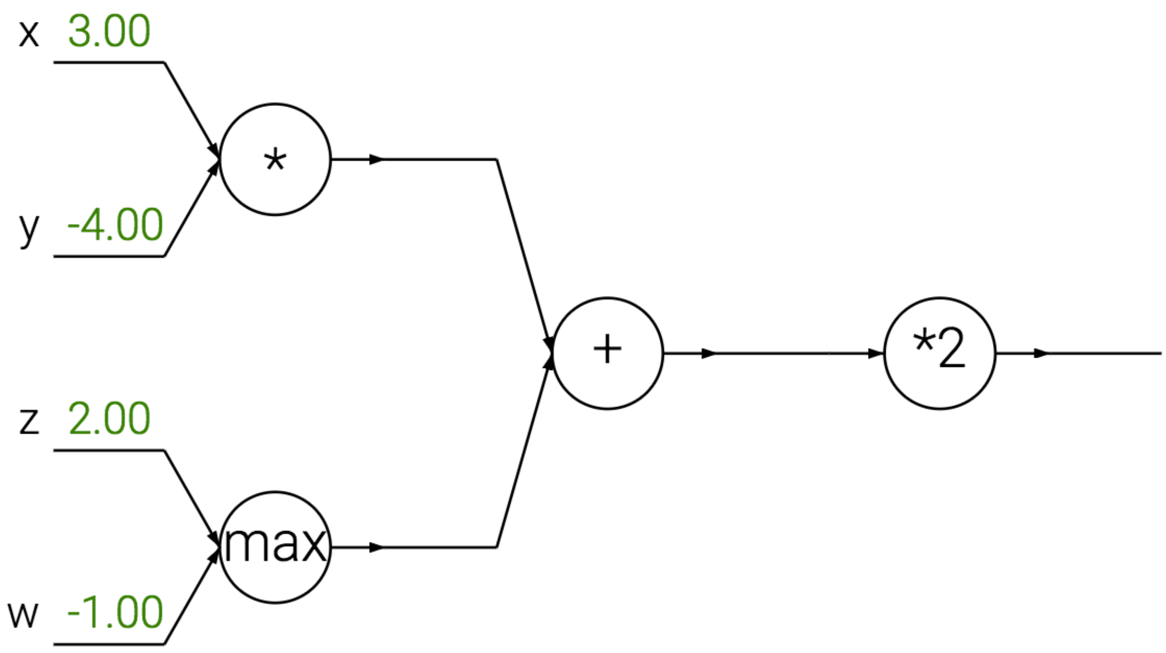
Sigmoid gate $\sigma(x) = \frac{1}{1+\exp(-x)}$

$$\sigma'(x) = \sigma(x)\big(1-\sigma(x)\big)$$

$$\sigma(1)\big(1-\sigma(1)\big) = 0.73 * (1-0.73) = 0.20$$

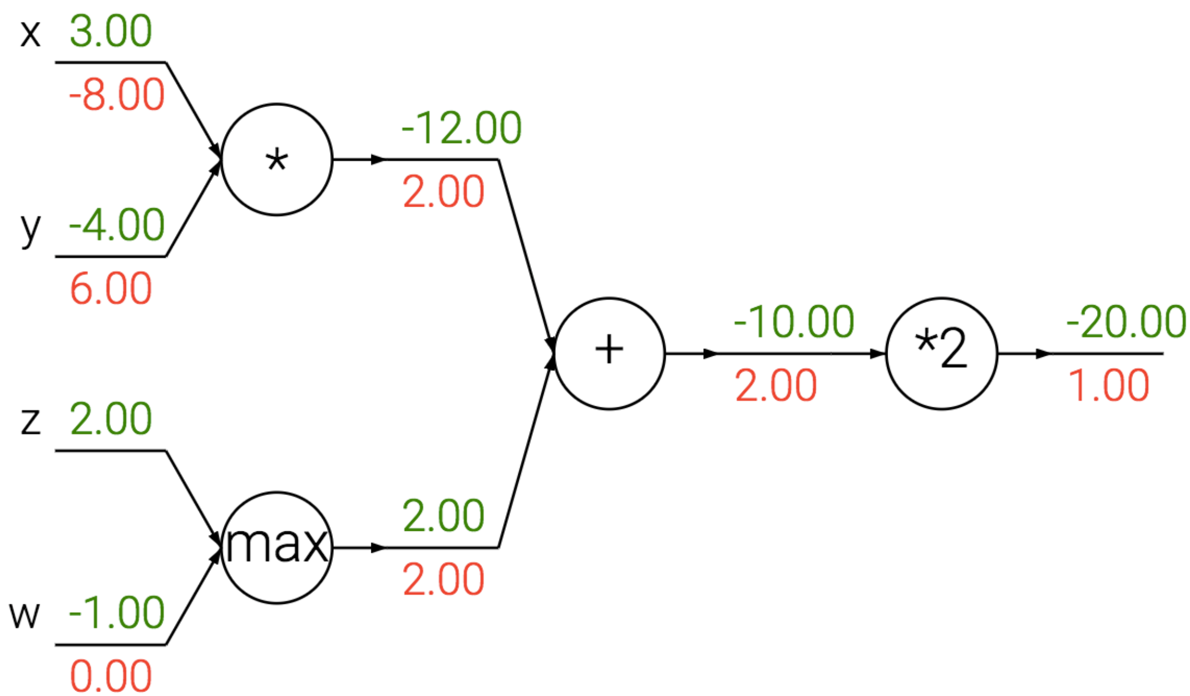
Source: Stanford 231n

Patterns in gradient flow



Source: [Stanford 231n]

Patterns in gradient flow



Add gate: “gradient distributor”

Multiply gate: “gradient switcher”

Max gate: “gradient router”

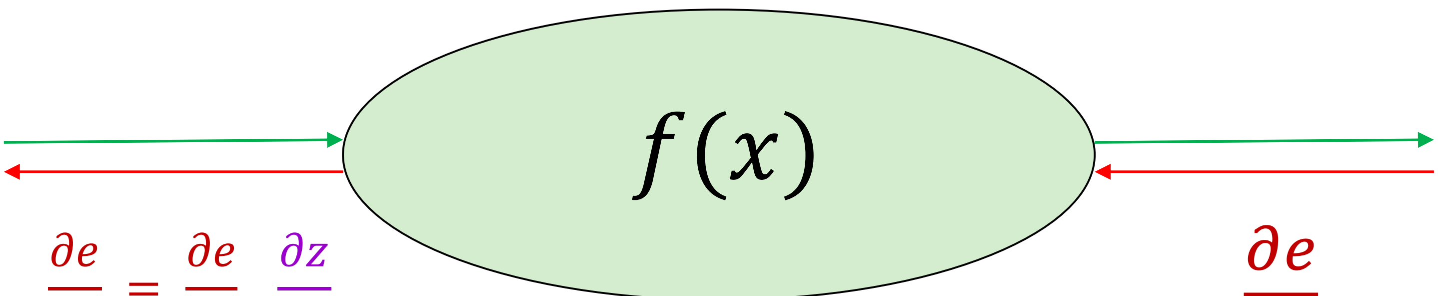
Source: Stanford 231n

Dealing with vectors

$$\frac{\partial z}{\partial x} = \begin{pmatrix} \frac{\partial z_1}{\partial x_1} & \cdots & \frac{\partial z_1}{\partial x_M} \\ \vdots & \ddots & \vdots \\ \frac{\partial z_N}{\partial x_1} & \cdots & \frac{\partial z_N}{\partial x_M} \end{pmatrix}$$

$N \times M$
Jacobian

x



z

$1 \times M$

$$\frac{\partial e}{\partial x} = \frac{\partial e}{\partial z} \frac{\partial z}{\partial x}$$

$1 \times M$ $1 \times N$$N \times M$

$\frac{\partial e}{\partial z}$

$1 \times N$

$1 \times N$

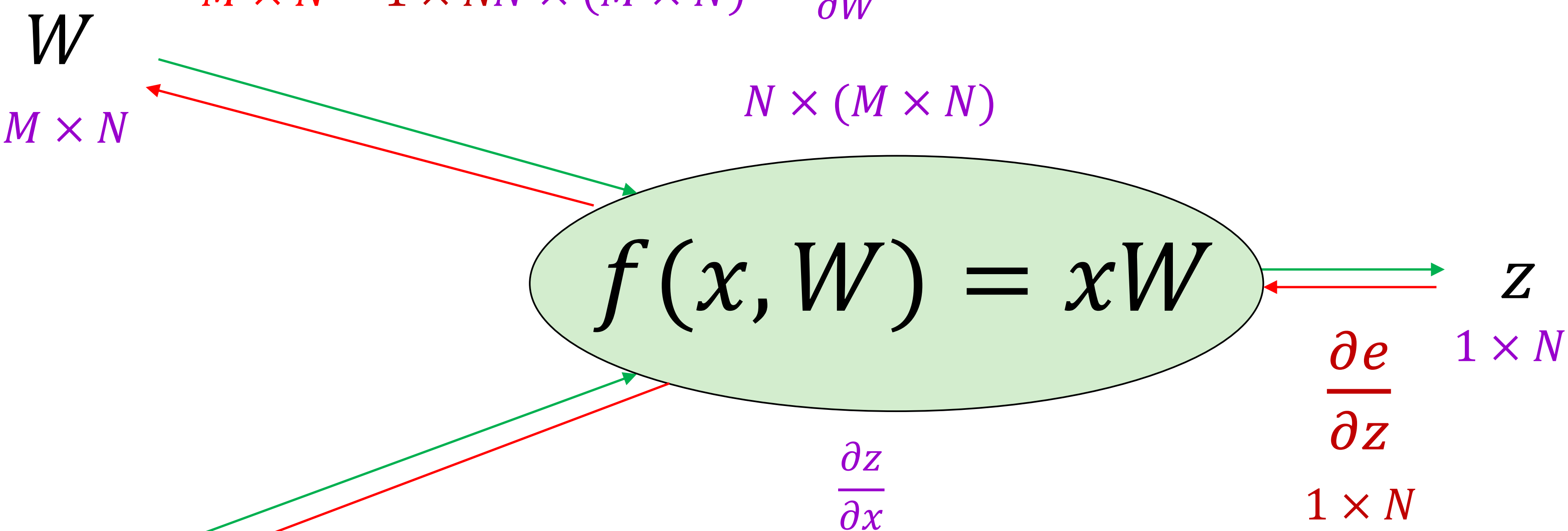
Matrix-vector multiplication

$$\frac{\partial e}{\partial W} = \frac{\partial e}{\partial z} \frac{\partial z}{\partial W}$$

$M \times N \quad 1 \times N N \times (M \times N)$

$\frac{\partial z}{\partial W}$

$N \times (M \times N)$



$1 \times N$

x

$1 \times M$

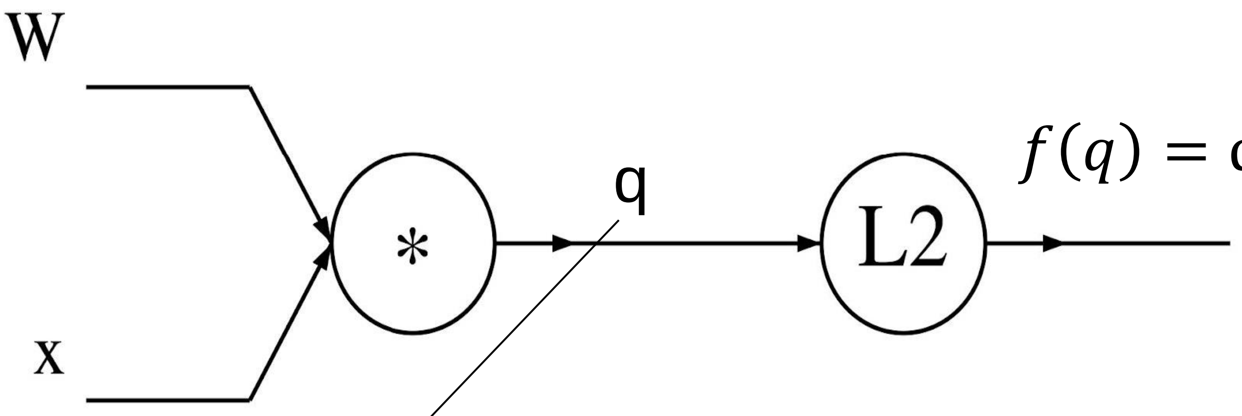
$$\frac{\partial e}{\partial x} = \frac{\partial e}{\partial z} \frac{\partial z}{\partial x}$$

$N \times M$

$1 \times M \quad 1 \times N N \times M$

A vectorized example:

$$f(\mathrm{x}, W) = \sum_{i=1}^{n} (W \cdot \mathrm{x})_i^2$$



$$f(q) = \mathrm{q}_1^2 + \cdots + \mathrm{q}_n^2$$

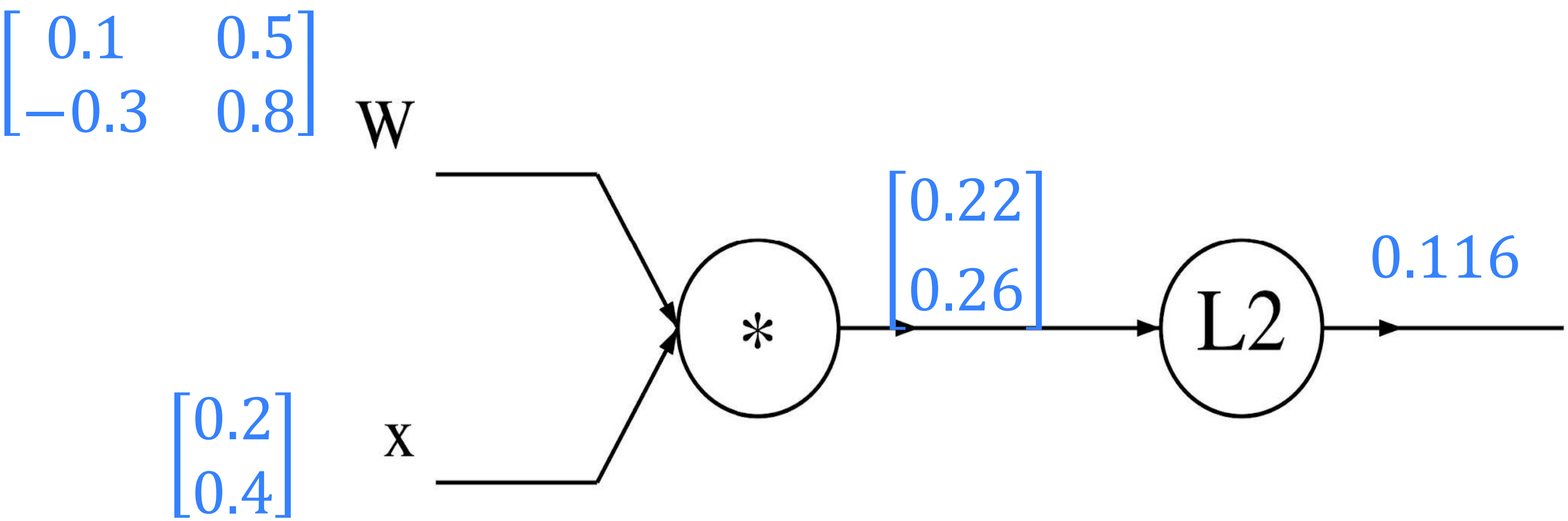
$$q = W\mathrm{x} = \begin{pmatrix} w_{11}x_1 + \cdots + w_{1n}x_n \\ \vdots \\ w_{n1}x_1 + \cdots + w_{nn}x_n \end{pmatrix}$$

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

Feed-forward:

$$q = Wx = \begin{pmatrix} w_{11}x_1 + w_{12}x_2 \\ w_{21}x_1 + w_{22}x_2 \end{pmatrix} = \begin{bmatrix} 0.22 \\ 0.26 \end{bmatrix}$$

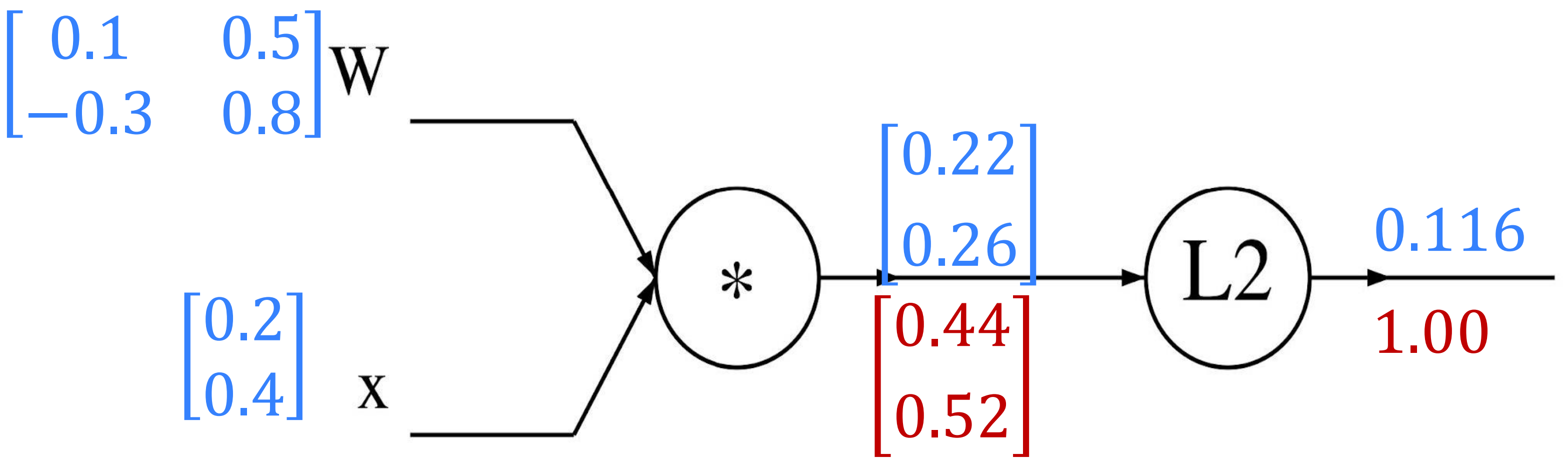
$$f(q) = \sum_{i=1}^{2} \mathrm{q}_{\mathrm{i}}^2 = 0.116$$



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

Backward:

$$f(q) = \sum_{i=1}^{2} q_i^2 \quad \Longrightarrow \quad \frac{\partial f}{\partial q_i} = 2q_i \quad \Longrightarrow \quad \boxed{\nabla_q f = 2q}$$

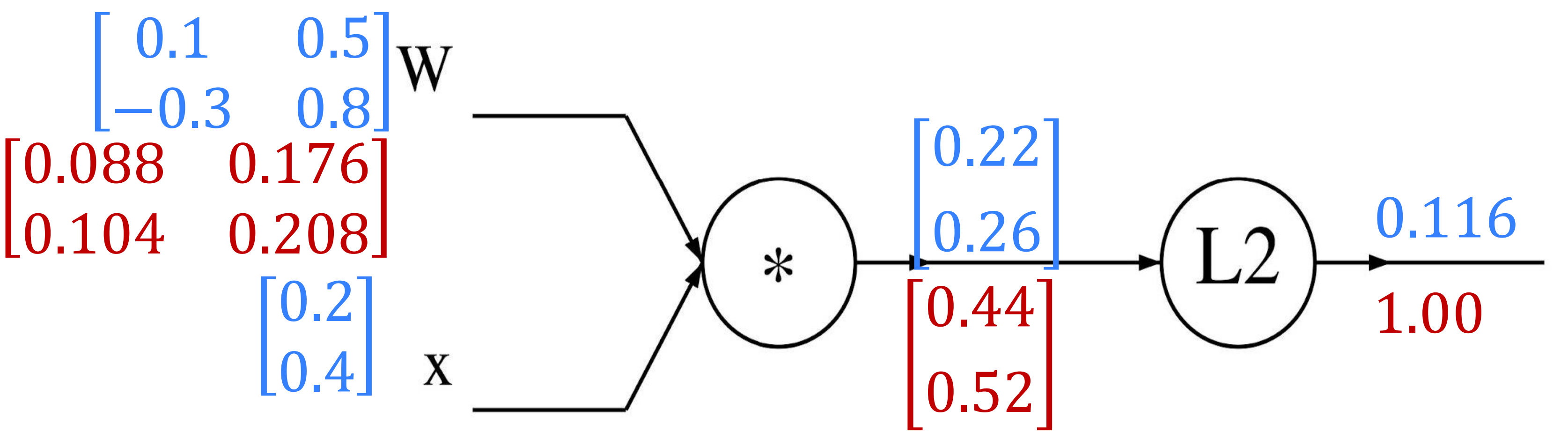


A vectorized example: $f(\mathrm{x}, W) = \sum_{i=1}^{n} (W \cdot \mathrm{x})_i^2$

Backward:

$$q = Wx = \begin{pmatrix} w_{11}x_1 + w_{12}x_2 \\ w_{21}x_1 + w_{22}x_2 \end{pmatrix} \Rightarrow \quad \frac{\partial q}{\partial w_{11}} = x_1 \quad \frac{\partial q}{\partial w_{12}} = x_2 \quad \frac{\partial q}{\partial w_{21}} = x_1 \quad \frac{\partial q}{\partial w_{22}} = x_2$$

$$\frac{\partial f}{\partial w_{ij}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial w_{ij}} = 2 q_i x_j \longrightarrow \boxed{\nabla_W \mathrm{f} = 2 q x^T}$$

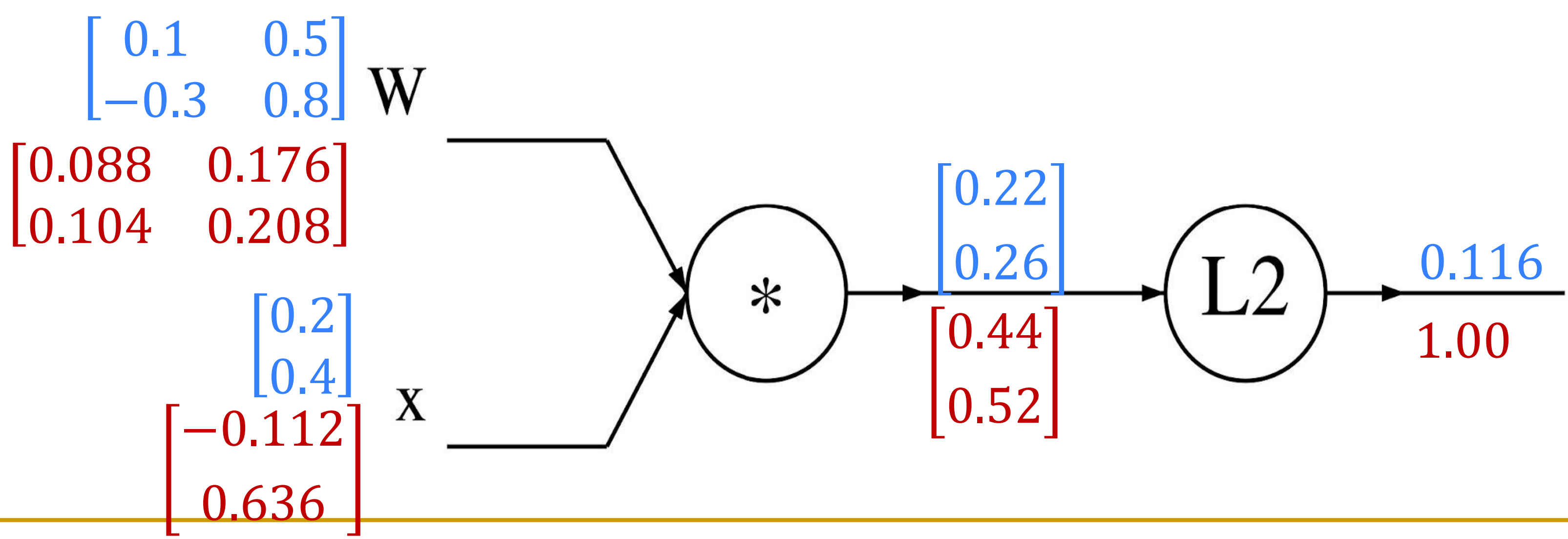


A vectorized example: $f(\mathrm{x}, W) = \sum_{i=1}^{n} (W \cdot \mathrm{x})_i^2$

Backward:

$$q = Wx = \begin{pmatrix} w_{11}x_1 + w_{12}x_2 \\ w_{21}x_1 + w_{22}x_2 \end{pmatrix} \Rightarrow \frac{\partial q}{\partial x_1} = \begin{pmatrix} w_{11} \\ w_{21} \end{pmatrix} \quad \frac{\partial q}{\partial x_2} = \begin{pmatrix} w_{12} \\ w_{22} \end{pmatrix}$$

$$\frac{\partial f}{\partial x_i} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial x_i} = \sum_k 2\, q_k W_{ki} \Rightarrow \boxed{\nabla_x \mathrm{f} = W^T 2q}$$



General tips for computation

- Derive error signal (upstream gradient) directly, avoid explicit computation of huge local derivatives
- Write out expression for a single element of the Jacobian, then deduce the overall formula
- Keep consistent indexing conventions, order of operations
- Use dimension analysis

Outline

- Modeling one neuron
- Activation functions
- Fully connected feed-forward network
- How to train a multi-layer network
- Representational power of NN

Representational power

- Neural Networks with fully-connected layers define a family of functions that are parameterized by the weights of the network.
- A Neural Network with at least one hidden layer are *universal approximators*, *which* means that it can approximate any continuous function.

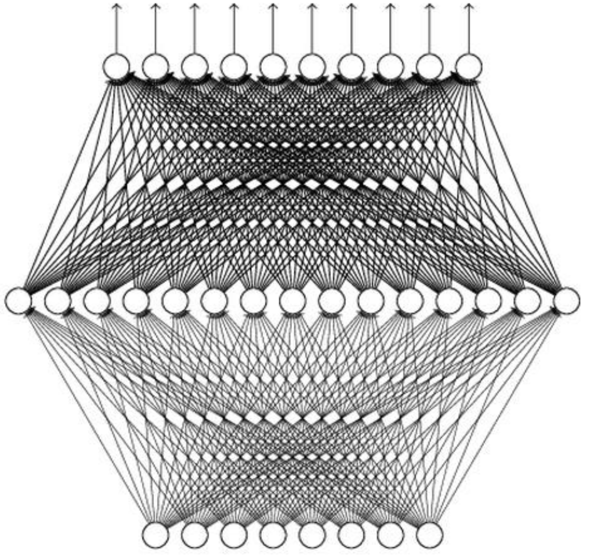
Universality Theorem

Any continuous function f

$$f : R^N \rightarrow R^M$$

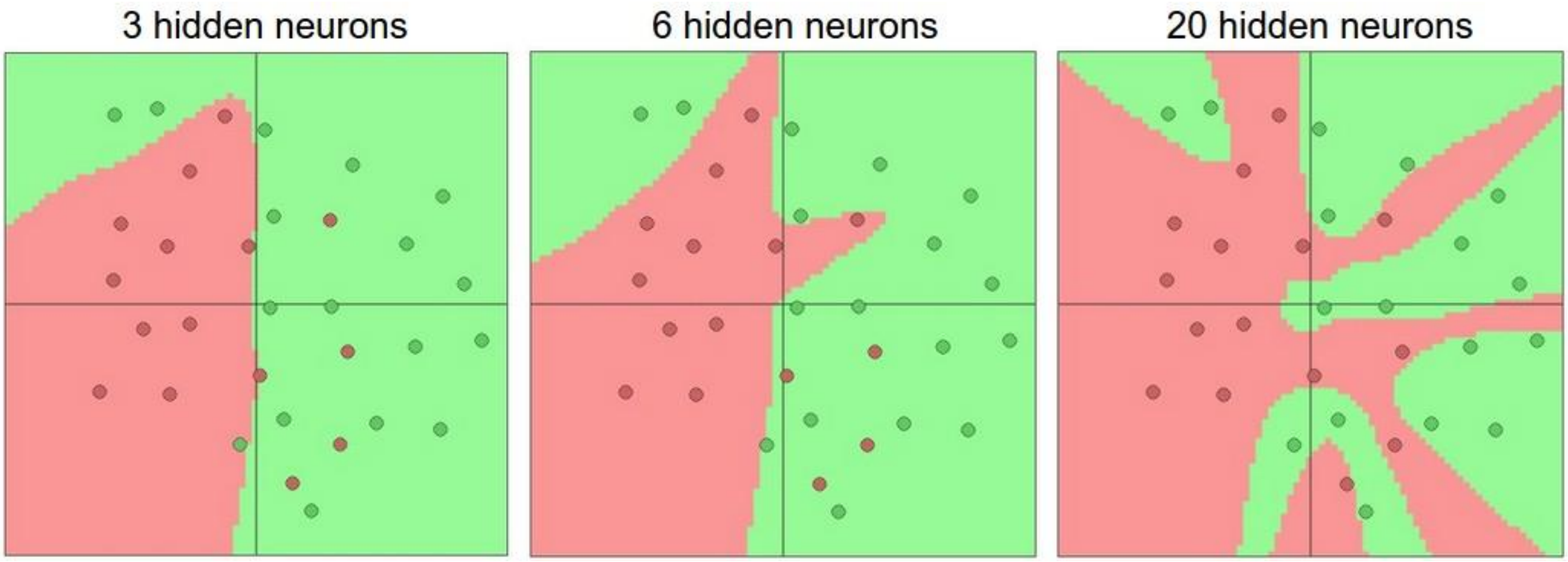
can be realized by a network with one hidden layer

(given **enough** hidden neurons)



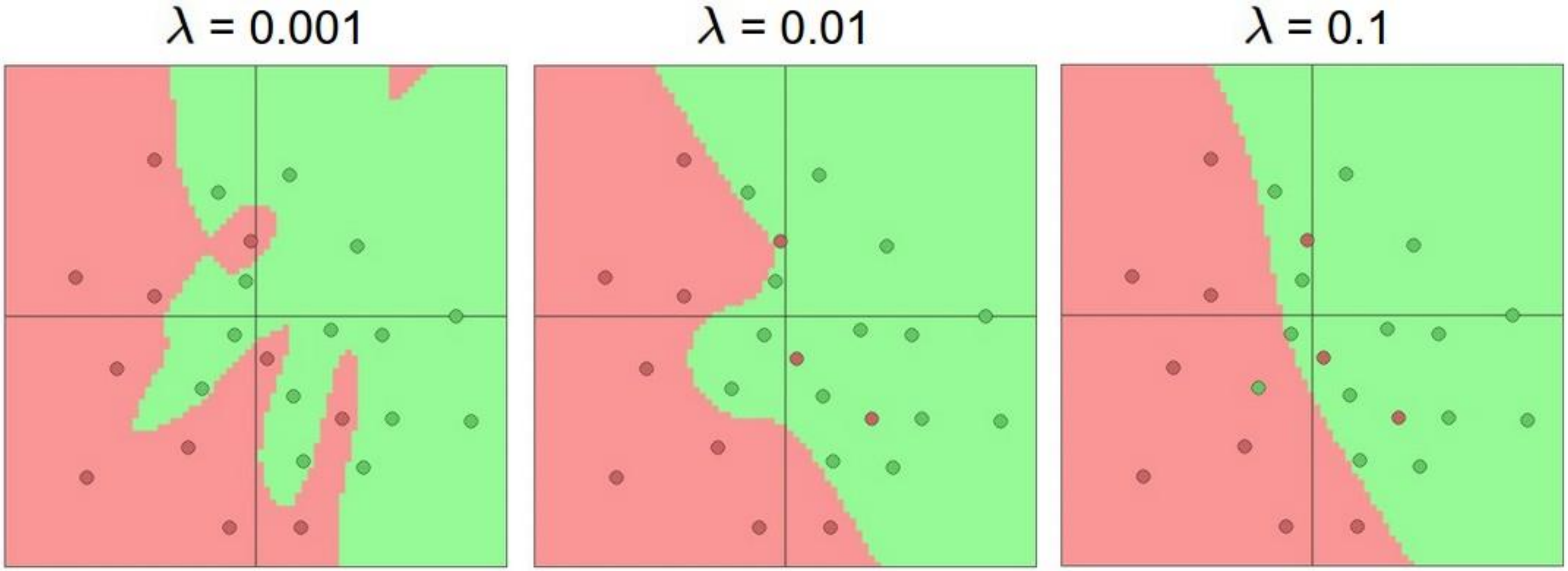
Reference for the reason: http://neuralnetworksanddeeplearning.com/chap4.html

Setting number of layers and their sizes

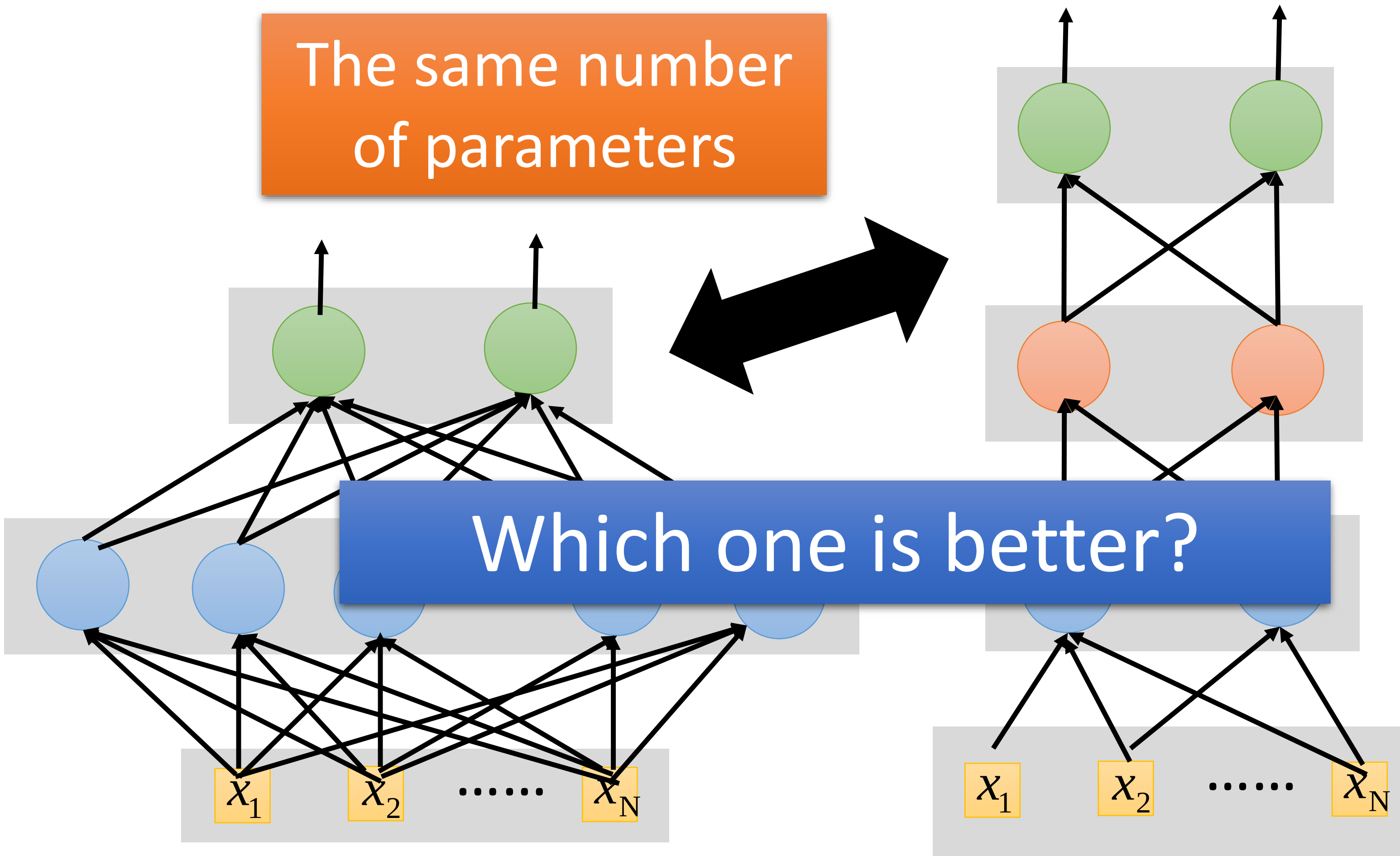


- Neural Networks with more neurons can express more complicated functions.
- But a model with high capacity fits the noise in the data instead of the (assumed) underlying relationship : **overfitting**.

❖ Use as big of a neural network as your computational budget allows, and use other regularization techniques to control overfitting.



Fat + Short vs. Thin + Tall



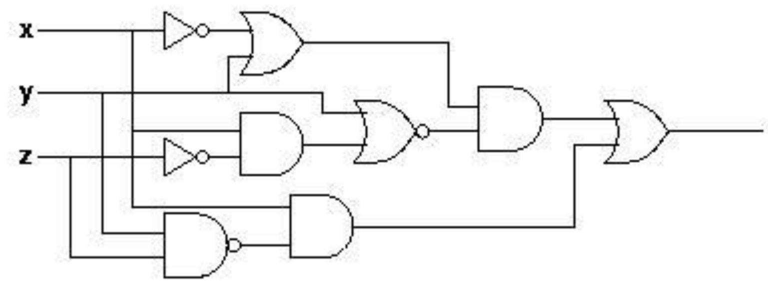
Shallow Deep

Thin + Tall vs. Fat + Short

Layer X Size	Word Error Rate (%)	Layer X Size	Word Error Rate (%)
1 X 2k	24.2		
2 X 2k	20.4		
3 X 2k	18.4		
4 X 2k	17.8		
5 X 2k	17.2	1 X 3772	22.5
7 X 2k	17.1	1 X 4634	22.6
		1 X 16k	22.1

Why?

Analogy

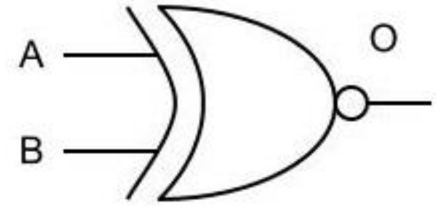


Logic circuits	Neural network
• Logic circuits consists of **gates**	• Neural network consists of **neurons**
• **A two layers of logic gates** can represent **any Boolean function.**	• **A hidden layer network** can represent **any continuous function.**
• Using multiple layers of logic gates to build some functions are much simpler	• Using multiple layers of neurons to represent some functions are much simpler

→ less gates needed

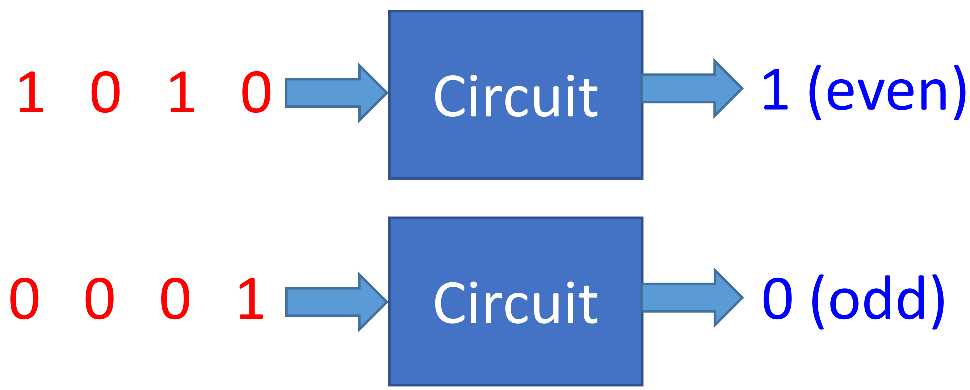
→ less parameters → less data?

Analogy



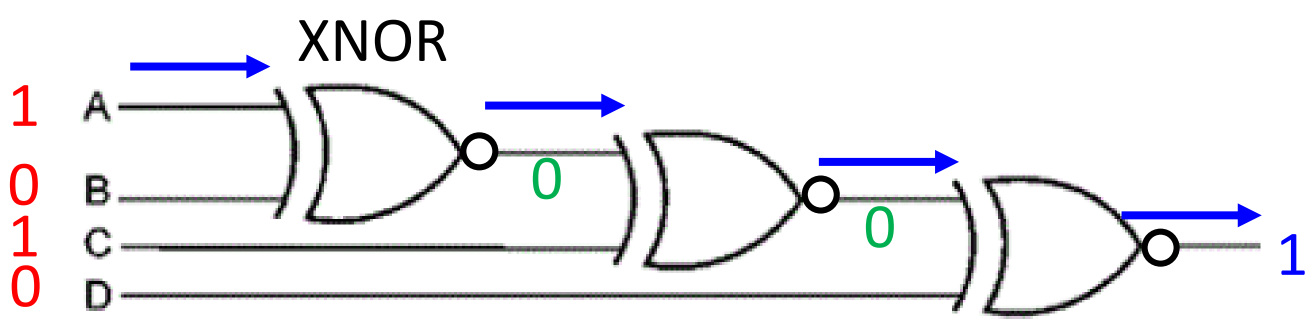
A	B	O
0	0	1
0	1	0
1	0	0
1	1	1

- E.g. ***parity check***



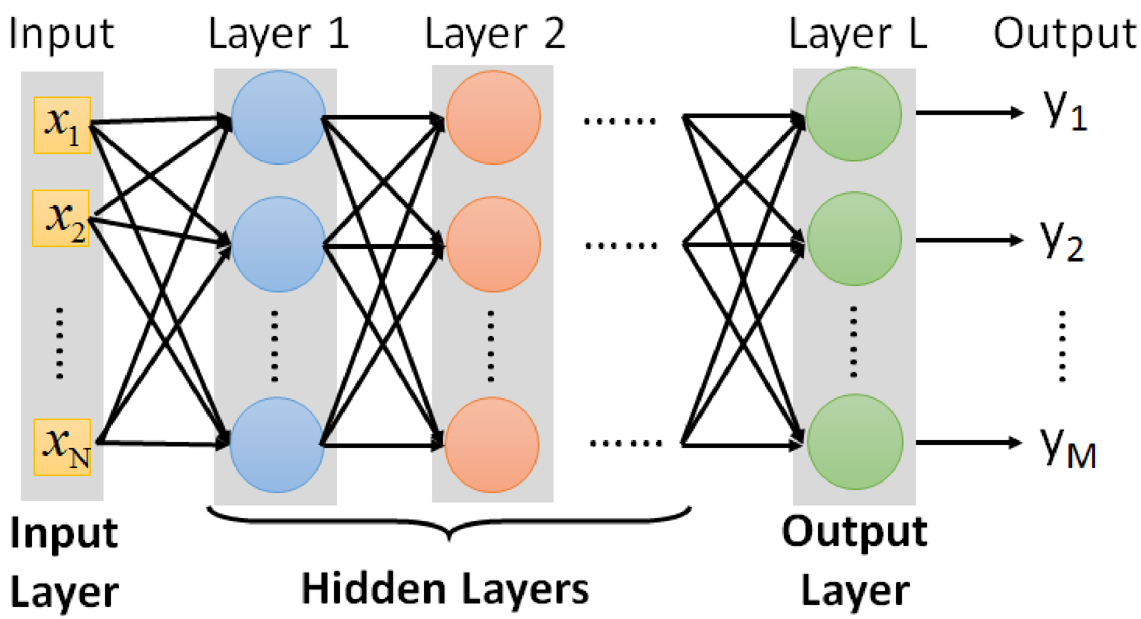
For input sequence with d bits,

Two-layer circuit need $O(2^d)$ gates.



With multiple layers, we need only O(d) gates.

Design the Network



- How many layers? How many neurons for each layer?

Trial and Error + Intuition

课程部分材料来自他人和网络，仅限教学使用，请勿传播，谢谢！