Machine Learning

Neural Network 1

Hiroshi Shimodaira and Hao Tang

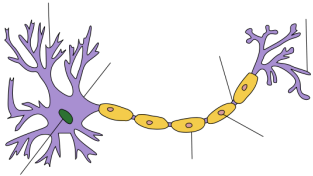
2025

Ver. 1.0

Topics - you should be able to explain after this week

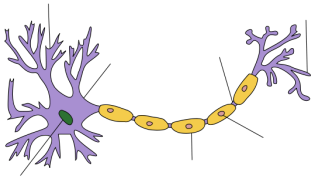
- Perceptron
- Perceptron learning algorithm (perceptron error correction algorithm)
- Linearly separable vs linearly non-separable
- Logical operations with perceptron
- Multilayer perceptron (MLP)
- Activation functions
- Universal approximation theorem

Background of Perceptron

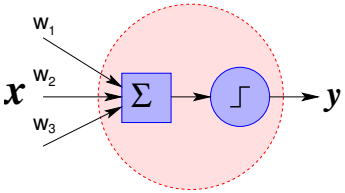


(https://en.wikipedia.org/wiki/File:Neuron_Hand-tuned.svg)

Background of Perceptron

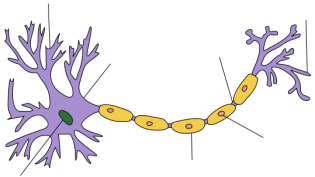


(https://en.wikipedia.org/wiki/File:Neuron_Hand-tuned.svg)

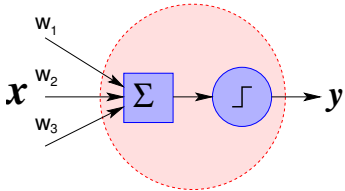


(a) function unit

Background of Perceptron



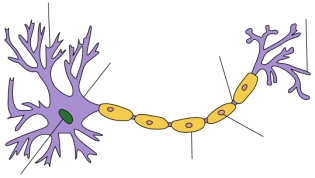
(https://en.wikipedia.org/wiki/File:Neuron_Hand-tuned.svg)



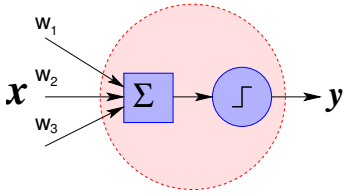
(a) function unit

1940s Warren McCulloch and Walter Pitts : 'threshold logic'
Donald Hebb : 'Hebbian learning'

Background of Perceptron



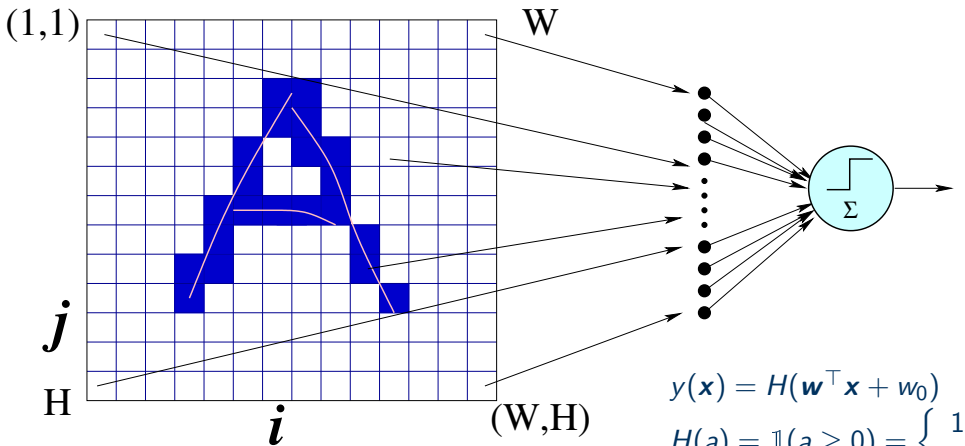
(https://en.wikipedia.org/wiki/File:Neuron_Hand-tuned.svg)



(a) function unit

1940s Warren McCulloch and Walter Pitts : 'threshold logic'
Donald Hebb : 'Hebbian learning'
1957 Frank Rosenblatt : 'Perceptron'

Character recognition with Perceptron



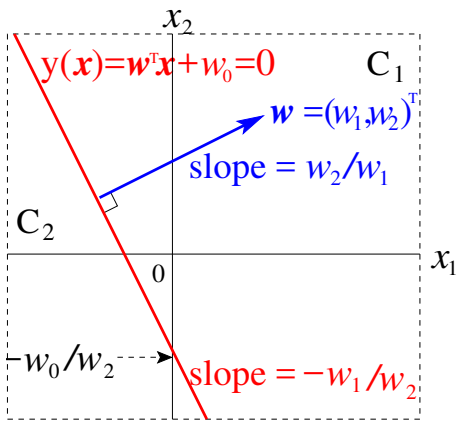
$$y(\boldsymbol{x}) = H(\boldsymbol{w}^\top \boldsymbol{x} + w_0)$$

$$H(a) = \mathbb{1}(a \geq 0) = \begin{cases} 1, & \text{if } a \geq 0, \\ 0, & \text{if } a < 0 \end{cases}$$

Heaviside step function

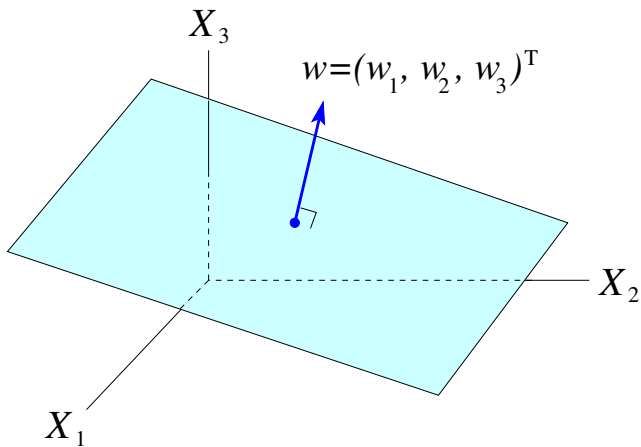
Decision boundary of linear discriminant (2D)

$$y(\boldsymbol{x}) = w_1 x_1 + w_2 x_2 + w_0 = 0 \quad \left(x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2}, \text{ when } w_2 \neq 0\right)$$



Decision boundary of linear discriminant (3D)

$$y(\boldsymbol{x}) = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_0 = 0$$



Decision boundary of linear discriminants

- Decision boundary:

$$y(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + w_0 = 0$$

Dimension	Decision boundary	
2	line	$w_1 x_1 + w_2 x_2 + w_0 = 0$
3	plane	$w_1 x_1 + w_2 x_2 + w_3 x_3 + w_0 = 0$
d	hyperplane	$(\sum_{i=1}^{d} w_i x_i) + w_0 = 0$

NB: $\boldsymbol{w}$ is a normal vector to the hyperplane

Training of Perceptron

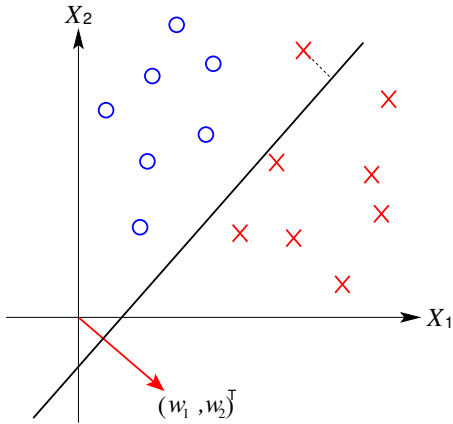
- A discriminant for a two-class problem:

$$y(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + w_0$$

Training of Perceptron

- A discriminant for a two-class problem:

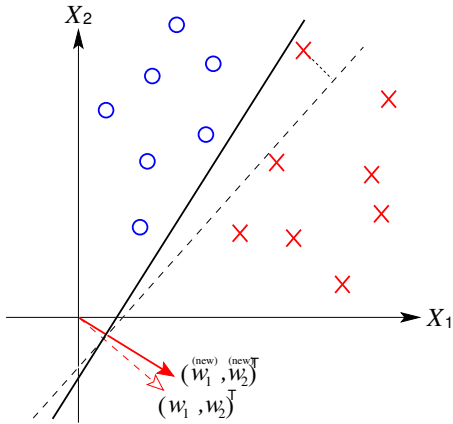
$$y(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + w_0$$



Training of Perceptron

- A discriminant for a two-class problem:

$$y(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + w_0$$



Perceptron error correction algorithm

$$a(\dot{\mathbf{x}}) = \boldsymbol{w}^\top \boldsymbol{x} + w_0 \;=\; \dot{\mathbf{w}}^\top \dot{\mathbf{x}}$$

where $\dot{\mathbf{w}} = (w_0, \boldsymbol{w}^\top)^\top,\ \dot{\mathbf{x}} = (1, \boldsymbol{x}^\top)^\top$

Let's just use $\boldsymbol{w}$ and $\boldsymbol{x}$ to denote $\dot{\mathbf{w}}$ and $\dot{\mathbf{x}}$ from now on!

Perceptron error correction algorithm

$$a(\dot{\mathbf{x}}) = \boldsymbol{w}^\top \boldsymbol{x} + w_0 \;=\; \dot{\mathbf{w}}^\top \dot{\mathbf{x}}$$

where $\dot{\mathbf{w}} = (w_0, \boldsymbol{w}^\top)^\top$, $\dot{\mathbf{x}} = (1, \boldsymbol{x}^\top)^\top$

Let's just use $\boldsymbol{w}$ and $\boldsymbol{x}$ to denote $\dot{\mathbf{w}}$ and $\dot{\mathbf{x}}$ from now on!

$$y(\boldsymbol{x}) = g(\, a(\boldsymbol{x})\,) \;=\; g(\, \boldsymbol{w}^\top \boldsymbol{x}\,)$$

Perceptron error correction algorithm

$$a(\dot{\mathbf{x}}) = \boldsymbol{w}^\top \boldsymbol{x} + w_0 \;=\; \dot{\mathbf{w}}^\top \dot{\mathbf{x}}$$

where $\dot{\mathbf{w}} = (w_0, \boldsymbol{w}^\top)^\top$, $\dot{\mathbf{x}} = (1, \boldsymbol{x}^\top)^\top$

Let's just use $\boldsymbol{w}$ and $\boldsymbol{x}$ to denote $\dot{\mathbf{w}}$ and $\dot{\mathbf{x}}$ from now on!

$$y(\boldsymbol{x}) = g(\, a(\boldsymbol{x})\,) \;=\; g(\, \boldsymbol{w}^\top \boldsymbol{x}\,)$$

where $g(a) = \mathbb{1}(a \geq 0) = \begin{cases} 1, & \text{if } a \geq 0, \\ 0, & \text{if } a < 0 \end{cases}$

$g(a)$: activation / transfer function. $g(a) = H(a)$ for perceptron

Perceptron error correction algorithm

$$a(\dot{\mathbf{x}}) = \boldsymbol{w}^\top \boldsymbol{x} + w_0 \;=\; \dot{\mathbf{w}}^\top \dot{\mathbf{x}}$$

where $\dot{\mathbf{w}} = (w_0, \boldsymbol{w}^\top)^\top$, $\dot{\mathbf{x}} = (1, \boldsymbol{x}^\top)^\top$

Let's just use $\boldsymbol{w}$ and $\boldsymbol{x}$ to denote $\dot{\mathbf{w}}$ and $\dot{\mathbf{x}}$ from now on!

$$y(\boldsymbol{x}) = g(\, a(\boldsymbol{x})\,) \;=\; g(\, \boldsymbol{w}^\top \boldsymbol{x}\,)$$

where $g(a) = \mathbb{1}(a \geq 0) = \begin{cases} 1, & \text{if } a \geq 0, \\ 0, & \text{if } a < 0 \end{cases}$

$g(a)$: activation / transfer function. $g(a) = H(a)$ for perceptron

- Training set : $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, $\quad y_i \in \{0, 1\}$: target value or label
- Initialise $\boldsymbol{w}$

Perceptron error correction algorithm

$$a(\dot{\mathbf{x}}) = \boldsymbol{w}^\top \boldsymbol{x} + w_0 \;=\; \dot{\mathbf{w}}^\top \dot{\mathbf{x}}$$

where $\dot{\mathbf{w}} = (w_0, \boldsymbol{w}^\top)^\top$, $\dot{\mathbf{x}} = (1, \boldsymbol{x}^\top)^\top$

Let's just use $\boldsymbol{w}$ and $\boldsymbol{x}$ to denote $\dot{\mathbf{w}}$ and $\dot{\mathbf{x}}$ from now on!

$$y(\boldsymbol{x}) = g(\, a(\boldsymbol{x})\,) \;=\; g(\, \boldsymbol{w}^\top \boldsymbol{x}\,)$$

where $g(a) = \mathbb{1}(a \geq 0) = \begin{cases} 1, & \text{if } a \geq 0, \\ 0, & \text{if } a < 0 \end{cases}$

$g(a)$: activation / transfer function. $g(a) = H(a)$ for perceptron

- Training set : $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, $\quad y_i \in \{0, 1\}$: target value or label
- Initialise $\boldsymbol{w}$
- Modify $\boldsymbol{w}$ if $\mathbf{x}_i$ was misclassified

$$\boldsymbol{w}^{(\text{new})} \;\leftarrow\; \boldsymbol{w} + \eta\,(y_i - y(\mathbf{x}_i))\,\mathbf{x}_i \qquad (0 < \eta < 1)$$

learning rate

Perceptron error correction algorithm

$$a(\dot{\mathbf{x}}) = \boldsymbol{w}^\top \boldsymbol{x} + w_0 \;=\; \dot{\mathbf{w}}^\top \dot{\mathbf{x}}$$

where $\dot{\mathbf{w}} = (w_0, \boldsymbol{w}^\top)^\top$, $\dot{\mathbf{x}} = (1, \boldsymbol{x}^\top)^\top$

Let's just use $\boldsymbol{w}$ and $\boldsymbol{x}$ to denote $\dot{\mathbf{w}}$ and $\dot{\mathbf{x}}$ from now on!

$$y(\boldsymbol{x}) = g(\, a(\boldsymbol{x})\,) \;=\; g(\, \boldsymbol{w}^\top \boldsymbol{x}\,)$$

where $g(a) = \mathbb{1}(a \geq 0) = \begin{cases} 1, & \text{if } a \geq 0, \\ 0, & \text{if } a < 0 \end{cases}$

$g(a)$: activation / transfer function. $g(a) = H(a)$ for perceptron

- Training set : $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, $\quad y_i \in \{0, 1\}$: target value or label
- Initialise $\boldsymbol{w}$
- Modify $\boldsymbol{w}$ if $\mathbf{x}_i$ was misclassified

$$\boldsymbol{w}^{(\text{new})} \;\leftarrow\; \boldsymbol{w} + \eta\,(y_i - y(\mathbf{x}_i))\,\mathbf{x}_i \qquad (0 < \eta < 1)$$

learning rate

NB:

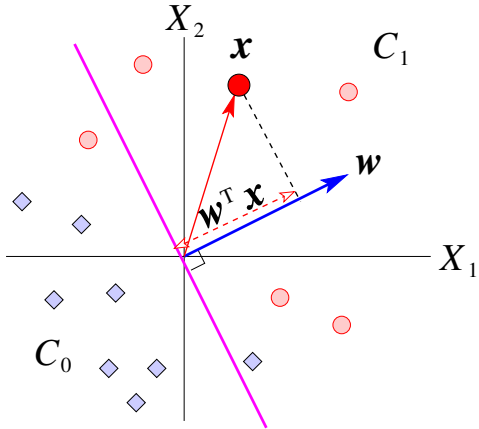
$$(\boldsymbol{w}^{(\text{new})})^\top \mathbf{x}_i = \boldsymbol{w}^\top \mathbf{x}_i + \eta\,(y_i - y(\mathbf{x}_i))\,\|\mathbf{x}_i\|^2$$

Geometry of Perceptron error correction

$$y(\mathbf{x}_i) = g(\boldsymbol{w}^\top \mathbf{x}_i)$$
$$\boldsymbol{w}^{(\text{new})} \leftarrow \boldsymbol{w} + \eta\,(y_i - y(\mathbf{x}_i))\,\mathbf{x}_i \qquad (0 < \eta < 1)$$

$y_i - y(\mathbf{x}_i)$		$y(\mathbf{x}_i)$ 0	$y(\mathbf{x}_i)$ 1
y_i	0	0	-1
	1	1	0

$$\boldsymbol{w}^\top \boldsymbol{x} = \|\boldsymbol{w}\|\|\boldsymbol{x}\| \cos\theta$$

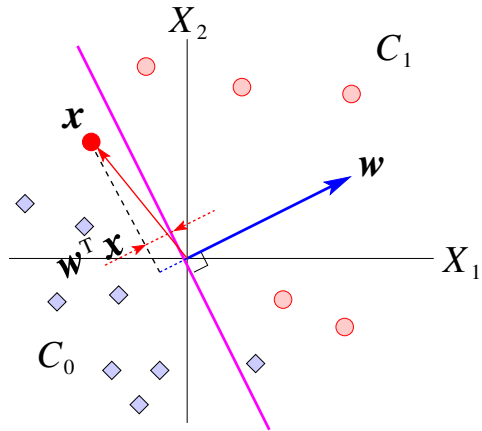


Geometry of Perceptron error correction (*cont.*)

$$y(\mathbf{x}_i) = g(\boldsymbol{w}^\top \mathbf{x}_i)$$
$$\boldsymbol{w}^{(\text{new})} \leftarrow \boldsymbol{w} + \eta\,(y_i - y(\mathbf{x}_i))\,\mathbf{x}_i \qquad (0 < \eta < 1)$$

$y_i - y(\mathbf{x}_i)$		$y(\mathbf{x}_i)$	
		0	1
y_i	0	0	-1
	1	**1**	0

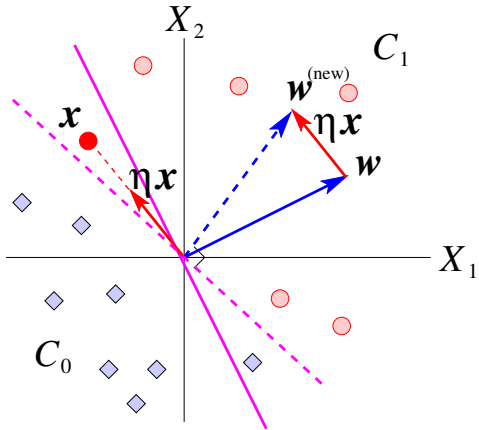
$$\boldsymbol{w}^\top \boldsymbol{x} = \|\boldsymbol{w}\|\|\boldsymbol{x}\| \cos\theta$$



Geometry of Perceptron error correction (*cont.*)

$$y(\mathbf{x}_i) = g(\boldsymbol{w}^\top \mathbf{x}_i)$$
$$\boldsymbol{w}^{(\text{new})} \leftarrow \boldsymbol{w} + \eta\,(y_i - y(\mathbf{x}_i))\,\mathbf{x}_i \qquad (0 < \eta < 1)$$

$y_i - y(\mathbf{x}_i)$		$y(\mathbf{x}_i)$ **0**	1
y_i	0	0	-1
	1	**1**	0



$$\boldsymbol{w}^\top \boldsymbol{x} = \|\boldsymbol{w}\|\|\boldsymbol{x}\| \cos\theta$$

The Perceptron learning algorithm

Incremental (online) Perceptron algorithm:

for $i = 1, \ldots, N$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \left(y_i - y(\mathbf{x}_i) \right) \mathbf{x}_i$$

The Perceptron learning algorithm

Incremental (online) Perceptron algorithm:

$$
\begin{aligned}
&\text{for } i = 1, \ldots, N \\
&\quad \boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \left(y_i - y(\mathbf{x}_i) \right) \mathbf{x}_i
\end{aligned}
$$

Batch Perceptron algorithm:

$$
\begin{aligned}
&\boldsymbol{v}_{sum} = \mathbf{0} \\
&\text{for } i = 1, \ldots, N \\
&\quad \boldsymbol{v}_{sum} = \boldsymbol{v}_{sum} + \left(y_i - y(\mathbf{x}_i) \right) \mathbf{x}_i \\
&\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \, \boldsymbol{v}_{sum}
\end{aligned}
$$

The Perceptron learning algorithm

Incremental (online) Perceptron algorithm:

for $i = 1, \ldots, N$
$\quad \boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \left(y_i - y(\mathbf{x}_i)\right) \mathbf{x}_i$

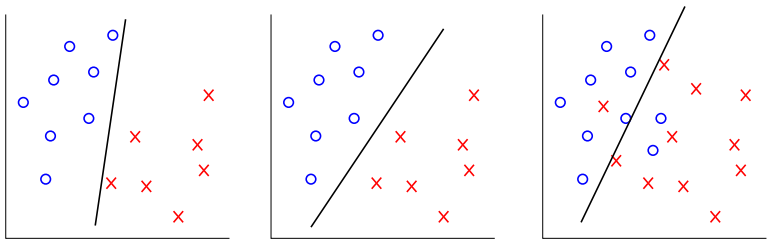
Batch Perceptron algorithm:

$\boldsymbol{v}_{sum} = \mathbf{0}$
for $i = 1, \ldots, N$
$\quad \boldsymbol{v}_{sum} = \boldsymbol{v}_{sum} + \left(y_i - y(\mathbf{x}_i)\right) \mathbf{x}_i$
$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \, \boldsymbol{v}_{sum}$

What about convergence?

The Perceptron learning algorithm terminates if training samples are linearly separable.

Linearly separable vs linearly non-separable

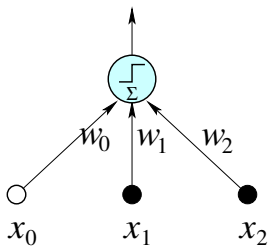


(a–1) (a–2) (b)

Linearly separable Linearly non–separable

Perceptron structures and decision boundaries

$$
\begin{aligned}
y(\boldsymbol{x}) &= g(a(\boldsymbol{x})) && \boldsymbol{w} = (w_0, w_1, \ldots, w_d)^\top \\
&= g(\,\boldsymbol{w}^\top \boldsymbol{x}\,) && \boldsymbol{x} = (1, x_1, \ldots, x_d)^\top \\
&\text{where } g(a) = \begin{cases} 1, & \text{if } a \geq 0, \\ 0, & \text{if } a < 0 \end{cases} && \text{cf. sigmoid function}
\end{aligned}
$$

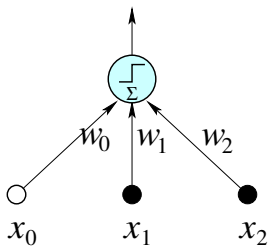


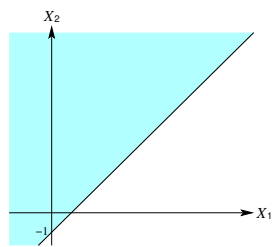
Perceptron structures and decision boundaries

$$y(\boldsymbol{x}) = g(a(\boldsymbol{x})) \qquad \boldsymbol{w} = (w_0, w_1, \ldots, w_d)^\top$$

$$= g(\boldsymbol{w}^\top \boldsymbol{x}) \qquad \boldsymbol{x} = (1, x_1, \ldots, x_d)^\top$$

$$\text{where } g(a) = \begin{cases} 1, & \text{if } a \geq 0, \\ 0, & \text{if } a < 0 \end{cases} \qquad \text{cf. sigmoid function}$$





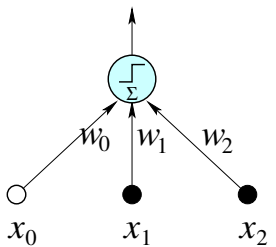
Perceptron structures and decision boundaries

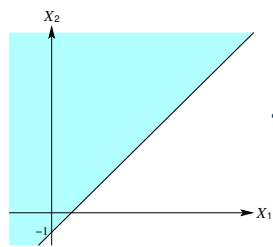
$$y(\boldsymbol{x}) = g(a(\boldsymbol{x})) \qquad \boldsymbol{w} = (w_0, w_1, \ldots, w_d)^\top$$

$$= g(\boldsymbol{w}^\top \boldsymbol{x}) \qquad \boldsymbol{x} = (1, x_1, \ldots, x_d)^\top$$

$$\text{where } g(a) = \begin{cases} 1, & \text{if } a \geq 0, \\ 0, & \text{if } a < 0 \end{cases}$$

cf. sigmoid function





$$x_2 > x_1 - 1$$

$$a(\boldsymbol{x}) = 1 - x_1 + x_2$$

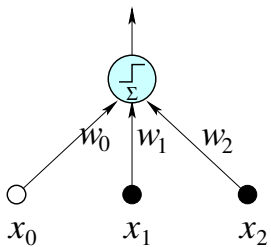
$$= w_0 + w_1 x_1 + w_2 x_2$$

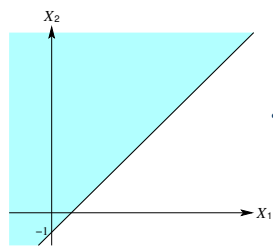
$$w_0 = 1, w_1 = -1, w_2 = 1$$

Perceptron structures and decision boundaries

$$
\begin{aligned}
y(\boldsymbol{x}) &= g(a(\boldsymbol{x})) \qquad & \boldsymbol{w} &= (w_0, w_1, \ldots, w_d)^\top \\
&= g(\boldsymbol{w}^\top \boldsymbol{x}) \qquad & \boldsymbol{x} &= (1, x_1, \ldots, x_d)^\top
\end{aligned}
$$

where $g(a) = \begin{cases} 1, & \text{if } a \geq 0, \\ 0, & \text{if } a < 0 \end{cases}$ cf. sigmoid function





$x_2 > x_1 - 1$

$$
\begin{aligned}
a(\boldsymbol{x}) &= 1 - x_1 + x_2 \\
&= w_0 + w_1 x_1 + w_2 x_2
\end{aligned}
$$

$w_0 = 1, w_1 = -1, w_2 = 1$

NB: A one node/neuron constructs a decision boundary, which splits the input space into two regions

Perceptron as a logical function

NOT

x_1	y
0	1
1	0

OR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

NAND

x_1	x_2	y
0	0	1
0	1	1
1	0	1
1	1	0

XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

Perceptron as a logical function

NOT

x_1	y
0	1
1	0

OR

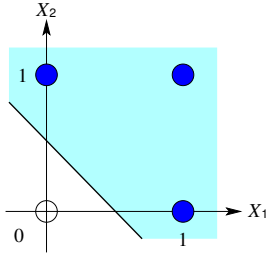
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

NAND

x_1	x_2	y
0	0	1
0	1	1
1	0	1
1	1	0

XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



Perceptron as a logical function

NOT

x_1	y
0	1
1	0

OR

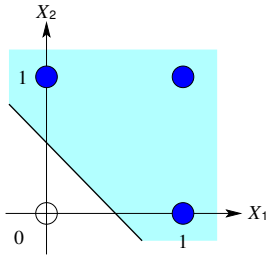
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

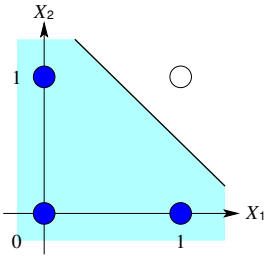
NAND

x_1	x_2	y
0	0	1
0	1	1
1	0	1
1	1	0

XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0





Perceptron as a logical function

NOT

x_1	y
0	1
1	0

OR

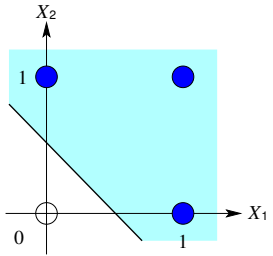
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

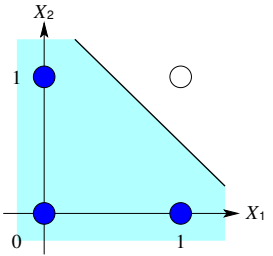
NAND

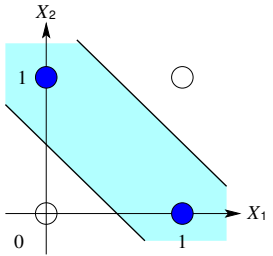
x_1	x_2	y
0	0	1
0	1	1
1	0	1
1	1	0

XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0







Perceptron as a logical function

NOT

x_1	y
0	1
1	0

OR

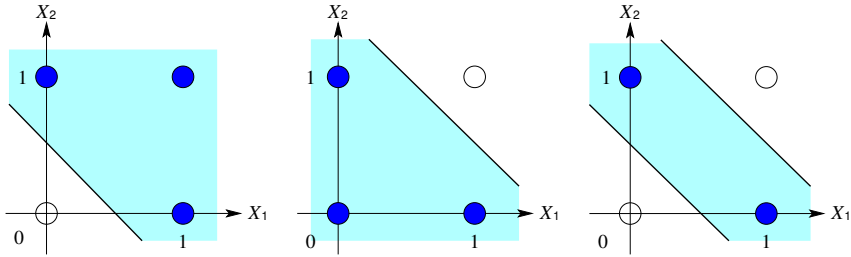
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

NAND

x_1	x_2	y
0	0	1
0	1	1
1	0	1
1	1	0

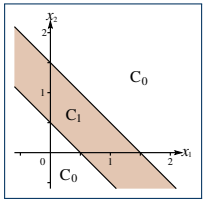
XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

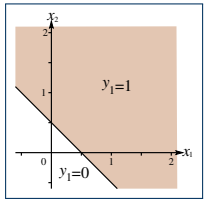


Question: find the weights for each function

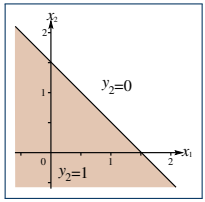
A perceptron for XOR

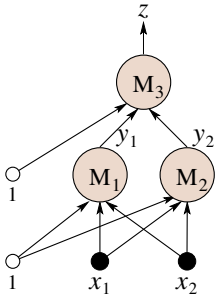


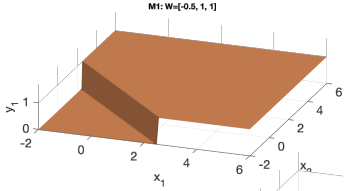
$\Leftarrow$

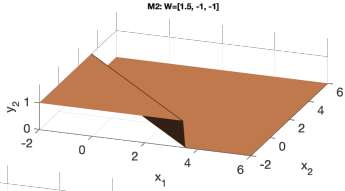


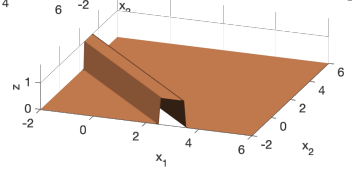
AND



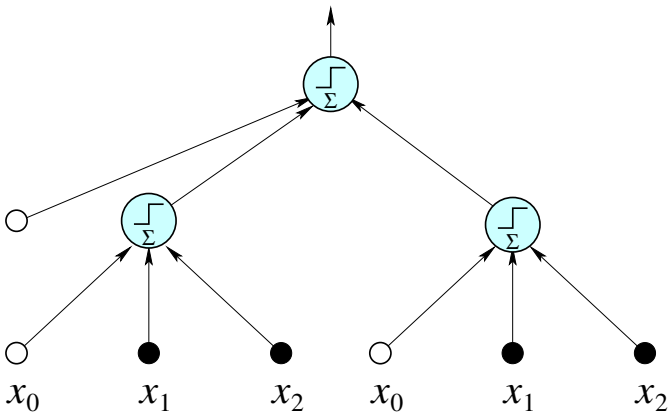




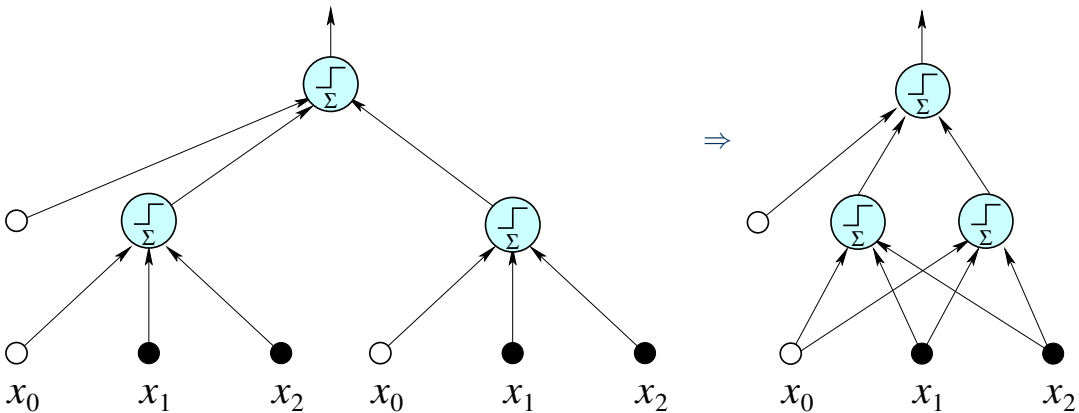




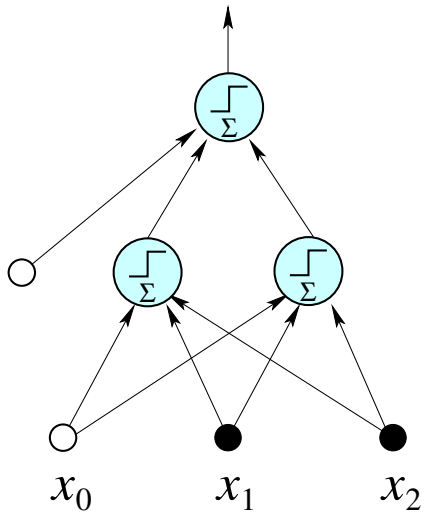
Perceptron structures and decision boundaries (*cont.*)



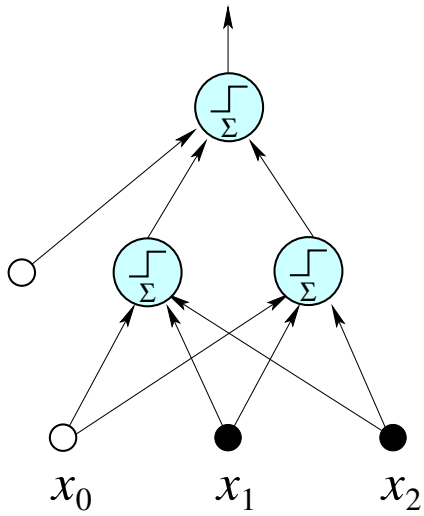
Perceptron structures and decision boundaries (*cont.*)

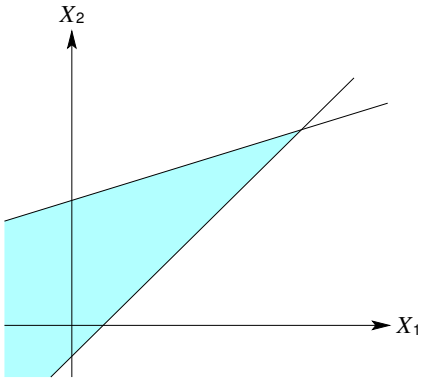


Perceptron structures and decision boundaries (*cont.*)

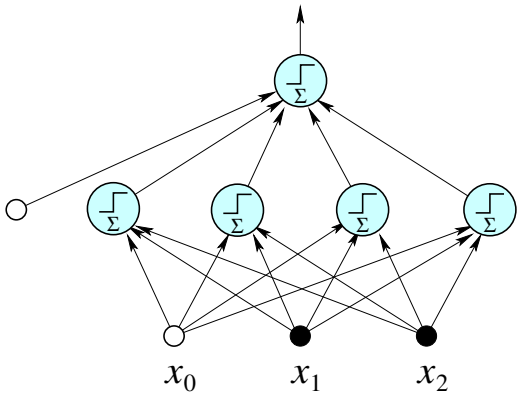


Perceptron structures and decision boundaries (*cont.*)

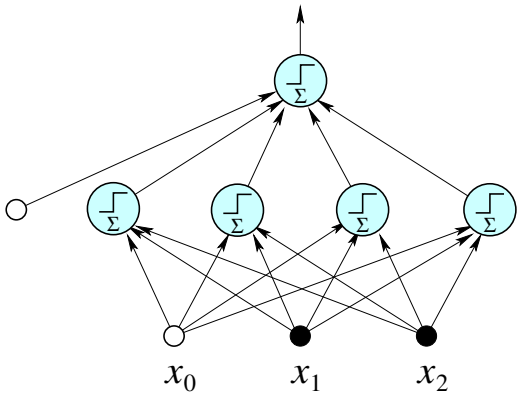


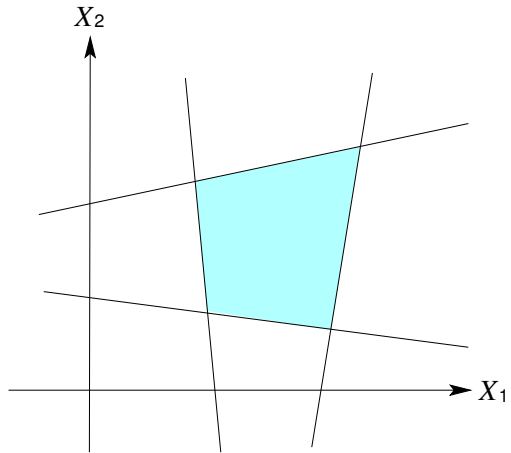


Perceptron structures and decision boundaries (*cont.*)

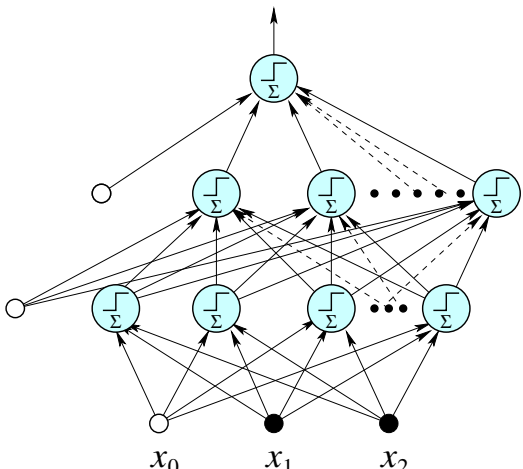


Perceptron structures and decision boundaries (*cont.*)

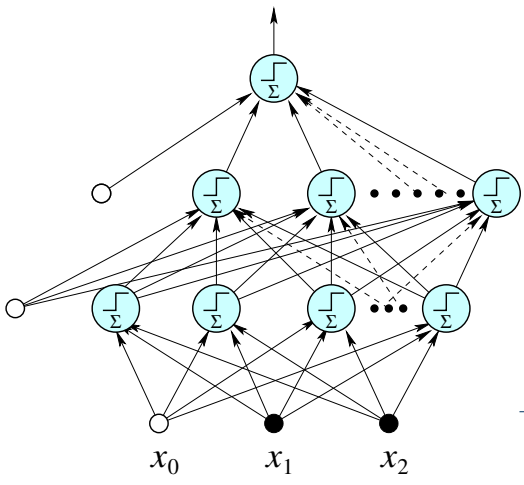


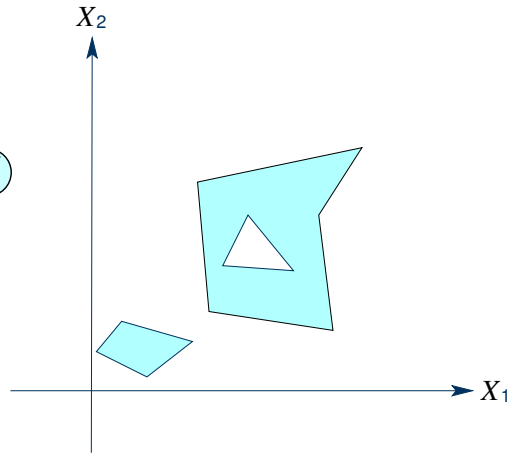


Perceptron structures and decision boundaries (*cont.*)

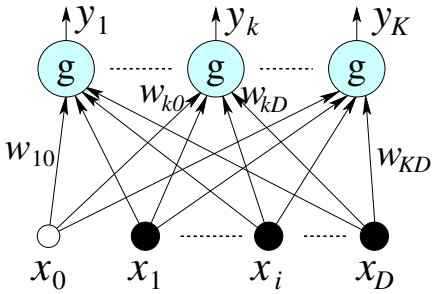


Perceptron structures and decision boundaries (*cont.*)





Single-layer network with multiple output nodes



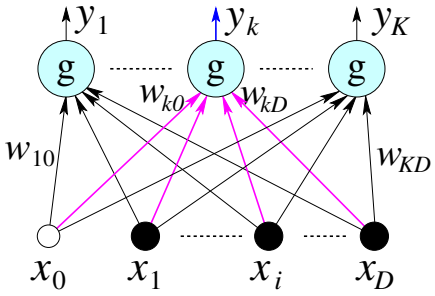
$$y_1(\mathbf{x}) = g(\mathbf{w}_1^\top \mathbf{x} + w_{10})$$
$$\vdots$$
$$y_K(\mathbf{x}) = g(\mathbf{w}_K^\top \mathbf{x} + w_{K0})$$

$$\begin{pmatrix} y_1 \\ \vdots \\ y_K \end{pmatrix} = g\left(\begin{pmatrix} w_{10} & w_{11} \dots w_{1d} \\ \vdots & \ddots \quad \vdots \\ w_{K0} & w_{K1} \dots w_{Kd} \end{pmatrix} \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{pmatrix} \right)$$

$$\mathbf{y} = g(\mathbf{W}\dot{\mathbf{x}})$$

- K output nodes: $y_1, \dots, y_K$. NB: we sometimes use y to denote $\hat{y}$ for simplicity's sake.

Single-layer network with multiple output nodes



$$y_1(\mathbf{x}) = g(\mathbf{w}_1^\top \mathbf{x} + w_{10})$$
$$\vdots$$
$$y_K(\mathbf{x}) = g(\mathbf{w}_K^\top \mathbf{x} + w_{K0})$$

$$\begin{pmatrix} y_1 \\ \vdots \\ y_K \end{pmatrix} = g\left(\begin{pmatrix} w_{10} & w_{11} \ldots w_{1d} \\ \vdots & \ddots \ \vdots \\ w_{K0} & w_{K1} \ldots w_{Kd} \end{pmatrix} \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{pmatrix} \right)$$

$$\mathbf{y} = g(\mathbf{W}\dot{\mathbf{x}})$$

- K output nodes: $y_1, \ldots, y_K$. NB: we sometimes use y to denote $\hat{y}$ for simplicity's sake.
- For $\mathbf{x}_n = (x_{n0}, \ldots, x_{nD})^\top$,

$$\hat{y}_{nk} \;=\; g\Big(\sum_{d=0}^{d} w_{kd}\, x_{nd}\Big) = g(a_{nk})\,, \qquad a_{nk} \;=\; \sum_{d=0}^{d} w_{kd}\, x_{nd}$$

Limitations of Perceptron

- Single-layer perceptron is just a linear classifier (Marvin Minsky and Seymour Papert, 1969)

Limitations of Perceptron

- Single-layer perceptron is just a linear classifier
  (Marvin Minsky and Seymour Papert, 1969)

- Training does not stop if data are linearly non-separable

Limitations of Perceptron

- Single-layer perceptron is just a linear classifier
  (Marvin Minsky and Seymour Papert, 1969)
- Training does not stop if data are linearly non-separable
- Weights $\boldsymbol{w}$ are adjusted for misclassified data only (correctly classified data are not considered at all)

Limitations of Perceptron

- Single-layer perceptron is just a linear classifier
  (Marvin Minsky and Seymour Papert, 1969)
- Training does not stop if data are linearly non-separable
- Weights $\boldsymbol{w}$ are adjusted for misclassified data only (correctly classified data are not considered at all)
- Multi-layer perceptron can form complex decision boundaries (piecewise-linear), but the Perceptron training algorithm is not applicable.

How can we resolve the problem of training?

- Use the least squares error criterion for training

$$E_2(\boldsymbol{w}) = \sum_{n=1}^{N} (\hat{y}_n - y_n)^2$$

How can we resolve the problem of training?

- Use the least squares error criterion for training

$$E_2(\boldsymbol{w}) = \sum_{n=1}^{N} (\hat{y}_n - y_n)^2$$

- Replace $g( )$ with a differentiable function

How can we resolve the problem of training?

- Use the least squares error criterion for training

$$E_2(\boldsymbol{w}) = \sum_{n=1}^{N} (\hat{y}_n - y_n)^2$$

- Replace $g( )$ with a differentiable function
  What about removing $g( )$ in the hidden layers?

How can we resolve the problem of training?

- Use the least squares error criterion for training

$$E_2(\boldsymbol{w}) = \sum_{n=1}^{N} (\hat{y}_n - y_n)^2$$

- Replace $g(\,)$ with a differentiable function
  What about removing $g(\,)$ in the hidden layers?

$$\hat{y}_n = g(W^{(2)} g(W^{(1)} \mathbf{x}_n)) \quad \Rightarrow \quad \hat{y}_n = g(W^{(2)} W^{(1)} \mathbf{x}_n) = g(W \mathbf{x}_n)$$

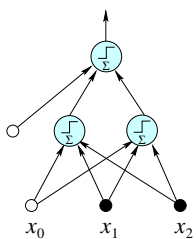
How can we resolve the problem of training?

- Use the least squares error criterion for training

$$E_2(\boldsymbol{w}) = \sum_{n=1}^{N} (\,\hat{y}_n - y_n\,)^2$$

- Replace $g(\,)$ with a differentiable function
  What about removing $g(\,)$ in the hidden layers?

$$\hat{y}_n = g(W^{(2)} g(W^{(1)} \mathbf{x}_n)) \quad \Rightarrow \quad \hat{y}_n = g(W^{(2)} W^{(1)} \mathbf{x}_n) = g(W \mathbf{x}_n)$$



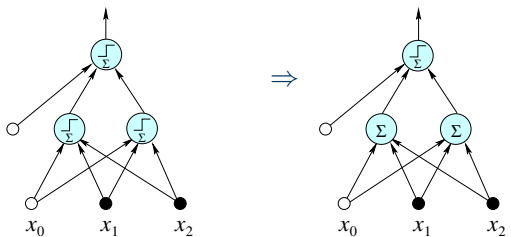
How can we resolve the problem of training?

- Use the least squares error criterion for training

$$E_2(\boldsymbol{w}) = \sum_{n=1}^{N} (\,\hat{y}_n - y_n\,)^2$$

- Replace $g(\,)$ with a differentiable function
  What about removing $g(\,)$ in the hidden layers?

$$\hat{y}_n = g(W^{(2)} g(W^{(1)} \mathbf{x}_n)) \quad \Rightarrow \quad \hat{y}_n = g(W^{(2)} W^{(1)} \mathbf{x}_n) = g(W \mathbf{x}_n)$$



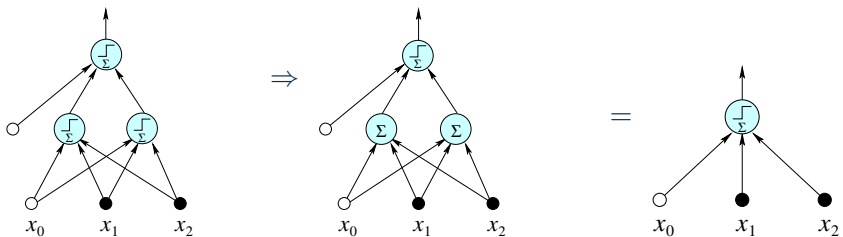
How can we resolve the problem of training?

- Use the least squares error criterion for training

$$E_2(\boldsymbol{w}) = \sum_{n=1}^{N} (\,\hat{y}_n - y_n\,)^2$$

- Replace $g(\,)$ with a differentiable function
  What about removing $g(\,)$ in the hidden layers?

$$\hat{y}_n = g(W^{(2)} g(W^{(1)} \mathbf{x}_n)) \quad \Rightarrow \quad \hat{y}_n = g(W^{(2)} W^{(1)} \mathbf{x}_n) = g(W \mathbf{x}_n)$$



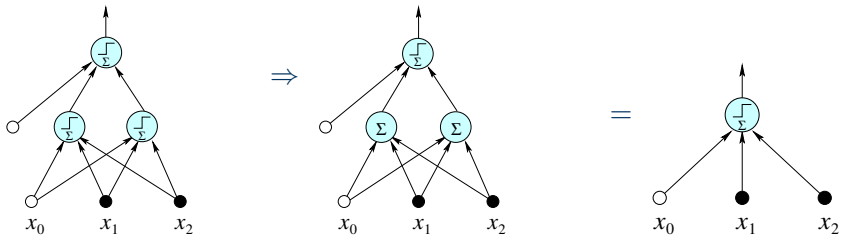
How can we resolve the problem of training?

- Use the least squares error criterion for training

$$E_2(\boldsymbol{w}) = \sum_{n=1}^{N} (\hat{y}_n - y_n)^2$$

- Replace $g(\,)$ with a differentiable function
  What about removing $g(\,)$ in the hidden layers?

$$\hat{y}_n = g(W^{(2)} g(W^{(1)} \mathbf{x}_n)) \quad \Rightarrow \quad \hat{y}_n = g(W^{(2)} W^{(1)} \mathbf{x}_n) = g(W \mathbf{x}_n)$$



Question: Show networks with linear hidden nodes reduce to single-layer networks

How can we resolve the problem of training? (*cont.*)

- Replace $g(\,)$ with a differentiable non-linear function

How can we resolve the problem of training? (*cont.*)

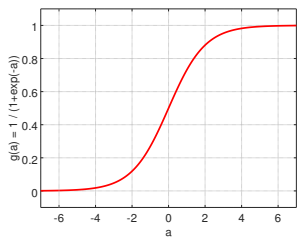
- Replace $g(\,)$ with a differentiable non-linear function
  e.g., Logistic sigmoid function:

$$g(a) = \frac{1}{1 + e^{-a}} = \frac{1}{1 + \exp(-a)}$$

How can we resolve the problem of training? (*cont.*)

- Replace $g(\,)$ with a differentiable non-linear function
  e.g., Logistic sigmoid function:

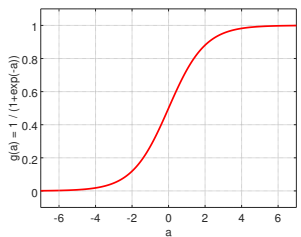
$$g(a) = \frac{1}{1 + e^{-a}} = \frac{1}{1 + \exp(-a)}$$



How can we resolve the problem of training? (*cont.*)

- Replace $g(\,)$ with a differentiable non-linear function
  e.g., Logistic sigmoid function:

$$g(a) = \frac{1}{1 + e^{-a}} = \frac{1}{1 + \exp(-a)}$$

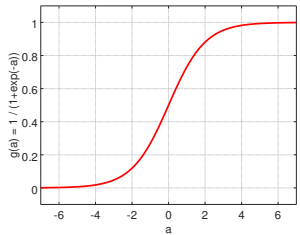


Mapping: $(-\infty, +\infty) \rightarrow (0, 1)$, Use a threshold=0.5 for binary classification

How can we resolve the problem of training? (*cont.*)

- Replace $g(\,)$ with a differentiable non-linear function
  e.g., Logistic sigmoid function:

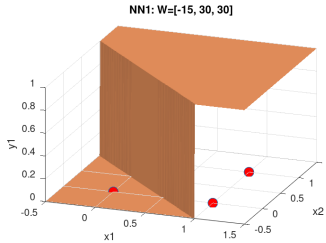
$$g(a) = \frac{1}{1+e^{-a}} = \frac{1}{1+\exp(-a)}$$

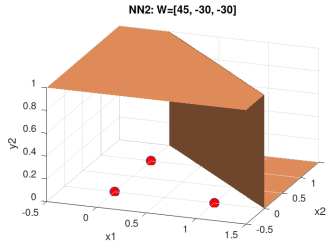


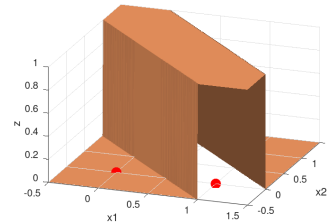
Mapping: $(-\infty, +\infty) \rightarrow (0, 1)$, Use a threshold=0.5 for binary classification

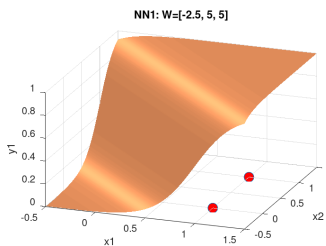
$$\frac{d}{da} g(a) = g'(a) = g(a)\,(1 - g(a))$$

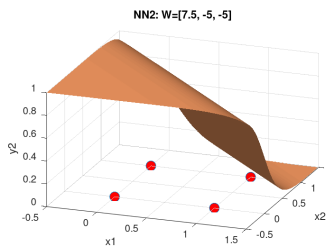
Output of NN – threshold func. vs sigmoid func.

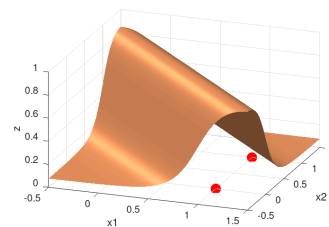












Input-Output – demos

Ability of neural networks

- Universal approximation theorem

Ability of neural networks

- Universal approximation theorem
  - "Univariate function and a set of affine functionals can uniformly approximate any continuous function of n real variables with support in the unit hypercube; only mild conditions are imposed on the univariate function. " (G. Cybenko (1989)

    $\longrightarrow$

    A single-output node neural network with a single hidden layer with a finite neurons can approximate continuous (and discontinuous) functions.

Ability of neural networks

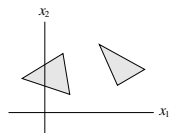
- Universal approximation theorem
  - "Univariate function and a set of affine functionals can uniformly approximate any continuous function of n real variables with support in the unit hypercube; only mild conditions are imposed on the univariate function. " (G. Cybenko (1989)

    $\longrightarrow$

    A single-output node neural network with a single hidden layer with a finite neurons can approximate continuous (and discontinuous) functions.
  - K. Hornik (1990) doi:10.1016/0893-6080(91)90009-T
  - N. Guliyev, V. Ismailov (2018) https://doi.org/10.1016/j.neunet.2017.12.007
  - V. Ismailov (2023) "A three layer neural network can represent any multivariate function" https://doi.org/10.1016/j.jmaa.2023.127096

Quizzes

- Answer the question of slide 16
- Find the structure of the perceptron for a two-class classification problem that gives the decision boundaries and decision regions shown in the figure below, in which grey areas correspond to one class, and the white areas to the other class. You do not need to identify the weight values, but you need to describe how many nodes are required and how each node should be connected to other nodes.



- Derive the derivative of the logistic sigmoid function
- Discuss how the decision boundary will change if you replace the Heaviside step function in a Rosenblatt's perceptron with a logistic sigmoid function in which you use a threshold=0.5 for classification.

References

- [LWLS] Section 6.1
- Neural Networks and Deep Learning by Michael Nielsen (http://neuralnetworksanddeeplearning.com/)
- [M1] Chapter 13