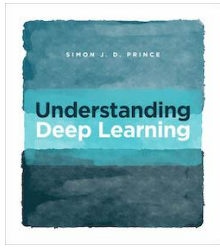
深度學習實作與應用
Deep learning and its applications

2. Basic Neural Networks:
From regression to neural networks (2/3)

IM5062, Spring 2024



CH3, 4, 5

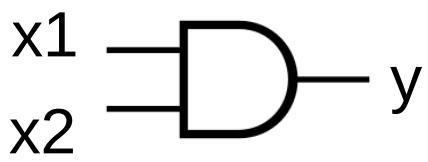
黃意婷

Outline

- **Perceptron**
- **Neural Network (**Multiple Layer perceptron)
  - Shallow Neural Networks (with a single hidden layer)
    - Neural units
    - Activation functions
    - Hidden units
    - Arbitrary inputs, hidden units, outputs
  - Deep Neural Networks (with multiple hidden layers)
- **Loss functions**

Can neural units compute simple functions of input?

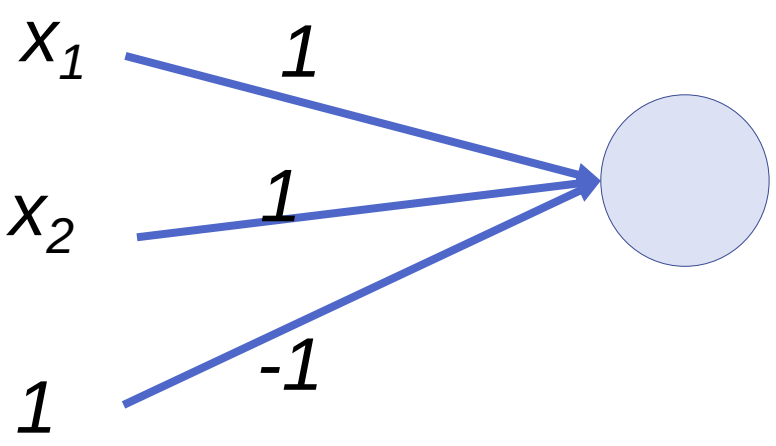
AND		
x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1



OR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1

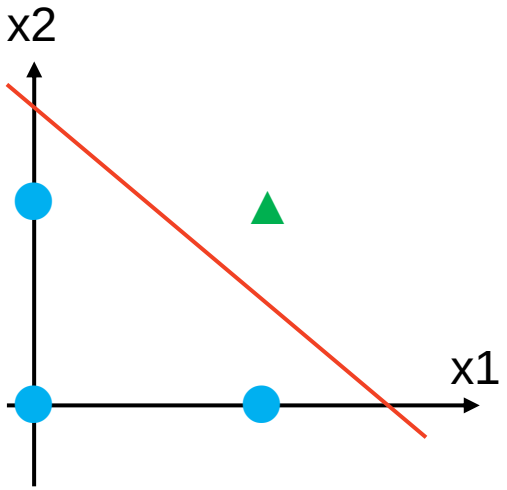
XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

Easy to build AND with perceptrons

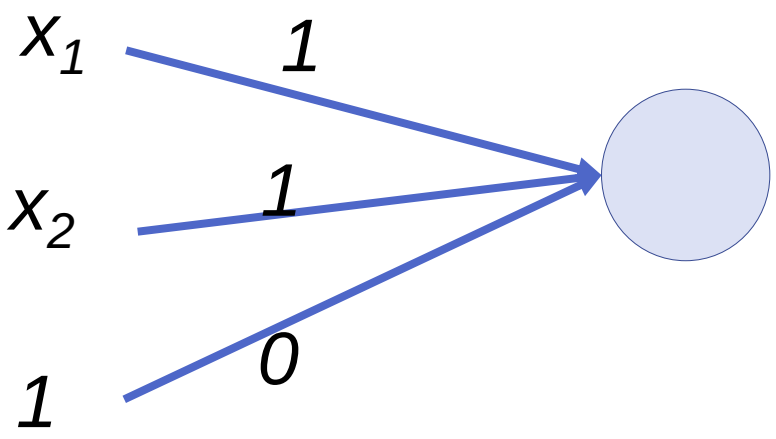


$$y = \begin{cases} 0 & if\ w \cdot x + b \leq 0 \\ 1 & if\ w \cdot x + b > 0 \end{cases}$$

AND						
w1	* x1	+ w2	* x2	+ b	= z	= y
1	0	1	0	-1	-1	0
1	0	1	1	-1	0	0
1	1	1	0	-1	0	0
1	1	1	1	-1	1	1

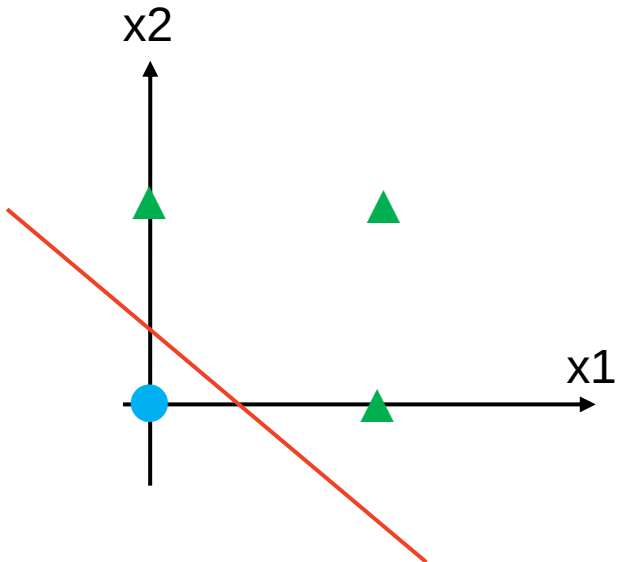


Easy to build OR with perceptrons



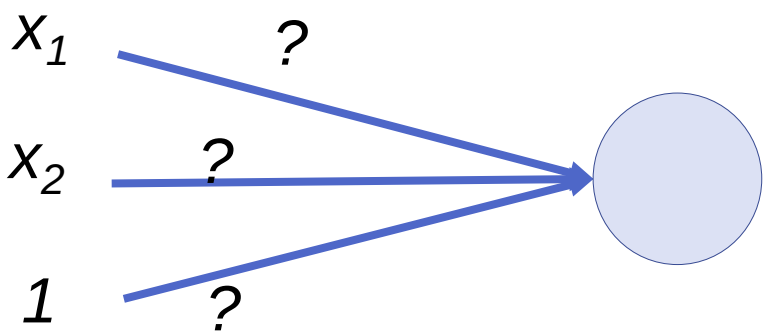
$$y = \begin{cases} 0 & if\ w \cdot x + b \leq 0 \\ 1 & if\ w \cdot x + b > 0 \end{cases}$$

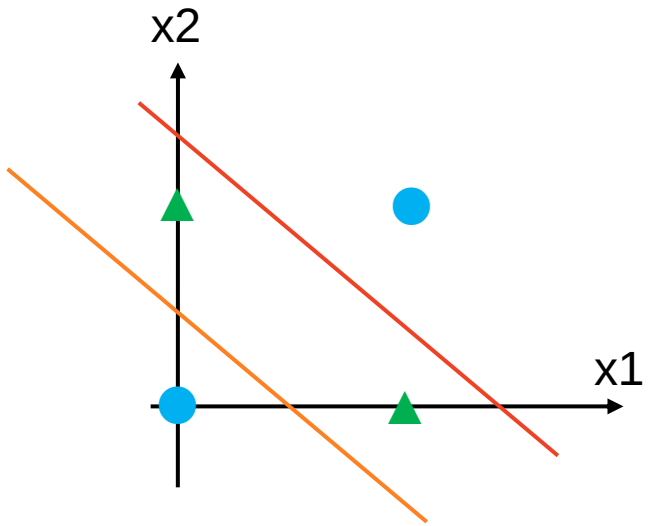
OR						
w1	* x1	+ w2	* x2	+ b	= z	= y
1	0	1	0	0	0	0
1	0	1	1	0	1	1
1	1	1	0	0	1	1
1	1	1	1	0	1	1



How about XOR with perceptrons?

- Try it for yourself!





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

XOR						
w1	* x1	+ w2	* x2	+ b	= z	= y
	0		0			0
	0		1			1
	1		0			1
	1		1			0

XOR is not a linearly separable function!

Why? Perceptrons are linear classifiers

- Perceptron equation given x1 and x2, is the equation of a line $w_1 x_1 + w_2 x_2 + b = 0$

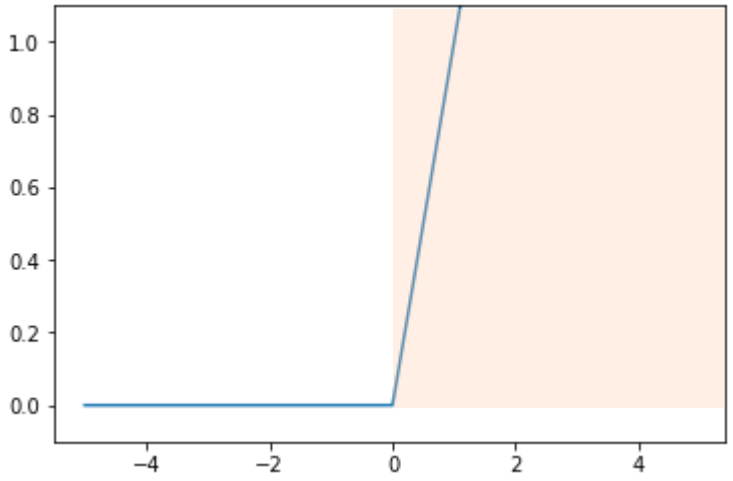
- This line acts as a **decision boundary**
  - 0 if input is one on side of the line
  - 1 if on the other side of the line

Solution to the XOR problem

- XOR **can't** be calculated by a single perceptron.
- XOR **can** be calculated a layered network of units.
  - + a layer
  - + activation function (ReLU)

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

$$ReLU(x) = \begin{cases} x & if\ x > 0 \\ 0 & if\ x \leq 0 \end{cases}$$

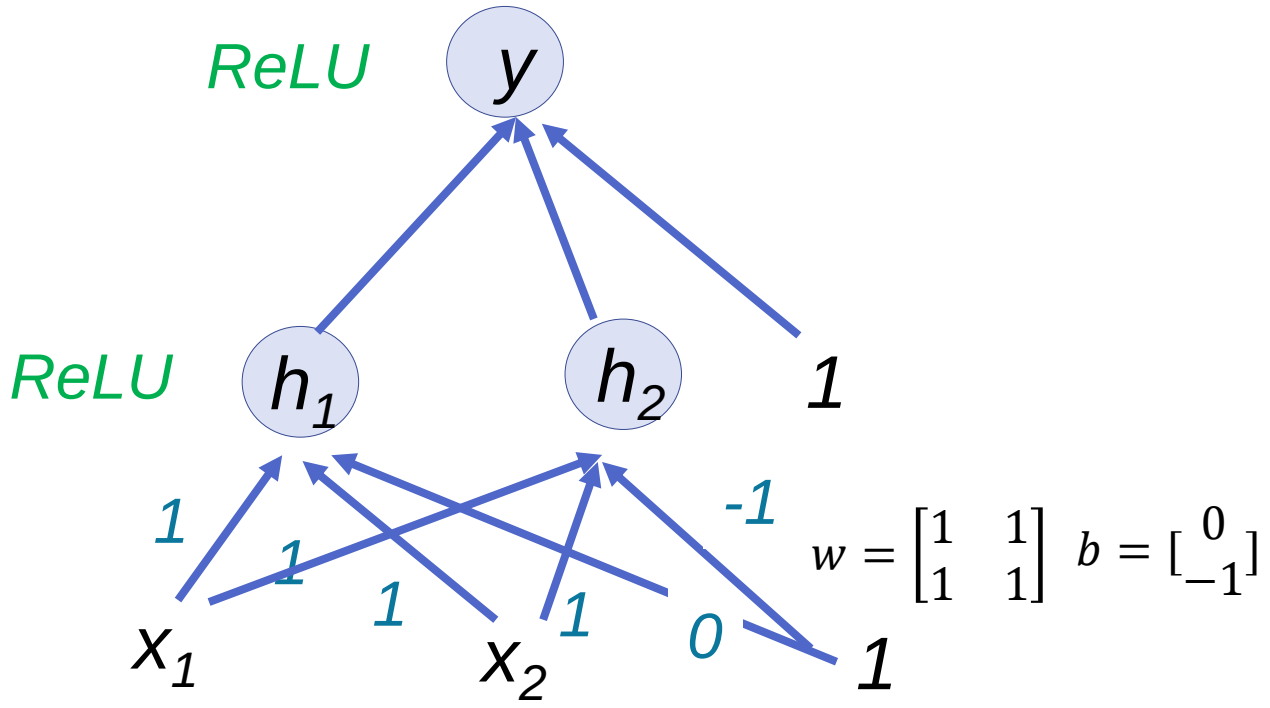


Solution to the XOR problem

- XOR **can't** be calculated by a single perceptron.
- XOR **can** be calculated a layered network of units.

XOR				
x1	x2	h1	h2	y
0	0	0	0	
0	1			
1	0			
1	1			

$$h_1 = 0 \times 1 + 0 \times 1 + 0 = 0$$
$$h_2 = 0 \times 1 + 0 \times 1 - 1 = -1$$

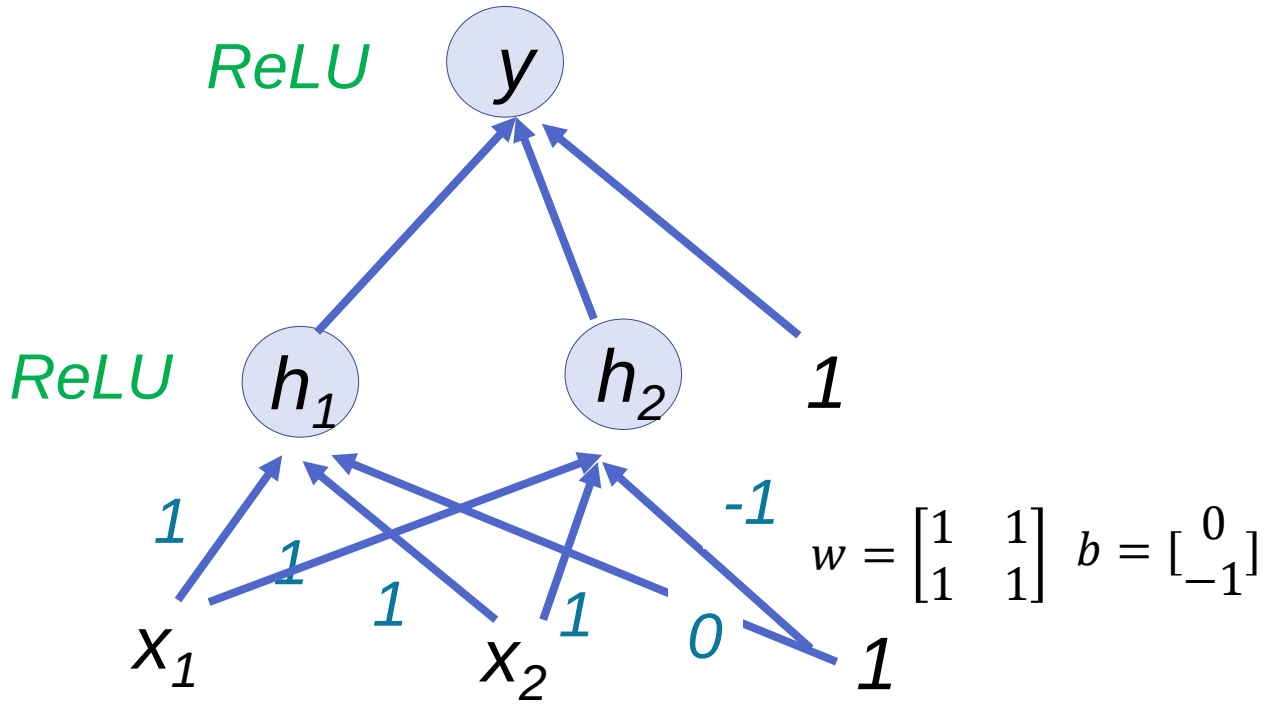


Solution to the XOR problem

- XOR **can't** be calculated by a single perceptron.
- XOR **can** be calculated a layered network of units.

XOR				
x1	x2	h1	h2	y
0	0	0	0	
0	1	1	0	
1	0	1	0	
1	1	2	1	

$h_1 = 0 \times 1 + 0 \times 1 + 0 = 0$
$h_2 = 0 \times 1 + 0 \times 1 - 1 = -1$

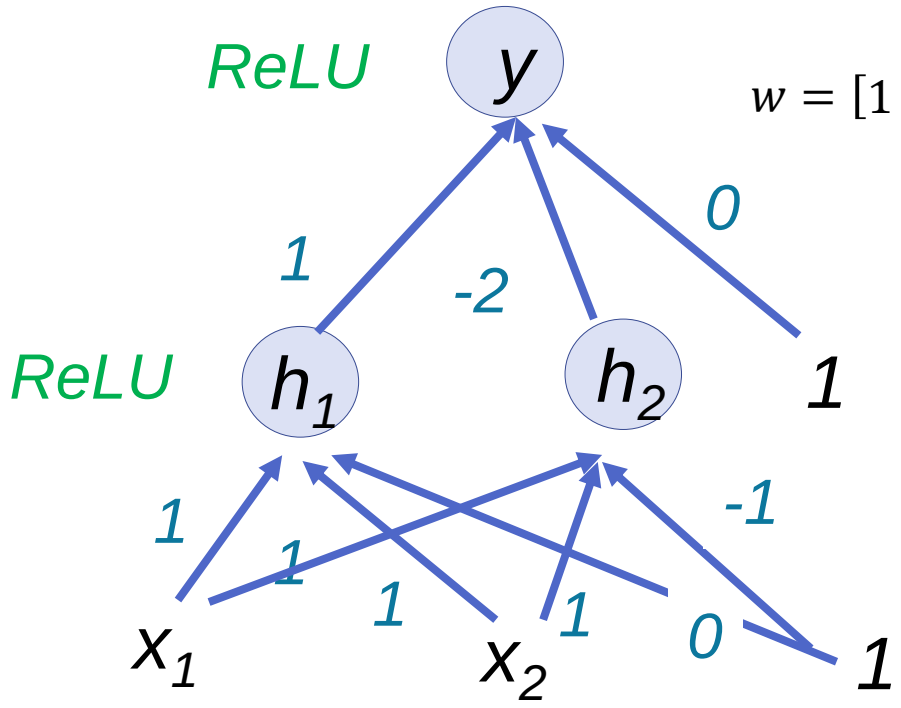


Solution to the XOR problem

- XOR **can't** be calculated by a single perceptron.
- XOR **can** be calculated a layered network of units.

XOR				
x1	x2	h1	h2	y
0	0	0	0	
0	1	1	0	
1	0	1	0	1
1	1	2	1	0

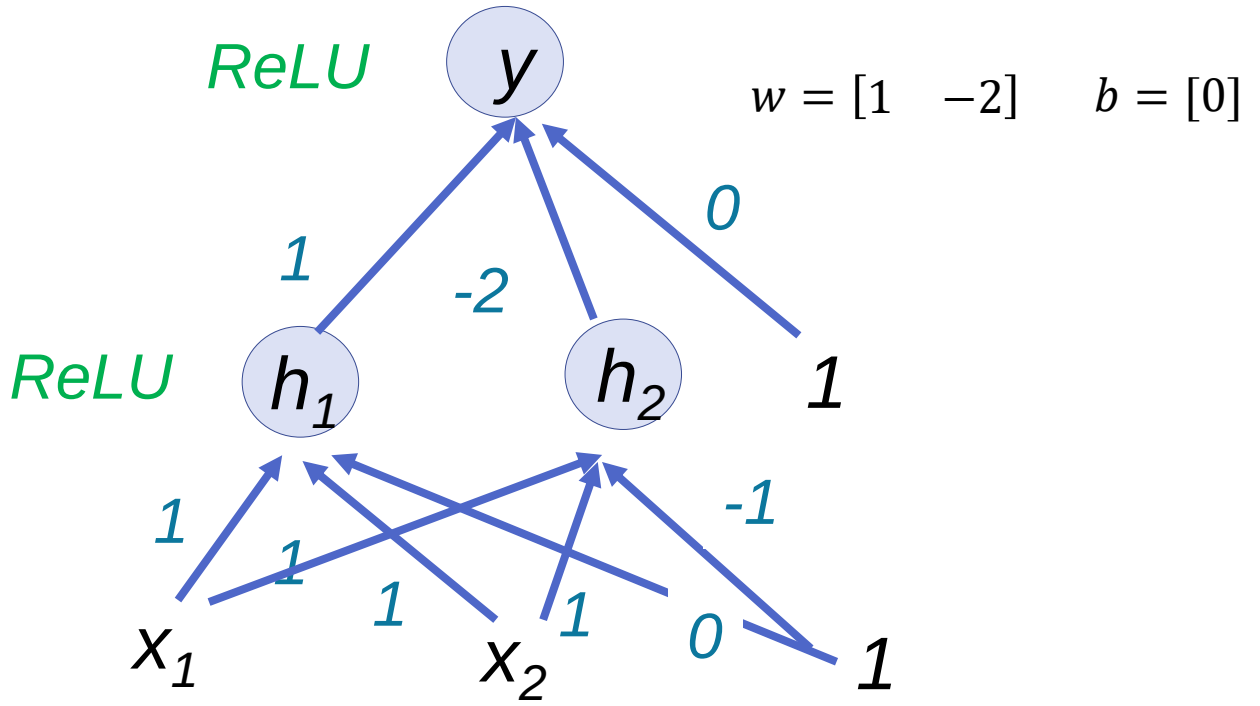
$$y = 1 \times 1 + 0 \times (-2) + 0 = 1$$
$$y = 2 \times 1 + 1 \times (-2) + 0 = 0$$



Solution to the XOR problem

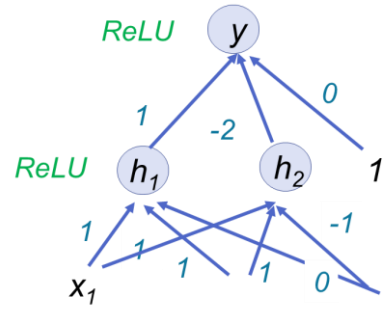
- XOR **can't** be calculated by a single perceptron.
- XOR **can** be calculated a layered network of units.

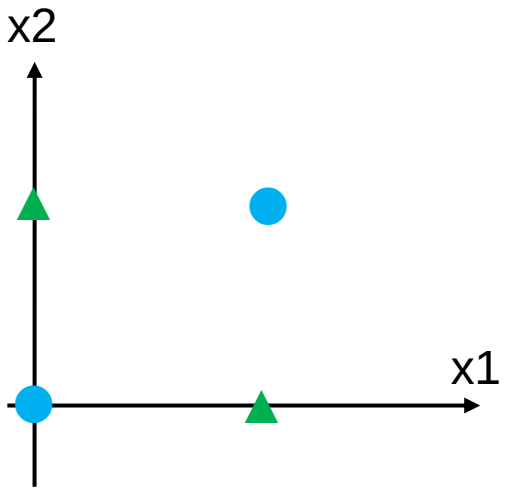
XOR				
x1	x2	h1	h2	y
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	2	1	0



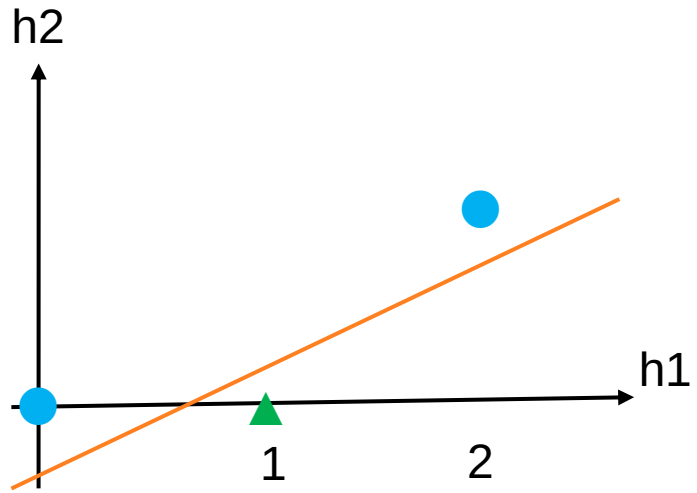
The hidden representation *h*

XOR				
x1	x2	h1	h2	y
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	2	1	0





The original x space



The new (linearly separable) *h* space

- With learning: hidden layers will learn to form useful representations.

Outline

- **Perceptron**
- **Neural Network** (Multiple Layer perceptron)
  - Shallow Neural Networks (with a single hidden layer)
    - Neural units
    - Activation functions
    - Hidden units
    - Arbitrary inputs, hidden units, outputs
  - Deep Neural Networks (with multiple hidden layers)
- **Loss functions**

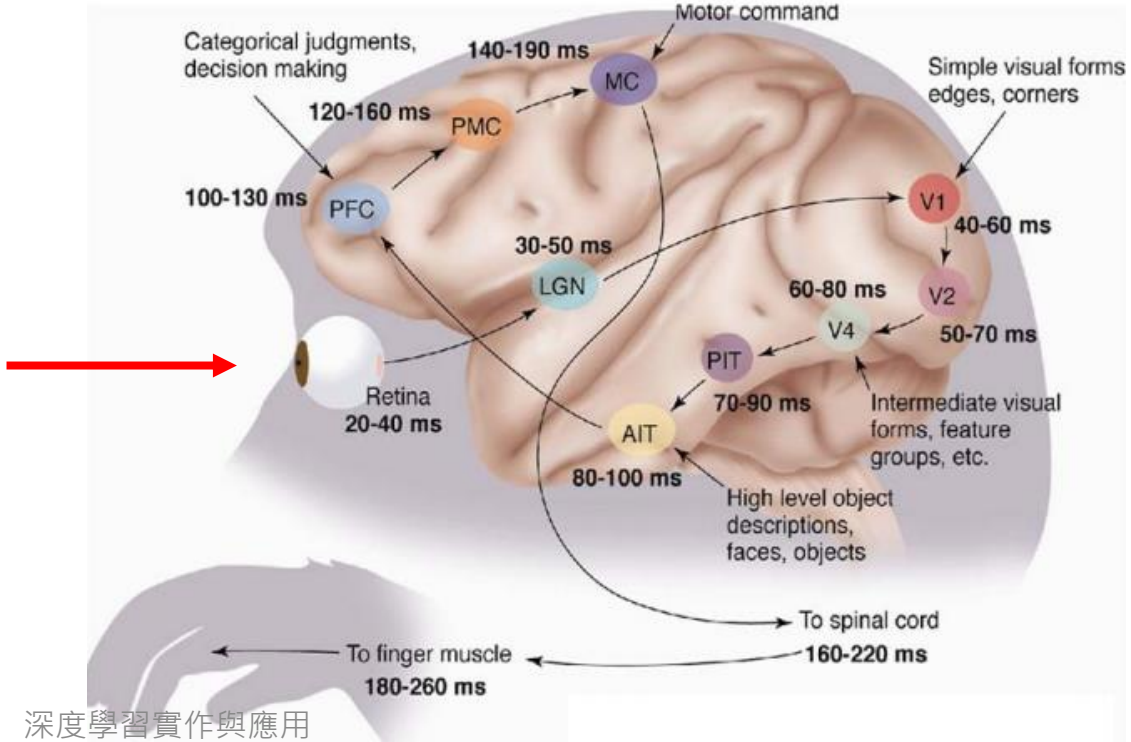
為啥先介紹神經元?

- 從上個世紀晚期以來，在神經網絡發展過程中起到重要作用的研究大多數都不是在數學和物理實驗室中完成的，而要
  歸功於心理學和神經生理學學科的研究者們。

認知心理學

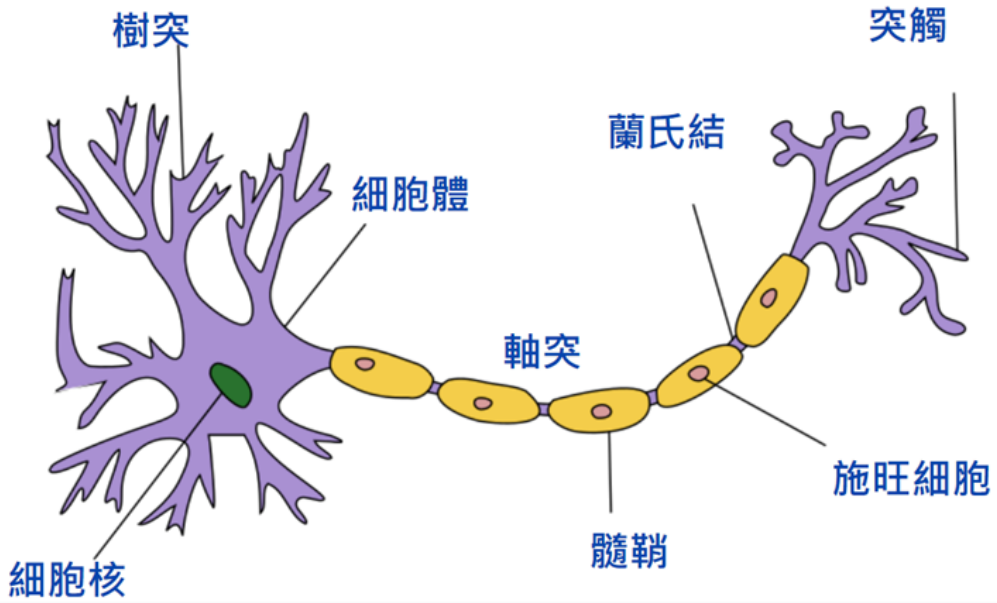
- 認知心理學（英語：Cognitive psychology）是對諸如**注意力**、語言使用、**記憶**、感知、問題解決、創造力和思考等心理過程的科學研究。
- 認知心理取向的重點便在認知的訊息處理模式 - - 一種以心智處理來思考與推理的模式。因此，思考與推理在人類大腦中的運作便像電腦軟體在電腦裡運作相似。認知心理學理論時常談到輸入、表徵、計算或處理，以及輸出等概念。

- Inspiration from visual cortex



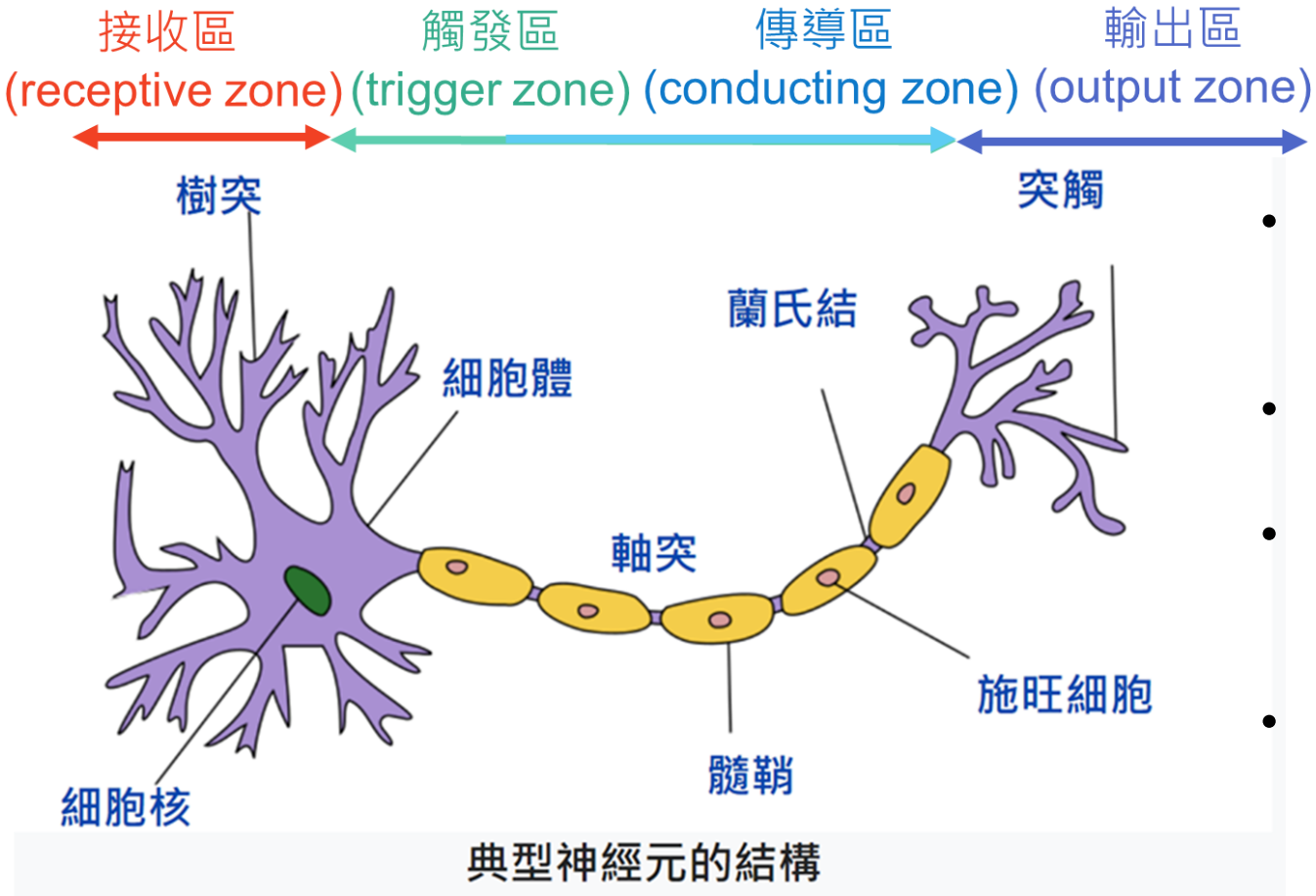
神經元

- 神經元主要的作用：**感知**環境的變化，再將訊息**傳遞**給其他的神經元，來進行作用。
- 神經的構造由突觸、樹突、軸突、細胞核等等所組成。



典型神經元的結構

Source: Wikipedia



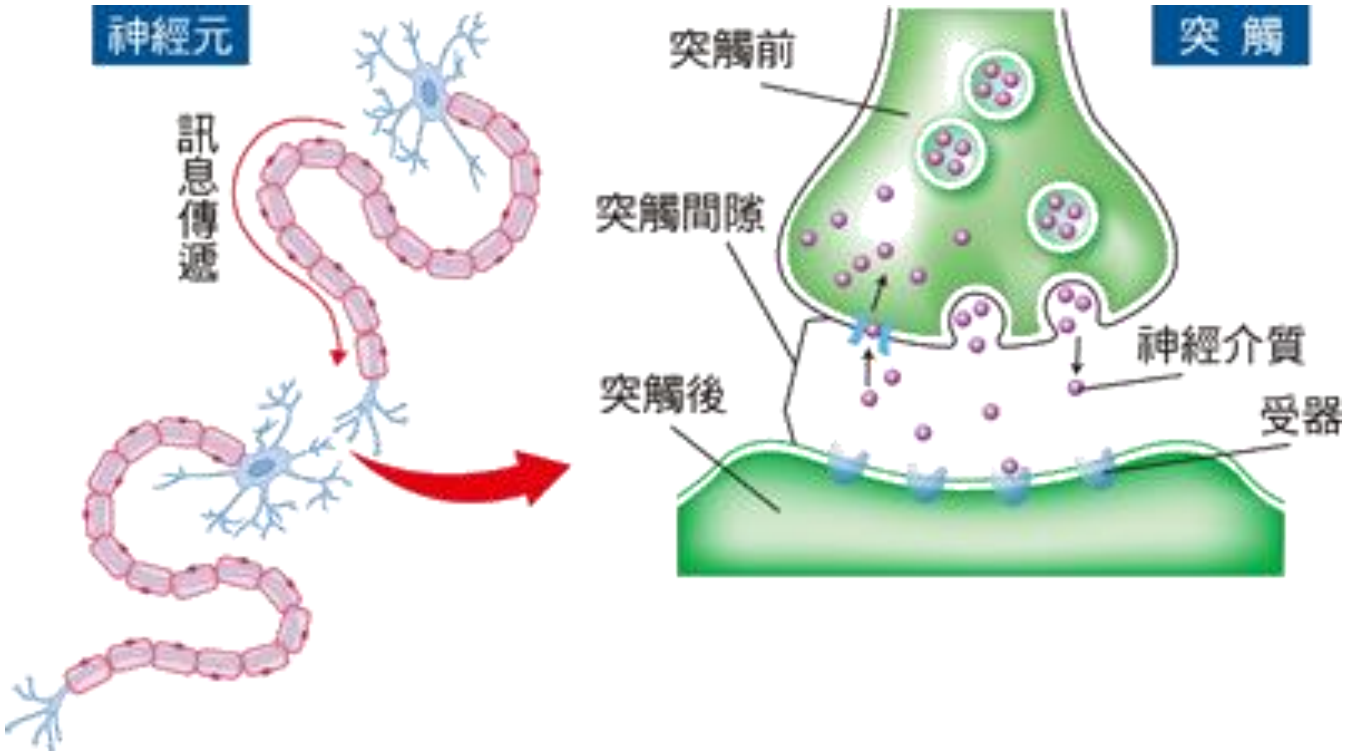
典型神經元的結構

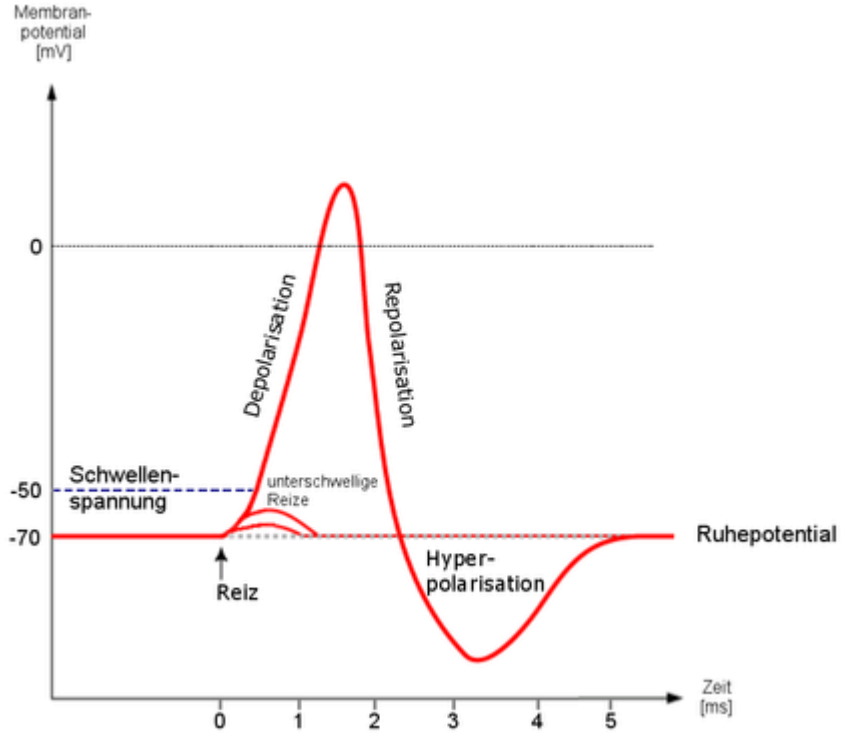
- 接收區（receptive zone）：指的是樹突接受到其他來源的突觸，如果接收的來源越多，對胞體膜電位的影響越大。
- 觸發區（trigger zone）：在胞體整合的電位，決定是否產生神經衝動的起始點。
- 傳導區（conducting zone）：是指軸突的部份，當產生動作電位（action potential）時，傳導區能遵守全有全無的定律來傳導神經衝動。
- 輸出區（output zone）：突觸的神經傳遞物質或電力釋出，影響下一個接受的細胞（神經元、肌肉細胞或是腺體細胞），此稱為突觸傳遞。

Source: Wikipedia

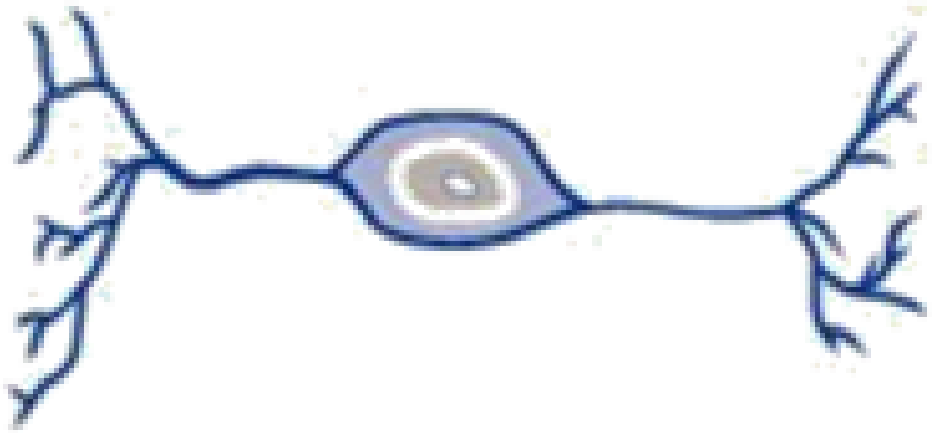
神經傳導

在傳遞的時候，會形成電流，從某一端傳到另外一端在適當的量傳遞後，兩個突觸間會形成電流傳導。

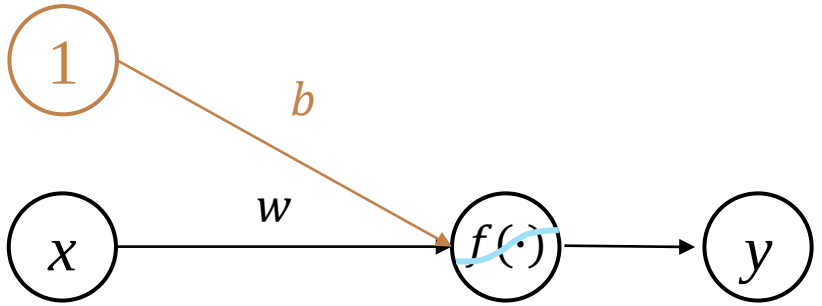




數學模擬

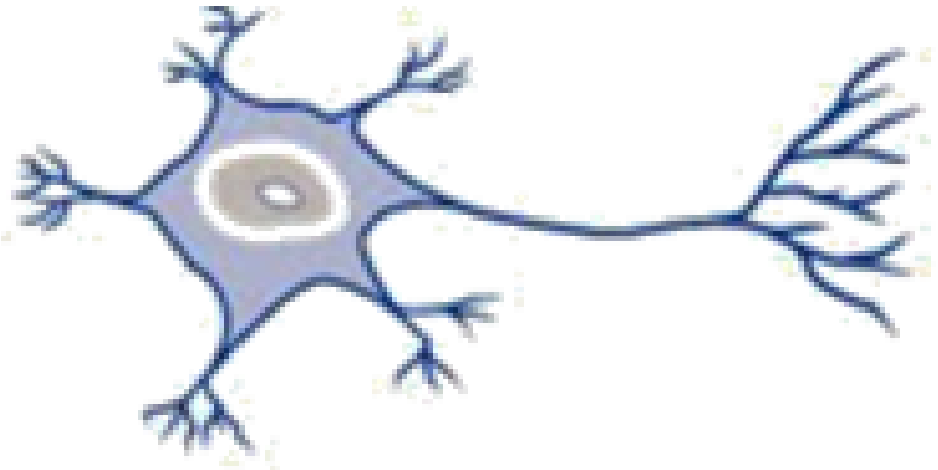


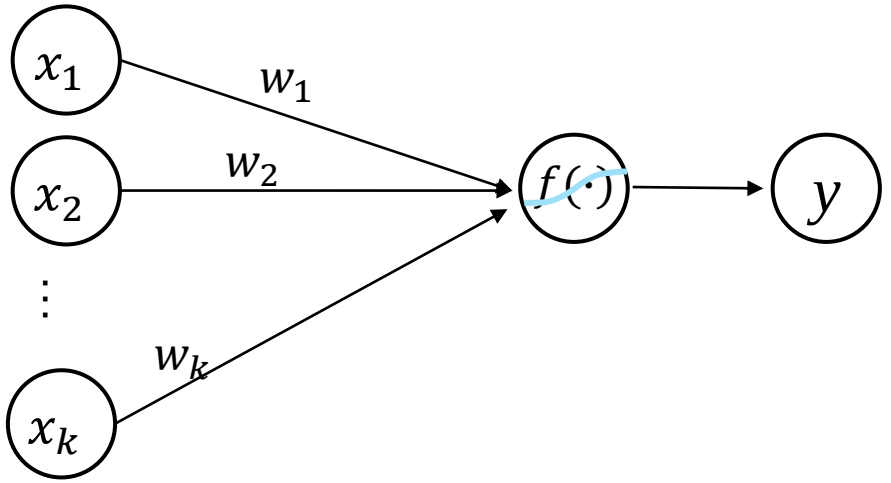
$$\mathbf{y} = f(\mathbf{wx} + \mathbf{b})$$



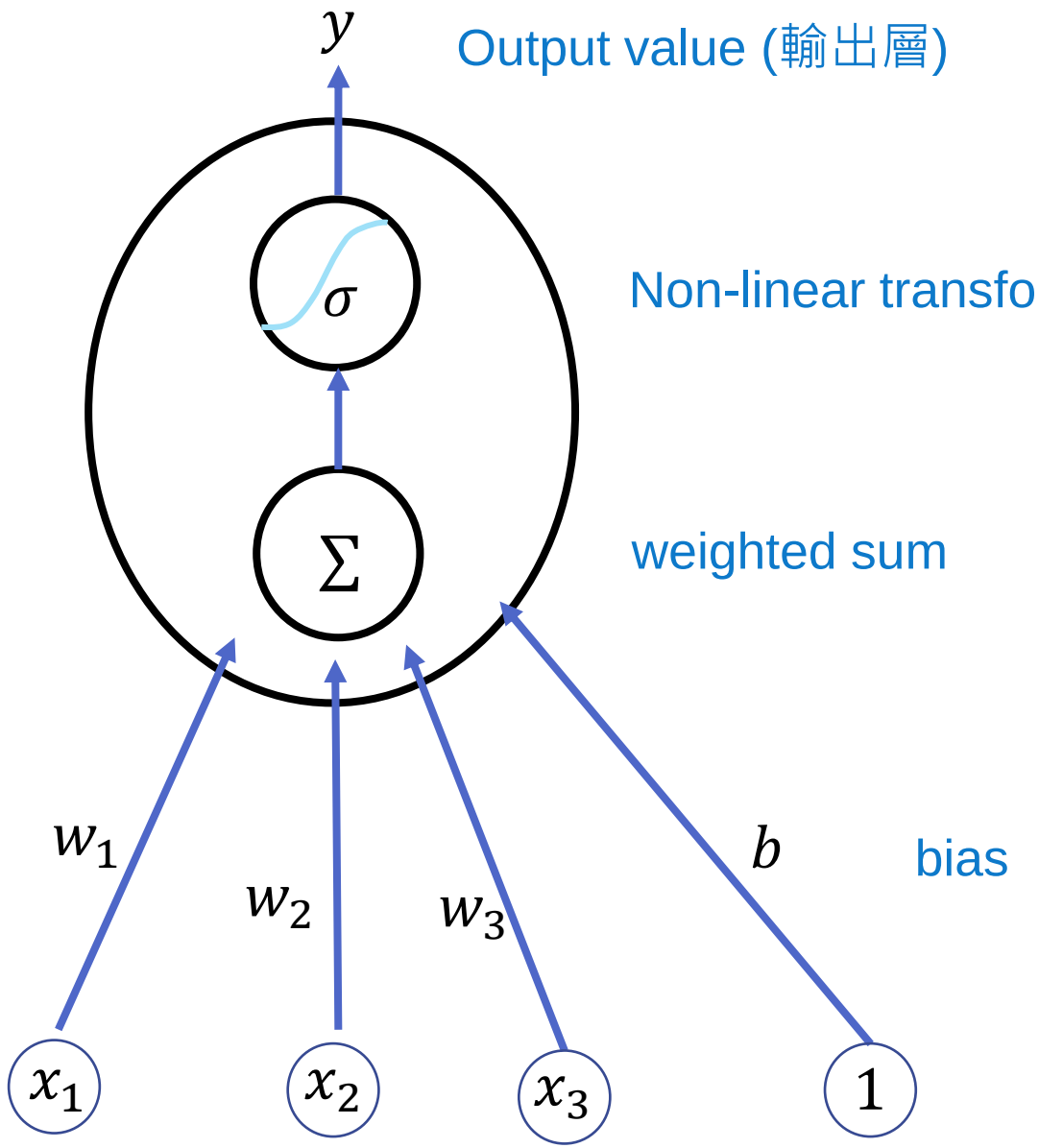
數學模擬

$$\mathbf{y} = f(\mathbf{w}_1\mathbf{x}_1, \mathbf{w}_2\mathbf{x}_2, \ldots, \mathbf{w}_k\mathbf{x}_k)$$





Neural Unit


y
Output value (輸出層)
σ
Non-linear transform
Σ
weighted sum
w_1
w_2
w_3
b
bias
Weights (觸發區)
x_1
x_2
x_3
1
Input layer (接收區)

Neural Unit

- Take weighted sum of inputs, plus a bias

$$z = b + \sum_i w_i x_i$$

$$z = w \cdot x + b$$

- Instead of just using z, we will apply a **nonlinear activation function** σ:

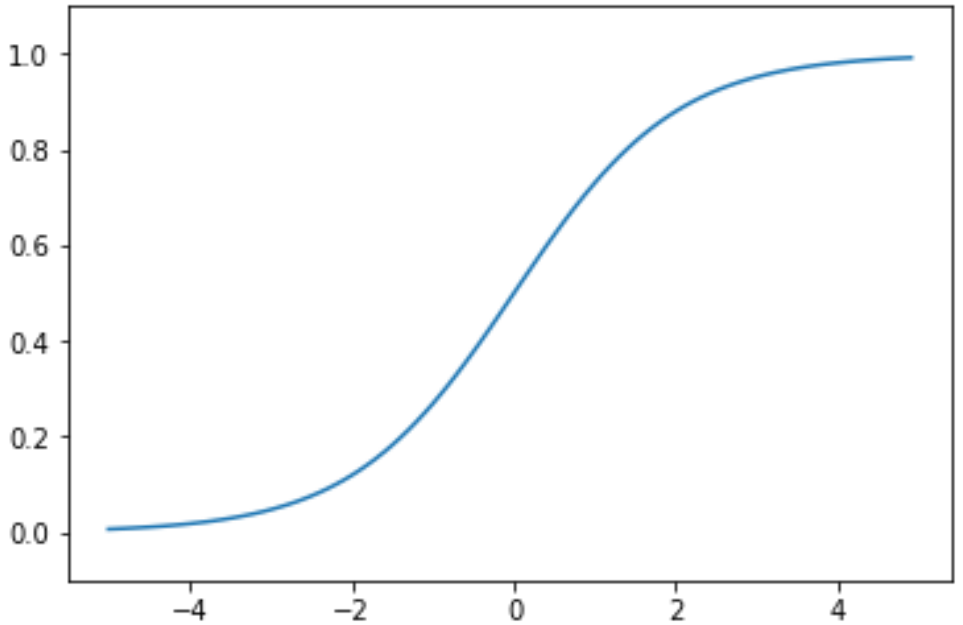
$$y = \sigma(z)$$

Non-Linear Activation Functions

- We're already seen the sigmoid for logistic regression:

Sigmoid activation:
Map inputs into (0, 1)

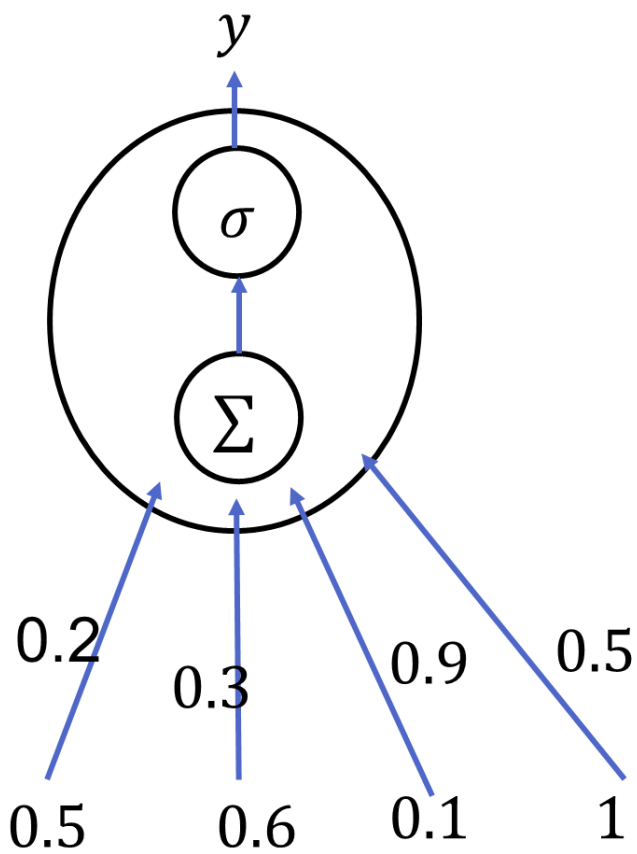
$$y = \sigma(z) = \frac{1}{1 + \exp(-z)}$$



An example

- Suppose a unit has:
  - w = [0.2, 0.3, 0.9]
  - b = 0.5

- What happens with input x:
  - x = [0.5, 0.6, 0.1]



$$y = \sigma(w \cdot x + b) = \frac{1}{1+\exp(-(w \cdot x + b))} = \frac{1}{1+\exp(-(.5*2+.6*.3+.1*.9+.5))}$$

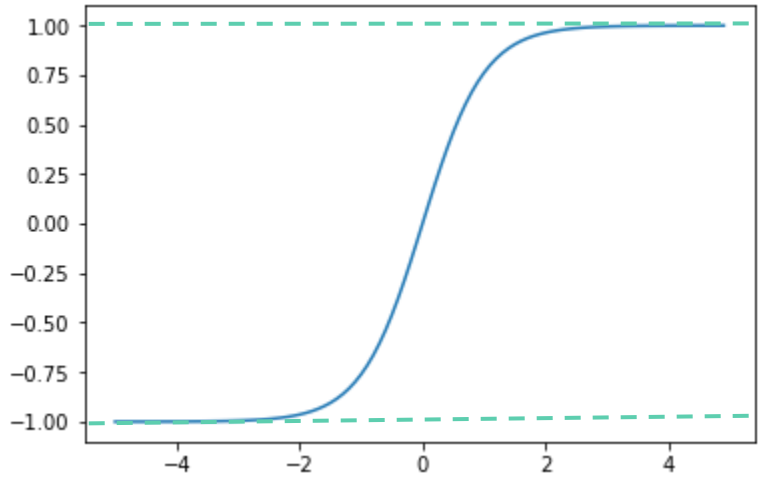
$$= \frac{1}{1+\exp(-.87)} = .70$$

Outline

- **Perceptron**
- **Neural Network (**Multiple Layer perceptron)
  - Shallow Neural Networks (with a single hidden layer)
    - Neural units
    - Activation functions
    - Hidden units
    - Arbitrary inputs, hidden units, outputs
  - Deep Neural Networks (with multiple hidden layers)
- **Loss functions**

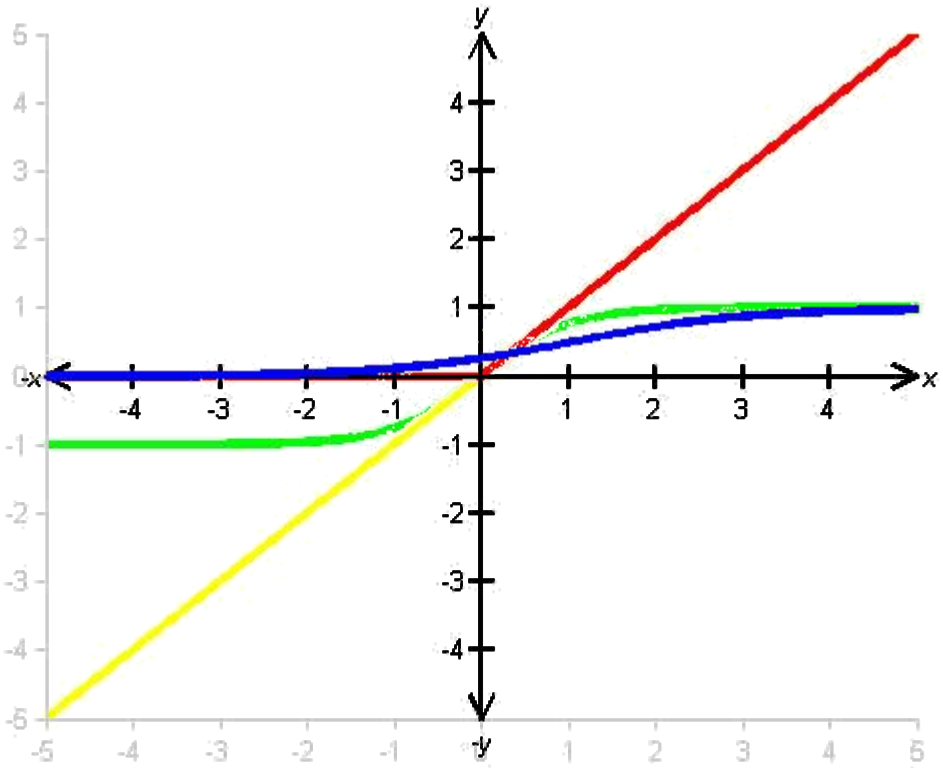
Activation functions

- Linear: $f(x) = x$
- ReLU: $f(x) = \max(x, 0)$
- Sigmoid: $f(x) = 1/(1 + e^{-x})$
- tanh: $f(x) = (e^{x} - e^{-x})/(e^{x} + e^{-x})$
  - Tanh (Hyperbolic Tangent) : Map inputs into (-1,1)



Activation functions

- Linear
- Rectified Linear Unit (ReLU)
- Sigmoid
- Hyper Tangent (tanh)



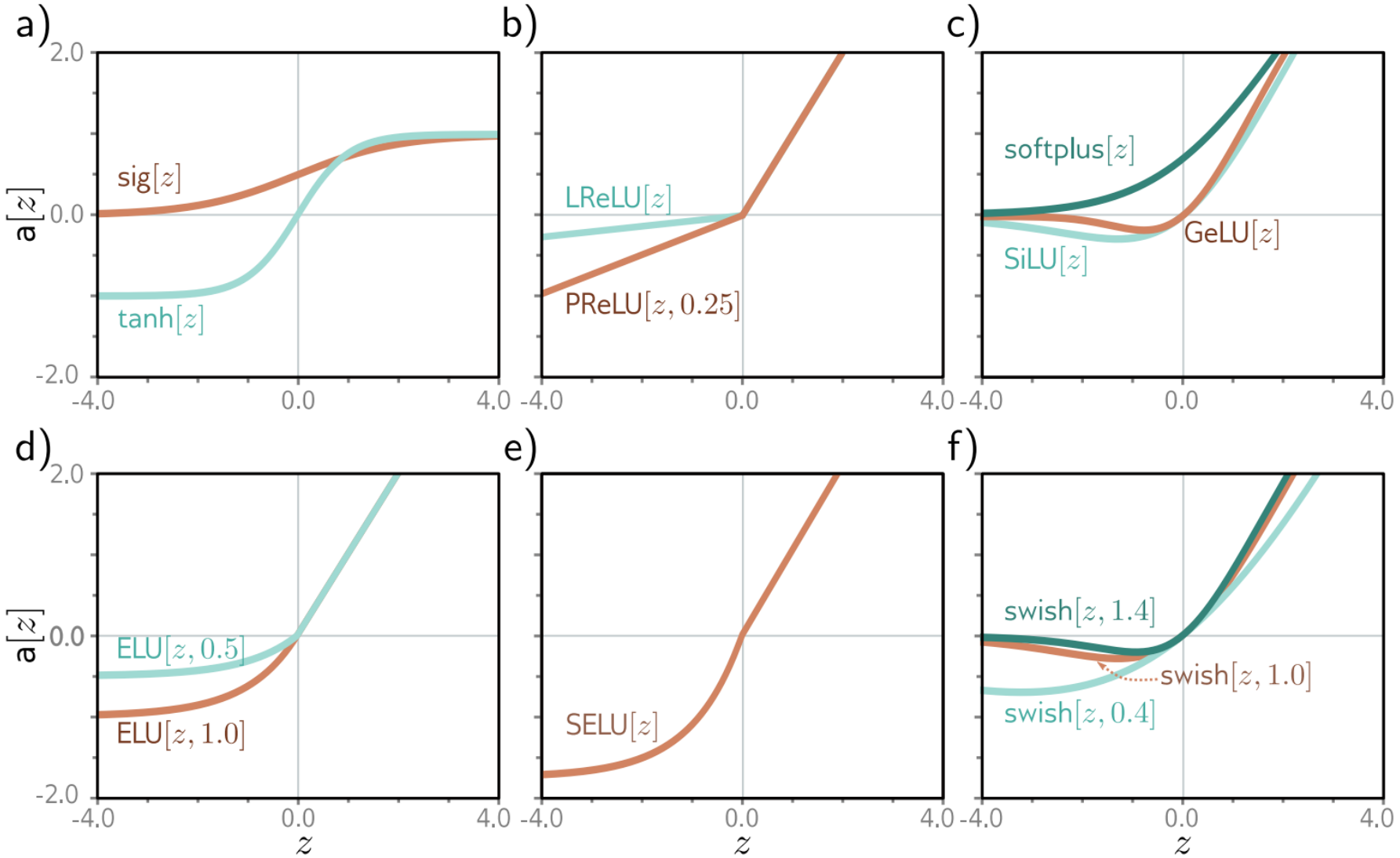
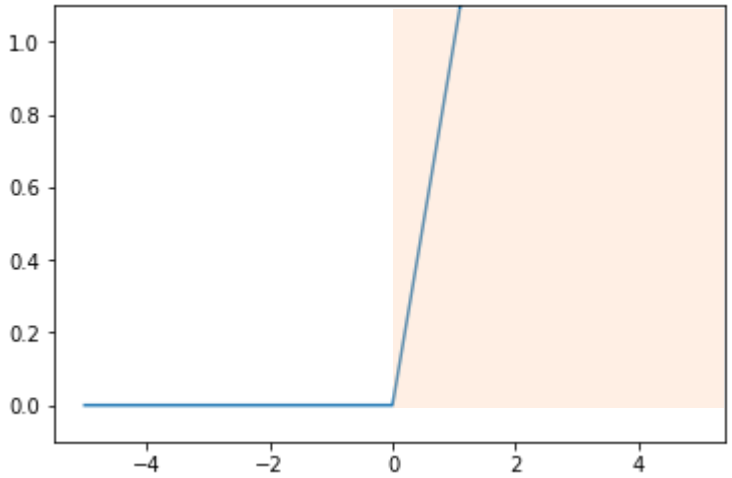


Figure 3.13 Activation functions. a) Logistic sigmoid and tanh functions. b) Leaky ReLU and parametric ReLU with parameter 0.25. c) SoftPlus, Gaussian error linear unit, and sigmoid linear unit. d) Exponential linear unit with parameters 0.5 and 1.0, e) Scaled exponential linear unit. f) Swish with parameters 0.4, 1.0, and 1.4.

$ReLU$

- ReLU is an important part of the success story of modern neural networks.
  - The derivative of the output with respect to the input is always one for inputs greater than zero.
  - This contributes to the stability and effciency of training and contrasts with the derivatives of sigmoid activation functions, which saturate (become close to zero) for large positive and large negative inputs.

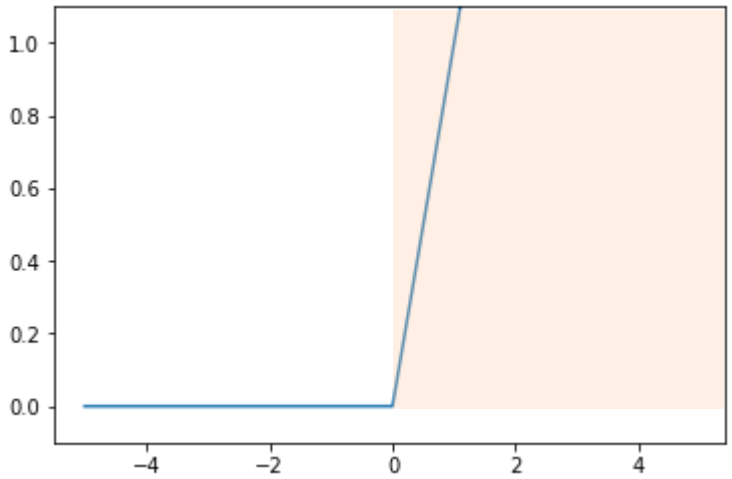


$$ReLU(x) = \max(x, 0)$$

$$ReLU(x) = \begin{cases} x & if\ x > 0 \\ 0 & if\ x \leq 0 \end{cases}$$

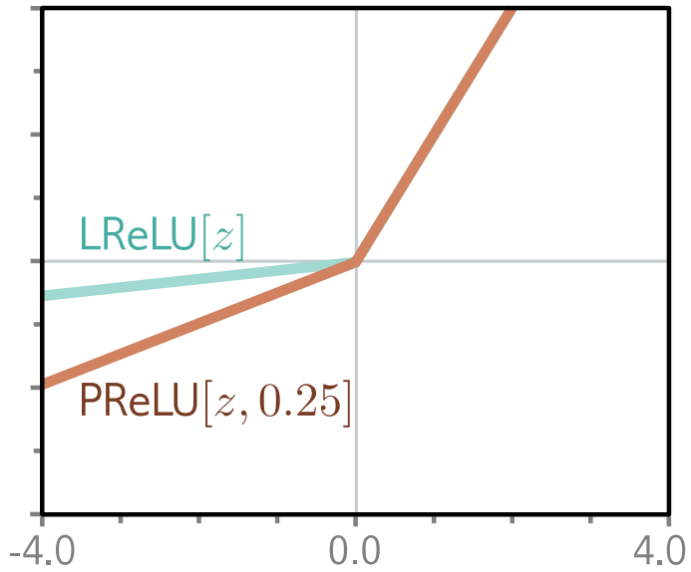
Dying ReLU problem

- ReLU function has the disadvantage that its derivative is zero for negative inputs.
- If all the training examples produce negative inputs to a given ReLU function, then we cannot improve the parameters feeding into this ReLU during training.
  - The gradient with respect to the incoming weights is locally flat, so we cannot “walk downhill.”



Variations on the ReLU

- Leaky ReLU:
  - a linear output for negative values with a smaller slope of 0.1
- Parametric ReLU:
  - treats the slope of the negative portion as an unknown parameter



Why do we need an a nonlinear activation?

- Activation functions (f) decide whether a neuron should be activated or not.
- The purpose is to introduce ***non-linearity*** into the network.

- 2 layer linear functions:
  - $\boldsymbol{h} = W_1\boldsymbol{x} + \boldsymbol{b}_1$
  - $y = w_2^T\boldsymbol{h} + b_2 = w_2^T W_1 x + w_2 b_1 + b_2 = W' x + b'$

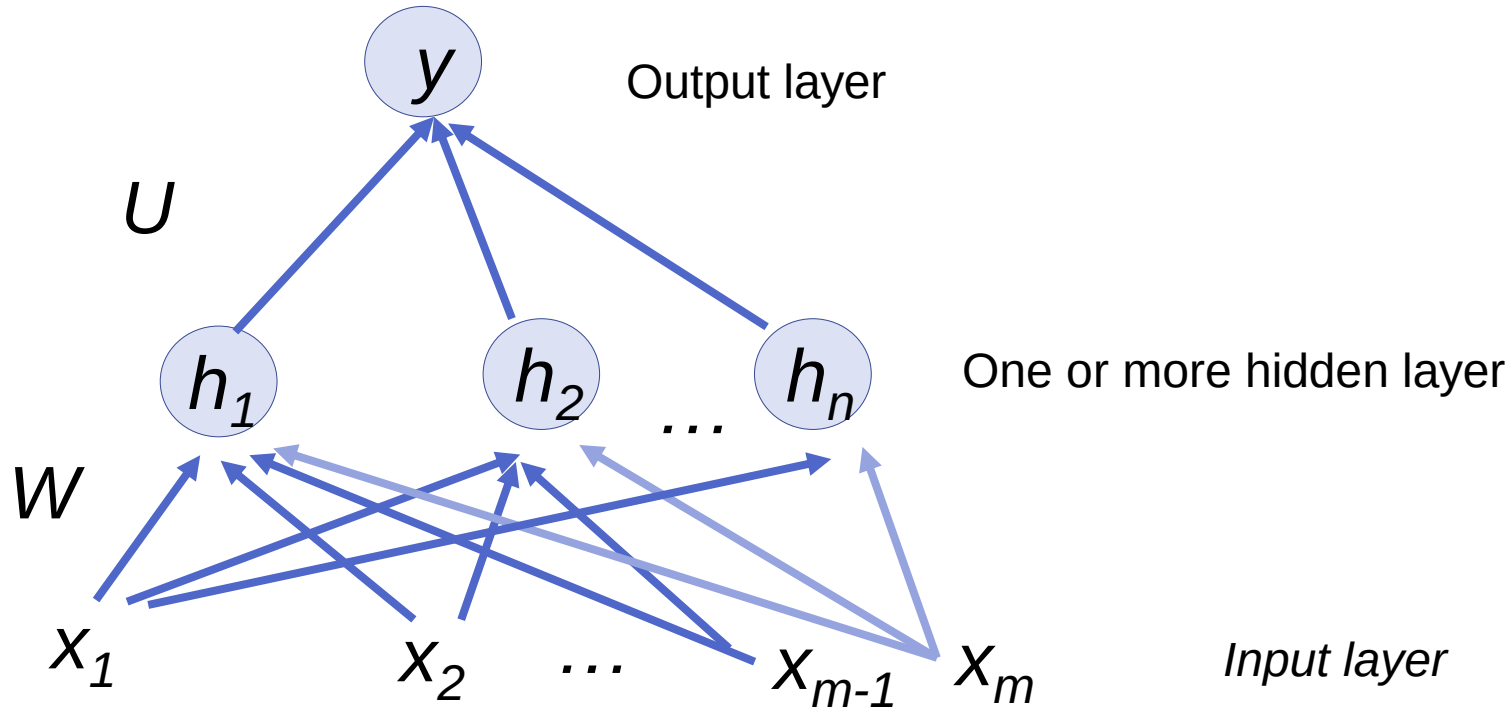
- 2 layer non-linear functions:
  - $h = f(Wx + b_1)$
  - $y = w_2^T h + b_2$

Outline

- **Perceptron**
- **Neural Network (**Multiple Layer perceptron)
  - Shallow Neural Networks (with a single hidden layer)
    - Neural units
    - Activation functions
    - Hidden units
    - Arbitrary inputs, hidden units, outputs
  - Deep Neural Networks (with multiple hidden layers)
- **Loss functions**

Multi-layer perceptrons (or MLPs)

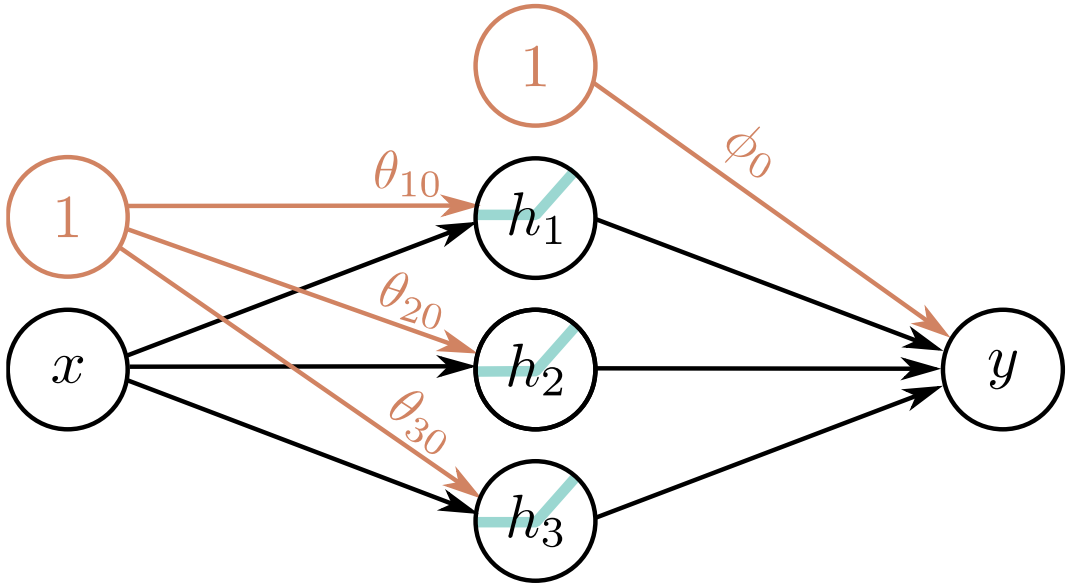
- **Multi-layer perceptrons** for historical reasons
  - Activation function
  - Hidden layer
  - Fully-connected



Example shallow network: 1 input, 1 output

With 3 hidden units:

$$y = \phi_0 + \phi_1 \mathrm{a}[\theta_{10} + \theta_{11}x] + \phi_2 \mathrm{a}[\theta_{20} + \theta_{21}x] + \phi_3 \mathrm{a}[\theta_{30} + \theta_{31}x].$$



Hidden units

$$y = \phi_0 + \phi_1 \mathrm{a}[\theta_{10} + \theta_{11}x] + \phi_2 \mathrm{a}[\theta_{20} + \theta_{21}x] + \phi_3 \mathrm{a}[\theta_{30} + \theta_{31}x].$$

Break down into two parts:

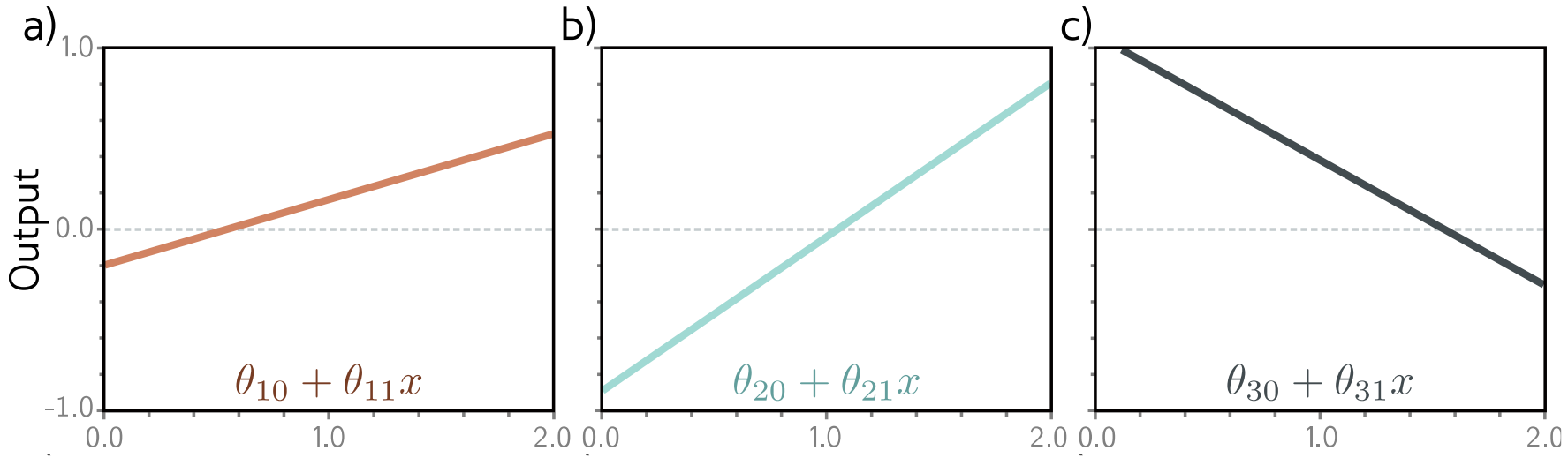
$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$$

where:

Hidden units

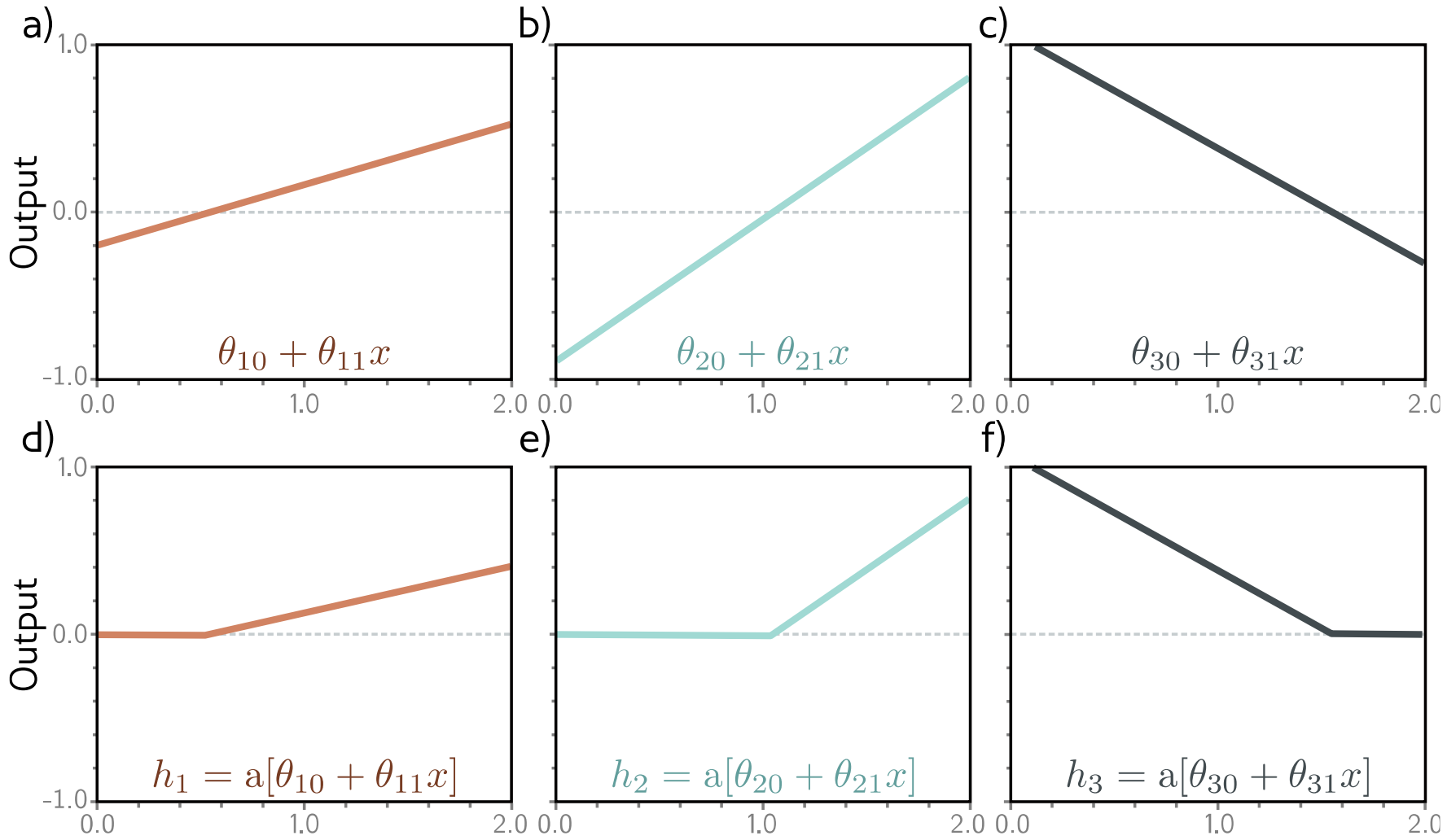
$$\begin{cases} h_1 = \mathrm{a}[\theta_{10} + \theta_{11}x] \\ h_2 = \mathrm{a}[\theta_{20} + \theta_{21}x] \\ h_3 = \mathrm{a}[\theta_{30} + \theta_{31}x] \end{cases}$$

1. compute three linear functions

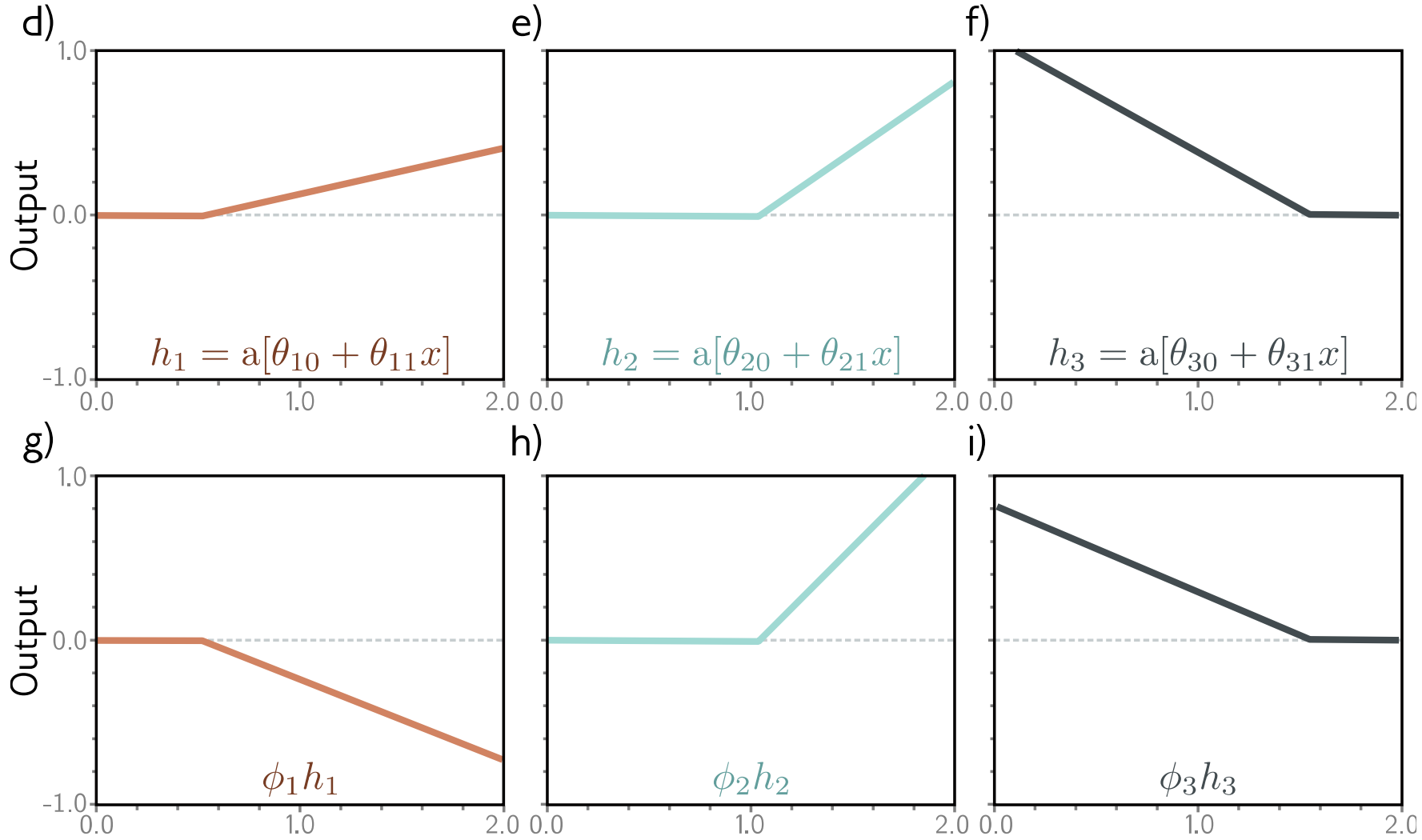


2. Pass through ReLU functions (creates hidden units)

$$h_1 = \mathrm{a}[\theta_{10} + \theta_{11}x]$$
$$h_2 = \mathrm{a}[\theta_{20} + \theta_{21}x]$$
$$h_3 = \mathrm{a}[\theta_{30} + \theta_{31}x],$$

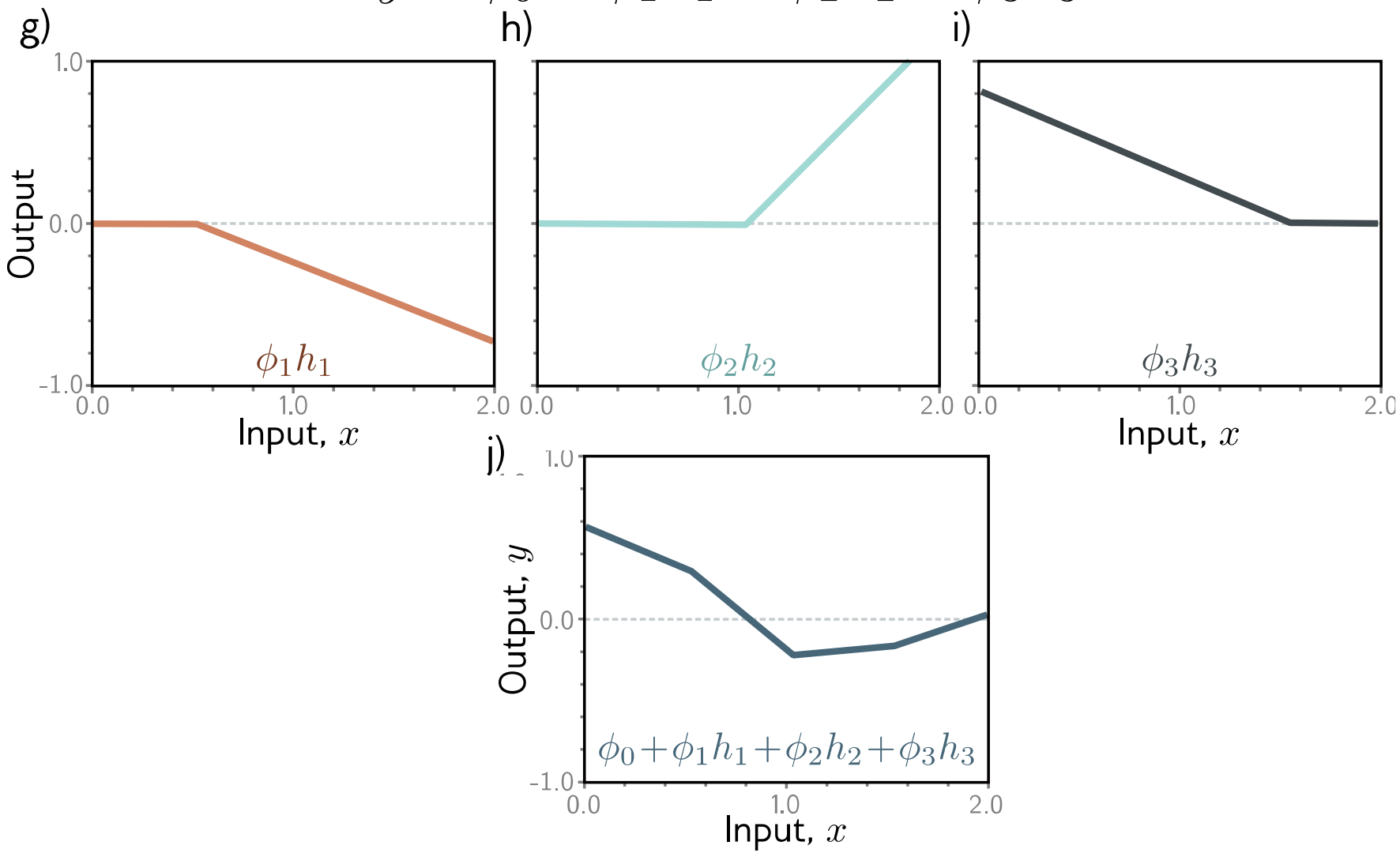


2. Weight the hidden units

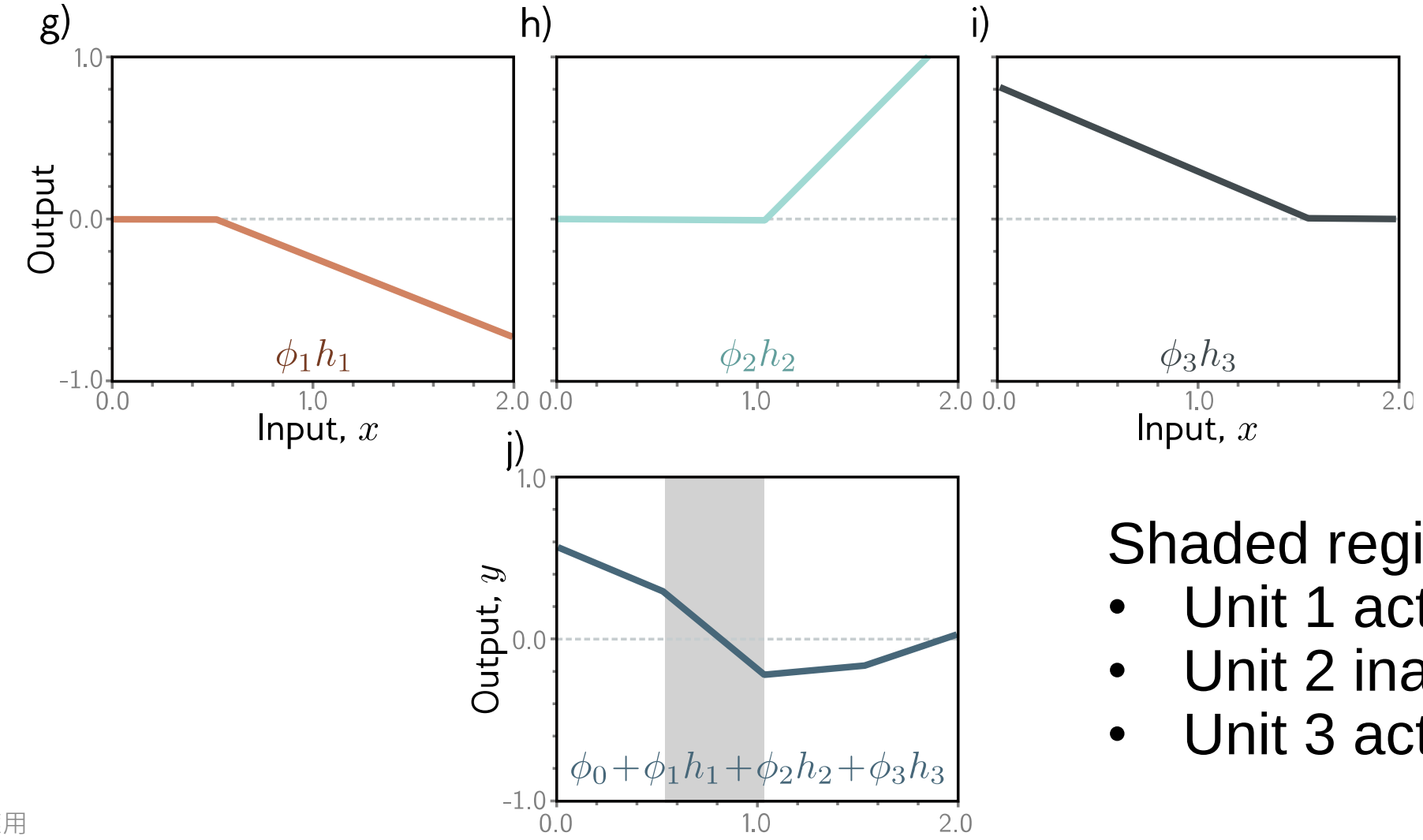


4. Sum the weighted hidden units to create output

$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$$



Activation pattern = which hidden units are activated

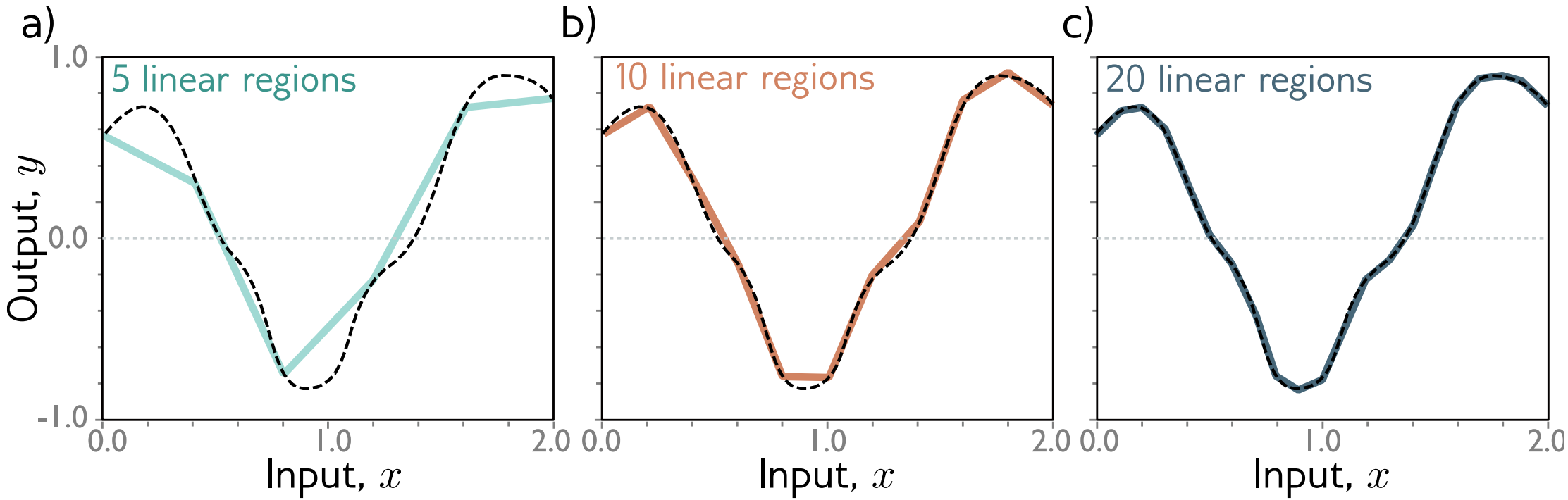


Shaded region:

- Unit 1 active
- Unit 2 inactive
- Unit 3 active

With enough hidden units…

… we can describe any 1D function to arbitrary accuracy



Outline

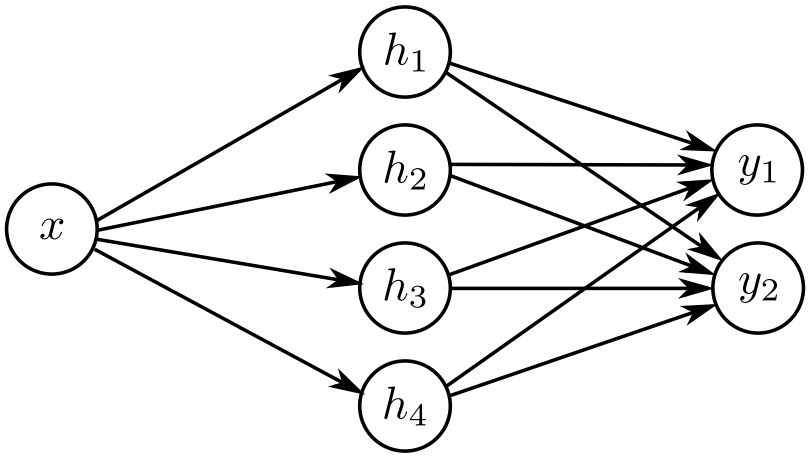
- **Perceptron**
- **Neural Network (**Multiple Layer perceptron)
  - Shallow Neural Networks (with a single hidden layer)
    - Neural units
    - Activation functions
    - Hidden units
    - Arbitrary inputs, hidden units, outputs
  - Deep Neural Networks (with multiple hidden layers)
- **Loss functions**

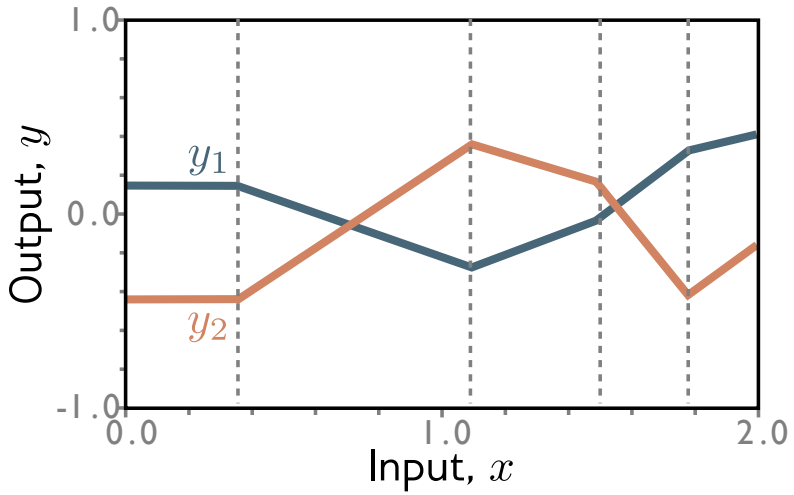
Two outputs

- 1 input, 4 hidden units, 2 outputs

$$
\begin{aligned}
h_1 &= \mathrm{a}[\theta_{10} + \theta_{11}x] \\
h_2 &= \mathrm{a}[\theta_{20} + \theta_{21}x] \\
h_3 &= \mathrm{a}[\theta_{30} + \theta_{31}x] \\
h_4 &= \mathrm{a}[\theta_{40} + \theta_{41}x]
\end{aligned}
$$

$$
\begin{aligned}
y_1 &= \phi_{10} + \phi_{11}h_1 + \phi_{12}h_2 + \phi_{13}h_3 + \phi_{14}h_4 \\
y_2 &= \phi_{20} + \phi_{21}h_1 + \phi_{22}h_2 + \phi_{23}h_3 + \phi_{24}h_4
\end{aligned}
$$



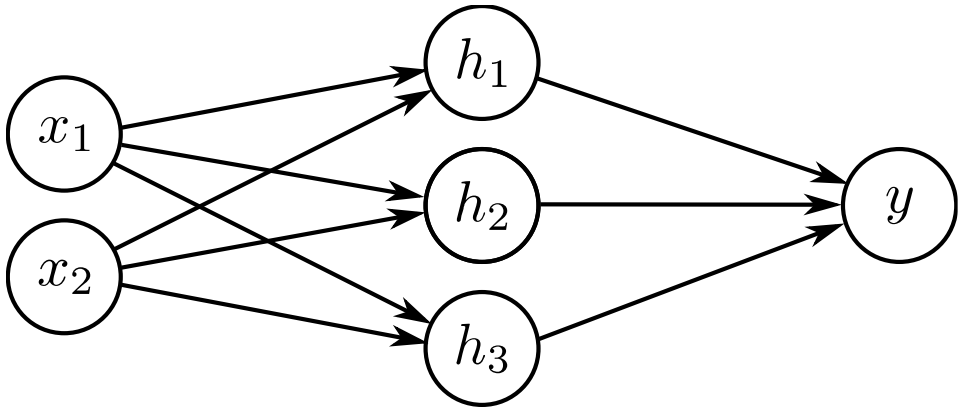


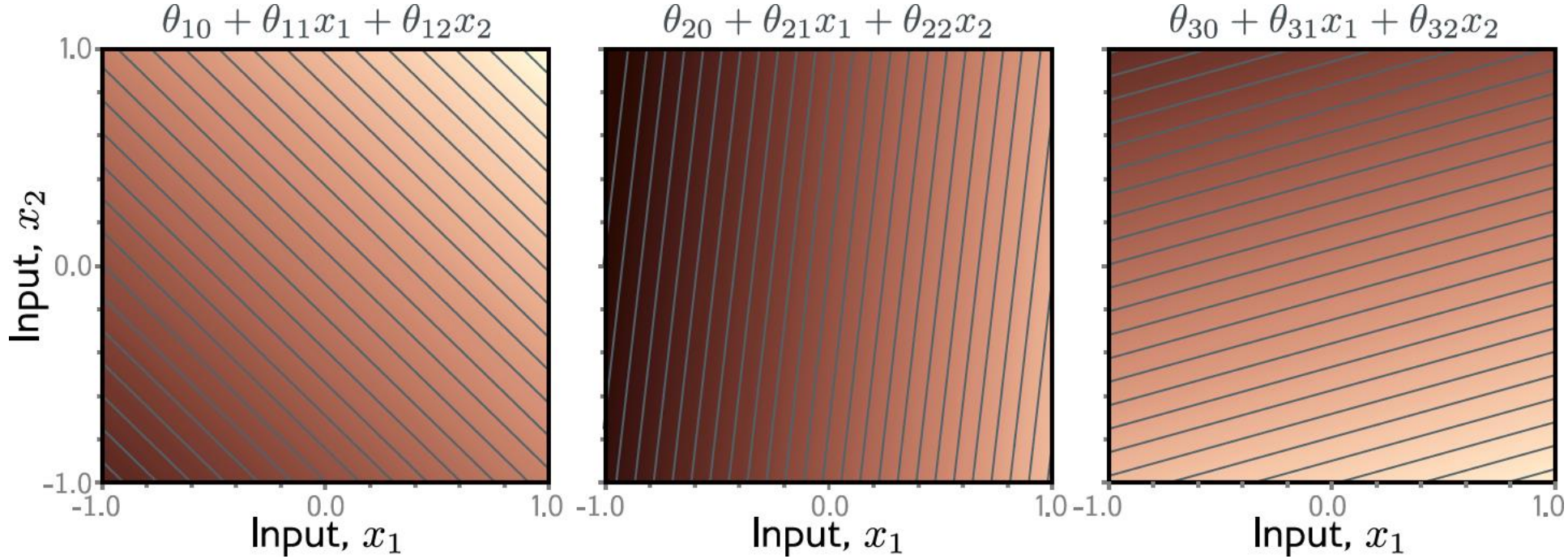
Two inputs

- 2 inputs, 3 hidden units, 1 output

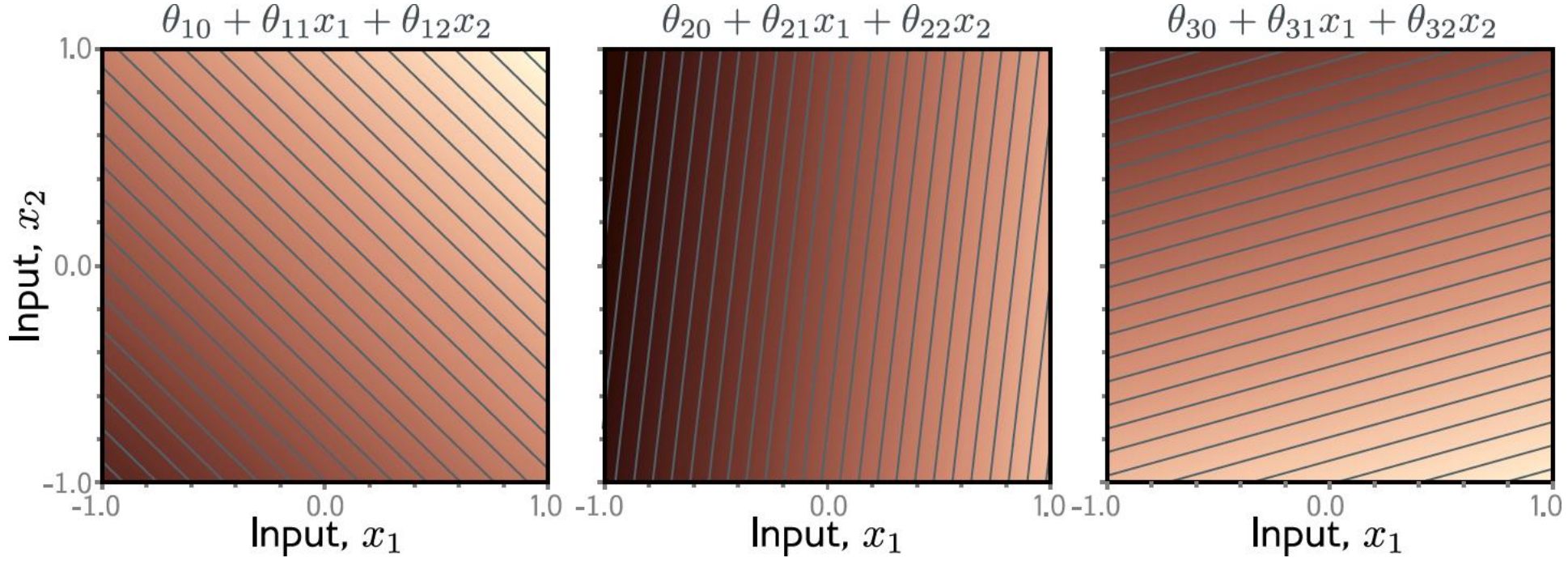
$$h_1 = \mathrm{a}[\theta_{10} + \theta_{11}x_1 + \theta_{12}x_2]$$
$$h_2 = \mathrm{a}[\theta_{20} + \theta_{21}x_1 + \theta_{22}x_2]$$
$$h_3 = \mathrm{a}[\theta_{30} + \theta_{31}x_1 + \theta_{32}x_2]$$

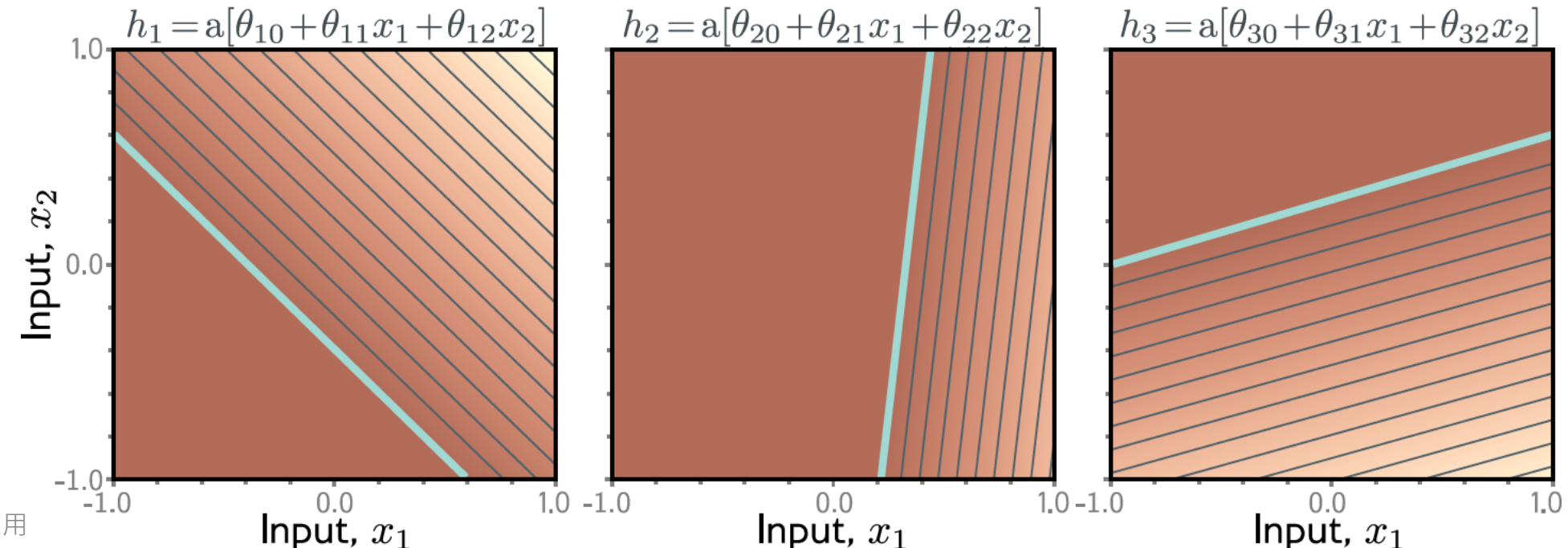
$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$$

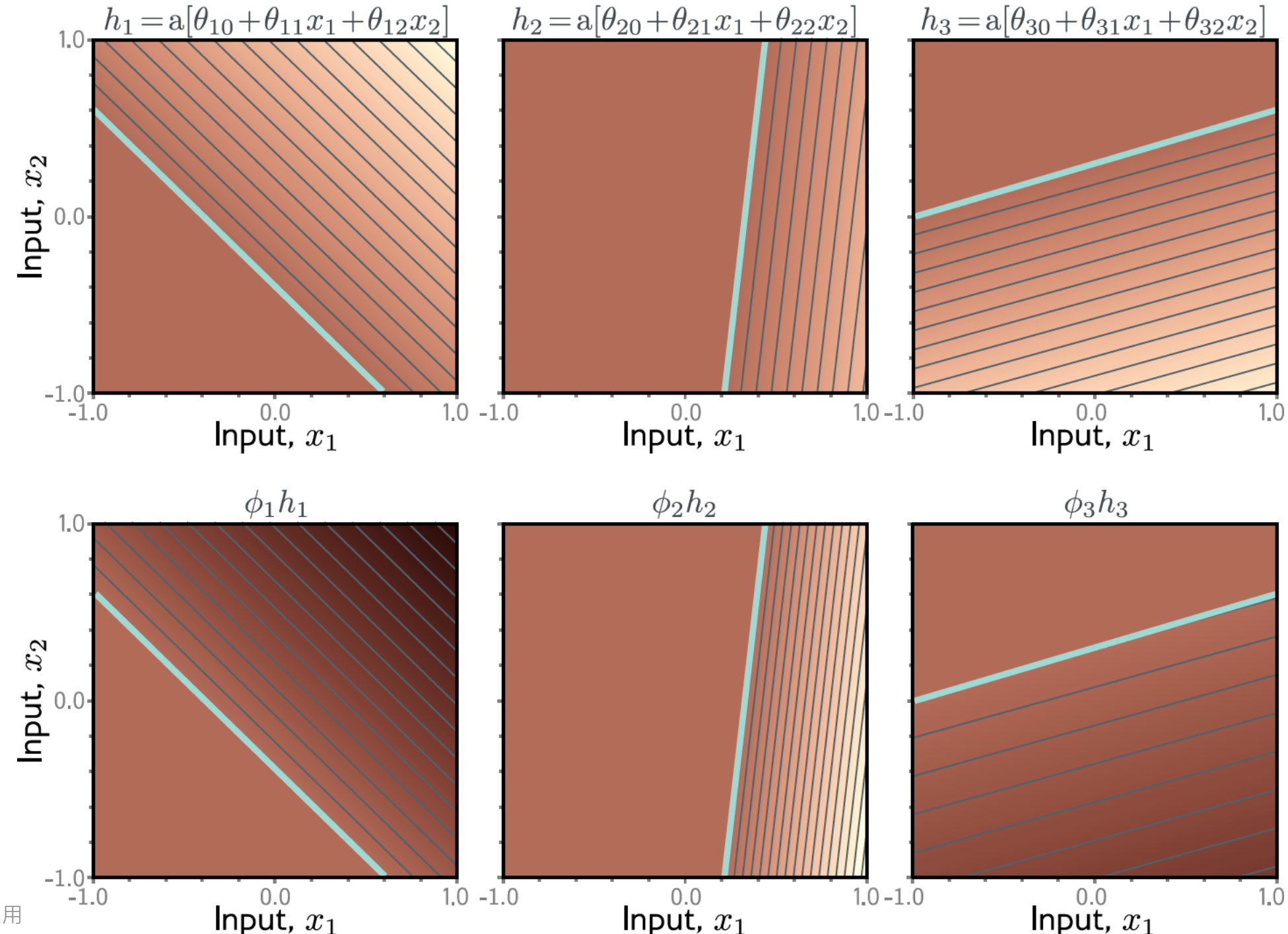


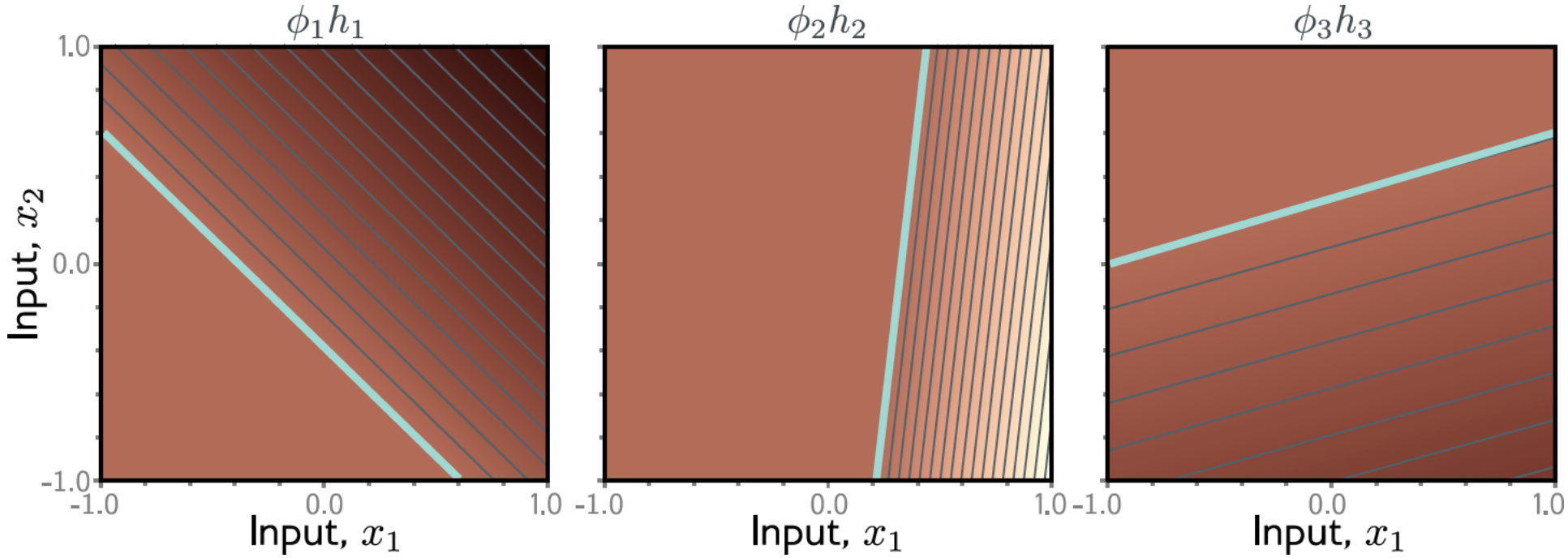

$\theta_{10} + \theta_{11}x_1 + \theta_{12}x_2$
$\theta_{20} + \theta_{21}x_1 + \theta_{22}x_2$
$\theta_{30} + \theta_{31}x_1 + \theta_{32}x_2$
Input, x_2
1.0
0.0
-1.0
-1.0
0.0
1.0
Input, x_1

ReLU activation function

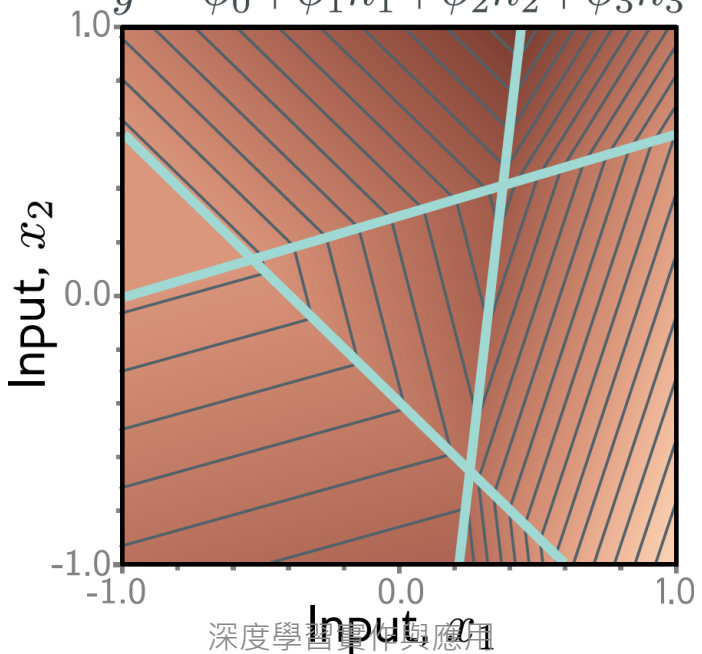





$h_1 = a[\theta_{10} + \theta_{11}x_1 + \theta_{12}x_2]$
$h_2 = a[\theta_{20} + \theta_{21}x_1 + \theta_{22}x_2]$
$h_3 = a[\theta_{30} + \theta_{31}x_1 + \theta_{32}x_2]$
$\phi_1 h_1$
$\phi_2 h_2$
$\phi_3 h_3$
Input, x_1
Input, x_2
1.0
0.0
-1.0



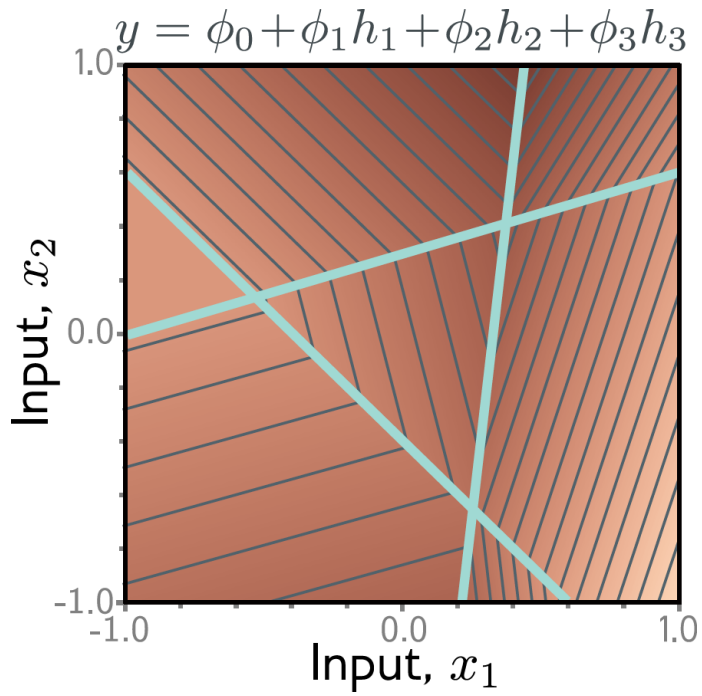
$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$$

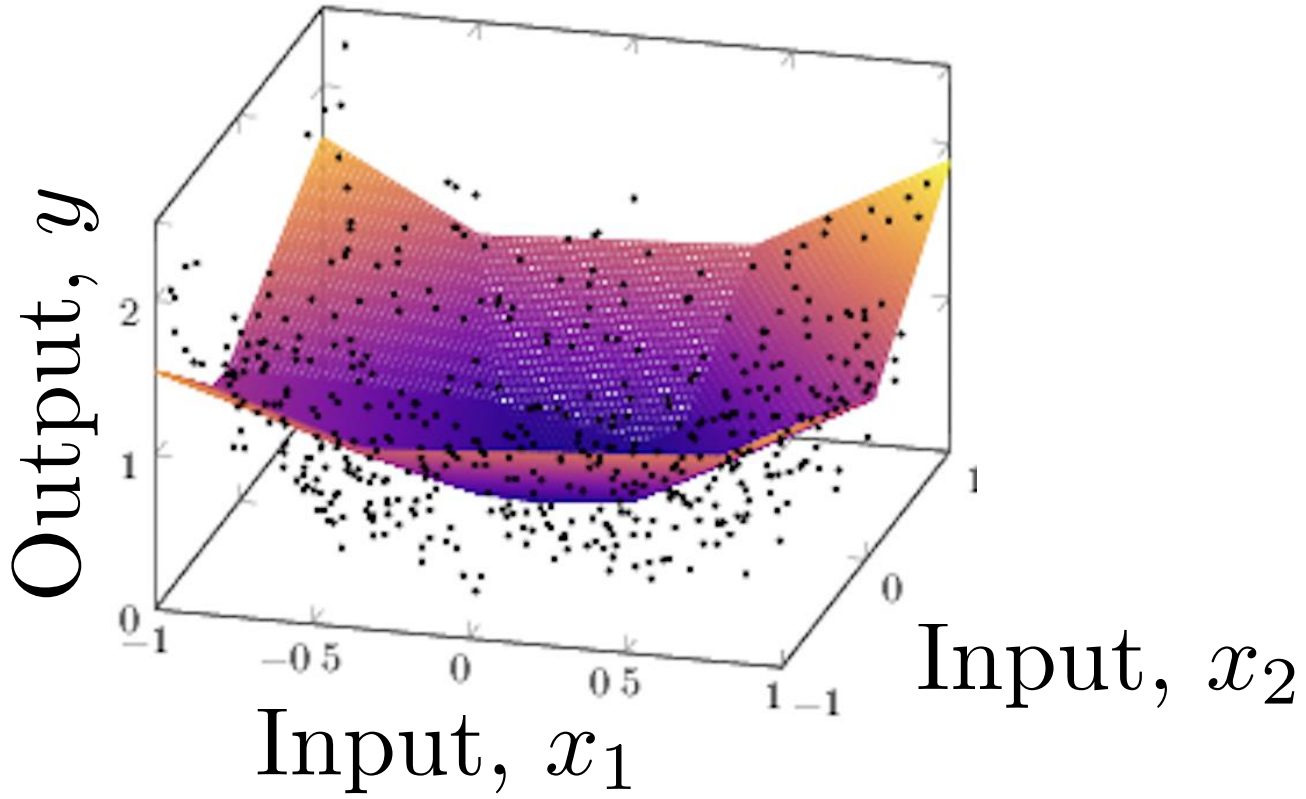


- Each region corresponds to a different activation pattern.
- For example, in the central triangular region, the first and third hidden units are active, and the second is inactive

Fitting

Convex polygons
(凸多邊形)





More than two inputs

- When there are more than two inputs to the model, it becomes difficult to visualize.
- However, the interpretation is similar. The output will be a continuous piecewise linear function of the input, where the linear regions are now convex polytopes in the multidimensional input space.

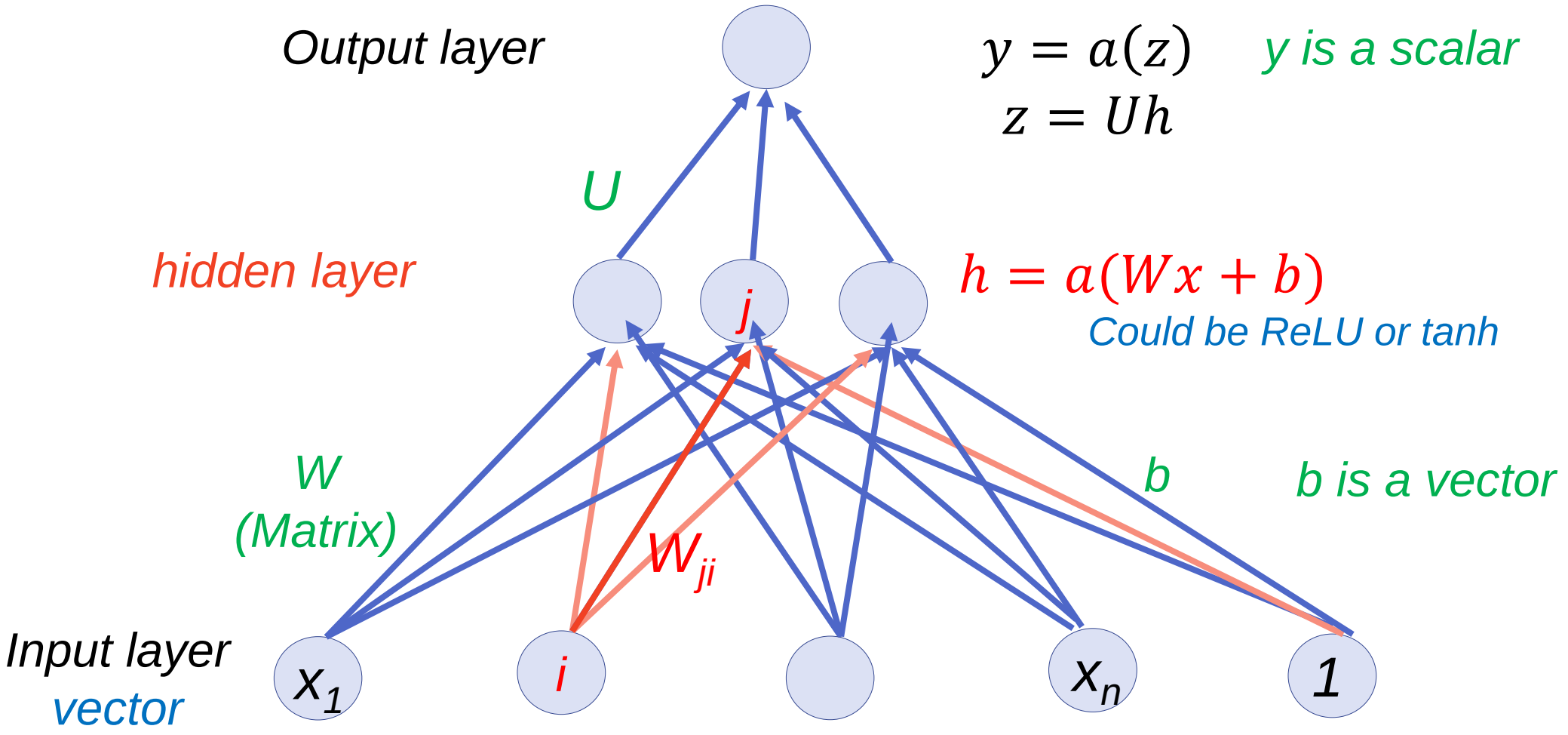
Arbitrary inputs, hidden units (1 layer), outputs

- D_o Outputs, D hidden units, and D_i inputs

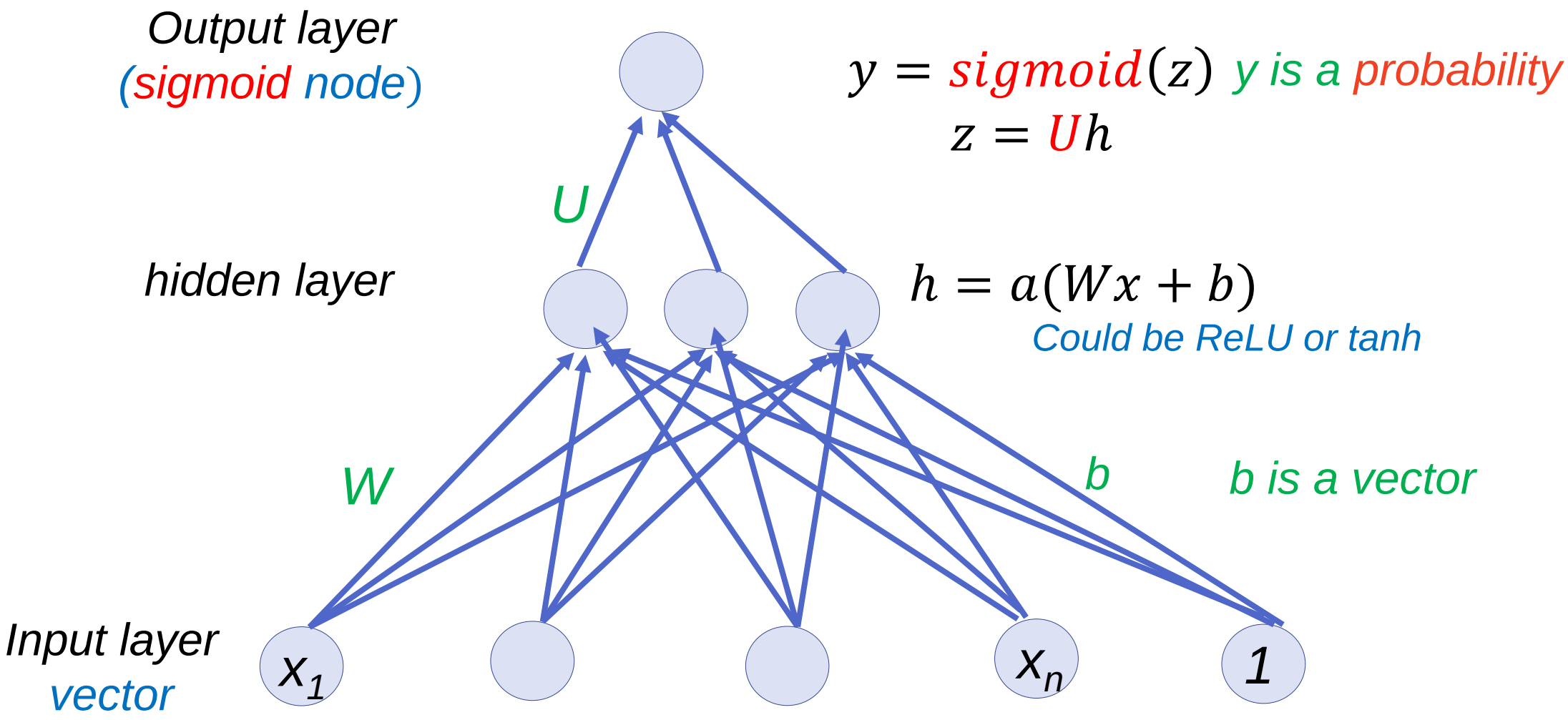
$$h_d = \mathrm{a}\left[\theta_{d0} + \sum_{i=1}^{D_i} \theta_{di} x_i\right] \qquad y_j = \phi_{j0} + \sum_{d=1}^{D} \phi_{jd} h_d$$

  - Regression task: y is a scalar
  - Binary classification: y is a probability
  - Multi-class classification: y_j is a vector, presenting probabilities

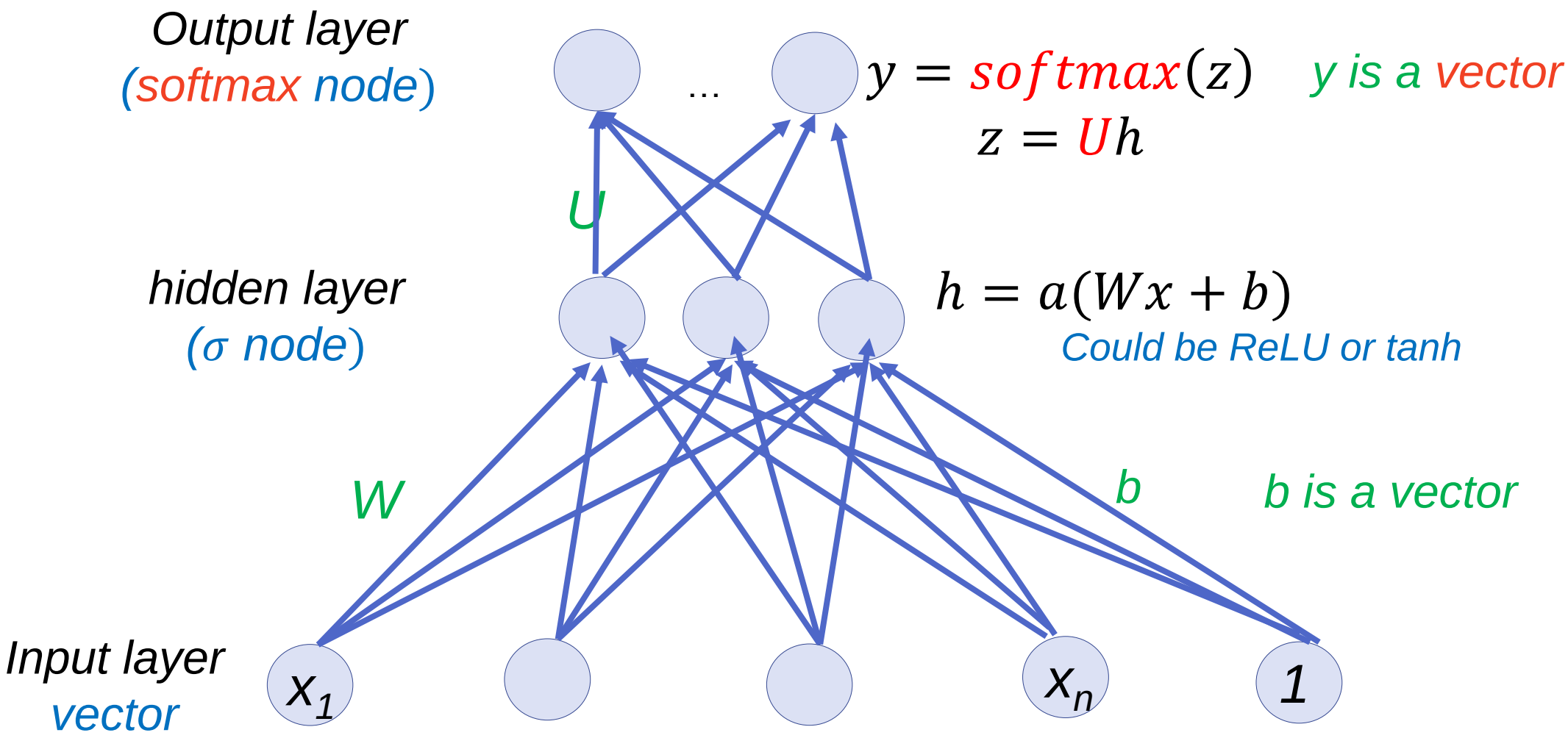
Shallow neural networks with scalar output



Shallow neural networks with sigmoid output



Shallow neural networks with softmax output



Outline

- **Perceptron**
- **Neural Network (**Multiple Layer perceptron)
  - Shallow Neural Networks (with a single hidden layer)
    - Neural units
    - Activation functions
    - Hidden units
    - Arbitrary inputs, hidden units, outputs
  - Deep Neural Networks (with multiple hidden layers)
- **Loss functions**

Multi-layers

$$for\ i\ in\ 1 \ldots n$$
$$z^i = W^i a^{i-1} + b^i$$
$$a^i = g^i(z^i)$$
$$\hat{y} = a^n$$

$$y = a^2$$

sigmoid or softmax

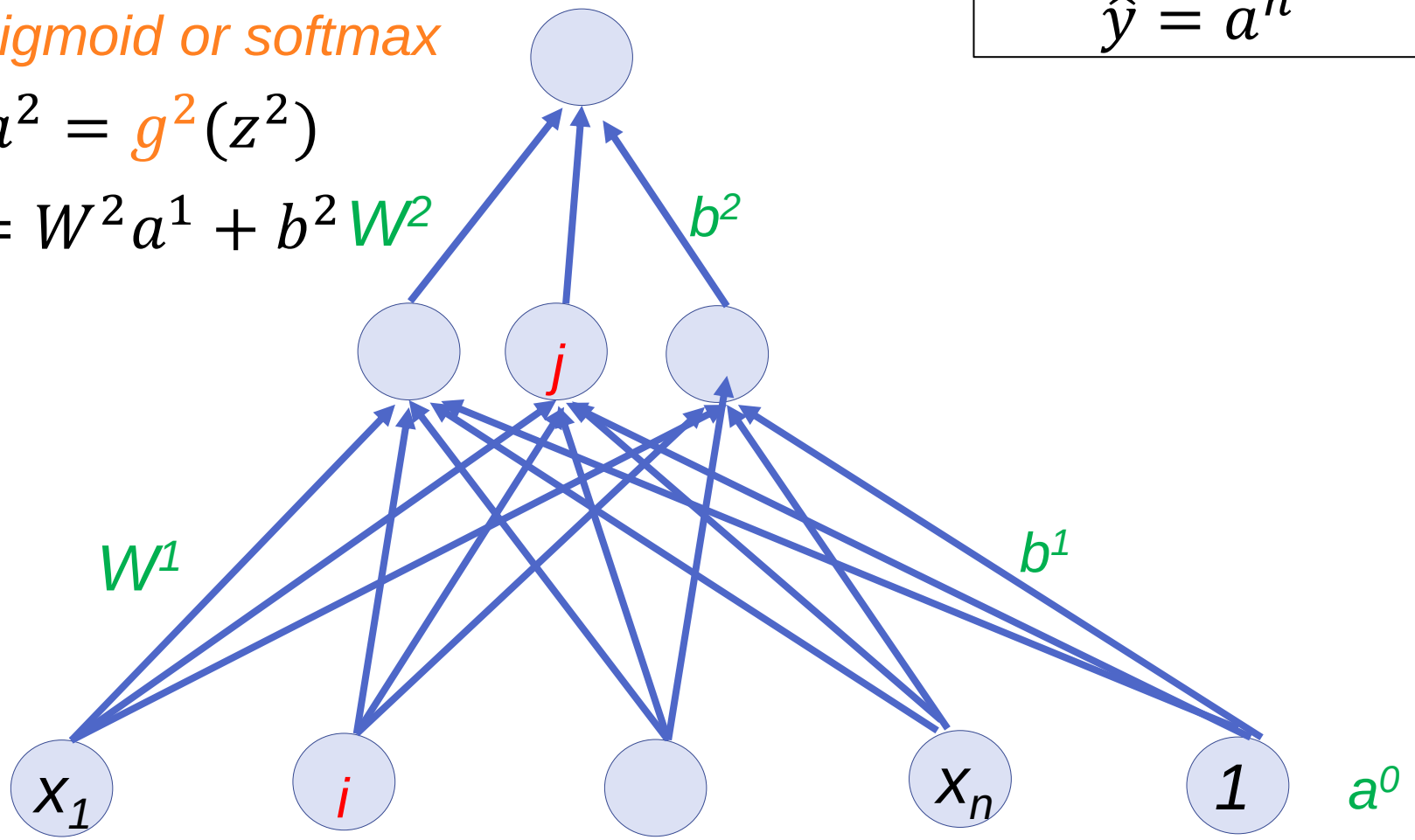
$$a^2 = g^2(z^2)$$

$$z^2 = W^2 a^1 + b^2$$

ReLU or tanh

$$a^1 = g^1(z^1)$$

$$z^1 = W^1 a^0 + b^1$$



Hyperparameters

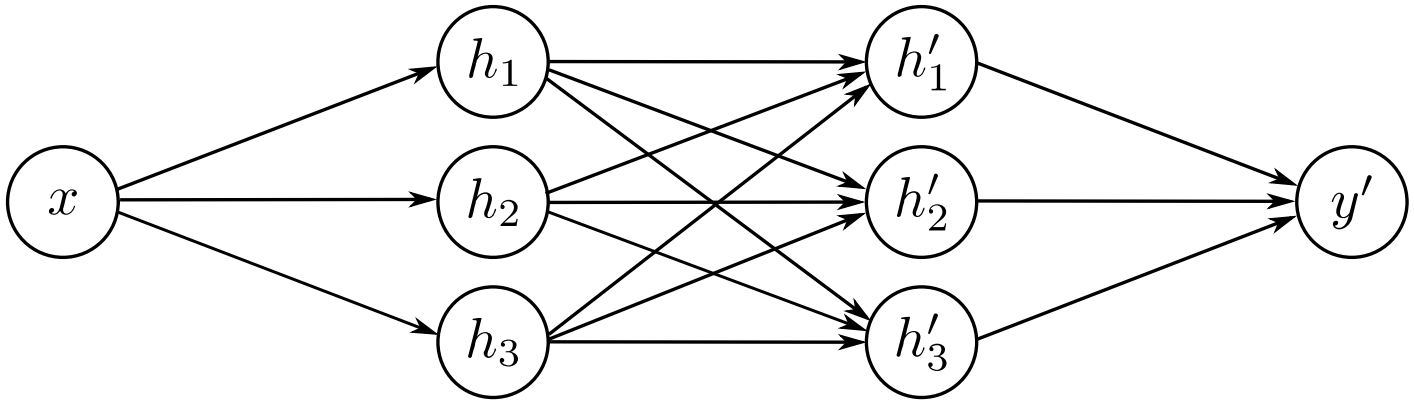
- K layers = depth of network
- D_k hidden units per layer = width of network

- These are called hyperparameters – chosen before training the network
- Can try retraining with different hyperparameters – hyperparameter optimization or hyperparameter search

Two-layer network

$$h_1 = \mathrm{a}[\theta_{10} + \theta_{11}x] \qquad h'_1 = \mathrm{a}[\psi_{10} + \psi_{11}h_1 + \psi_{12}h_2 + \psi_{13}h_3]$$
$$h_2 = \mathrm{a}[\theta_{20} + \theta_{21}x] \qquad h'_2 = \mathrm{a}[\psi_{20} + \psi_{21}h_2 + \psi_{22}h_2 + \psi_{23}h_3]$$
$$h_3 = \mathrm{a}[\theta_{30} + \theta_{31}x] \qquad h'_3 = \mathrm{a}[\psi_{30} + \psi_{31}h_2 + \psi_{32}h_2 + \psi_{33}h_3]$$

$$y' = \phi'_0 + \phi'_1 h'_1 + \phi'_2 h'_2 + \phi'_3 h'_3$$



Notation change #1

$$h_1 = \mathrm{a}[\theta_{10} + \theta_{11}x]$$
$$h_2 = \mathrm{a}[\theta_{20} + \theta_{21}x]$$
$$h_3 = \mathrm{a}[\theta_{30} + \theta_{31}x]$$

$$h'_1 = \mathrm{a}[\psi_{10} + \psi_{11}h_1 + \psi_{12}h_2 + \psi_{13}h_3]$$
$$h'_2 = \mathrm{a}[\psi_{20} + \psi_{21}h_1 + \psi_{22}h_2 + \psi_{23}h_3]$$
$$h'_3 = \mathrm{a}[\psi_{30} + \psi_{31}h_1 + \psi_{32}h_2 + \psi_{33}h_3]$$

$$y' = \phi'_0 + \phi'_1 h'_1 + \phi'_2 h'_2 + \phi'_3 h'_3$$

Notation change #1

$$
\begin{aligned}
h_1 &= \mathrm{a}[\theta_{10} + \theta_{11}x] \\
h_2 &= \mathrm{a}[\theta_{20} + \theta_{21}x] \\
h_3 &= \mathrm{a}[\theta_{30} + \theta_{31}x]
\end{aligned}
$$

$$
\begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = \mathbf{a}\left[\begin{bmatrix} \theta_{10} \\ \theta_{20} \\ \theta_{30} \end{bmatrix} + \begin{bmatrix} \theta_{11} \\ \theta_{21} \\ \theta_{31} \end{bmatrix} x\right]
$$

$$
\begin{aligned}
h'_1 &= \mathrm{a}[\psi_{10} + \psi_{11}h_1 + \psi_{12}h_2 + \psi_{13}h_3] \\
h'_2 &= \mathrm{a}[\psi_{20} + \psi_{21}h_1 + \psi_{22}h_2 + \psi_{23}h_3] \\
h'_3 &= \mathrm{a}[\psi_{30} + \psi_{31}h_1 + \psi_{32}h_2 + \psi_{33}h_3]
\end{aligned}
$$

$$
y' = \phi'_0 + \phi'_1 h'_1 + \phi'_2 h'_2 + \phi'_3 h'_3
$$

Notation change #1

$$
\begin{aligned}
h_1 &= \mathrm{a}[\theta_{10} + \theta_{11}x] \\
h_2 &= \mathrm{a}[\theta_{20} + \theta_{21}x] \\
h_3 &= \mathrm{a}[\theta_{30} + \theta_{31}x]
\end{aligned}
\longrightarrow
\begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = \mathbf{a}\left[\begin{bmatrix} \theta_{10} \\ \theta_{20} \\ \theta_{30} \end{bmatrix} + \begin{bmatrix} \theta_{11} \\ \theta_{21} \\ \theta_{31} \end{bmatrix} x\right]
$$

$$
\begin{aligned}
h'_1 &= \mathrm{a}[\psi_{10} + \psi_{11}h_1 + \psi_{12}h_2 + \psi_{13}h_3] \\
h'_2 &= \mathrm{a}[\psi_{20} + \psi_{21}h_1 + \psi_{22}h_2 + \psi_{23}h_3] \\
h'_3 &= \mathrm{a}[\psi_{30} + \psi_{31}h_1 + \psi_{32}h_2 + \psi_{33}h_3]
\end{aligned}
\longrightarrow
\begin{bmatrix} h'_1 \\ h'_2 \\ h'_3 \end{bmatrix} = \mathbf{a}\left[\begin{bmatrix} \psi_{10} \\ \psi_{20} \\ \psi_{30} \end{bmatrix} + \begin{bmatrix} \psi_{11} & \psi_{12} & \psi_{13} \\ \psi_{21} & \psi_{22} & \psi_{23} \\ \psi_{31} & \psi_{32} & \psi_{33} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}\right]
$$

Notation change #1

$$
\begin{aligned}
h_1 &= \mathrm{a}[\theta_{10} + \theta_{11}x] \\
h_2 &= \mathrm{a}[\theta_{20} + \theta_{21}x] \\
h_3 &= \mathrm{a}[\theta_{30} + \theta_{31}x]
\end{aligned}
\longrightarrow
\begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = \mathbf{a}\left[\begin{bmatrix} \theta_{10} \\ \theta_{20} \\ \theta_{30} \end{bmatrix} + \begin{bmatrix} \theta_{11} \\ \theta_{21} \\ \theta_{31} \end{bmatrix} x\right]
$$

$$
\begin{aligned}
h'_1 &= \mathrm{a}[\psi_{10} + \psi_{11}h_1 + \psi_{12}h_2 + \psi_{13}h_3] \\
h'_2 &= \mathrm{a}[\psi_{20} + \psi_{21}h_1 + \psi_{22}h_2 + \psi_{23}h_3] \\
h'_3 &= \mathrm{a}[\psi_{30} + \psi_{31}h_1 + \psi_{32}h_2 + \psi_{33}h_3]
\end{aligned}
\longrightarrow
\begin{bmatrix} h'_1 \\ h'_2 \\ h'_3 \end{bmatrix} = \mathbf{a}\left[\begin{bmatrix} \psi_{10} \\ \psi_{20} \\ \psi_{30} \end{bmatrix} + \begin{bmatrix} \psi_{11} & \psi_{12} & \psi_{13} \\ \psi_{21} & \psi_{22} & \psi_{23} \\ \psi_{32} & \psi_{32} & \psi_{33} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}\right]
$$

$$
y' = \phi'_0 + \phi'_1 h'_1 + \phi'_2 h'_2 + \phi'_3 h'_3
\longrightarrow
y' = \phi'_0 + \begin{bmatrix} \phi'_1 & \phi'_2 & \phi'_3 \end{bmatrix} \begin{bmatrix} h'_1 \\ h'_2 \\ h'_3 \end{bmatrix}
$$

Notation change #2

$$\begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = \mathbf{a}\left[\begin{bmatrix} \theta_{10} \\ \theta_{20} \\ \theta_{30} \end{bmatrix} + \begin{bmatrix} \theta_{11} \\ \theta_{21} \\ \theta_{31} \end{bmatrix} x\right] \longrightarrow \mathbf{h} = \mathbf{a}\left[\boldsymbol{\theta}_0 + \boldsymbol{\theta} x\right]$$

$$\begin{bmatrix} h'_1 \\ h'_2 \\ h'_3 \end{bmatrix} = \mathbf{a}\left[\begin{bmatrix} \psi_{10} \\ \psi_{20} \\ \psi_{30} \end{bmatrix} + \begin{bmatrix} \psi_{11} & \psi_{12} & \psi_{13} \\ \psi_{21} & \psi_{22} & \psi_{23} \\ \psi_{32} & \psi_{32} & \psi_{33} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}\right] \longrightarrow \mathbf{h}' = \mathbf{a}\left[\boldsymbol{\psi}_0 + \boldsymbol{\Psi}\mathbf{h}\right]$$

$$y' = \phi'_0 + \begin{bmatrix} \phi'_1 & \phi'_2 & \phi'_3 \end{bmatrix} \begin{bmatrix} h'_1 \\ h'_2 \\ h'_3 \end{bmatrix} \longrightarrow y = \boldsymbol{\phi}'_0 + \boldsymbol{\phi}'\mathbf{h}'$$

Notation change #3

$$\mathbf{h} = \mathbf{a}\left[\boldsymbol{\theta}_0 + \boldsymbol{\theta} x\right] \longrightarrow \mathbf{h}_1 = \mathbf{a}[\boldsymbol{\beta}_0 + \boldsymbol{\Omega}_0 \mathbf{x}]$$

$$\mathbf{h}' = \mathbf{a}\left[\boldsymbol{\psi}_0 + \boldsymbol{\Psi}\mathbf{h}\right] \longrightarrow \mathbf{h}_2 = \mathbf{a}[\boldsymbol{\beta}_1 + \boldsymbol{\Omega}_1 \mathbf{h}_1]$$

$$y = \boldsymbol{\phi}_0' + \boldsymbol{\phi}'\mathbf{h}' \longrightarrow \mathbf{y} = \boldsymbol{\beta}_2 + \boldsymbol{\Omega}_2 \mathbf{h}_2$$

Notation change #3

Bias vector

Weight matrix

$$\mathbf{h} = \mathbf{a}\left[\boldsymbol{\theta}_0 + \boldsymbol{\theta} x\right] \longrightarrow \mathbf{h}_1 = \mathbf{a}[\boldsymbol{\beta}_0 + \boldsymbol{\Omega}_0 \mathbf{x}]$$

$$\mathbf{h}' = \mathbf{a}\left[\boldsymbol{\psi}_0 + \boldsymbol{\Psi}\mathbf{h}\right] \longrightarrow \mathbf{h}_2 = \mathbf{a}[\boldsymbol{\beta}_1 + \boldsymbol{\Omega}_1 \mathbf{h}_1]$$

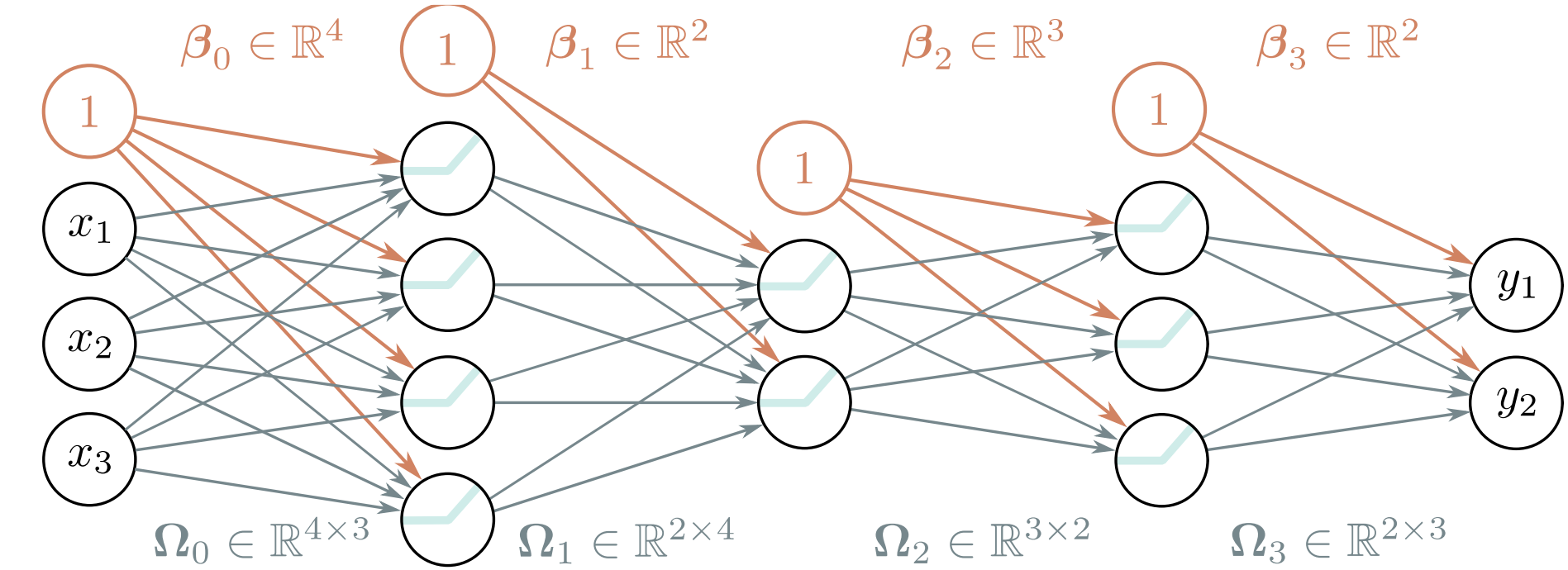
$$y = \boldsymbol{\phi}_0' + \boldsymbol{\phi}'\mathbf{h}' \longrightarrow \mathbf{y} = \boldsymbol{\beta}_2 + \boldsymbol{\Omega}_2 \mathbf{h}_2$$

General equations for deep network

$$
\begin{aligned}
\mathbf{h}_1 &= \mathbf{a}[\boldsymbol{\beta}_0 + \boldsymbol{\Omega}_0\mathbf{x}] \\
\mathbf{h}_2 &= \mathbf{a}[\boldsymbol{\beta}_1 + \boldsymbol{\Omega}_1\mathbf{h}_1] \\
\mathbf{h}_3 &= \mathbf{a}[\boldsymbol{\beta}_2 + \boldsymbol{\Omega}_2\mathbf{h}_2] \\
&\vdots \\
\mathbf{h}_K &= \mathbf{a}[\boldsymbol{\beta}_{K-1} + \boldsymbol{\Omega}_{K-1}\mathbf{h}_{K-1}] \\
\mathbf{y} &= \boldsymbol{\beta}_K + \boldsymbol{\Omega}_K\mathbf{h}_K,
\end{aligned}
$$

$$
\mathbf{y} = \boldsymbol{\beta}_K + \boldsymbol{\Omega}_K\mathbf{a}\left[\boldsymbol{\beta}_{K-1} + \boldsymbol{\Omega}_{K-1}\mathbf{a}\left[\ldots\boldsymbol{\beta}_2 + \boldsymbol{\Omega}_2\mathbf{a}\left[\boldsymbol{\beta}_1 + \boldsymbol{\Omega}_1\mathbf{a}\left[\boldsymbol{\beta}_0 + \boldsymbol{\Omega}_0\mathbf{x}\right]\right]\ldots\right]\right]
$$

Example



Input, $\mathbf{x}$	Hidden layer, $\mathbf{h}_1$	Hidden layer, $\mathbf{h}_2$	Hidden layer, $\mathbf{h}_3$	Output, $\mathbf{y}$
$D_i = 3$	$D_1 = 4$	$D_2 = 2$	$D_3 = 3$	$D_o = 2$

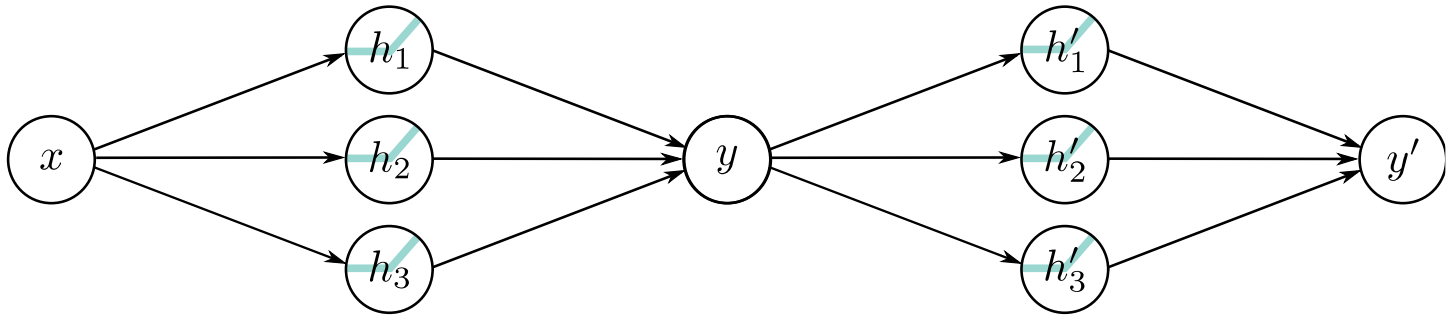
Shallow vs. deep networks

The best results are created by deep networks with many layers.

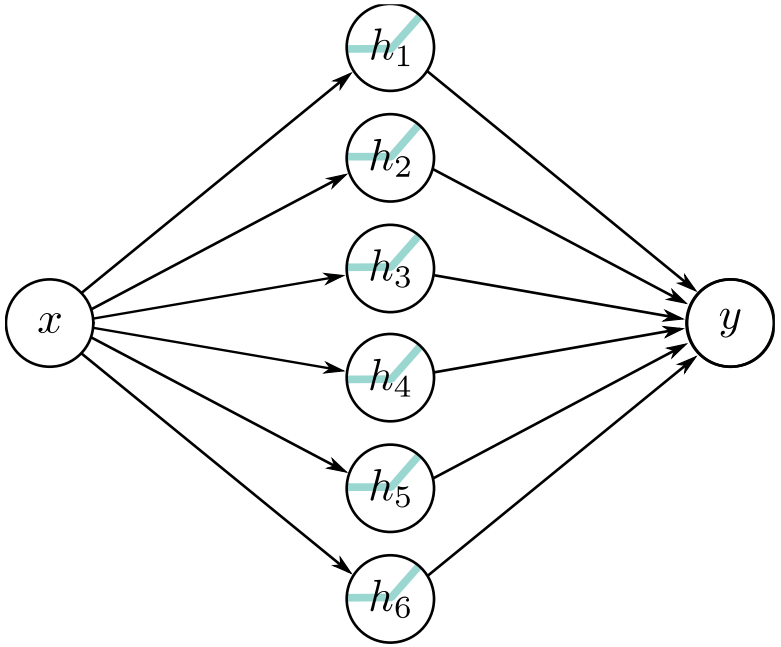
- 50-1000 layers for most applications
- Best results in
  - Computer vision
  - Natural language processing
  - Graph neural networks
  - Generative models
  - Reinforcement learning

All use deep networks.

Comparing to shallow with six hidden units



- 20 parameters
- (at least) 9 regions

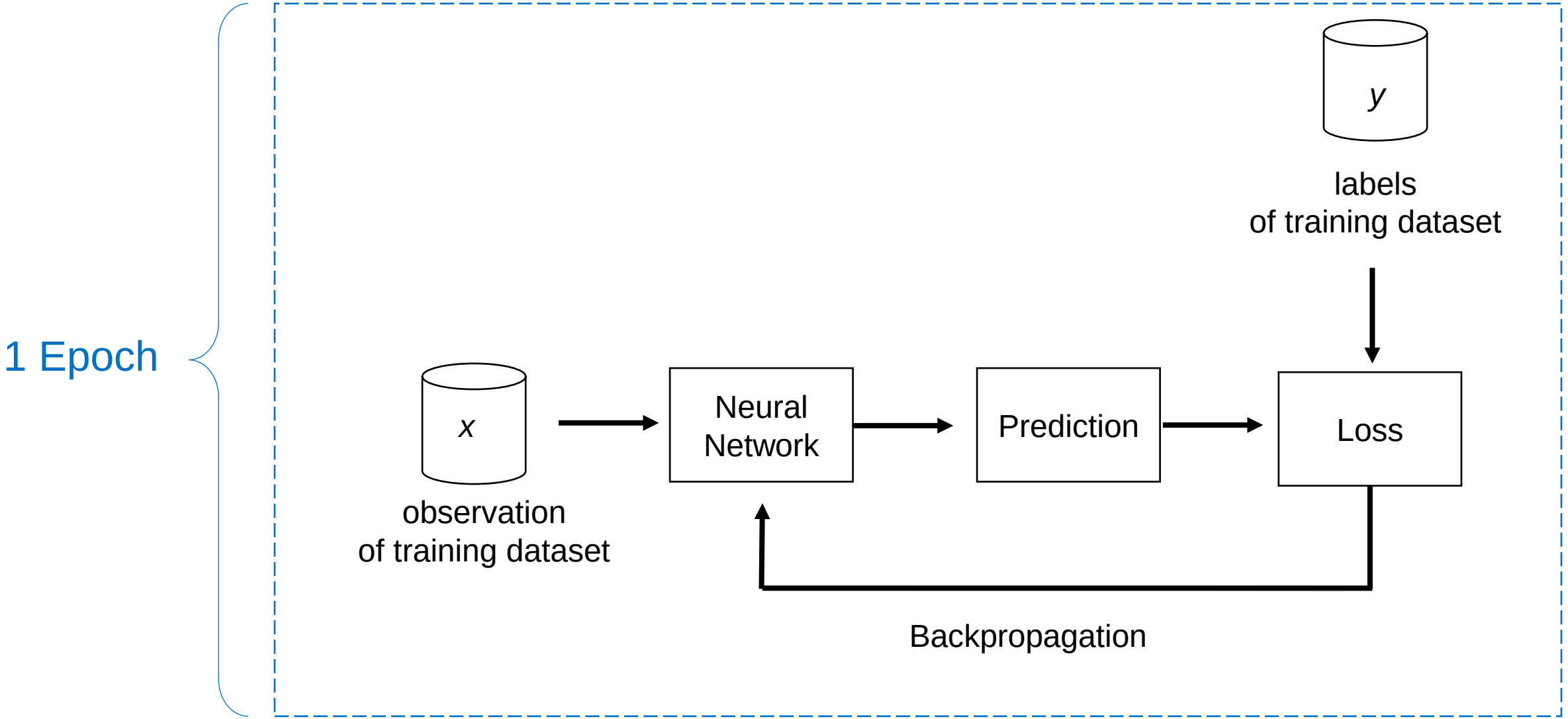


- 19 parameters
- Max 7 regions

Outline

- **Perceptron**
- **Neural Network (**Multiple Layer perceptron)
  - Shallow Neural Networks (with a single hidden layer)
    - Neural units
    - Activation functions
    - Hidden units
    - Arbitrary inputs, hidden units, outputs
  - Deep Neural Networks (with multiple hidden layers)
- **Loss functions**

Training cycle



Where did the W's come from?

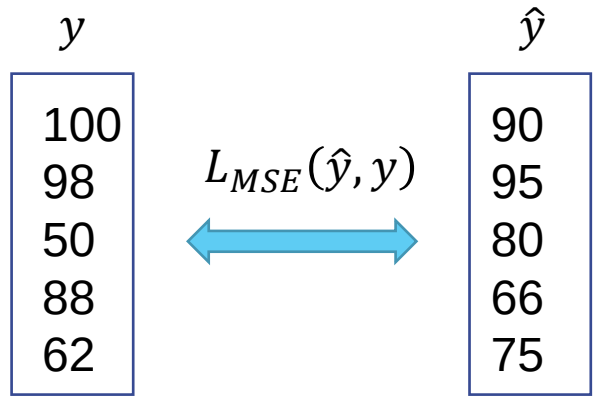
- Supervised classification:
  - We know the correct either real number or label y for each x.
  - But what the system produces is an estimate, $\hat{y}$
- We want to set *w* and *b* to minimize the distance between our estimate $\hat{y}_i$ and the true y_i .
  - We need a distance estimator: a **loss function**
  - We need an **optimization** algorithm to **iteratively** update *w* and *b* to minimize the loss.。

Learning components

- A loss function:
  - Mean square error

$$L(y, \hat{y}) = \frac{1}{m}\sum_{i}^{m}\left(y_i - \hat{y}_i\right)^2$$

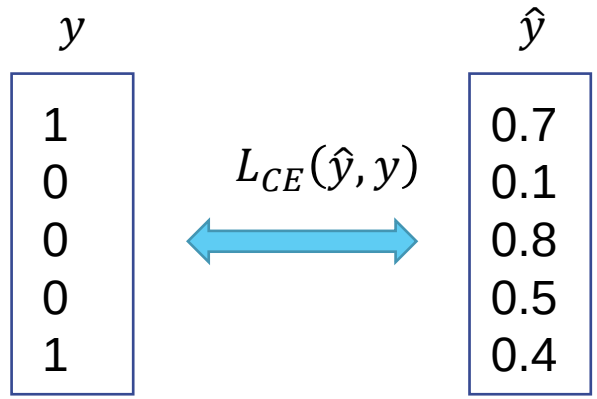
where *m* is the number of data in the training data



  - Cross-entropy loss

$$L_{CE}(\hat{y}, y) = -\sum_{i}^{m}\sum_{j}^{n} y_j \log(\hat{y}_j)$$

where *n* is the number of class,
m is the number of data in the training data



Measure estimation quality for regression

- Compare the true value vs. the estimated value
  - e.g., Real sale price vs. estimated house price

- Let y the true value, and $\hat{y}$ the estimated value, we can compare the loss

$$L(y, \hat{y}) = (y - \hat{y})^2$$

- It is called squared loss.

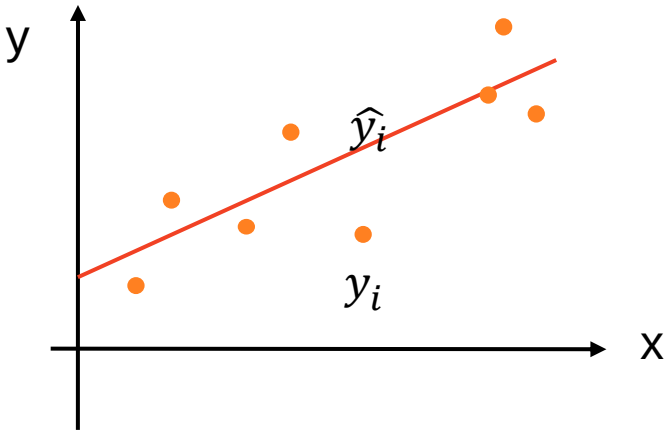
Measure estimation quality for regression

- Collect multiple data points to fit parameters, called the training data (the more the better).
  - e.g., Houses sold in the last 6 months
- Assume *n* examples

$$\boldsymbol{X} = [\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n}], \qquad \boldsymbol{y} = [y_1, y_2, \ldots, y_n]$$

- Training loss (Mean Squared Error):

$$L(y, \hat{y}) = \frac{1}{m}\sum_i^m \left(y_i - \hat{y}_i\right)^2$$



- Minimize loss to learn parameters

$$w^*, b^* = argmin_{\boldsymbol{w},b}\ L(y, \hat{y})$$

Loss functions

- Maximum likelihood
- Recipe for loss functions
- Example 1: univariate regression
- Example 2: binary classification
- Example 3: multiclass classification

How to construct loss functions

- Model predicts output *y* given input *x*

How to construct loss functions

- ~~Model predicts output *y* given input *x*~~

How to construct loss functions

- ~~Model predicts output *y* given input *x*~~
- Model predicts a conditional probability distribution:

$$Pr(\mathbf{y}|\mathbf{x})$$

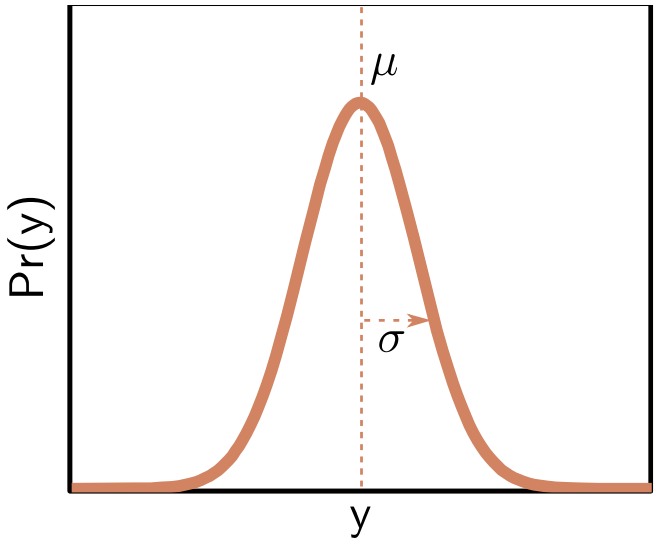
over outputs **y** given inputs **x**.

- Loss function aims to make the outputs have high probability

How can a model predict a probability distribution?

1. Pick a known distribution (e.g., normal distribution) to model output y with parameters $\boldsymbol{\theta}$

   e.g., the normal distribution $\boldsymbol{\theta} = \{\mu, \sigma^2\}$



2. Use model to predict parameters $\boldsymbol{\theta}$ of probability distribution

Maximum likelihood criterion

- The model now computes different distribution parameters $\boldsymbol{\theta}_i = \mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}]$ for each training input $\mathbf{x}_i$.
- Each observed training output $\mathbf{y}_i$ should have high probability under its corresponding distribution $Pr(\mathbf{y}_i|\boldsymbol{\theta}_i)$
- Hence, we choose the model parameters $\boldsymbol{\phi}$ so that they maximize the combined probability across all I training examples :

$$\begin{aligned}\hat{\boldsymbol{\phi}} &= \underset{\boldsymbol{\phi}}{\operatorname{argmax}} \left[\prod_{i=1}^{I} Pr(\mathbf{y}_i|\mathbf{x}_i)\right] \\ &= \underset{\boldsymbol{\phi}}{\operatorname{argmax}} \left[\prod_{i=1}^{I} Pr(\mathbf{y}_i|\boldsymbol{\theta}_i)\right] \\ &= \underset{\boldsymbol{\phi}}{\operatorname{argmax}} \left[\prod_{i=1}^{I} Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}])\right]\end{aligned}$$

likelihood of the parameters

When we consider this probability as a function of the parameters $\boldsymbol{\phi}$, we call it a likelihood.

Problem:

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmax}} \left[\prod_{i=1}^{I} Pr(\mathbf{y}_i | \mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}]) \right]$$

- The terms in this product might all be small
- The product might get so small that we can't easily represent it

Maximum log likelihood

$$\begin{aligned}\hat{\boldsymbol{\phi}} &= \underset{\boldsymbol{\phi}}{\operatorname{argmax}}\left[\prod_{i=1}^{I} Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i,\boldsymbol{\phi}])\right] \\ &= \underset{\boldsymbol{\phi}}{\operatorname{argmax}}\left[\log\left[\prod_{i=1}^{I} Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i,\boldsymbol{\phi}])\right]\right] \\ &= \underset{\boldsymbol{\phi}}{\operatorname{argmax}}\left[\sum_{i=1}^{I}\log\Big[Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i,\boldsymbol{\phi}])\Big]\right]\end{aligned}$$

Now it's a sum of terms, so doesn't matter so much if the terms are small

Minimizing negative log likelihood

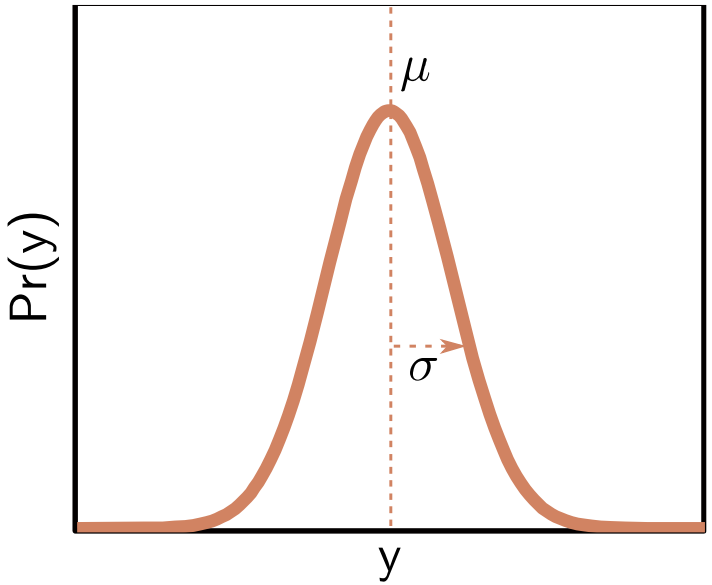
- By convention, we minimize things (i.e., a loss)

$$\begin{aligned}\hat{\boldsymbol{\phi}} &= \underset{\boldsymbol{\phi}}{\operatorname{argmax}} \left[\sum_{i=1}^{I} \log\Big[Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}])\Big]\right] \\ &= \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} \log\Big[Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}])\Big]\right] \\ &= \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \Big[\mathrm{L}[\boldsymbol{\phi}]\Big]\end{aligned}$$

Inference

- But now we predict a probability distribution
- We need an actual prediction (point estimate)
- Find the peak of the probability distribution (i.e., mean for normal)

$$\hat{\mathbf{y}} = \underset{\mathbf{y}}{\operatorname{argmax}} \left[Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]) \right]$$



Loss functions

- Maximum likelihood
- Recipe for loss functions
- Example 1: univariate regression
- Example 2: binary classification
- Example 3: multiclass classification

Recipe for loss functions

1. Choose a suitable probability distribution $Pr(\mathbf{y}|\boldsymbol{\theta})$ that is defined over the domain of the predictions $\mathbf{y}$ and has distribution parameters $\boldsymbol{\theta}$.

Recipe for loss functions

1. Choose a suitable probability distribution $Pr(\mathbf{y}|\boldsymbol{\theta})$ that is defined over the domain of the predictions $\mathbf{y}$ and has distribution parameters $\boldsymbol{\theta}$.
2. Set the machine learning model $\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ to predict one or more of these parameters so $\boldsymbol{\theta} = \mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ and $Pr(\mathbf{y}|\boldsymbol{\theta}) = Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}])$.

Recipe for loss functions

1. Choose a suitable probability distribution $Pr(\mathbf{y}|\boldsymbol{\theta})$ that is defined over the domain of the predictions $\mathbf{y}$ and has distribution parameters $\boldsymbol{\theta}$.
2. Set the machine learning model $\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ to predict one or more of these parameters so $\boldsymbol{\theta} = \mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ and $Pr(\mathbf{y}|\boldsymbol{\theta}) = Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}])$.
3. To train the model, find the network parameters $\hat{\boldsymbol{\phi}}$ that minimize the negative log-likelihood loss function over the training dataset pairs $\{\mathbf{x}_i, \mathbf{y}_i\}$:

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[L[\boldsymbol{\phi}]\right] = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} \log\Big[Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}])\Big] \right]. \tag{5.7}$$

Recipe for loss functions

1. Choose a suitable probability distribution $Pr(\mathbf{y}|\boldsymbol{\theta})$ that is defined over the domain of the predictions $\mathbf{y}$ and has distribution parameters $\boldsymbol{\theta}$.
2. Set the machine learning model $\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ to predict one or more of these parameters so $\boldsymbol{\theta} = \mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ and $Pr(\mathbf{y}|\boldsymbol{\theta}) = Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}])$.
3. To train the model, find the network parameters $\hat{\boldsymbol{\phi}}$ that minimize the negative log-likelihood loss function over the training dataset pairs $\{\mathbf{x}_i, \mathbf{y}_i\}$:

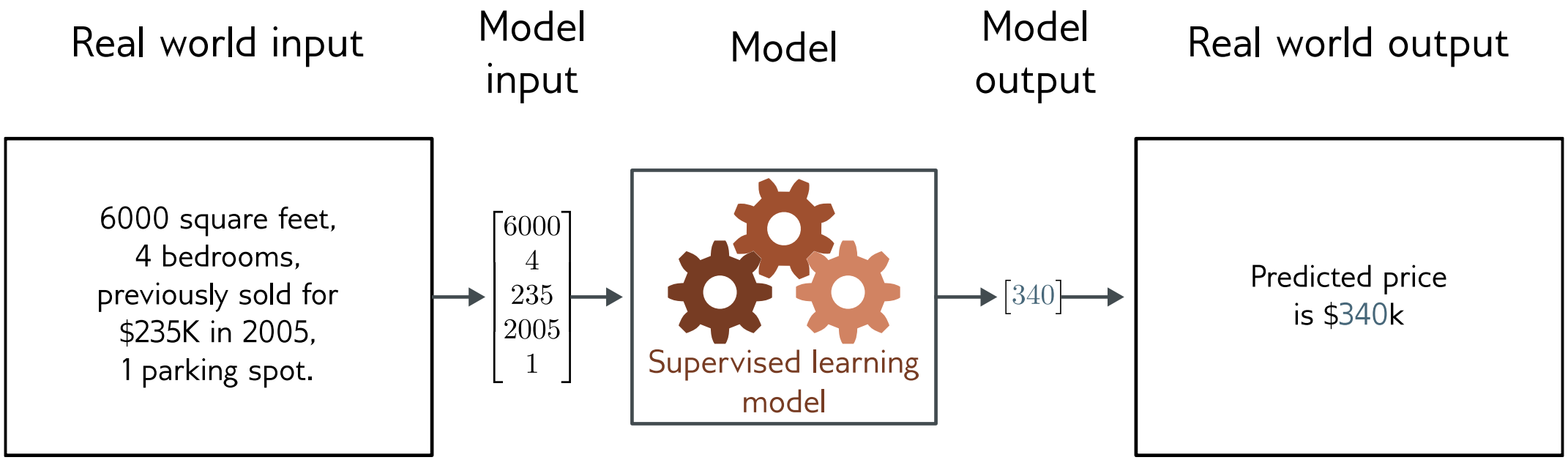
$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[L[\boldsymbol{\phi}]\right] = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} \log\Big[Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}])\Big] \right]. \tag{5.7}$$

4. To perform inference for a new test example $\mathbf{x}$, return either the full distribution $Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}, \hat{\boldsymbol{\phi}}])$ or the maximum of this distribution.

Loss functions

- Maximum likelihood
- Recipe for loss functions
- Example 1: univariate regression
- Example 2: binary classification
- Example 3: multiclass classification

Example 1: univariate regression

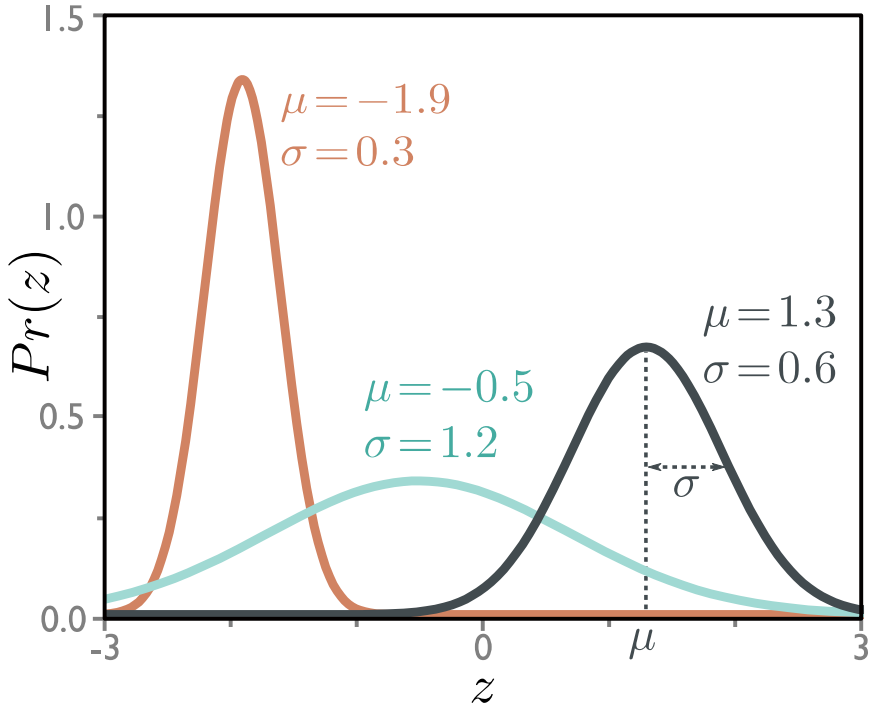


Example 1: univariate regression

1. Choose a suitable probability distribution $Pr(\mathbf{y}|\boldsymbol{\theta})$ that is defined over the domain of the predictions $\mathbf{y}$ and has distribution parameters $\boldsymbol{\theta}$.

- Predict scalar output: $y \in \mathbb{R}$
- Sensible probability distribution:
  - Normal distribution

$$Pr(y|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(y-\mu)^2}{2\sigma^2}\right]$$



Example 1: univariate regression

2. Set the machine learning model $\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ to predict one or more of these parameters so $\boldsymbol{\theta} = \mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ and $Pr(\mathbf{y}|\boldsymbol{\theta}) = Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}])$.

$$Pr(y|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(y-\mu)^2}{2\sigma^2}\right]$$

$$Pr(y|\mathrm{f}[\mathbf{x}, \boldsymbol{\phi}], \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(y-\mathrm{f}[\mathbf{x}, \boldsymbol{\phi}])^2}{2\sigma^2}\right]$$

Example 1: univariate regression

3. To train the model, find the network parameters $\hat{\boldsymbol{\phi}}$ that minimize the negative log-likelihood loss function over the training dataset pairs $\{\mathbf{x}_i, \mathbf{y}_i\}$:

$$L[\boldsymbol{\phi}] = -\sum_{i=1}^{I} \log\left[Pr(y_i|\mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}], \sigma^2)\right]$$

$$= -\sum_{i=1}^{I} \log\left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(y_i - \mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}])^2}{2\sigma^2}\right]\right]$$

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp \left[-\frac{(y_i - \mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}])^2}{2\sigma^2} \right] \right] \right]$$

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp \left[-\frac{(y_i - \mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}])^2}{2\sigma^2} \right] \right] \right]$$

$$= \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} \right] + \log \left[\exp \left[-\frac{(y_i - \mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}])^2}{2\sigma^2} \right] \right] \right]$$

$$\log[a \cdot b] = \log[a] + \log[b]$$

$$
\begin{aligned}
\hat{\boldsymbol{\phi}} &= \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp \left[-\frac{(y_i - \mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}])^2}{2\sigma^2} \right] \right] \right] \\
&= \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} \right] + \log \left[\exp \left[-\frac{(y_i - \mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}])^2}{2\sigma^2} \right] \right] \right] \\
&= \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} \right] - \frac{(y_i - \mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}])^2}{2\sigma^2} \right]
\end{aligned}
$$

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\text{argmin}} \left[-\sum_{i=1}^{I} \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp \left[-\frac{(y_i - \text{f}[\mathbf{x}_i, \boldsymbol{\phi}])^2}{2\sigma^2} \right] \right] \right]$$

$$= \underset{\boldsymbol{\phi}}{\text{argmin}} \left[-\sum_{i=1}^{I} \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} \right] + \log \left[\exp \left[-\frac{(y_i - \text{f}[\mathbf{x}_i, \boldsymbol{\phi}])^2}{2\sigma^2} \right] \right] \right]$$

$$= \underset{\boldsymbol{\phi}}{\text{argmin}} \left[-\sum_{i=1}^{I} \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} \right] - \frac{(y_i - \text{f}[\mathbf{x}_i, \boldsymbol{\phi}])^2}{2\sigma^2} \right]$$

$$= \underset{\boldsymbol{\phi}}{\text{argmin}} \left[-\sum_{i=1}^{I} -\frac{(y_i - \text{f}[\mathbf{x}_i, \boldsymbol{\phi}])^2}{2\sigma^2} \right]$$

We have removed the first because it does not depend on $\boldsymbol{\phi}$.

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp \left[-\frac{(y_i - \mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}])^2}{2\sigma^2} \right] \right] \right]$$

$$= \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} \right] + \log \left[\exp \left[-\frac{(y_i - \mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}])^2}{2\sigma^2} \right] \right] \right]$$

$$= \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} \right] - \frac{(y_i - \mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}])^2}{2\sigma^2} \right]$$

$$= \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} -\frac{(y_i - \mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}])^2}{2\sigma^2} \right]$$

$$= \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[\sum_{i=1}^{I} (y_i - \mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}])^2 \right],$$

We have removed the denominator as this is just a constant scaling factor that does not affect the position of the minimum.

Least squares!

Example 1: univariate regression

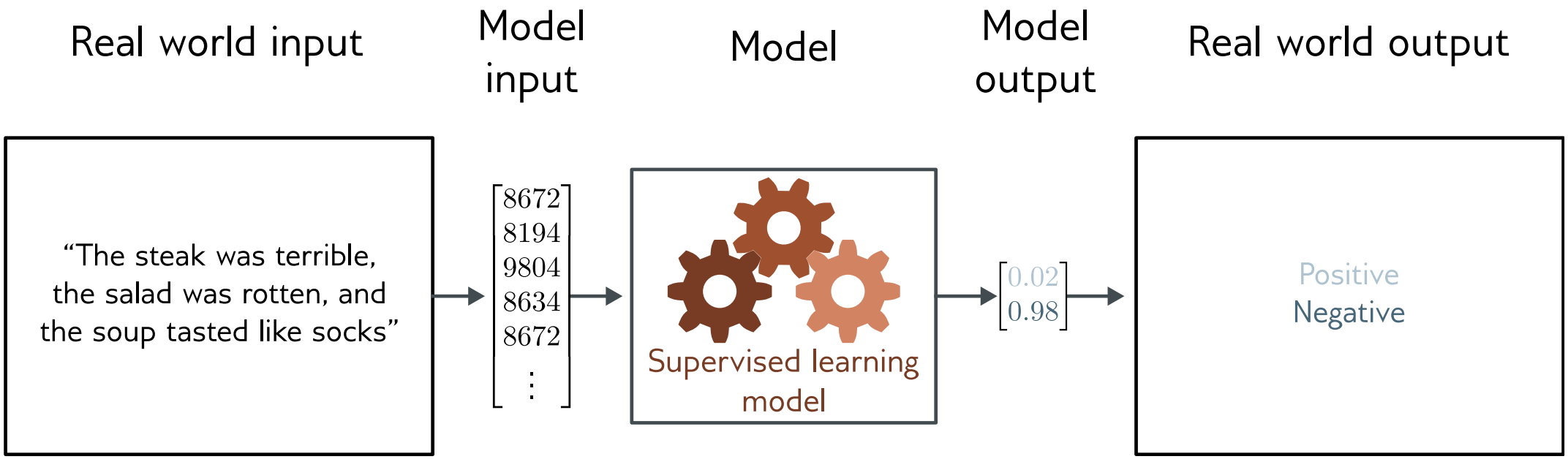
4. To perform inference for a new test example $\mathbf{x}$, return either the full distribution $Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}, \hat{\boldsymbol{\phi}}])$ or the maximum of this distribution.

$$\hat{y} = \mathrm{f}[\mathbf{x}, \hat{\boldsymbol{\phi}}]$$

Loss functions

- Maximum likelihood
- Recipe for loss functions
- Example 1: univariate regression
- Example 2: binary classification
- Example 3: multiclass classification

Example 2: binary classification



- Goal: predict which of two classes $y \in \{0, 1\}$ the input *x* belongs to

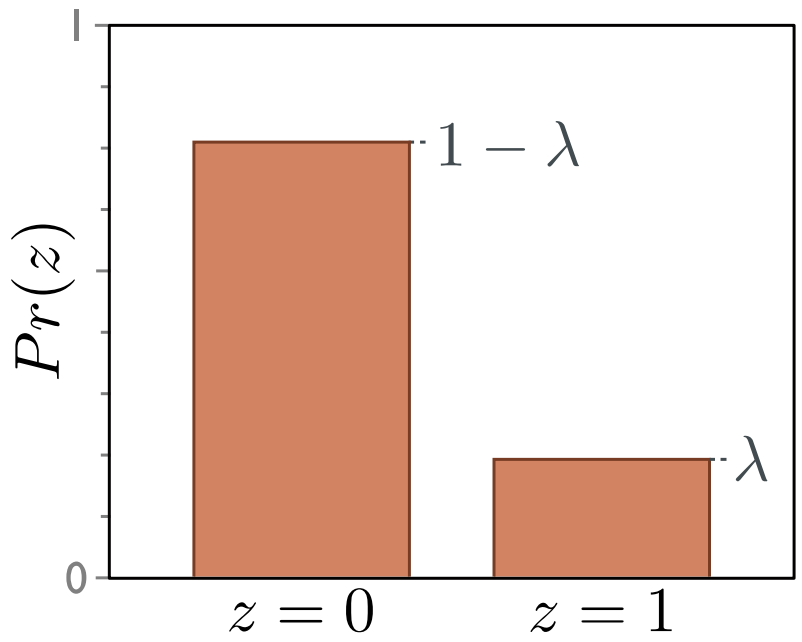
Example 2: binary classification

1. Choose a suitable probability distribution $Pr(\mathbf{y}|\boldsymbol{\theta})$ that is defined over the domain of the predictions $\mathbf{y}$ and has distribution parameters $\boldsymbol{\theta}$.

- Domain: $y \in \{0, 1\}$
- Bernoulli distribution
- One parameter $\lambda \in [0,1]$

$$Pr(y|\lambda) = \begin{cases} 1-\lambda & y = 0 \\ \lambda & y = 1 \end{cases}$$

$$Pr(y|\lambda) = (1-\lambda)^{1-y} \cdot \lambda^{y}$$



Example 2: binary classification

2. Set the machine learning model $\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ to predict one or more of these parameters so $\boldsymbol{\theta} = \mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ and $Pr(\mathbf{y}|\boldsymbol{\theta}) = Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}])$.

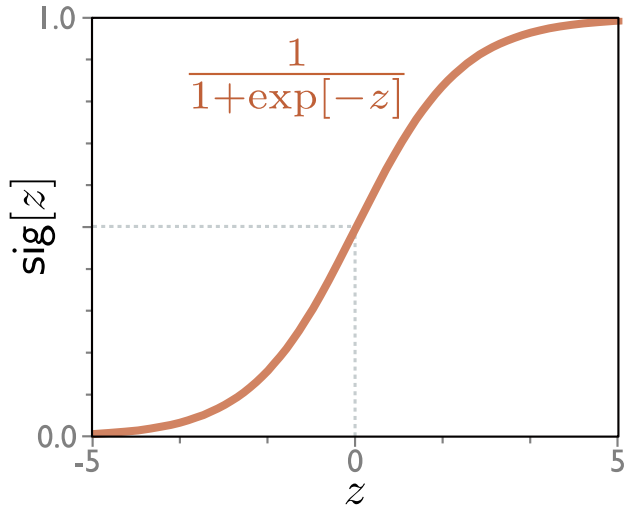
Problem:

- Parameter $\lambda \in [0,1]$
- Output of neural network can be anything, we cannot guarantee that the network output will lie in this range.

Solution:

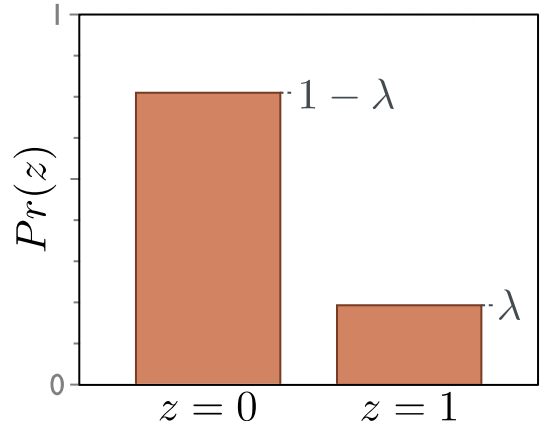
- Pass through function that maps “anything” to [0,1]

$$\text{sig}[z] = \frac{1}{1+\exp[-z]}$$



Example 2: binary classification

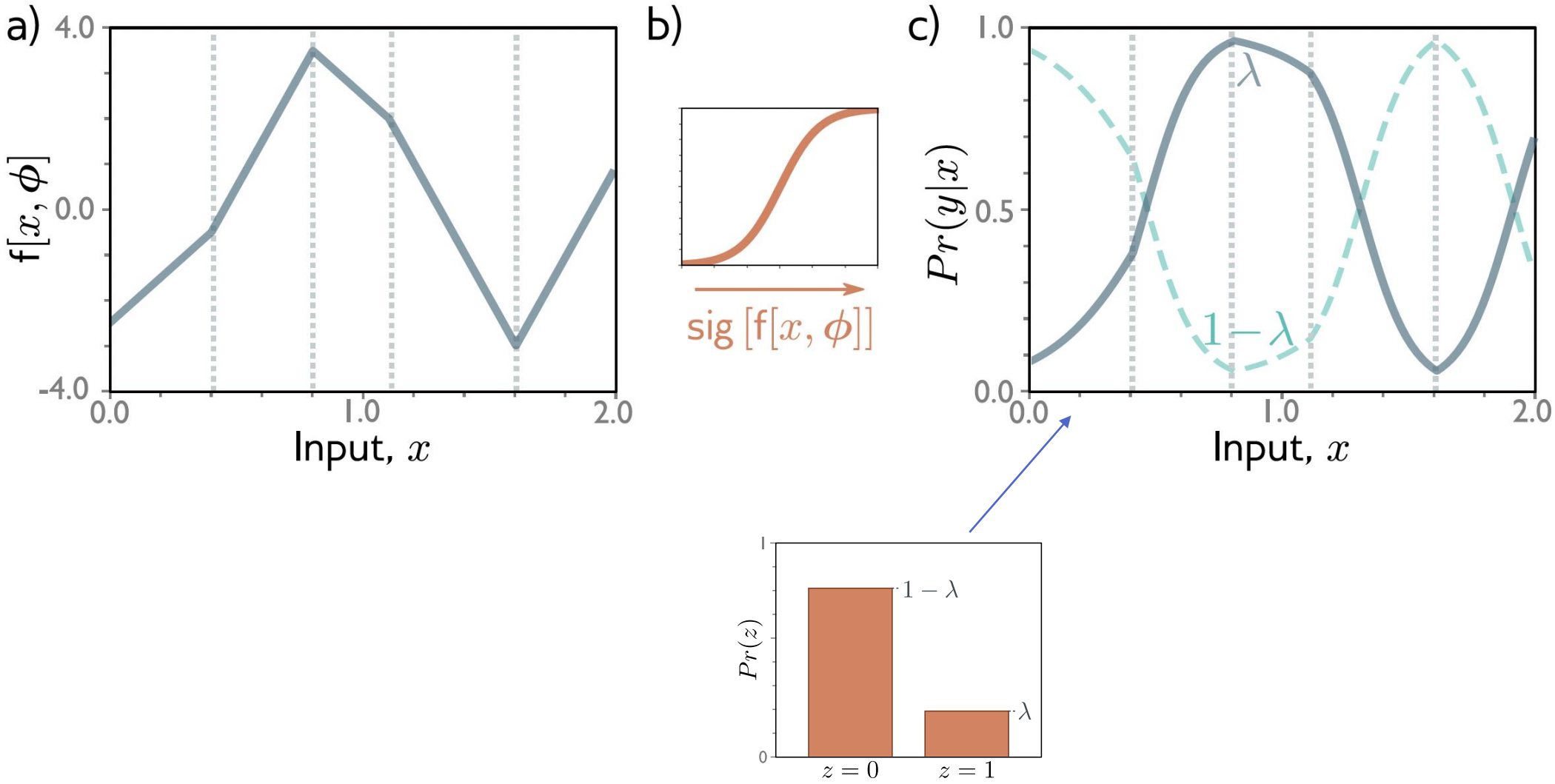
2. Set the machine learning model $\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ to predict one or more of these parameters so $\boldsymbol{\theta} = \mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ and $Pr(\mathbf{y}|\boldsymbol{\theta}) = Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}])$.



$$Pr(y|\lambda) = (1-\lambda)^{1-y} \cdot \lambda^{y}$$

$$Pr(y|\mathbf{x}) = (1-\text{sig}[\text{f}[\mathbf{x}|\boldsymbol{\phi}]])^{1-y} \cdot \text{sig}[\text{f}[\mathbf{x}|\boldsymbol{\phi}]]^{y}$$

Example 2: binary classification



Example 2: binary classification

3. To train the model, find the network parameters $\hat{\boldsymbol{\phi}}$ that minimize the negative log-likelihood loss function over the training dataset pairs $\{\mathbf{x}_i, \mathbf{y}_i\}$:

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[L[\boldsymbol{\phi}]\right] = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} \log\left[Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}])\right]\right]. \tag{5.7}$$

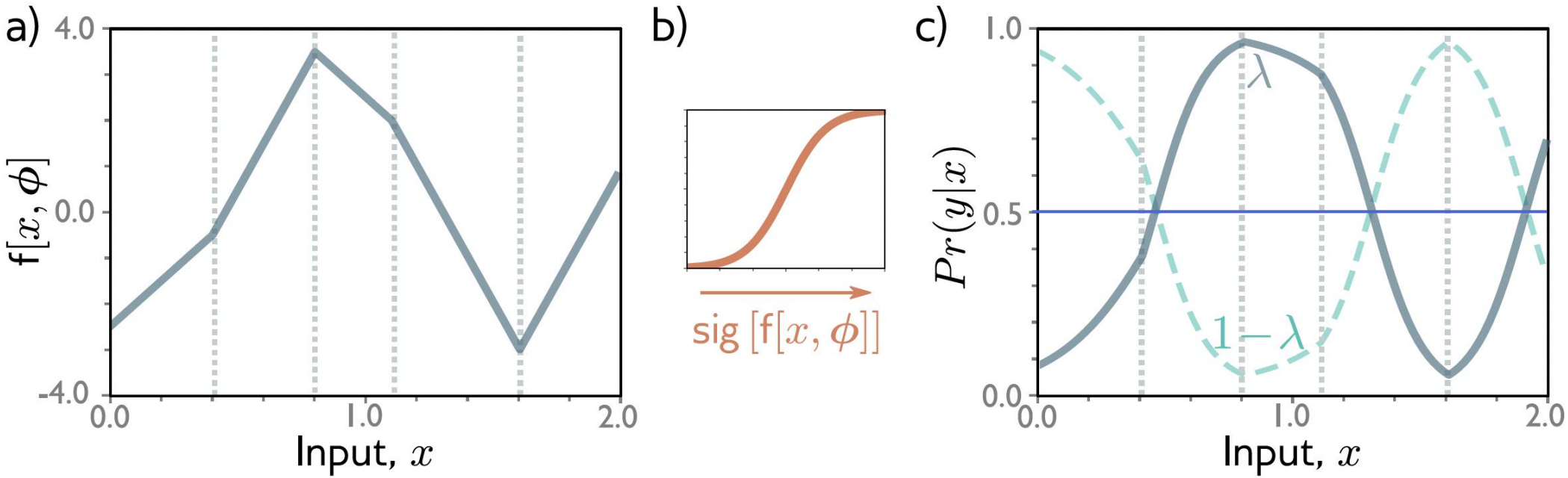
$$Pr(y|\mathbf{x}) = (1 - \text{sig}[\text{f}[\mathbf{x}|\boldsymbol{\phi}]])^{1-y} \cdot \text{sig}[\text{f}[\mathbf{x}|\boldsymbol{\phi}]]^{y}$$

$$L[\boldsymbol{\phi}] = \sum_{i=1}^{I} -(1 - y_i) \log\left[1 - \text{sig}[\text{f}[\mathbf{x}_i|\boldsymbol{\phi}]]\right] - y_i \log\left[\text{sig}[\text{f}[\mathbf{x}_i|\boldsymbol{\phi}]]\right]$$

Binary cross-entropy loss

Example 2: binary classification

4. To perform inference for a new test example $\mathbf{x}$, return either the full distribution $Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}, \hat{\boldsymbol{\phi}}])$ or the maximum of this distribution.

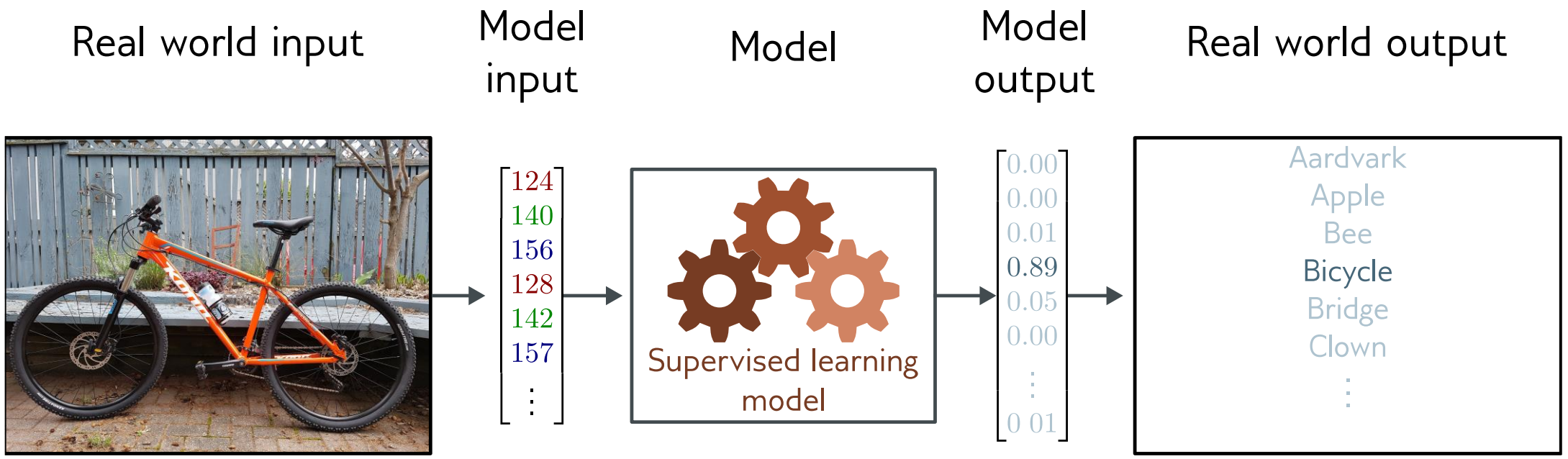


Choose y=1 where λ is greater than 0.5, otherwise 0

Loss functions

- Maximum likelihood
- Recipe for loss functions
- Example 1: univariate regression
- Example 2: binary classification
- Example 3: multiclass classification

Example 3: multiclass classification



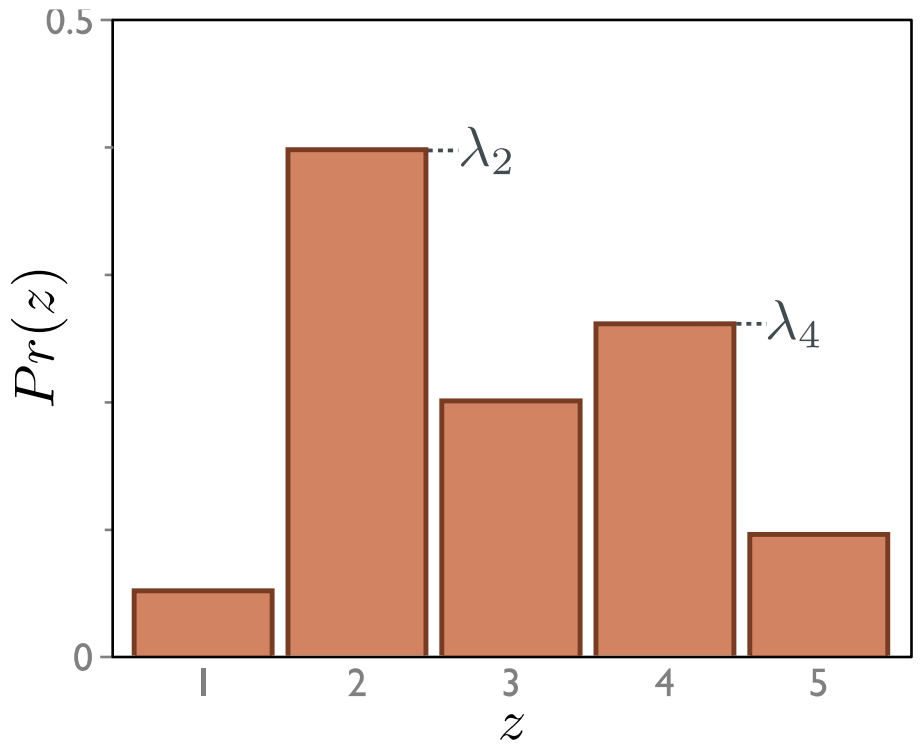
Goal: predict which of K classes $y \in \{1, 2, \ldots, K\}$ the input *x* belongs to

Example 3: multiclass classification

1. Choose a suitable probability distribution $Pr(\mathbf{y}|\boldsymbol{\theta})$ that is defined over the domain of the predictions $\mathbf{y}$ and has distribution parameters $\boldsymbol{\theta}$.

- Domain: $y \in \{1, 2, \ldots, K\}$
- Categorical distribution
- K parameters $\lambda_k \in [0,1]$
- Sum of all parameters = 1

$$Pr(y = k) = \lambda_k$$



Example 3: multiclass classification

2. Set the machine learning model $\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ to predict one or more of these parameters so $\boldsymbol{\theta} = \mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ and $Pr(\mathbf{y}|\boldsymbol{\theta}) = Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}])$.

Problem:

- Parameters $\lambda_k \in [0,1]$, sum to one
- However, output of neural network can be anything

Solution:

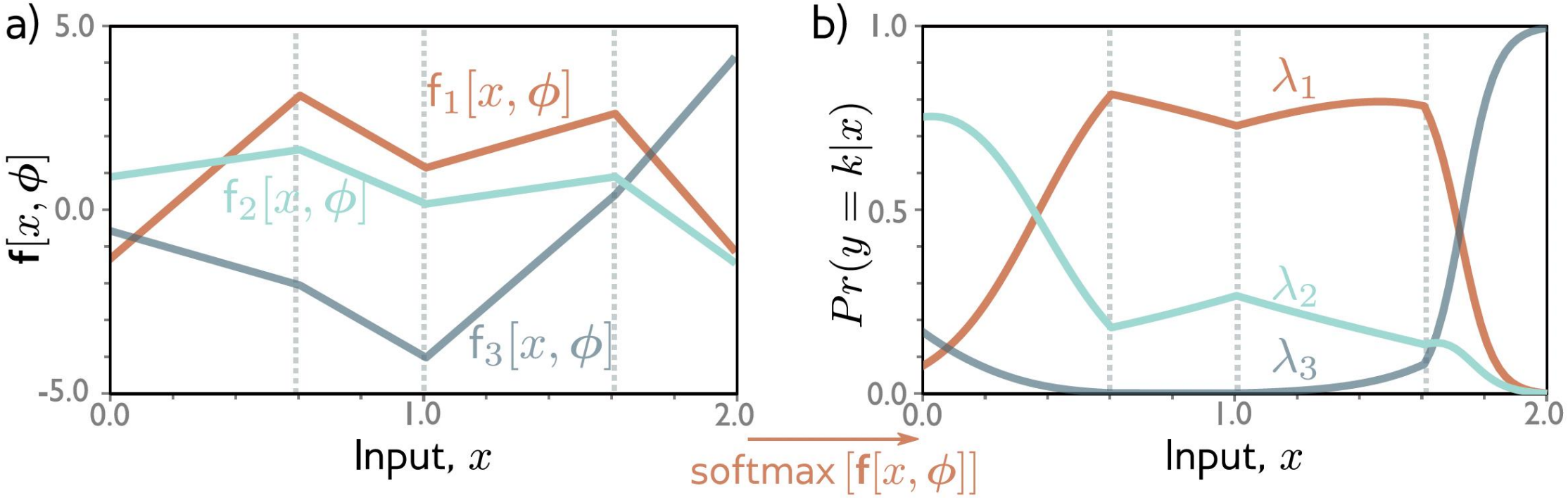
- Pass through function that maps "anything" to [0,1], sum to one

$$\text{softmax}_k[\mathbf{z}] = \frac{\exp[z_k]}{\sum_{k'=1}^{K} \exp[z_{k'}]}$$

$$Pr(y = k|\mathbf{x}) = \text{softmax}_k[\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]]$$

- the exponential functions ensure positivity
- the sum in the denominator ensures that the *K* numbers sum to one

Example 3: multiclass classification



$$Pr(y = k|\mathbf{x}) = \text{softmax}_k[\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]]$$

Example 3: multiclass classification

3. To train the model, find the network parameters $\hat{\boldsymbol{\phi}}$ that minimize the negative log-likelihood loss function over the training dataset pairs $\{\mathbf{x}_i, \mathbf{y}_i\}$:

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[L[\boldsymbol{\phi}] \right] = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} \log \left[Pr(\mathbf{y}_i | \mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}]) \right] \right]. \tag{5.7}$$

$$L[\boldsymbol{\phi}] = -\sum_{i=1}^{I} \log \left[\text{softmax}_{y_i} \left[\mathbf{f}\left[\mathbf{x}_i, \boldsymbol{\phi}\right] \right] \right]$$

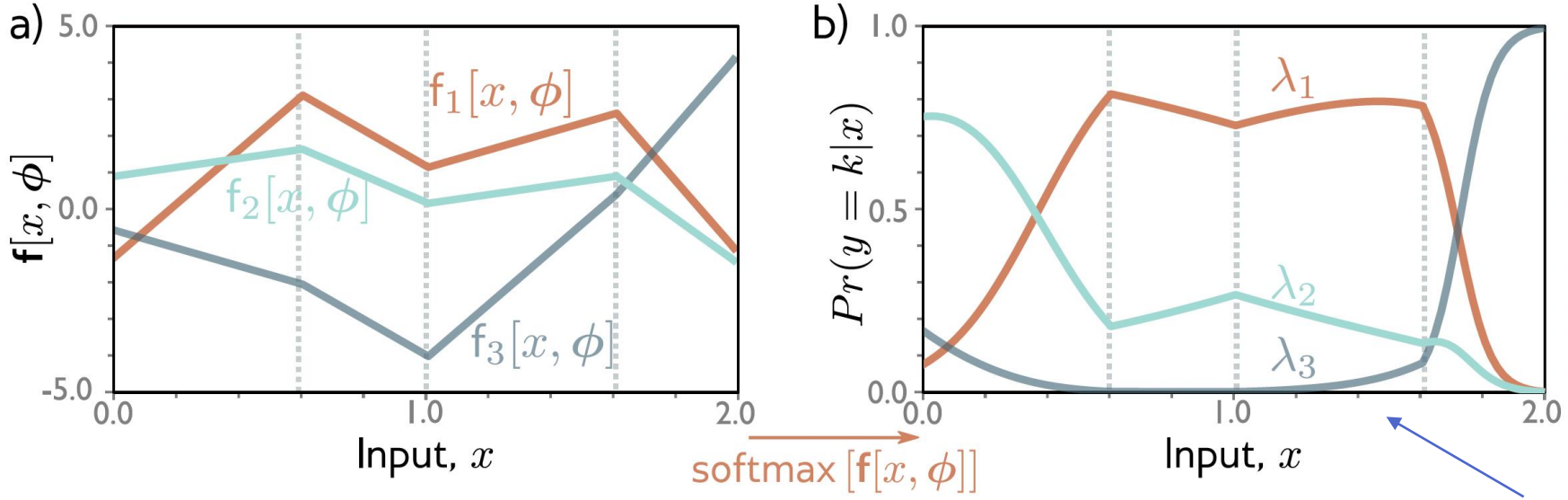
$$= -\sum_{i=1}^{I} \text{f}_{y_i} \left[\mathbf{x}_i, \boldsymbol{\phi} \right] - \log \left[\sum_{k=1}^{K} \exp \left[\text{f}_k \left[\mathbf{x}_i, \boldsymbol{\phi} \right] \right] \right]$$

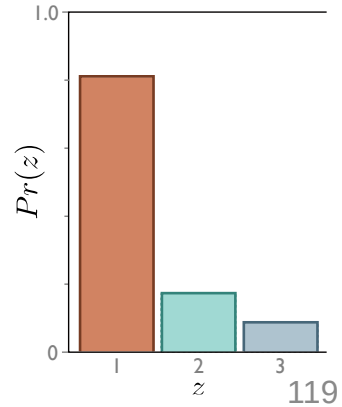
$$\text{softmax}_k[\mathbf{z}] = \frac{\exp[z_k]}{\sum_{k'=1}^{K} \exp[z_{k'}]}$$

Multiclass cross-entropy loss

Example 3: multiclass classification

4. To perform inference for a new test example $\mathbf{x}$, return either the full distribution $Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}, \hat{\boldsymbol{\phi}}])$ or the maximum of this distribution.





Choose the class with the largest probability