



南方科技大学

MAT8034: Machine Learning

Deep Learning

Fang Kong

<https://fangkongx.github.io/Teaching/MAT8034/Spring2026/index.html>

Outline

- History of artificial neural nets
- Perceptron
- Multilayer perceptron networks
- Activation functions
- Training: backpropagation
- Modules in modern neural networks

History of artificial neural nets

Brief history of artificial neural nets

■ The First wave

- 1943 McCulloch and Pitts proposed the McCulloch-Pitts neuron model
- 1958 Rosenblatt introduced the simple single layer networks now called Perceptrons
- 1969 Minsky and Papert's book Perceptrons demonstrated the limitation of single layer perceptrons, and almost the whole field went into hibernation

■ The Second wave

- 1986 The Back-Propagation learning algorithm for Multi-Layer Perceptrons was rediscovered and the whole field took off again

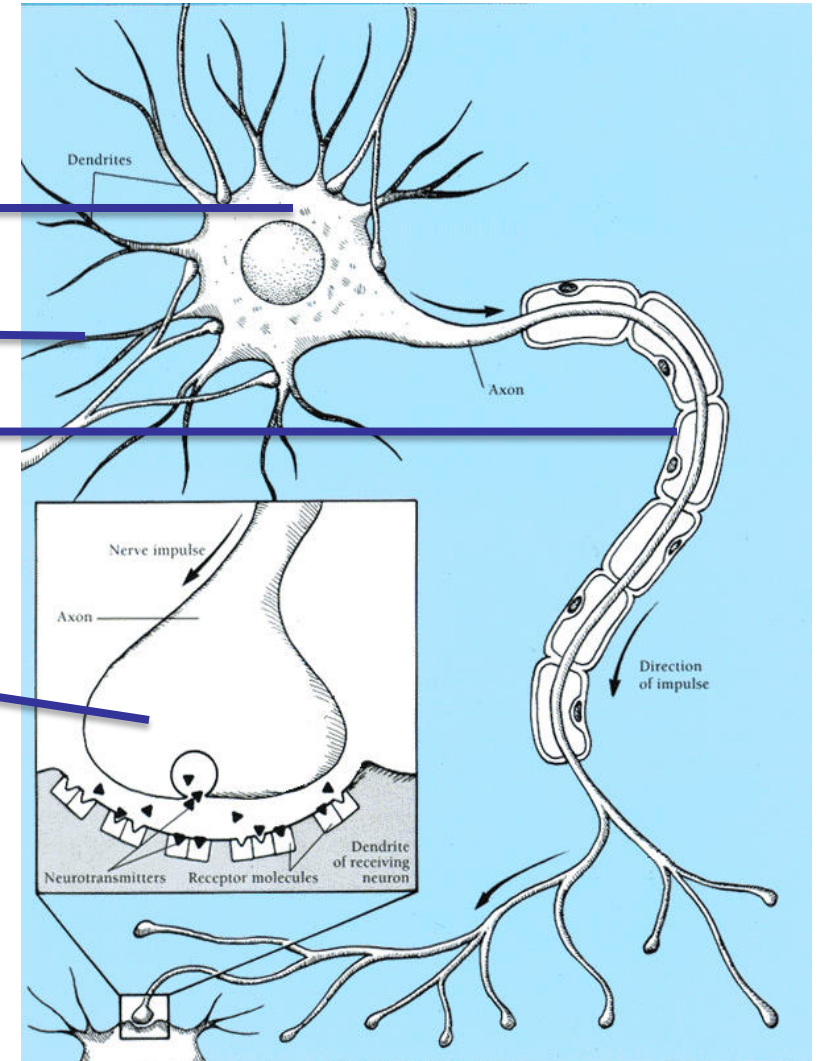
■ The Third wave

- 2006 Deep (neural networks) Learning gains popularity
- 2012 made significant break-through in many applications

Biological neuron structure

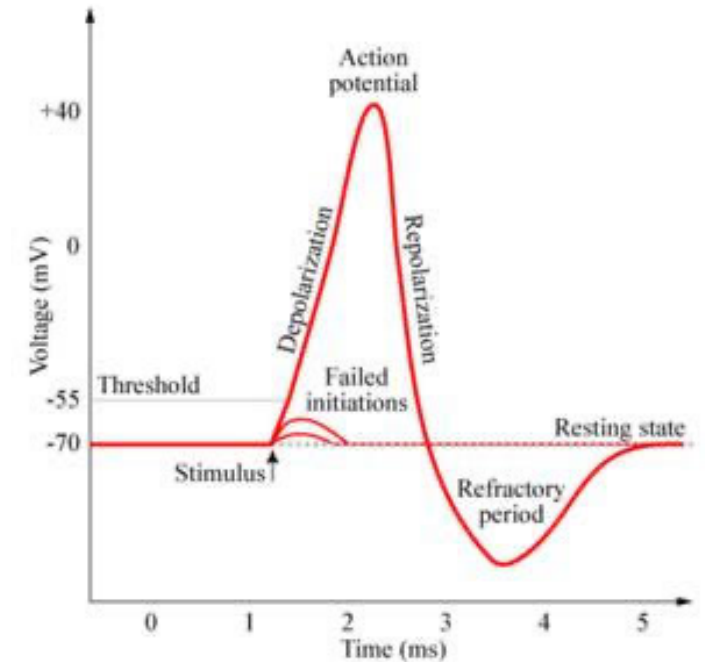
■ 细胞结构

- 细胞体
- 树突
- 轴突
- 突触末梢



Biological neural communication

- 细胞膜间的电位表现出的电信号称为动作电位
- 电信号从细胞体中产生，沿着轴突往下传，并且导致突触末梢释放神经递质介质
- 介质通过化学扩散从突触传递到其他神经元的树突
- 神经递质可以是兴奋的或者是抑制的
- 如果从其他神经元来的神经递质是兴奋的且超过某个阈值，将会触发一个动作电位

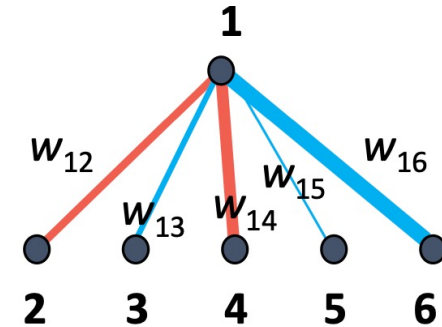


McCulloch-Pitts neuron model [1943]

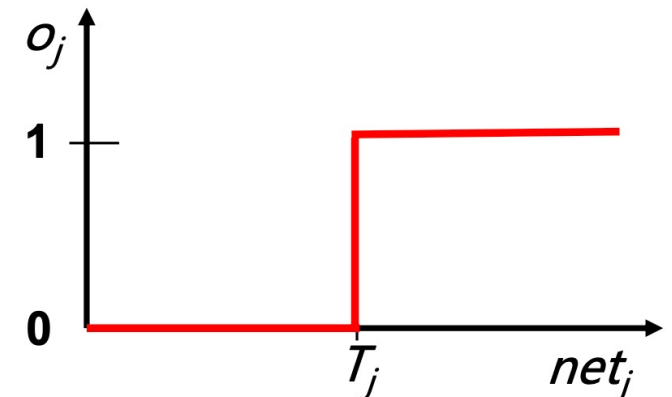
- Model the network as a graph, where the units are nodes, and the synaptic connections are weighted edges from node i to node j , with the weight as $w_{j,i}$

- The input of the unit is:

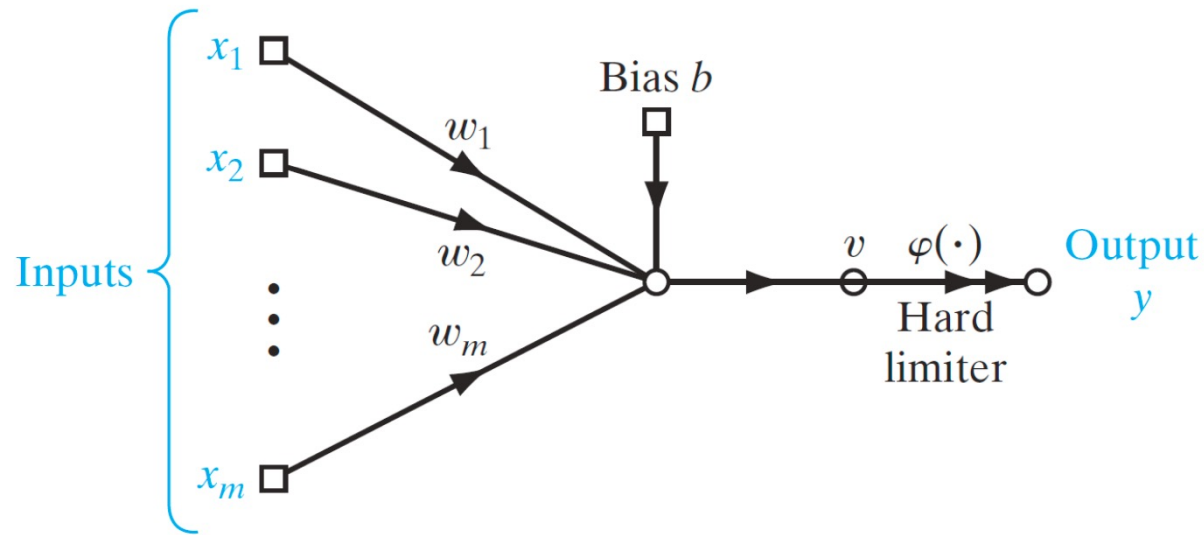
$$\text{net}_j = \sum_i w_{j,i} \cdot o_i$$



- The output of the unit is:
 - 0 if $\text{net}_j < T_j$; 1 otherwise
 - T_j is the threshold



Single-layer perceptron by Rosenblatt [1958]



预测

$$\hat{y} = \sigma \left(\sum_{i=1}^m w_i x_i + b \right)$$

激活函数

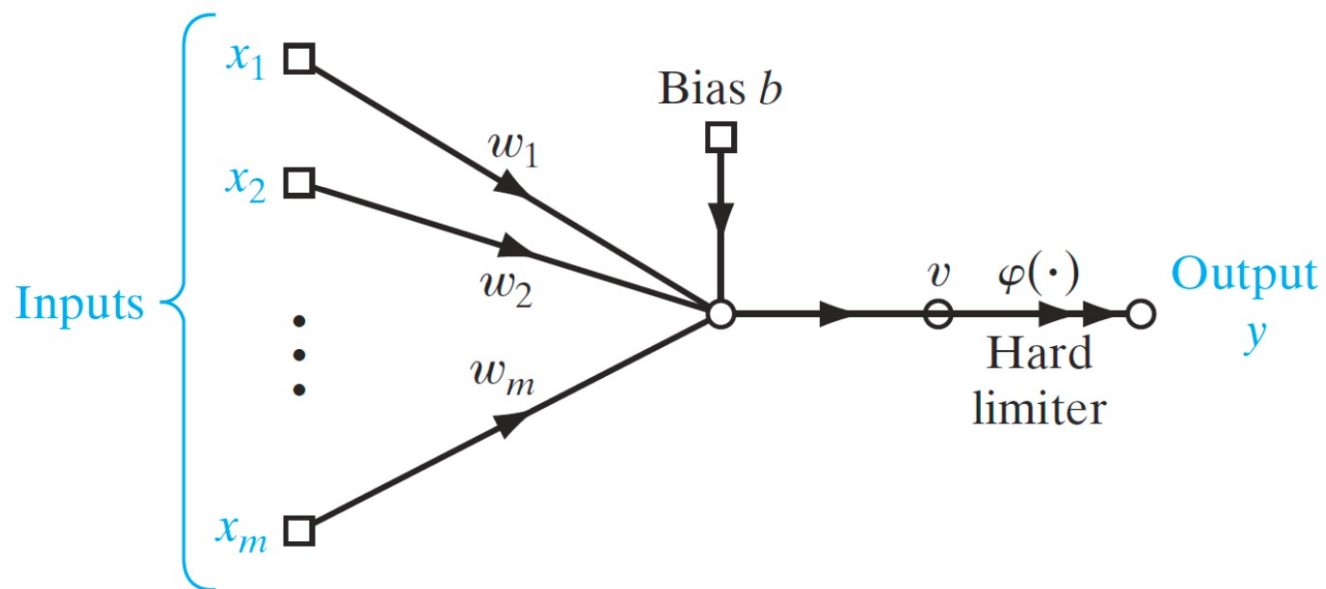
$$\sigma(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

□ Rosenblatt [1958] 进一步提出感知机作为第一个在“老师”指导下进行学习的模型 (即监督学习)

□ 专注在如何找到合适的用于二分类任务的权重 w_m

- $y = 1$: 类别1
- $y = -1$: 类别2

Training perception



预测

激活函数

$$\hat{y} = \sigma \left(\sum_{i=1}^m w_i x_i + b \right)$$

$$\sigma(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

□ 训练

$$w_i = w_i + \eta(y - \hat{y})x_i$$

$$b = b + \eta(y - \hat{y})$$

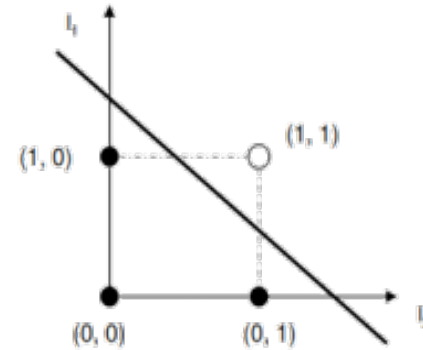
□ 下列规则等价:

- 如果输出正确, 则不进行操作
- 如果输出高了, 降低正输入的权重
- 如果输出低了, 增加正输入的权重

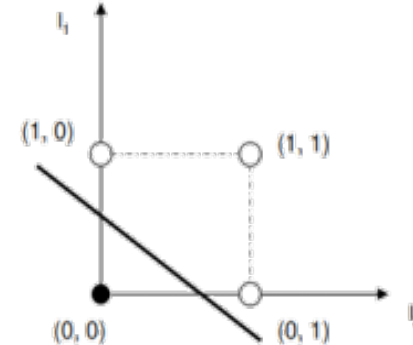
Limitation of perception

- Minsky and Papert [1969] showed that some rather elementary computations, such as XOR problem, could not be done by Rosenblatt's one-layer perceptron
- However Rosenblatt believed the limitations could be overcome if more layers of units to be added, but no learning algorithm known to obtain the weights yet

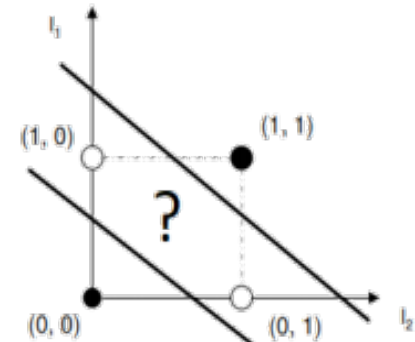
AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1



OR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	1

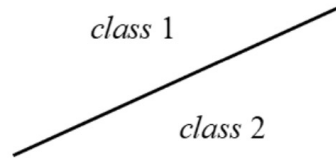


XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0

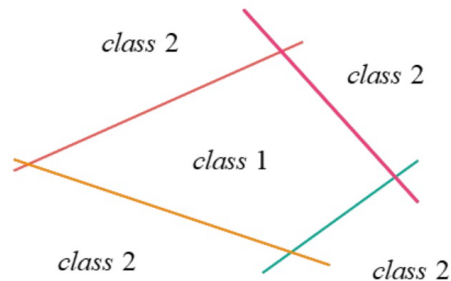
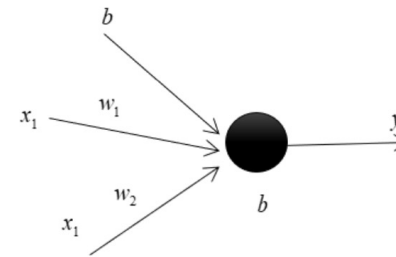


Solution: Add hidden layers

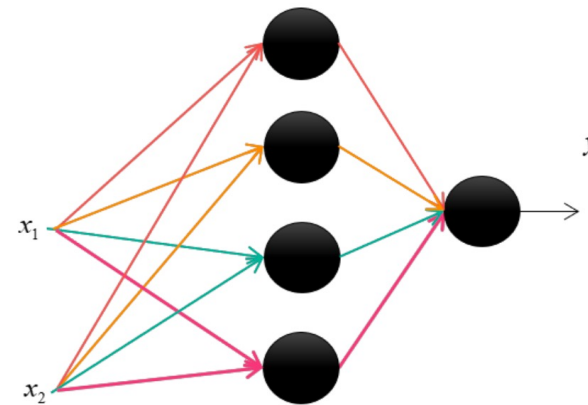
- Adding hidden layers to learn more general scenarios



决策边界: $x_1w_1 + x_2w_2 + b = 0$



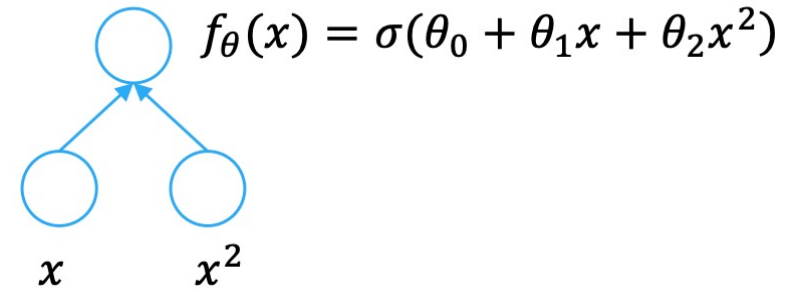
每个隐含节点负责实现凸区域的一条边界线



Computation

- Single-layer function

- $f_{\theta}(x) = \sigma(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

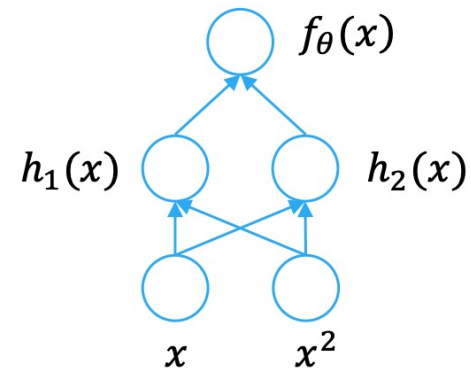


- Multi-layer function

- $h_1(x) = \sigma(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

- $h_2(x) = \sigma(\theta_3 + \theta_4 x_1 + \theta_5 x_2)$

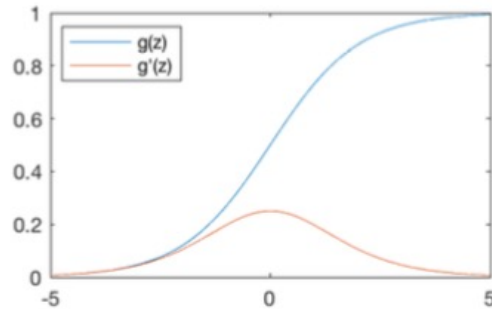
- $f_{\theta}(x) = \sigma(\theta_6 + \theta_7 h_1 + \theta_8 h_2)$



Non-linear activation functions

- Adding non-linearity allows the network to learn and represent complex patterns in the data
- Common non-linear activation functions

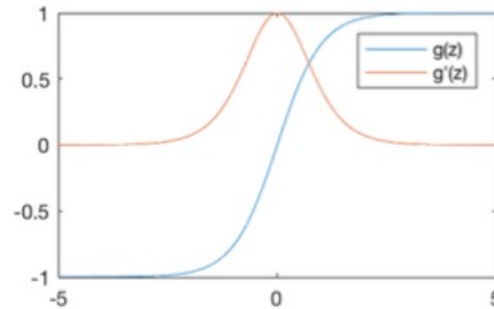
Sigmoid Function



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

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

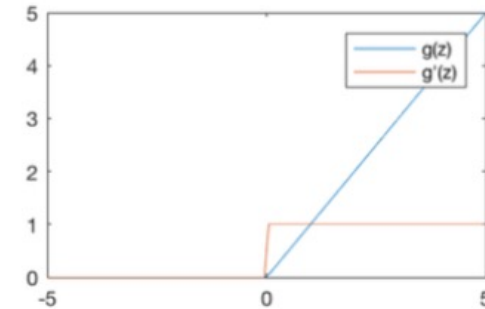
Hyperbolic Tangent



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

$$\sigma'(z) = 1 - \sigma(z)^2$$

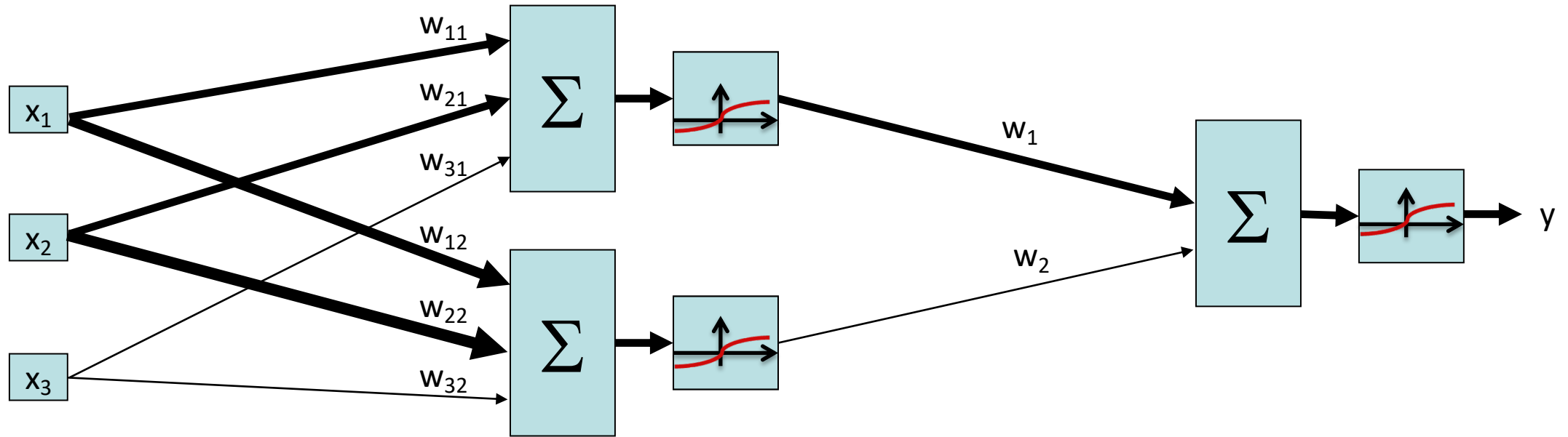
Rectified Linear Unit (ReLU)



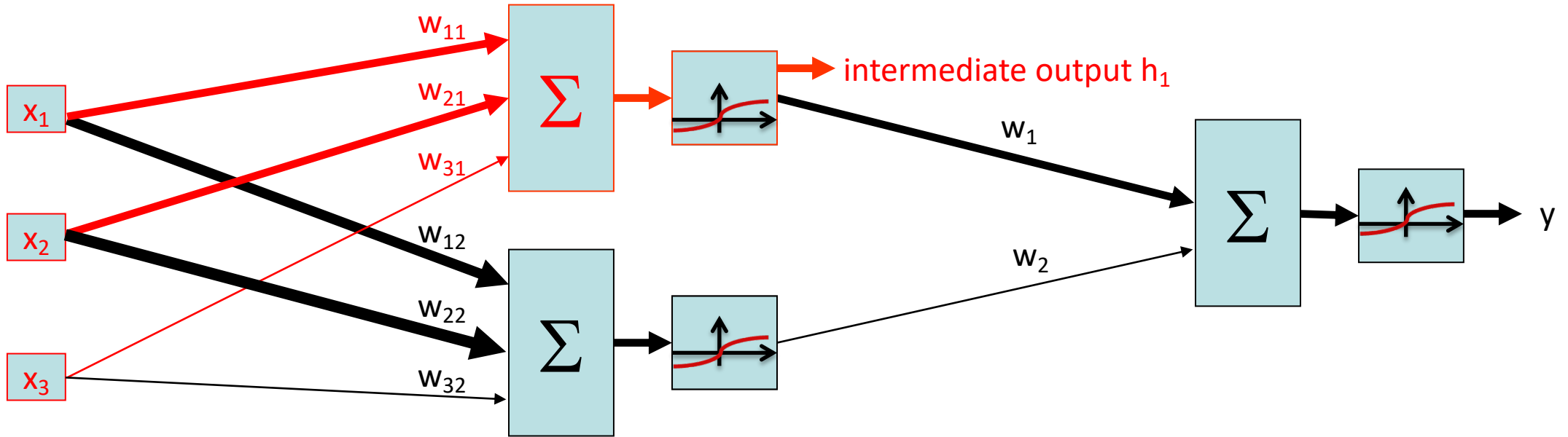
$$\sigma(z) = \max(0, z)$$

$$\sigma'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

2-Layer, 2-Neuron Neural Network

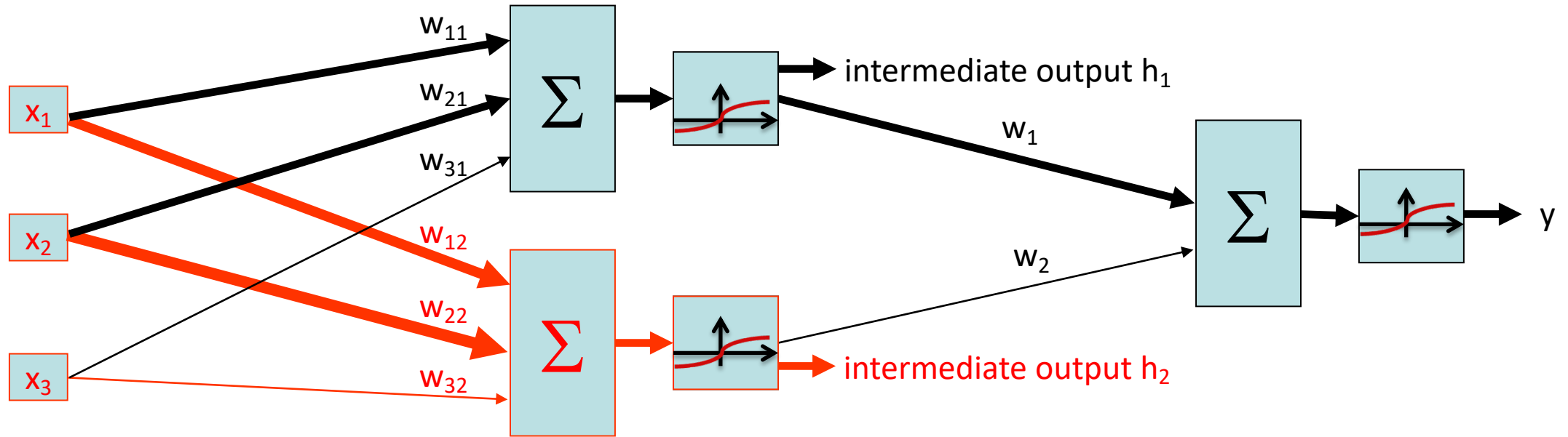


2-Layer, 2-Neuron Neural Network



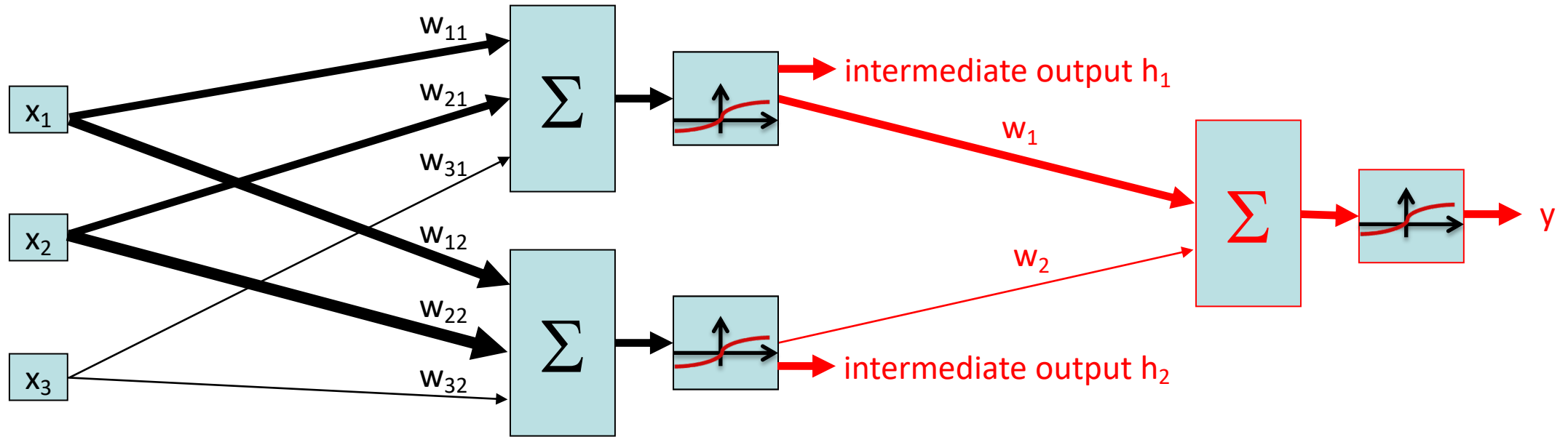
$$\begin{aligned} \text{intermediate output } h_1 &= \sigma(w_{11}x_1 + w_{21}x_2 + w_{31}x_3) \\ &= \frac{1}{1 + e^{-(w_{11}x_1 + w_{21}x_2 + w_{31}x_3)}} \end{aligned}$$

2-Layer, 2-Neuron Neural Network



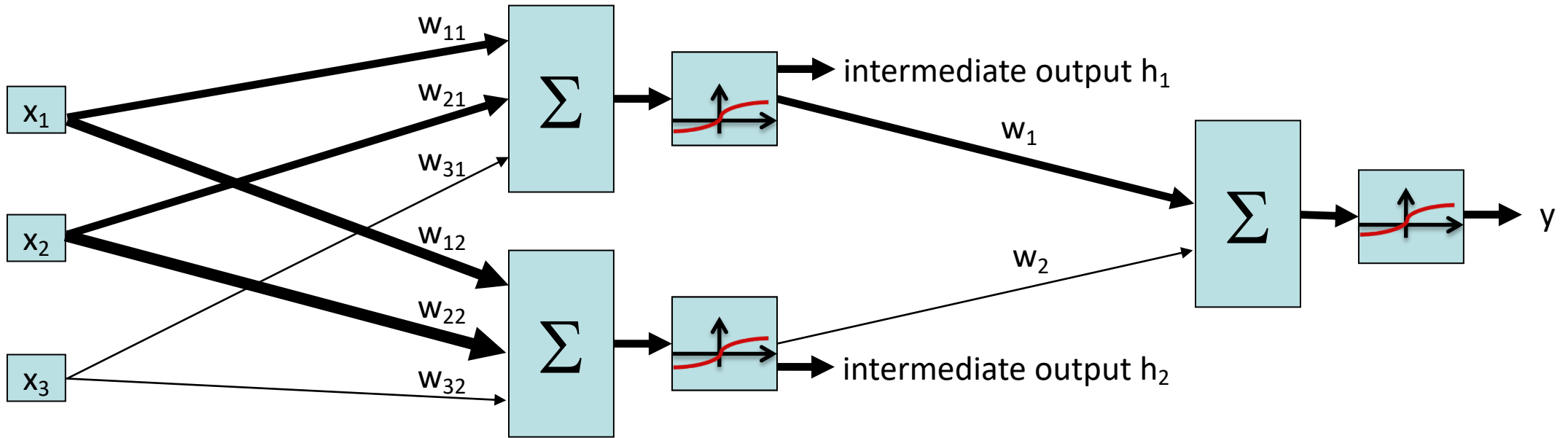
$$\begin{aligned} \text{intermediate output } h_2 &= \sigma(w_{12}x_1 + w_{22}x_2 + w_{32}x_3) \\ &= \frac{1}{1 + e^{-(w_{12}x_1 + w_{22}x_2 + w_{32}x_3)}} \end{aligned}$$

2-Layer, 2-Neuron Neural Network



$$y = \sigma(w_1 h_1 + w_2 h_2)$$
$$= \frac{1}{1 + e^{-(w_1 h_1 + w_2 h_2)}}$$

2-Layer, 2-Neuron Neural Network



$$y = \sigma(w_1h_1 + w_2h_2)$$
$$= \sigma(w_1\sigma(w_{11}x_1 + w_{21}x_2 + w_{31}x_3) + w_2\sigma(w_{12}x_1 + w_{22}x_2 + w_{32}x_3))$$

Vectorization

$$\begin{aligned}y &= \sigma(w_1 h_1 + w_2 h_2) \\ &= \sigma(w_1 \sigma(w_{11} x_1 + w_{21} x_2 + w_{31} x_3) + w_2 \sigma(w_{12} x_1 + w_{22} x_2 + w_{32} x_3))\end{aligned}$$

The same equation, formatted with matrices:

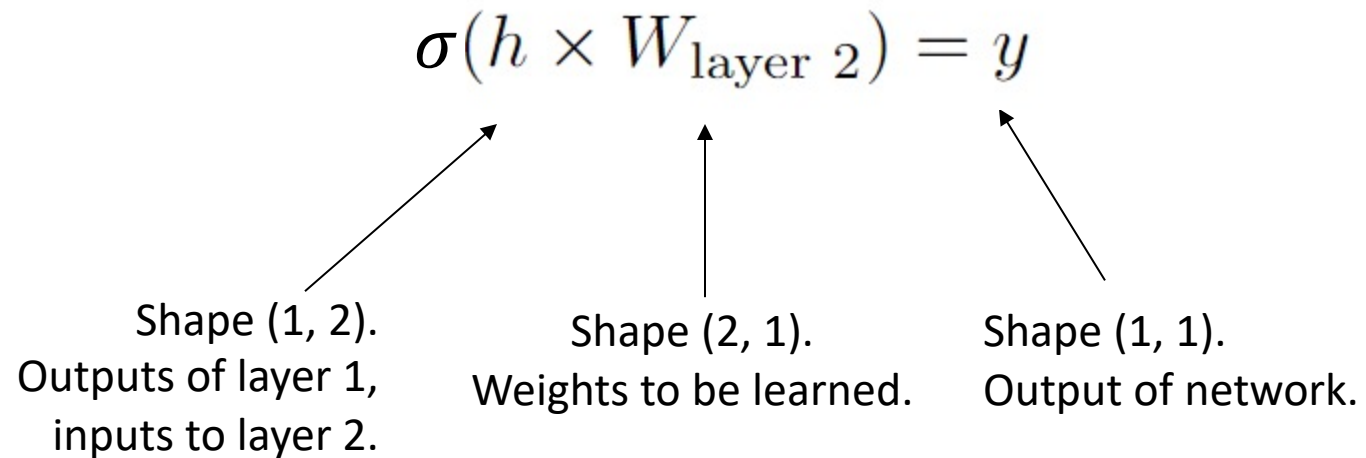
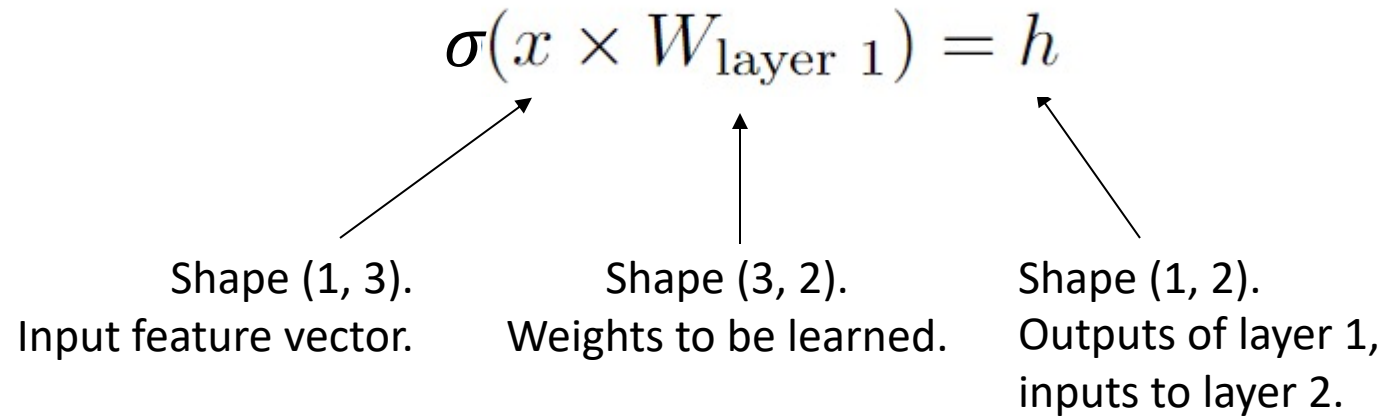
$$\begin{aligned}& \sigma \left(\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} \right) \\ &= \sigma \left(\begin{bmatrix} w_{11} x_1 + w_{21} x_2 + w_{31} x_3 & w_{12} x_1 + w_{22} x_2 + w_{32} x_3 \end{bmatrix} \right) \\ &= \begin{bmatrix} h_1 & h_2 \end{bmatrix}\end{aligned}$$

$$\sigma \left(\begin{bmatrix} h_1 & h_2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \right) = \sigma(w_1 h_1 + w_2 h_2) = y$$

The same equation, formatted more compactly by introducing variables representing each matrix:

$$\sigma(x \times W_{\text{layer 1}}) = h \qquad \sigma(h \times W_{\text{layer 2}}) = y$$

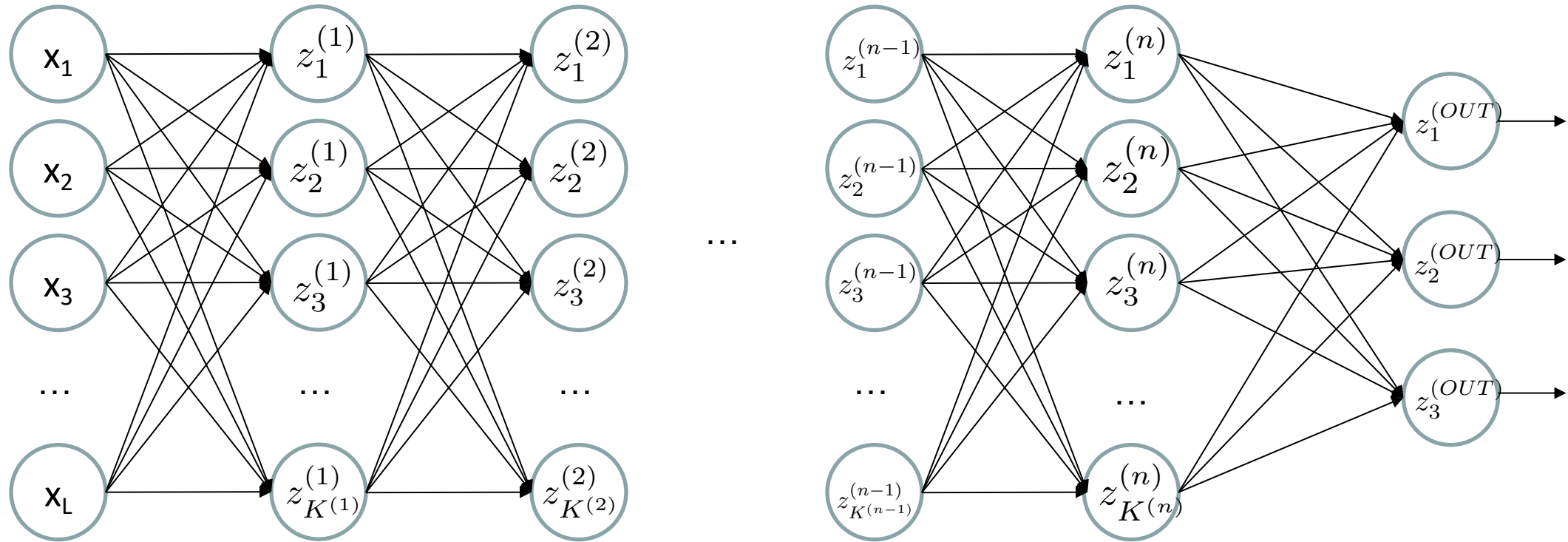
2-Layer, 2-Neuron Neural Network



Multi-Layer Neural Network

- Input to a layer: some $dim(x)$ -dimensional input vector
- Output of a layer: some $dim(y)$ -dimensional output vector
 - $dim(y)$ is the number of neurons in the layer (1 output per neuron)
- Process of converting input to output:
 - Multiply the $(1, dim(x))$ input vector with a $(dim(x), dim(y))$ weight vector. The result has shape $(1, dim(y))$.
 - Apply some non-linear function (e.g. sigmoid) to the result. The result still has shape $(1, dim(y))$.
- Big idea: Chain layers together
 - The input could come from a previous layer's output
 - The output could be used as the input to the next layer

Deep Neural Network

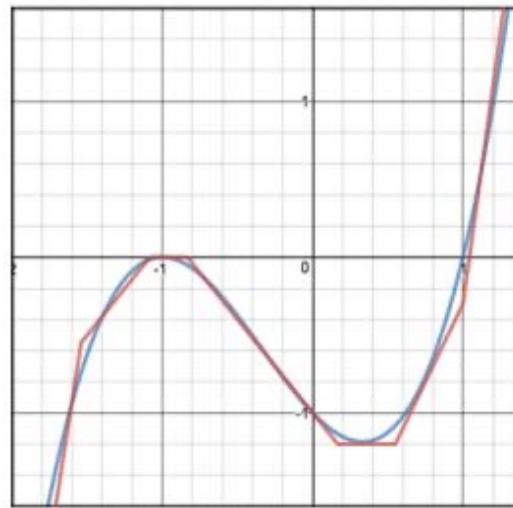


$$z_i^{(k)} = \sigma \left(\sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)} \right)$$

σ = nonlinear activation function

Universal approximation theorem

- Theorem (Universal Function Approximators). A two-layer neural network with a sufficient number of neurons can approximate any continuous function to any desired accuracy.



$$n_1(x) = \text{Relu}(-5x - 7.7)$$

$$n_2(x) = \text{Relu}(-1.2x - 1.3)$$

$$n_3(x) = \text{Relu}(1.2x + 1)$$

$$n_4(x) = \text{Relu}(1.2x - .2)$$

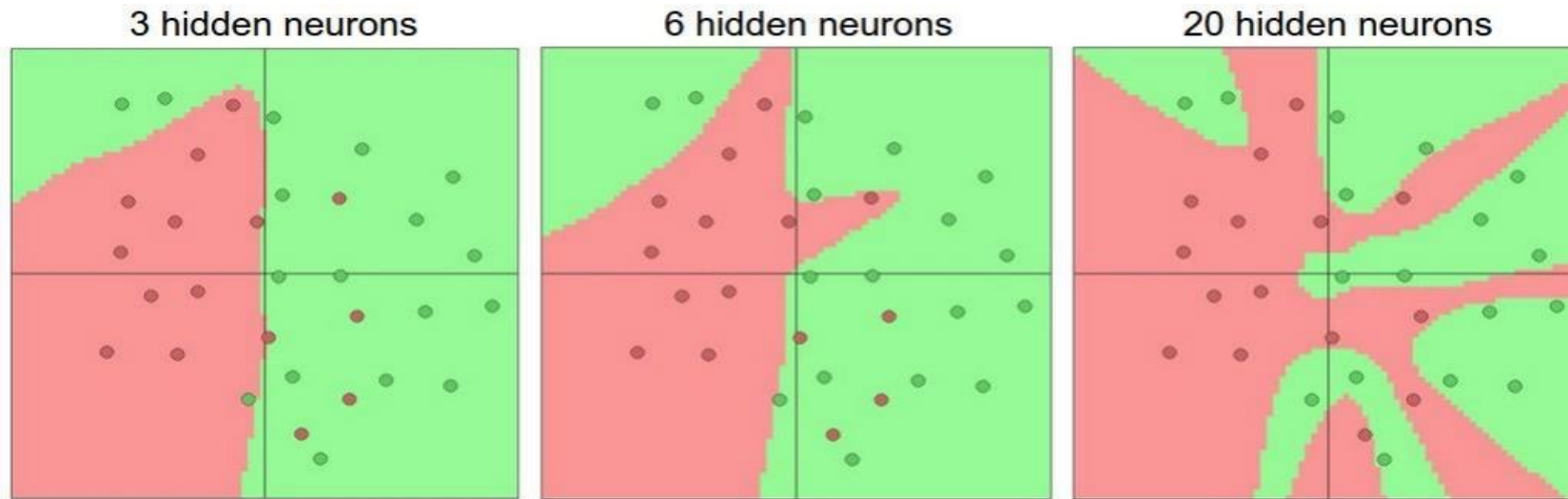
$$n_5(x) = \text{Relu}(2x - 1.1)$$

$$n_6(x) = \text{Relu}(5x - 5)$$

$$Z(x) = -n_1(x) - n_2(x) - n_3(x) \\ + n_4(x) + n_5(x) + n_6(x)$$

Increasing power of approximation

- With more neurons, its approximation power increases. The decision boundary covers more details (risk of overfitting)



- Usually in applications, we use more layers with structures to approximate complex functions instead of one hidden layer with many neurons

Training: Backpropagation

Review: Derivatives and Gradients

- What is the derivative of the function $g(x) = x^2 + 3$?

$$\frac{dg}{dx} = 2x$$

- What is the derivative of $g(x)$ at $x=5$?

$$\left. \frac{dg}{dx} \right|_{x=5} = 10$$

Review: Derivatives and Gradients

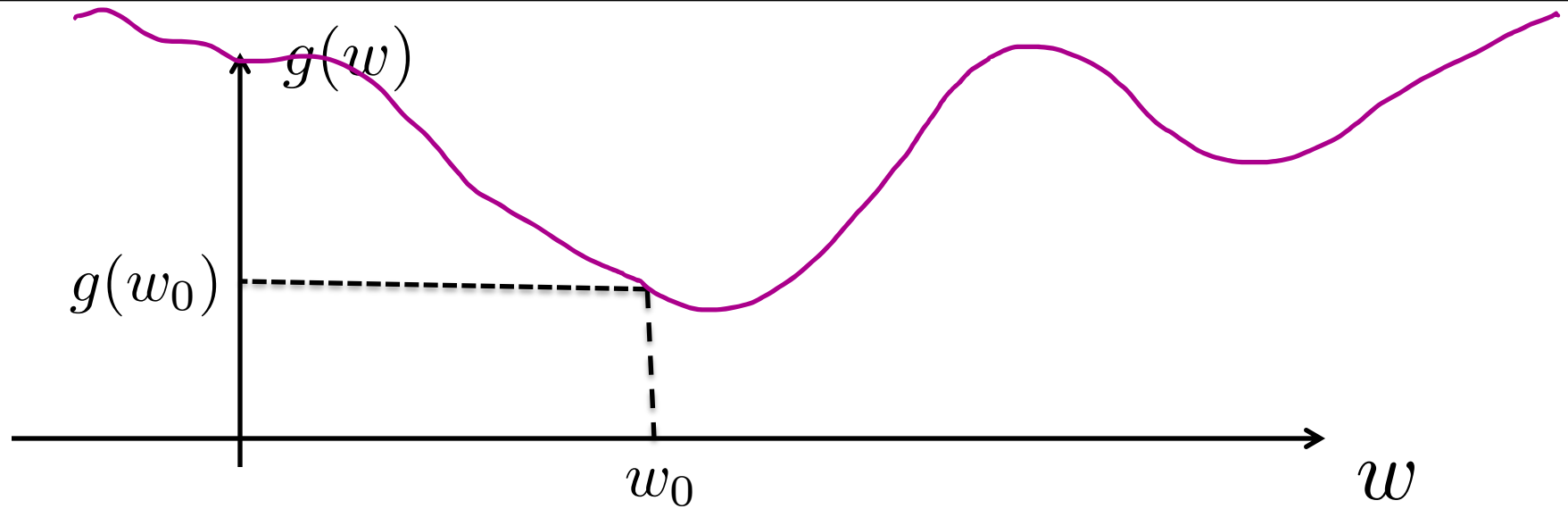
- What is the gradient of the function $g(x, y) = x^2y$?
 - Recall: Gradient is a vector of partial derivatives with respect to each variable

$$\nabla g = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 2xy \\ x^2 \end{bmatrix}$$

- What is the derivative of $g(x, y)$ at $x=0.5, y=0.5$?

$$\nabla g|_{x=0.5, y=0.5} = \begin{bmatrix} 2(0.5)(0.5) \\ (0.5^2) \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.25 \end{bmatrix}$$

1-D Optimization

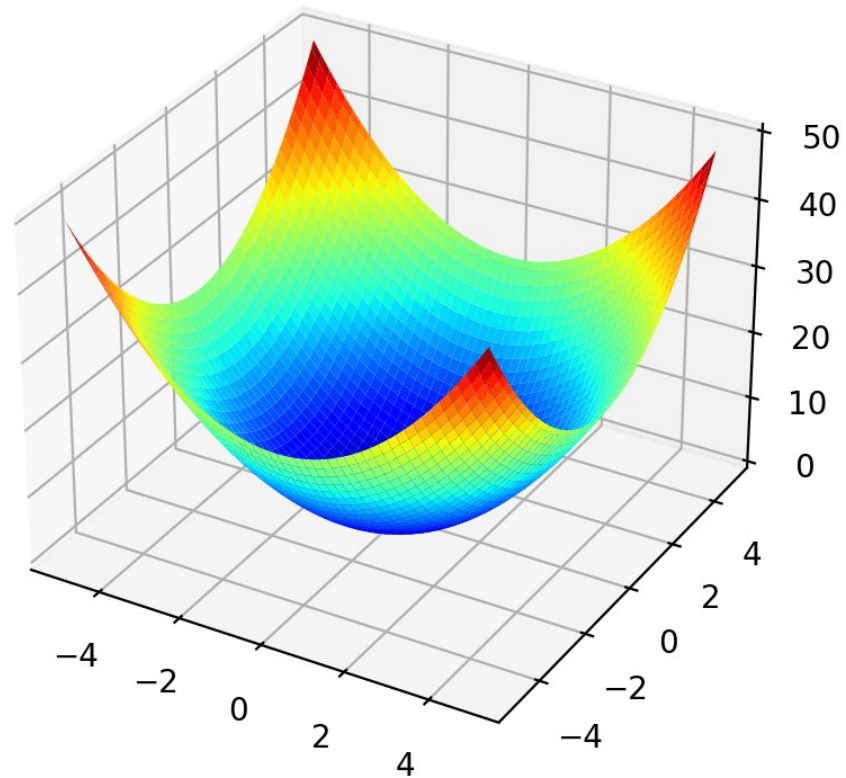


- Could evaluate $g(w_0 + h)$ and $g(w_0 - h)$
 - Then step in best direction

- Or, evaluate derivative:
$$\frac{\partial g(w_0)}{\partial w} = \lim_{h \rightarrow 0} \frac{g(w_0 + h) - g(w_0 - h)}{2h}$$

- Tells which direction to step into

2-D Optimization



Gradient descent

- Perform update in downhill direction for each coordinate
- The steeper the slope (i.e. the higher the derivative) the bigger the step for that coordinate
- E.g., consider: $g(w_1, w_2)$

- Updates:

$$w_1 \leftarrow w_1 - \alpha * \frac{\partial g}{\partial w_1}(w_1, w_2)$$

$$w_2 \leftarrow w_2 - \alpha * \frac{\partial g}{\partial w_2}(w_1, w_2)$$

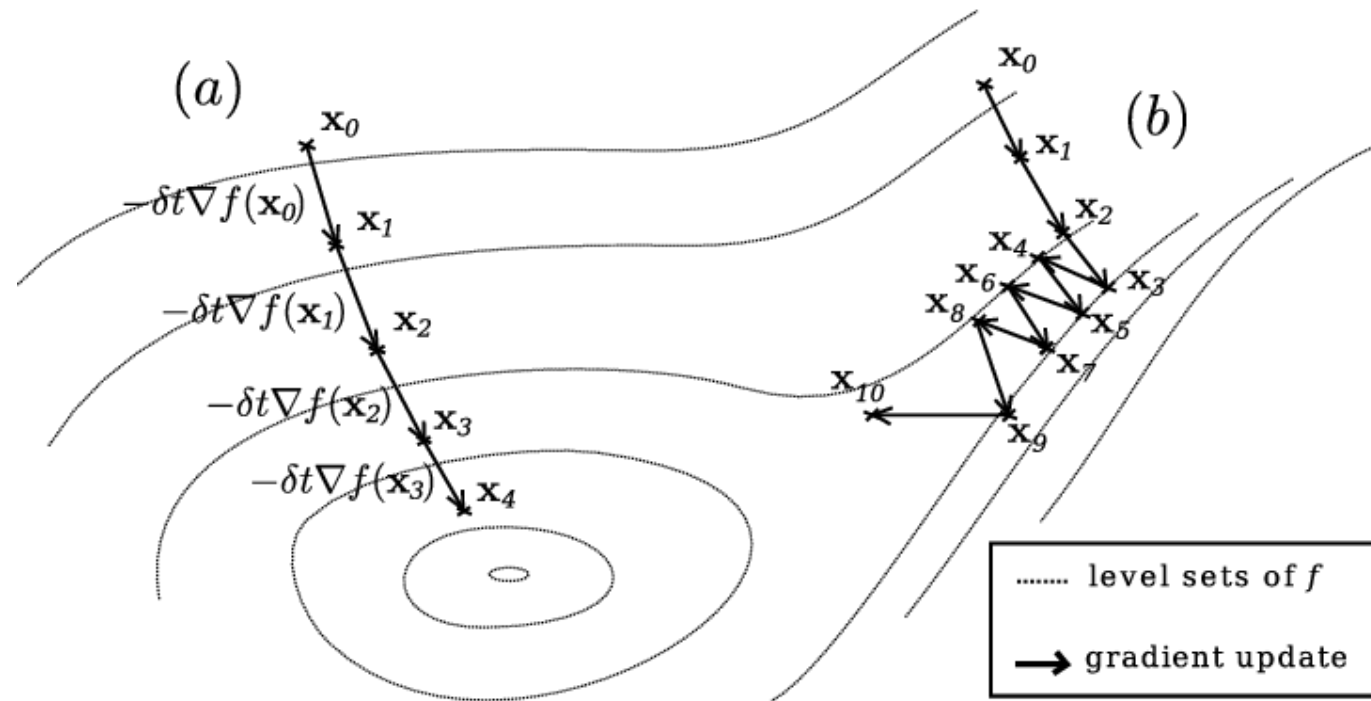
- Updates in vector notation:

$$w \leftarrow w - \alpha * \nabla_w g(w)$$

$$\text{with: } \nabla_w g(w) = \begin{bmatrix} \frac{\partial g}{\partial w_1}(w) \\ \frac{\partial g}{\partial w_2}(w) \end{bmatrix} = \text{gradient}$$

Gradient descent

- Idea:
 - Start somewhere
 - Repeat: Take a step in the gradient direction



Gradient in n dimensions

$$\nabla g = \begin{bmatrix} \frac{\partial g}{\partial w_1} \\ \frac{\partial g}{\partial w_2} \\ \dots \\ \frac{\partial g}{\partial w_n} \end{bmatrix}$$

Optimization procedure: Gradient descent

```
■ init  $w$   
■ for iter = 1, 2, ...  
 $w \leftarrow w - \alpha * \nabla g(w)$ 
```

- α : learning rate --- tweaking parameter that needs to be chosen carefully

Training Neural Networks

- Step 1: For each input in the training (sub)set x , predict a classification y using the current weights

$$\sigma(x \times W_{\text{layer 1}}) = h \qquad \sigma(h \times W_{\text{layer 2}}) = y$$

- Step 2: Compare predictions with the true y values, using a **loss function**
 - Higher value of loss function = bad model
 - Lower value of loss function = good model
 - Example: **zero-one loss**: count the number of misclassified inputs
 - Example: **log loss** (derived from maximum likelihood)
 - Example: **sum of squared errors** (more on this soon)
- Step 3: Use numerical method (e.g. gradient descent) to minimize loss
 - Loss is a function of the weights. Optimization goal: find weights that minimize loss

Optimization Procedure: Gradient Descent

```
▪ init  $w$   
▪ for iter = 1, 2, ...  
     $w \leftarrow w - \alpha \nabla \text{Loss}(w)$ 
```

- α : learning rate --- tweaking parameter that needs to be chosen carefully

Computing Gradients

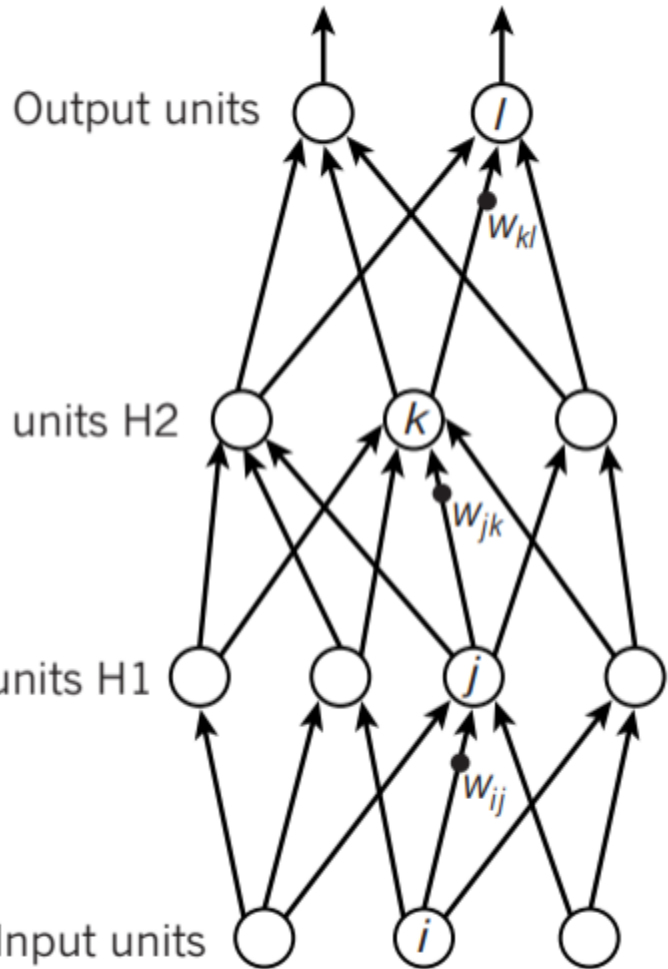
- How do we compute gradients of these loss functions?
 - Repeated application of the chain rule:

If $f(x) = g(h(x))$

Then $f'(x) = g'(h(x))h'(x)$

→ Derivatives can be computed by following well-defined procedures

Feed forward vs. Backpropagation



$$y_l = f(z_l)$$

$$z_l = \sum_{k \in H2} w_{kl} y_k$$

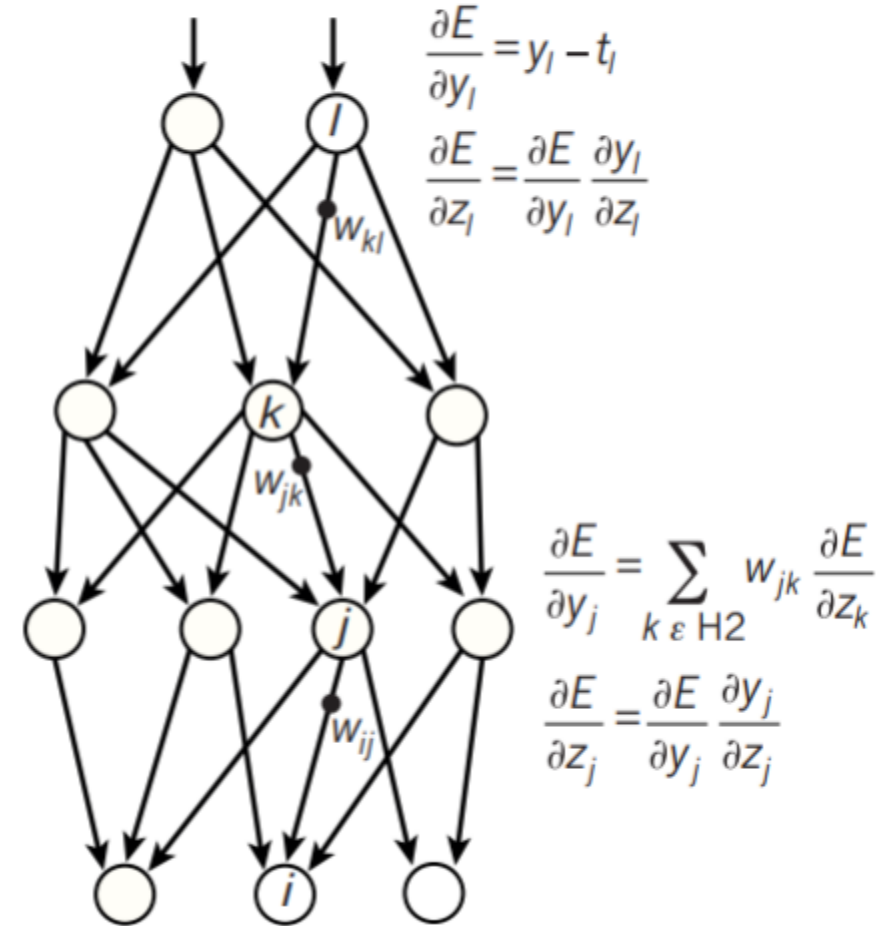
$$y_k = f(z_k)$$

$$z_k = \sum_{j \in H1} w_{jk} y_j$$

$$y_j = f(z_j)$$

$$z_j = \sum_{i \in \text{Input}} w_{ij} x_i$$

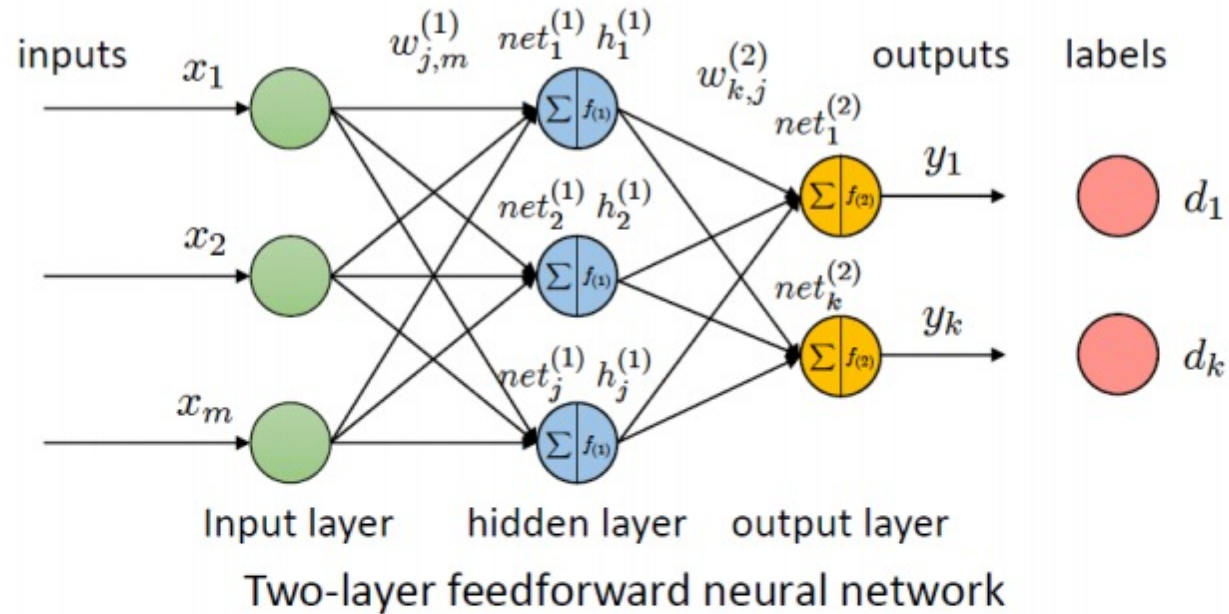
Compare outputs with correct answer to get error derivatives



Backpropagation - demo

- [Backpropagation demo](#)

Make a prediction



Feed-forward prediction:

$$x = (x_1, \dots, x_m) \xrightarrow{\quad} h_j^{(1)} \xrightarrow{\quad} y_k$$

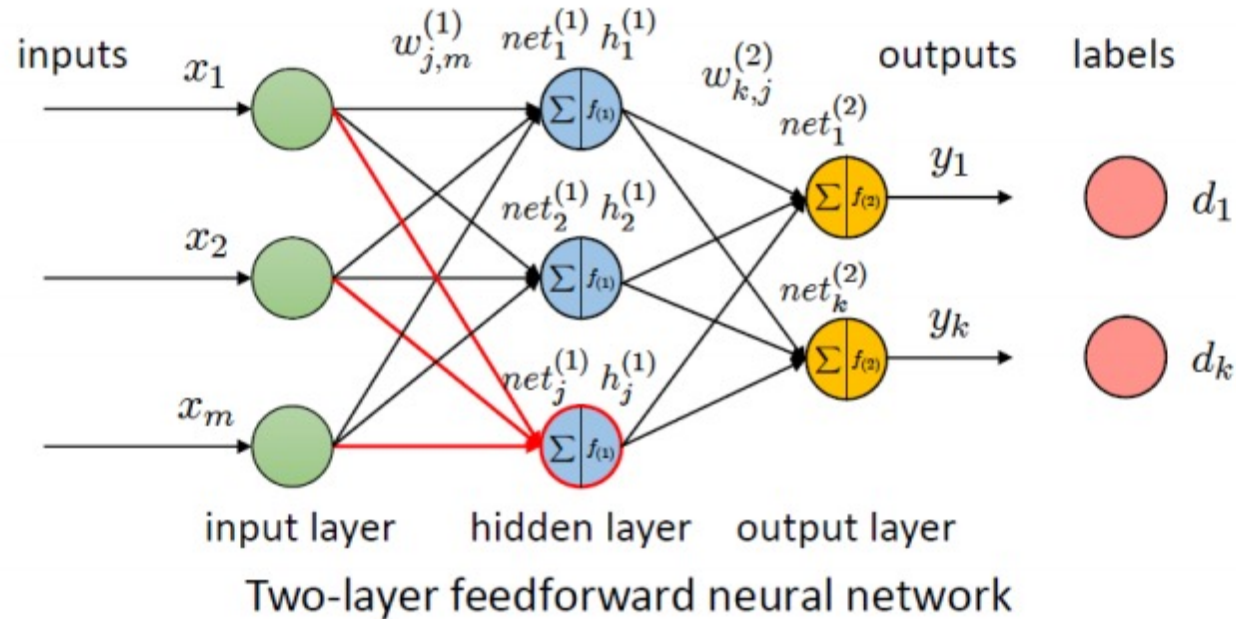
$$h_j^{(1)} = f_{(1)}(net_j^{(1)}) = f_{(1)}\left(\sum_m w_{j,m}^{(1)} x_m\right) \quad y_k = f_{(2)}(net_k^{(2)}) = f_{(2)}\left(\sum_j w_{k,j}^{(2)} h_j^{(1)}\right)$$

where

$$net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$$

$$net_k^{(2)} = \sum_j w_{k,j}^{(2)} h_j^{(1)}$$

Make a prediction (cont.)



Feed-forward prediction:

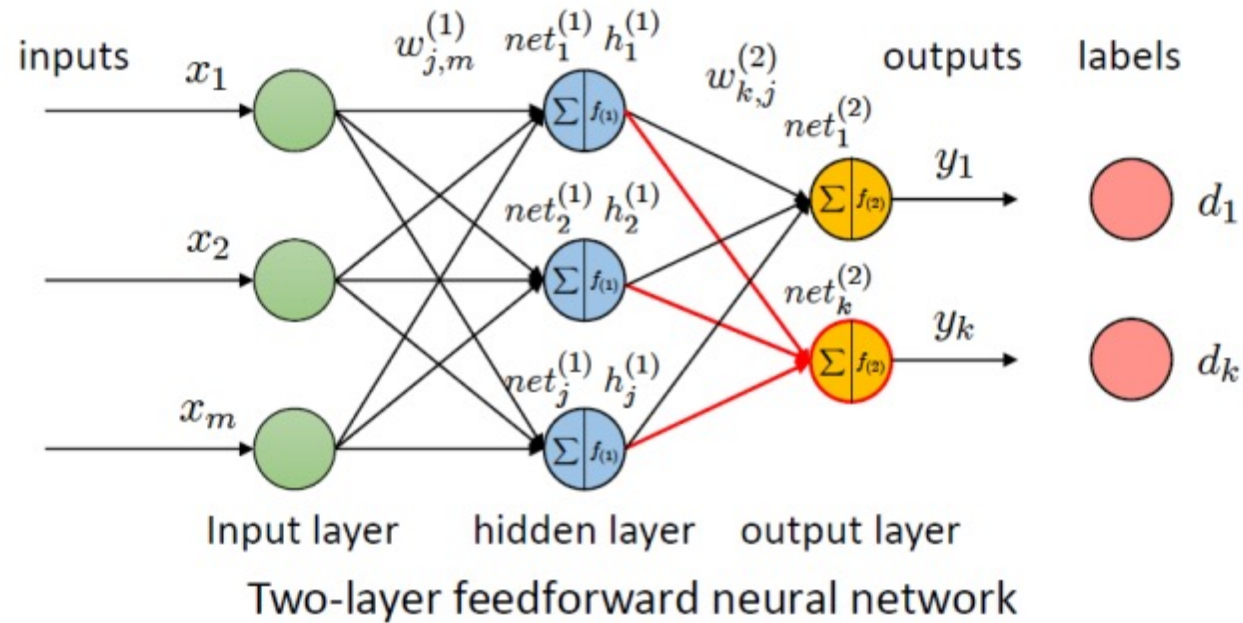
$$x = (x_1, \dots, x_m) \xrightarrow{h_j^{(1)}} h_j^{(1)} \xrightarrow{y_k} y_k$$

$$h_j^{(1)} = f_{(1)}(net_j^{(1)}) = f_{(1)}\left(\sum_m w_{j,m}^{(1)} x_m\right) \quad y_k = f_{(2)}(net_k^{(2)}) = f_{(2)}\left(\sum_j w_{k,j}^{(2)} h_j^{(1)}\right)$$

where

$$net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m \quad net_k^{(2)} = \sum_j w_{k,j}^{(2)} h_j^{(1)}$$

Make a prediction (cont.)



Feed-forward prediction:

$$x = (x_1, \dots, x_m) \longrightarrow h_j^{(1)} \longrightarrow y_k$$

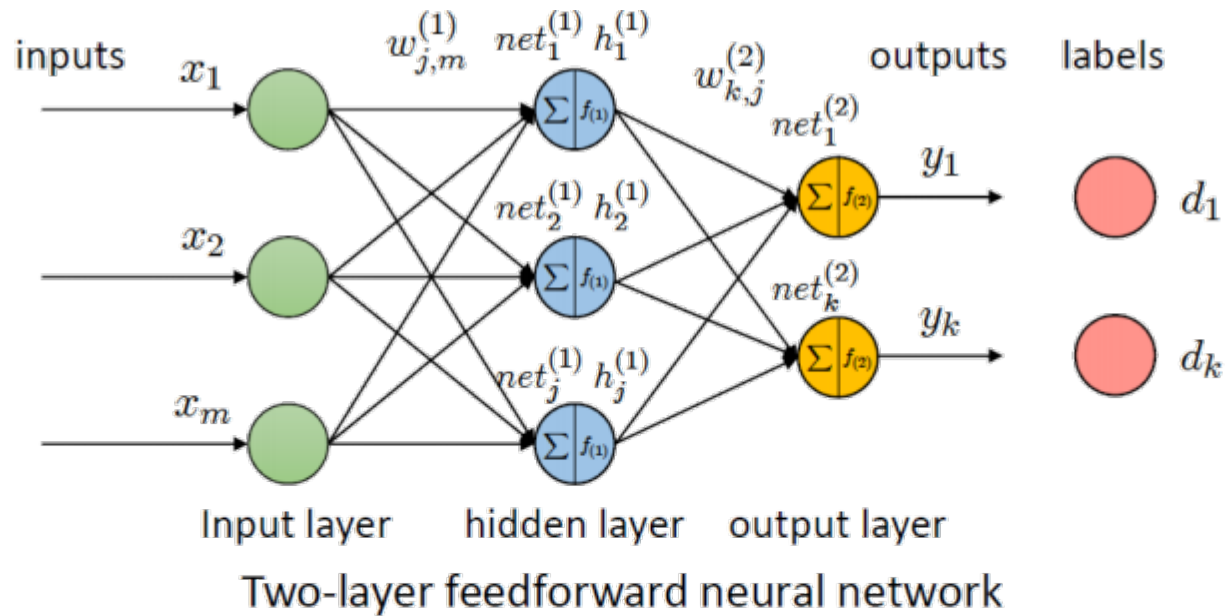
$$h_j^{(1)} = f_{(1)}(net_j^{(1)}) = f_{(1)}\left(\sum_m w_{j,m}^{(1)} x_m\right) \quad y_k = f_{(2)}(net_k^{(2)}) = f_{(2)}\left(\sum_j w_{k,j}^{(2)} h_j^{(1)}\right)$$

where

$$net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$$

$$net_k^{(2)} = \sum_j w_{k,j}^{(2)} h_j^{(1)}$$

Backpropagation



- Assume all the activation functions are **sigmoid**
- Error function $E = \frac{1}{2} \sum_k (y_k - d_k)^2$
- $\frac{\partial E}{\partial y_k} = y_k - d_k$
- $\frac{\partial y_k}{\partial w_{k,j}^{(2)}} = f'_{(2)}(net_k^{(2)})h_j^{(1)} = y_k(1 - y_k)h_j^{(1)}$
- $\Rightarrow \frac{\partial E}{\partial w_{k,j}^{(2)}} = (y_k - d_k)y_k(1 - y_k)h_j^{(1)}$
- $\Rightarrow w_{k,j}^{(2)} \leftarrow w_{k,j}^{(2)} - \eta \underbrace{(y_k - d_k)y_k(1 - y_k)h_j^{(1)}}_{\delta_k^{(2)}}$

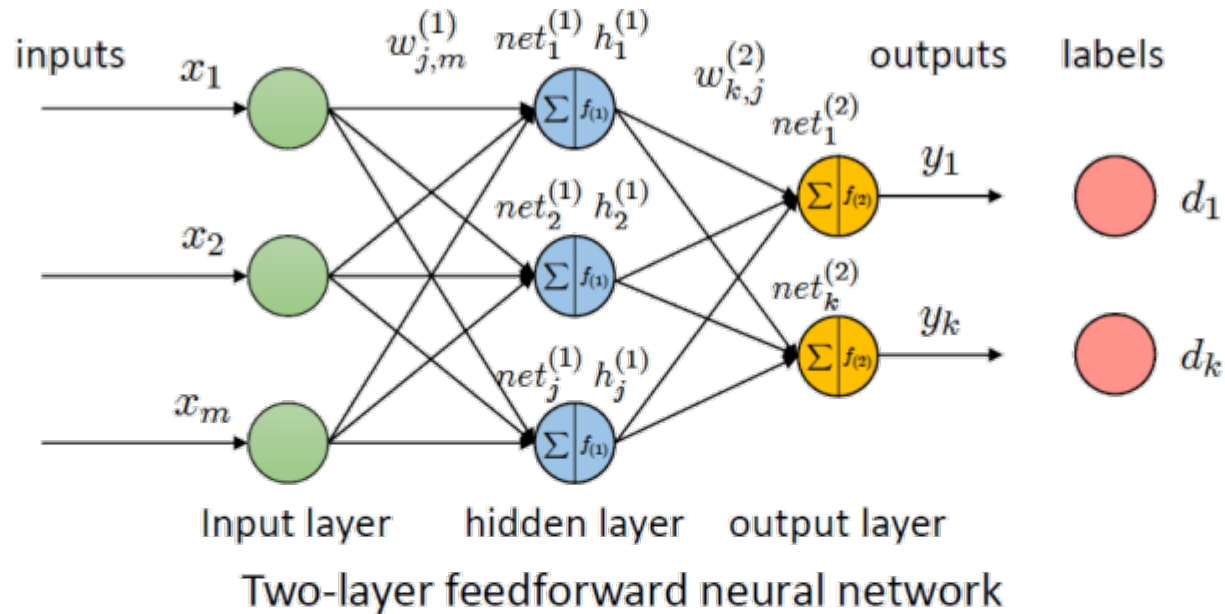
Feed-forward prediction:

$$h_j^{(1)} = f_{(1)}(net_j^{(1)}) = f_{(1)}\left(\sum_m w_{j,m}^{(1)} x_m\right) \quad y_k = f_{(2)}(net_k^{(2)}) = f_{(2)}\left(\sum_j w_{k,j}^{(2)} h_j^{(1)}\right)$$

$x = (x_1, \dots, x_m)$ $\xrightarrow{h_j^{(1)}}$ y_k

where $net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$ $net_k^{(2)} = \sum_j w_{k,j}^{(2)} h_j^{(1)}$

Backpropagation (cont.)



- Error function $E = \frac{1}{2} \sum_k (y_k - d_k)^2$
- $\frac{\partial E}{\partial y_k} = y_k - d_k$
- $\frac{\partial y_k}{\partial h_j^{(1)}} = y_k(1 - y_k)w_{k,j}^{(2)}$
- $\frac{\partial h_j^{(1)}}{\partial w_{j,m}^{(1)}} = f'_{(1)}(net_j^{(1)}) x_m = h_j^{(1)}(1 - h_j^{(1)}) x_m$
- $\Rightarrow \frac{\partial E}{\partial w_{j,m}^{(1)}} = h_j^{(1)}(1 - h_j^{(1)}) \sum_k w_{k,j}^{(2)}(y_k - d_k)y_k(1 - y_k)x_m$
- $\Rightarrow w_{j,m}^{(1)} \leftarrow w_{j,m}^{(1)} - \eta h_j^{(1)} \underbrace{\left(\sum_k w_{k,j}^{(2)}(y_k - d_k)y_k(1 - y_k) \right)}_{\delta_j^{(1)}} x_m$

$\delta_j^{(1)}$ $\delta_k^{(2)}$

Feed-forward prediction:

$$x = (x_1, \dots, x_m) \xrightarrow{\quad} h_j^{(1)} \xrightarrow{\quad} y_k$$

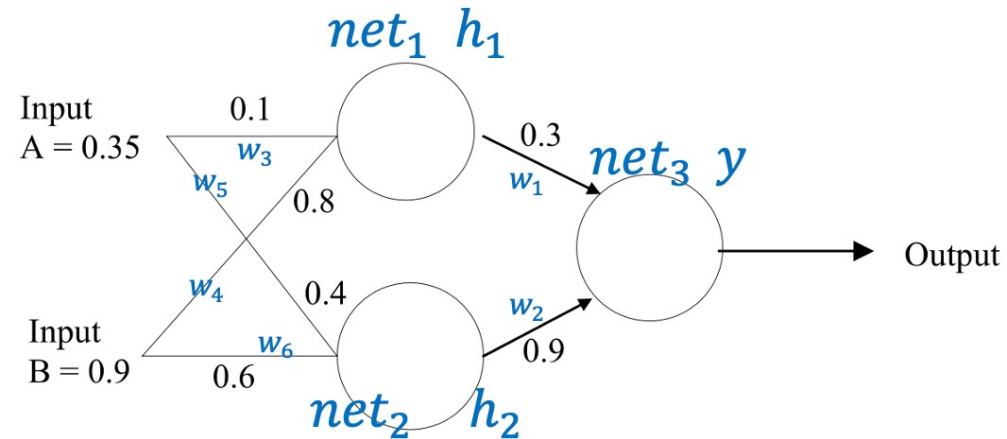
where

$$net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m \quad net_k^{(2)} = \sum_j w_{k,j}^{(2)} h_j^{(1)}$$

$$h_j^{(1)} = f_{(1)}(net_j^{(1)}) = f_{(1)}\left(\sum_m w_{j,m}^{(1)} x_m\right) \quad y_k = f_{(2)}(net_k^{(2)}) = f_{(2)}\left(\sum_j w_{k,j}^{(2)} h_j^{(1)}\right)$$

Backpropagation - example

- Consider the simple network below:



- Assume that the neurons have sigmoid activation function
 - Perform a forward pass on the network and find the predicted output
 - Perform a reverse pass (training) once (target = 0.5) with $\eta = 1$
 - Perform a further forward pass and comment on the result

Backpropagation – example (cont.)

(i)

Input to top neuron = $(0.35 \times 0.1) + (0.9 \times 0.8) = 0.755$. Out = 0.68.

Input to bottom neuron = $(0.9 \times 0.6) + (0.35 \times 0.4) = 0.68$. Out = 0.6637.

Input to final neuron = $(0.3 \times 0.68) + (0.9 \times 0.6637) = 0.80133$. Out = 0.69.

(ii) New weights for output layer:

$$\begin{aligned}w1^+ &= w1 - (y - t)y(1 - y)h_1 \\ &= 0.3 - (0.69 - 0.5) \times 0.69 \times (1 - 0.69) \times 0.68 = 0.272392.\end{aligned}$$

$$\begin{aligned}w2^+ &= w2 - (y - t)y(1 - y)h_2 \\ &= 0.9 - (0.69 - 0.5) \times 0.69 \times (1 - 0.69) \times 0.6637 = 0.87305.\end{aligned}$$

New weights for hidden layer:

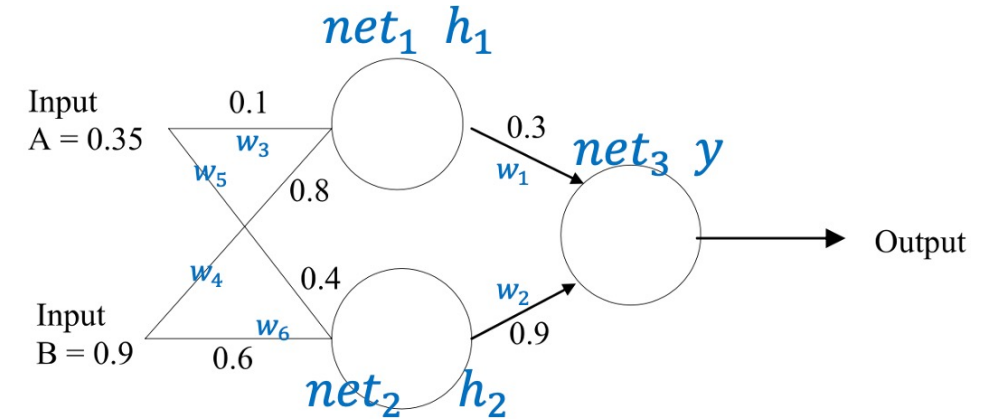
$$w3^+ = w3 - (y - t)y(1 - y)w_1 h_1 (1 - h_1) A$$

=?

$$w4^+ = ?$$

$$w5^+ = ?$$

$$w6^+ = ?$$



Fun Neural Net Demo Site

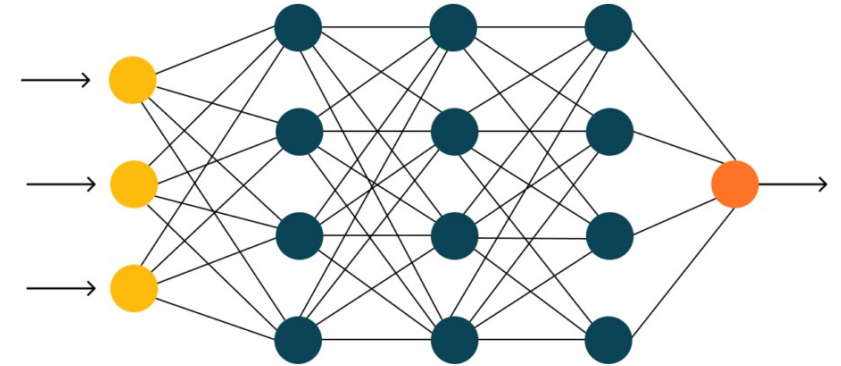
- Demo-site:
 - <http://playground.tensorflow.org/>

Modules in modern neural networks

Multi-layer perceptron (MLP)

- Denote the matrix multiplication operation with (W, b) as

- $\text{MM}_{W,b}(x) = Wx + b$

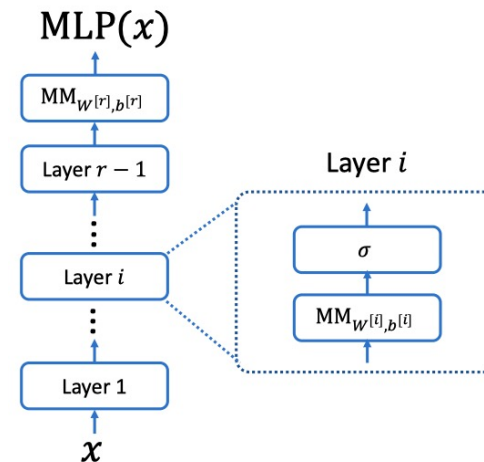


- Denote σ as the activation function
- Denote $W^{[r]}, b^{[r]}$ as the weight/bias of the r -th layer
- Then the MLP can be represented as

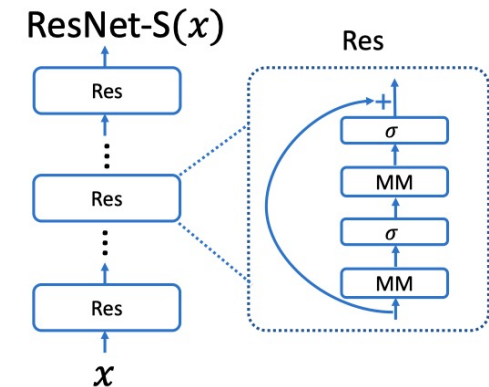
$$\text{MLP}(x) = \text{MM}_{W^{[r]}, b^{[r]}}(\sigma(\text{MM}_{W^{[r-1]}, b^{[r-1]}}(\sigma(\cdots \text{MM}_{W^{[1]}, b^{[1]}}(x))))))$$

Residual connections

- An important network structure in CV: ResNet



- Residual connections



$$\text{Res}(z) = z + \sigma(\text{MM}(\sigma(\text{MM}(z))))$$

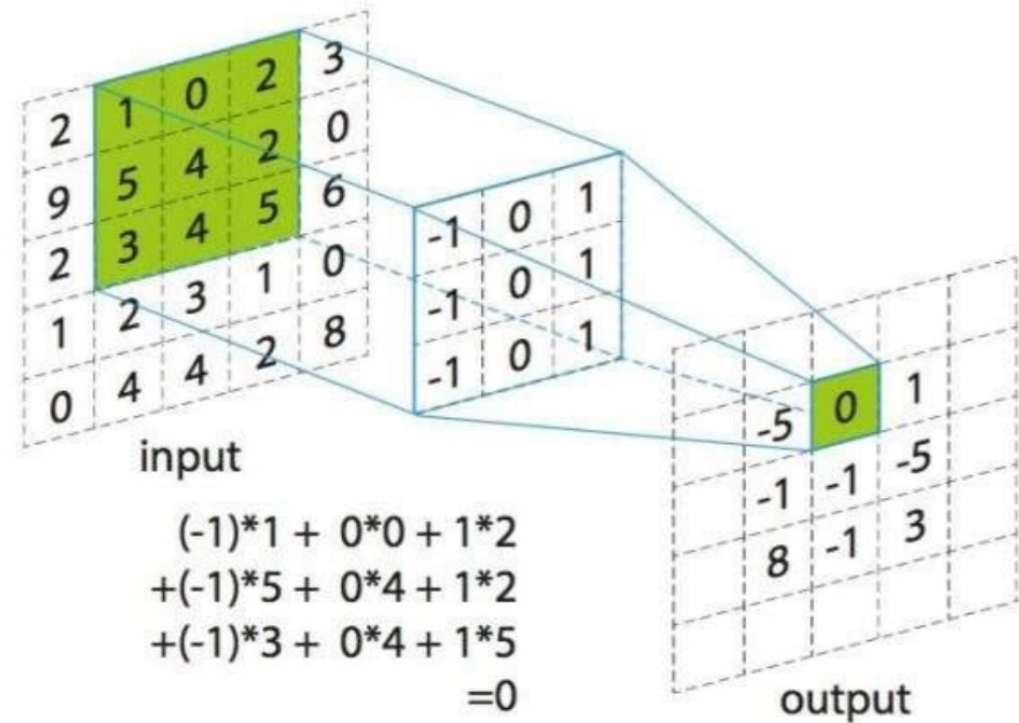
Residual connections (cont'd)

- Advantages of residual connections
 - Enable identity mapping, Improve the ability of model expression
 - Mitigate gradient disappearance, Ease training of deep networks
- Applications
 - Computer Vision (ResNet)
 - Natural Language Processing (Transformer encoder/decoder block)
 - Reinforcement Learning (policy/value networks)

Convolutional layers

- Intuition

- Given an input matrix (e.g. an image)
- Use a small matrix (called **filter** or **kernel**) to screening the input at every position of the input matrix
- Put the convolution results at corresponding positions

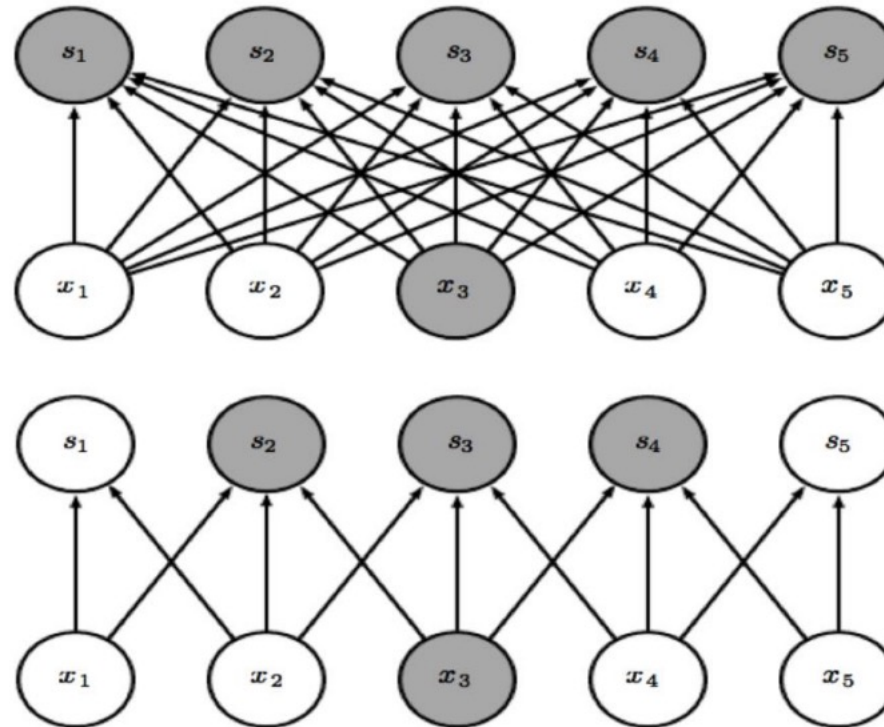


Convolutional layers (cont'd)

- Advantage

- Sparse connections

- Weight sharing



MLP
Edges: $5*5$
Parameters: $5*5$

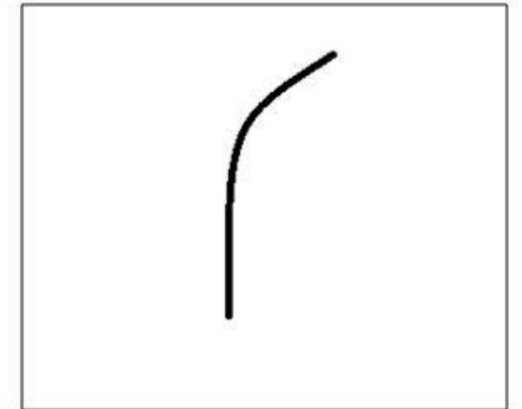
Convolution
Edges: $3*3+2*2$
Parameters: 3

Interpretation of convolution

- Convolution can be used to find an area with particular patterns
- Example
 - The filter in the left represents the edge in the right, which is the back of a mouse

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

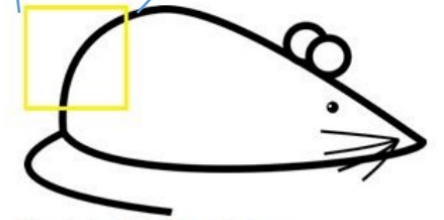
Pixel representation of filter



Visualization of a curve detector filter



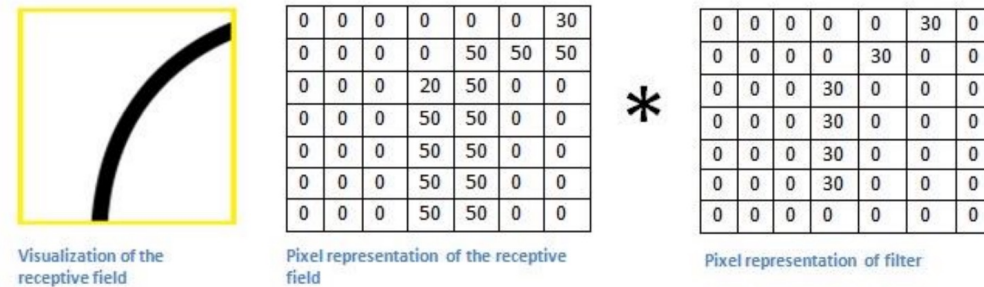
Original image



Visualization of the filter on the image

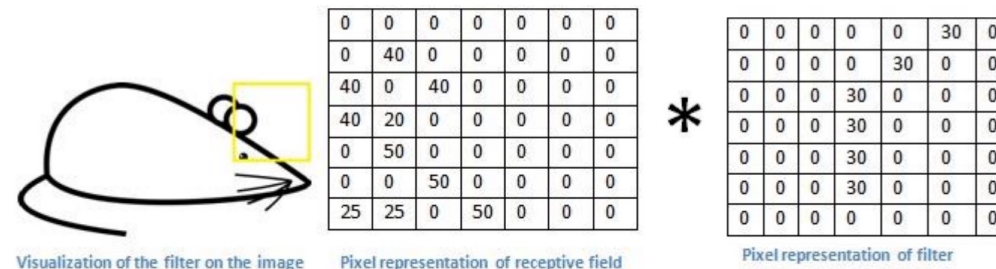
Interpretation of convolution (cont'd)

- When the filter moves to the back of the mouse, the convolution operation will generate a very large value



Multiplication and Summation = $(50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600$ (A large number!)

- Otherwise, it generates a very small value



Multiplication and Summation = 0

Layer normalization

- Maps a vector to a more normalized vector

- A sub-module of the layer normalization \longrightarrow LN-S(z) =

- $\hat{\mu} = \frac{\sum_{i=1}^m z_i}{m}$ is the empirical mean of the vector

- $\hat{\sigma} = \sqrt{\frac{\sum_{i=1}^m (z_i - \hat{\mu})^2}{m}}$ is the empirical standard deviation

- Intuition: normalized to having empirical mean zero and empirical standard deviation 1

$$\begin{bmatrix} \frac{z_1 - \hat{\mu}}{\hat{\sigma}} \\ \frac{z_2 - \hat{\mu}}{\hat{\sigma}} \\ \vdots \\ \frac{z_m - \hat{\mu}}{\hat{\sigma}} \end{bmatrix}$$

Layer normalization (cont'd)

- More general mean and variance

$$\text{LN}(z) = \beta + \gamma \cdot \text{LN-S}(z) = \begin{bmatrix} \beta + \gamma \left(\frac{z_1 - \hat{\mu}}{\hat{\sigma}} \right) \\ \beta + \gamma \left(\frac{z_2 - \hat{\mu}}{\hat{\sigma}} \right) \\ \vdots \\ \beta + \gamma \left(\frac{z_m - \hat{\mu}}{\hat{\sigma}} \right) \end{bmatrix}$$

- β, γ are learnable parameters
- Properties: Scaling-invariant

$$\text{LN}(\text{MM}_{\alpha W, \alpha b}(z)) = \text{LN}(\text{MM}_{W, b}(z)), \forall \alpha > 0.$$

- Applications

- Transformer / BERT / GPT / RL policy networks

Attention

- Intuition: the importance of different inputs depends on the content

The chicken didn't cross the road because it

What should be the properties of "it"?

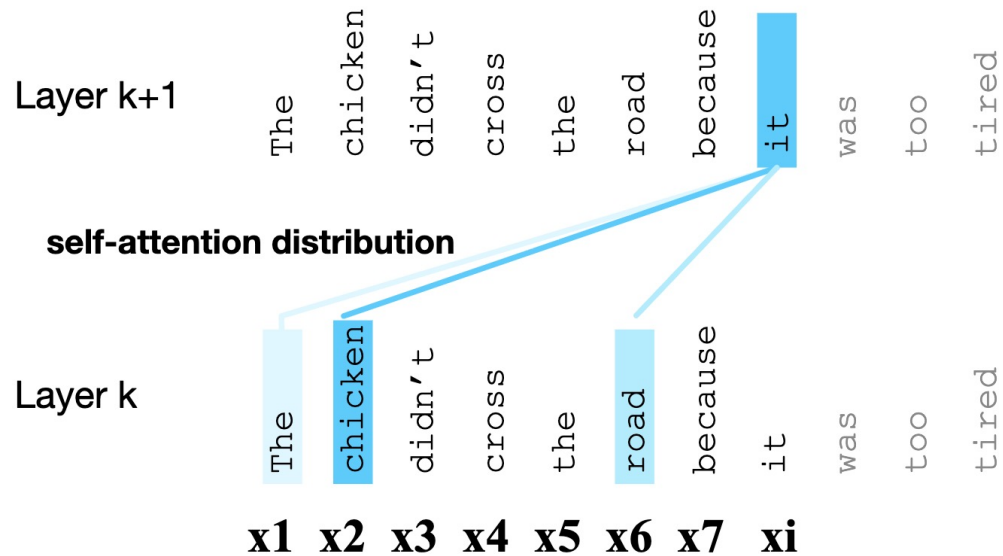
The chicken didn't cross the road because it was too **tired**

The chicken didn't cross the road because it was too **wide**

At this point in the sentence, it's probably referring to either the chicken or the street

- Attention: a mechanism that dynamically decides which inputs are important

Simplified version of attention



Given a sequence of token embeddings:

$$\mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_3 \quad \mathbf{x}_4 \quad \mathbf{x}_5 \quad \mathbf{x}_6 \quad \mathbf{x}_7 \quad \mathbf{x}_i$$

Produce: \mathbf{a}_i = a weighted sum of \mathbf{x}_1 through \mathbf{x}_7 (and \mathbf{x}_i)
Weighted by their similarity to \mathbf{x}_i

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$$

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \quad \forall j \leq i$$

$$\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j$$

An actual attention head

- High-level idea: instead of using vectors (like x_i and x_4) directly, we'll represent 3 separate roles each vector x_i plays:
 - query: what the current token is looking for
 - key: how this token can be matched
 - value: the information stored in the token
- use matrices to project each vector into a representation of its role as query, key, value
 - query: W^Q
 - key: W^K
 - value: W^V

$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^Q; \quad \mathbf{k}_i = \mathbf{x}_i \mathbf{W}^K; \quad \mathbf{v}_i = \mathbf{x}_i \mathbf{W}^V$$

Final equation for one attention head

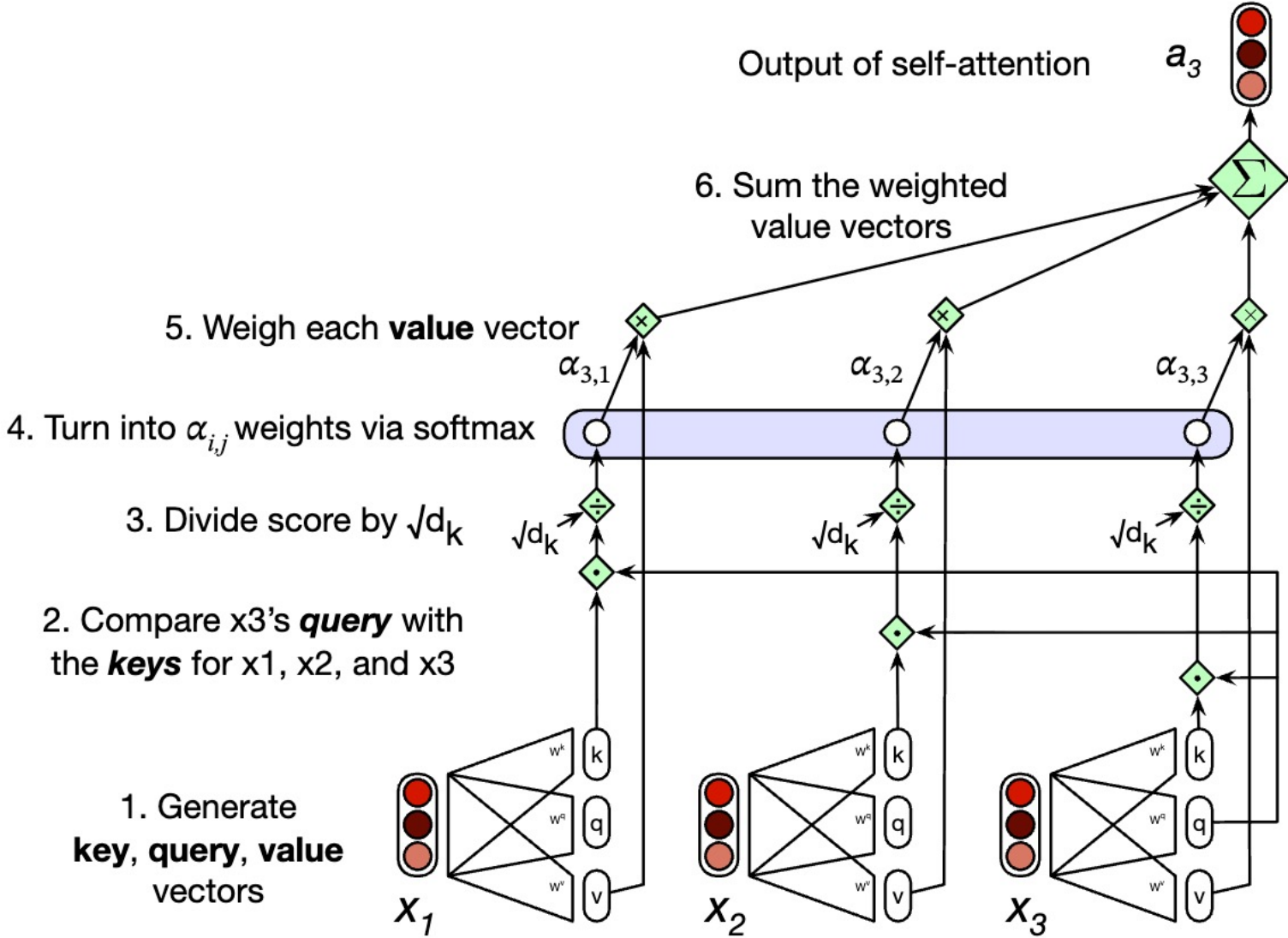
$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^Q; \quad \mathbf{k}_j = \mathbf{x}_j \mathbf{W}^K; \quad \mathbf{v}_j = \mathbf{x}_j \mathbf{W}^V$$

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$$

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \quad \forall j \leq i$$

$$\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j$$

Calculating the value of a_3



Multi-head attention

- Instead of one attention head, we'll have lots of them
- Intuition: each head might be attending to the context for different purposes
 - Different linguistic relationships or patterns in the context

$$\mathbf{q}_i^c = \mathbf{x}_i \mathbf{W}^{\mathbf{Q}c}; \quad \mathbf{k}_j^c = \mathbf{x}_j \mathbf{W}^{\mathbf{K}c}; \quad \mathbf{v}_j^c = \mathbf{x}_j \mathbf{W}^{\mathbf{V}c}; \quad \forall c \quad 1 \leq c \leq h$$

$$\text{score}^c(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i^c \cdot \mathbf{k}_j^c}{\sqrt{d_k}}$$

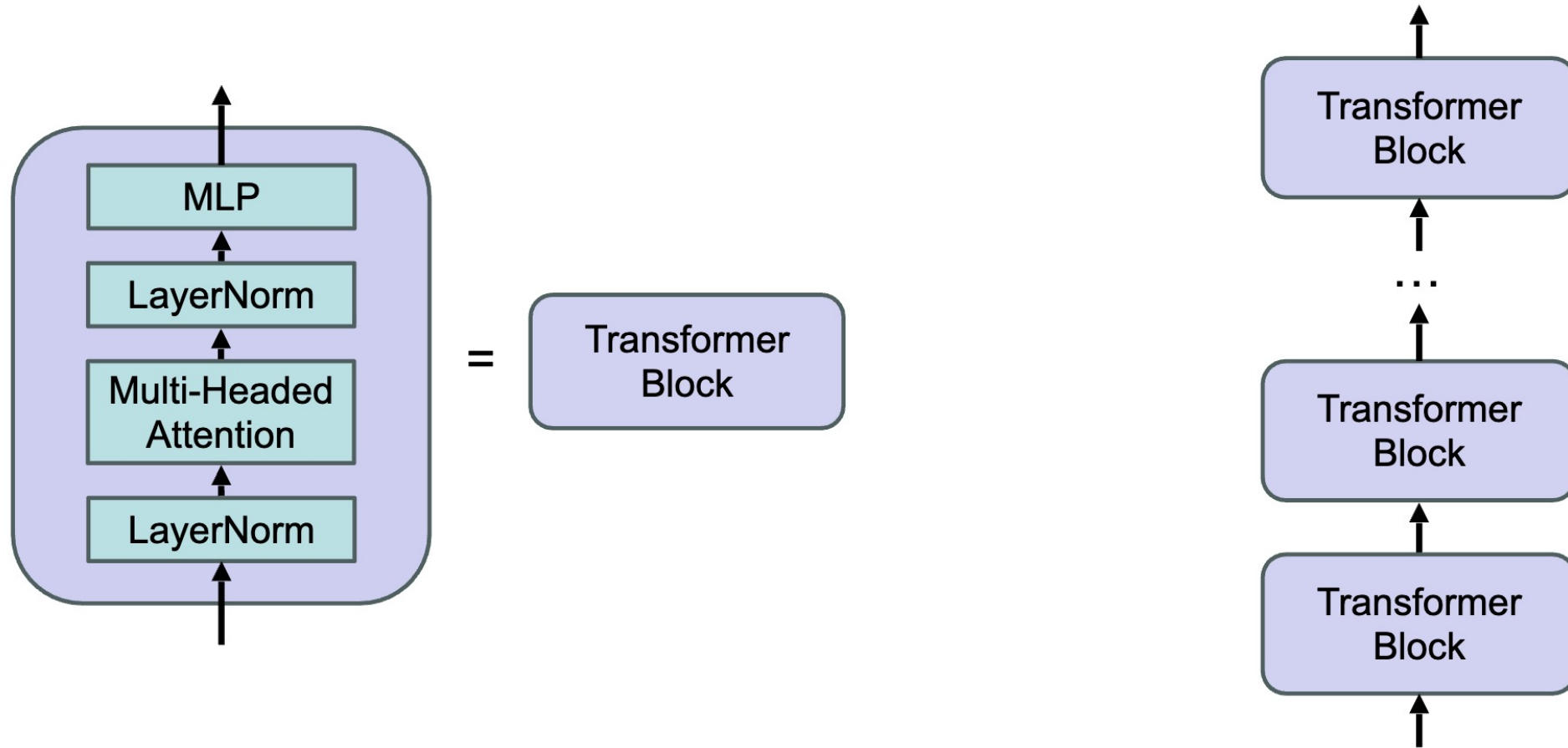
$$\alpha_{ij}^c = \text{softmax}(\text{score}^c(\mathbf{x}_i, \mathbf{x}_j)) \quad \forall j \leq i$$

$$\text{head}_i^c = \sum_{j \leq i} \alpha_{ij}^c \mathbf{v}_j^c$$

$$\mathbf{a}_i = (\text{head}^1 \oplus \text{head}^2 \dots \oplus \text{head}^h) \mathbf{W}^O$$

$$\text{MultiHeadAttention}(\mathbf{x}_i, [\mathbf{x}_1, \dots, \mathbf{x}_N]) = \mathbf{a}_i$$

Transformer Architecture



Summary

- History of artificial neural nets
- Perceptron
- Multilayer perceptron networks
- Activation functions
- Training: backpropagation
- Modules in modern neural networks