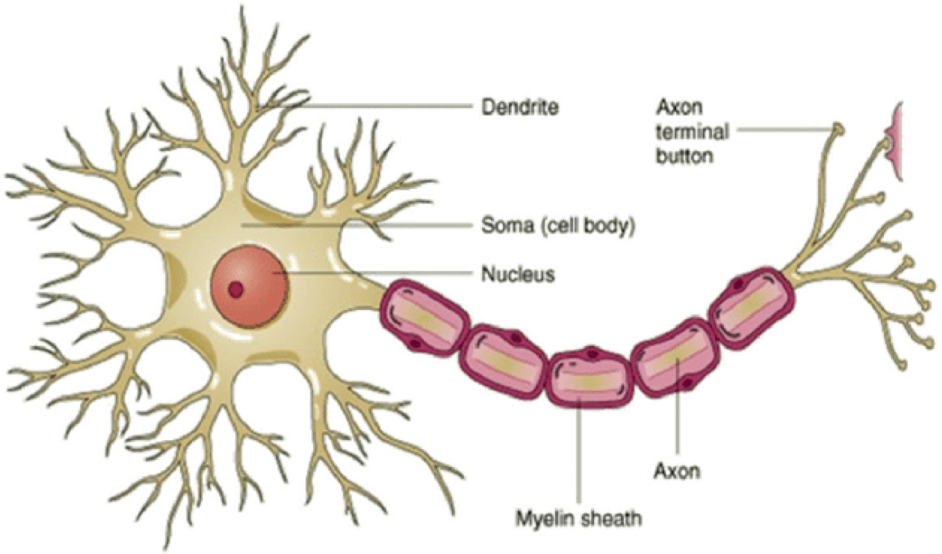
# Deep feed-forward networks

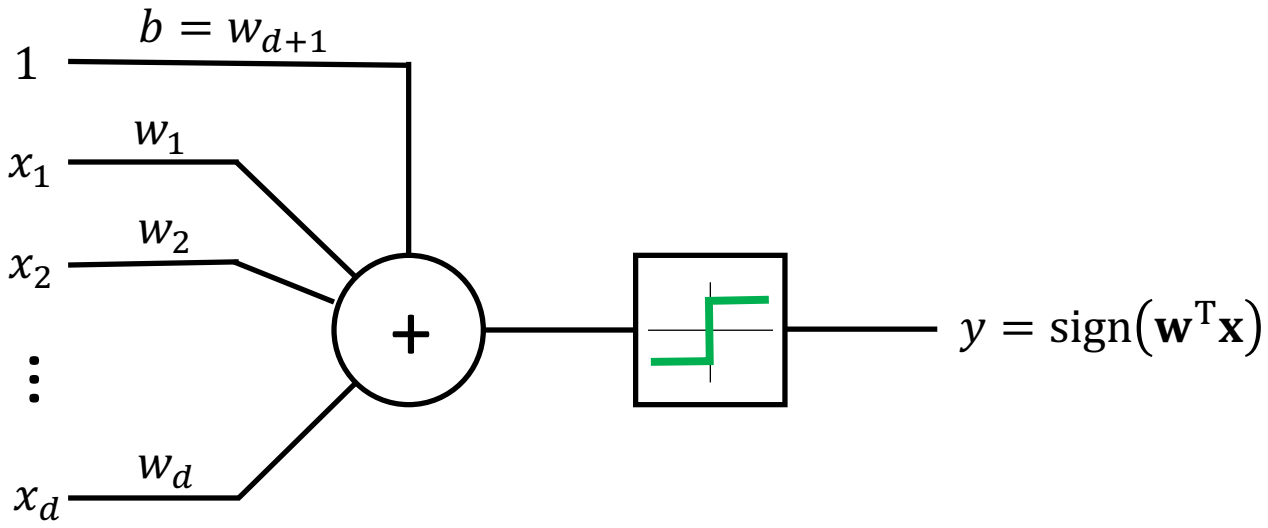
Goodfellow et al. 2016 (sec. 6)

# Perceptron


Rosenblatt 1957

# Perceptron

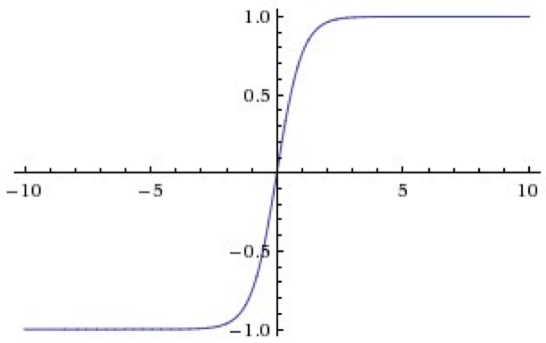



* Note the class labels here are +1 and -1 instead of +1 and 0 as before

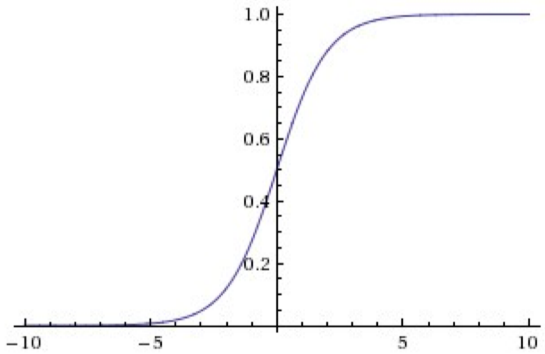
Rosenblatt 1957

# Activation functions

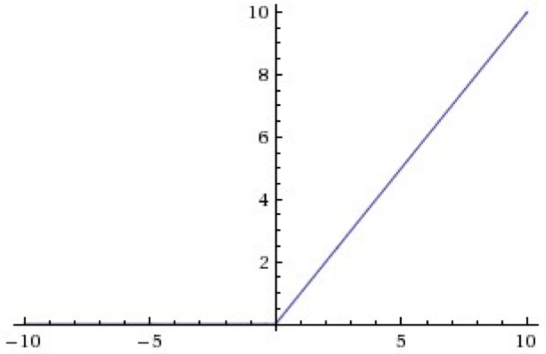
tanh




sigmoid



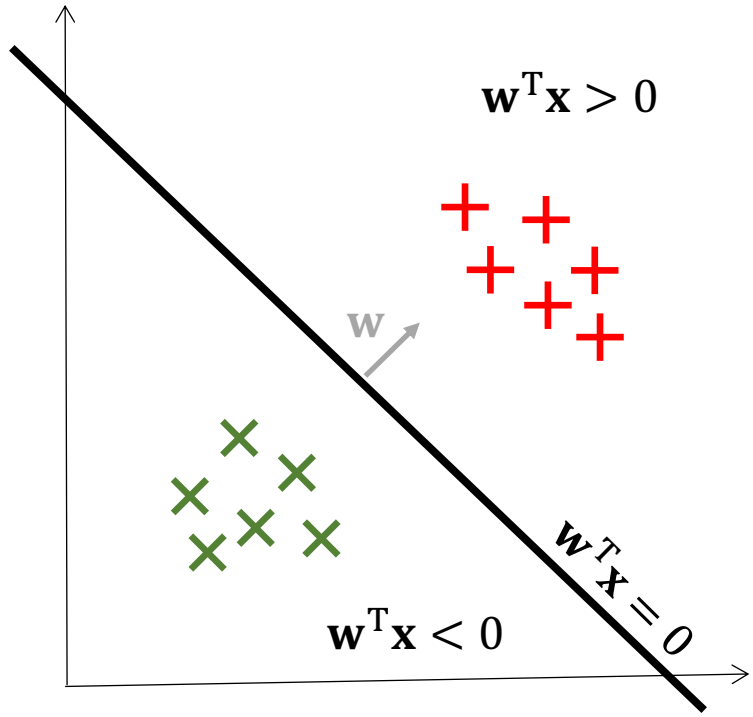

ReLU



# Original Perceptron learning algorithm

- Initialize $\mathbf{w} = \mathbf{0}$
- Given example $\mathbf{x}$, output

$$\hat{y} = \text{sign}\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}\right)$$

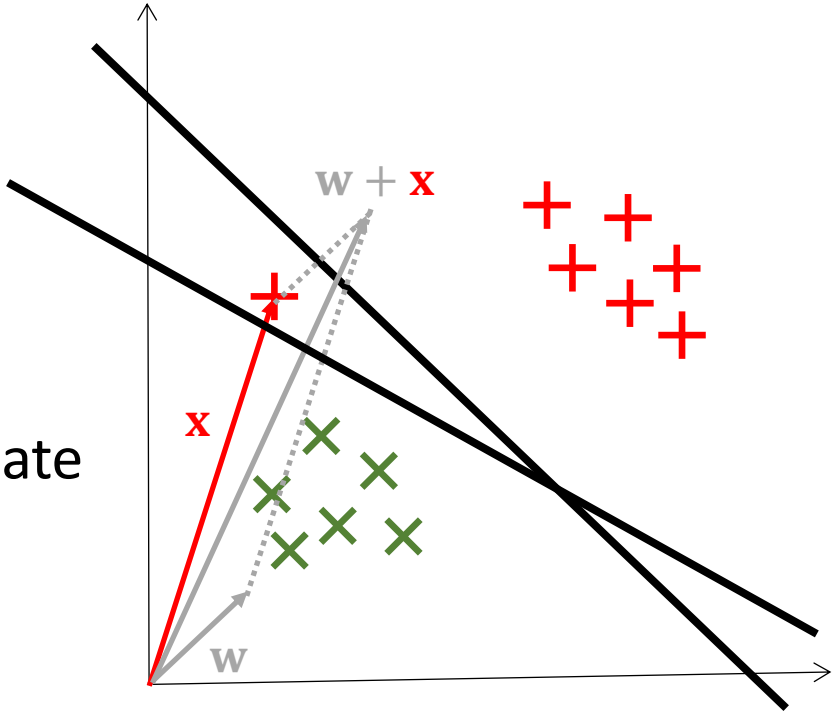


# Original Perceptron learning algorithm

- Initialize $\mathbf{w} = \mathbf{0}$
- Given example $\mathbf{x}$, output

$$\hat{y} = \text{sign}\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}\right)$$

- If predicted wrong ($\hat{y} \neq y$) update
  - If $y = 1$: $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}$

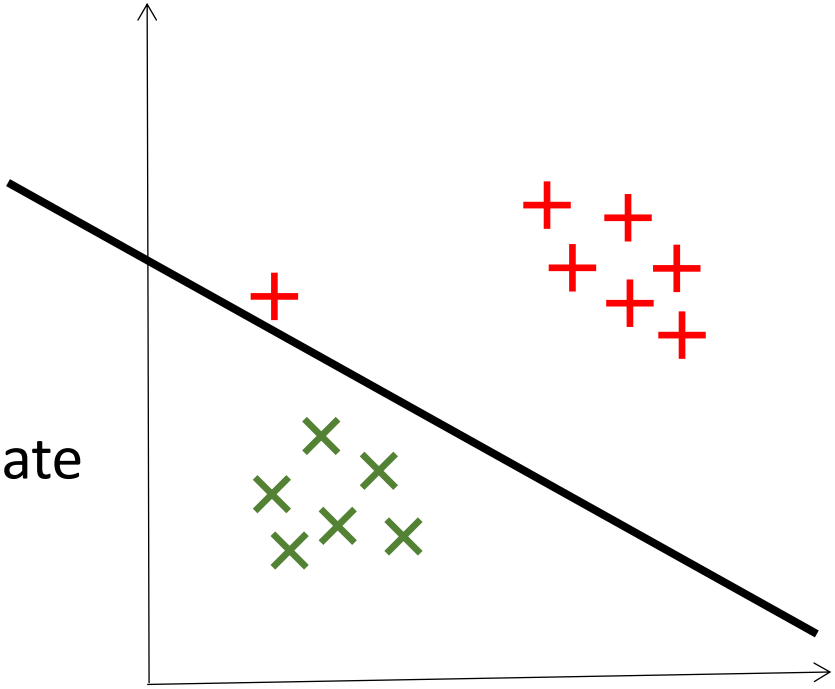


# Original Perceptron learning algorithm

- Initialize $\mathbf{w} = \mathbf{0}$
- Given example $\mathbf{x}$, output

$$\hat{y} = \text{sign}\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}\right)$$

- If predicted wrong ($\hat{y} \neq y$) update
  - If $y = 1$: $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}$
  - If $y = -1$: $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{x}$



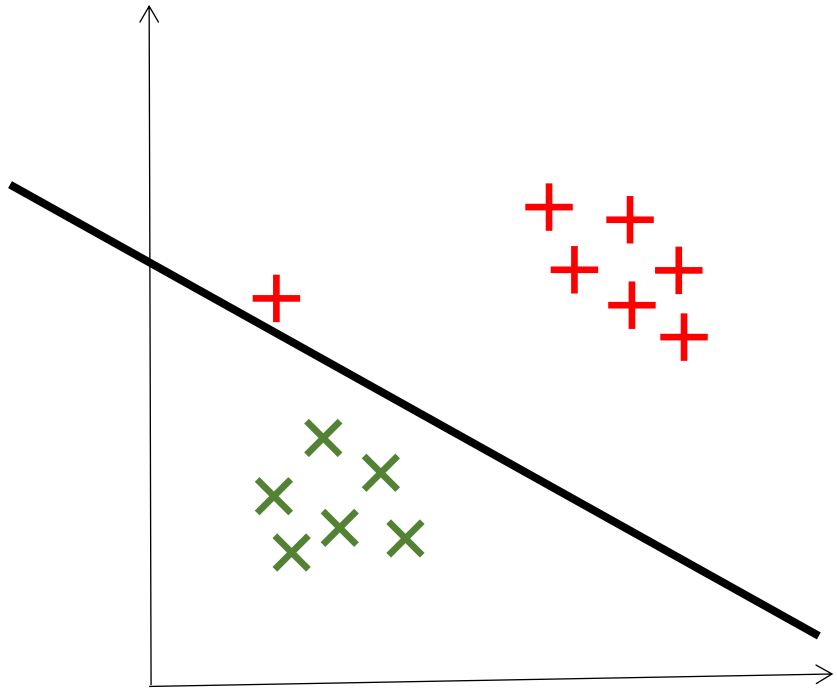
# Original Perceptron learning algorithm

- Initialize $\mathbf{w} = \mathbf{0}$
- Given example $\mathbf{x}$, output

$$\hat{y} = \text{sign}\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}\right)$$

- Update

$$\mathbf{w} \leftarrow \mathbf{w} + y\mathbf{x}\,\mathbb{I}_{\text{sign}\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}\right)\neq\widehat{y}}$$



# Perceptron training as SGD

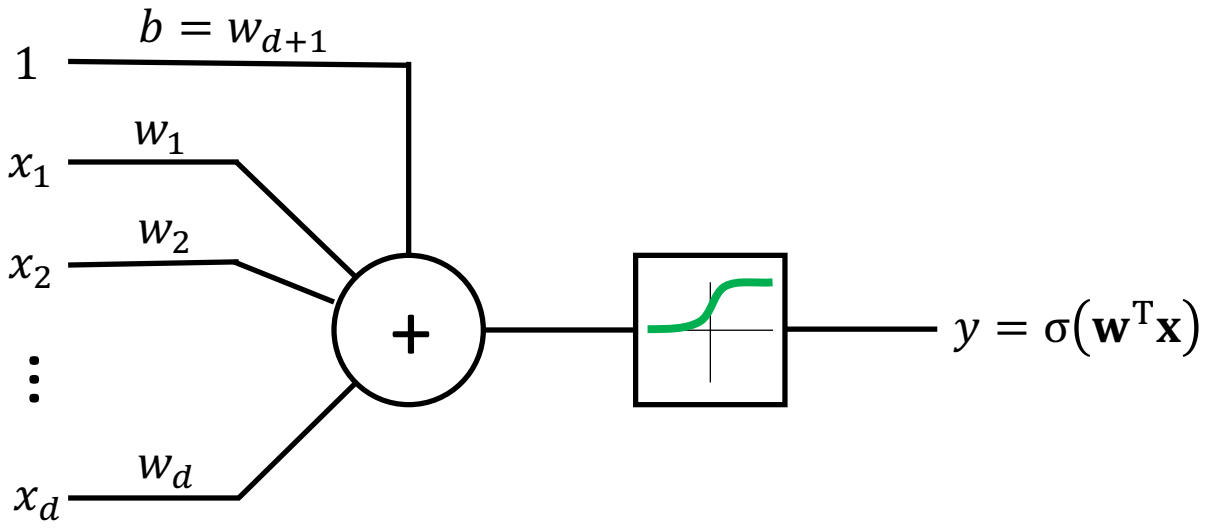
$$\hat{L}(\mathbf{w}) = -\sum_{i=1}^{n} y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i \mathbb{I}_{\mathrm{sign}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i) \neq y_i} \rightarrow \min$$

- SGD with step size $\alpha^{(t)} = 1$

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \nabla_{\mathbf{w}} \left( -y_t \mathbf{w}^{\mathrm{T}} \mathbf{x}_t \mathbb{I}_{\mathrm{sign}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_t) \neq y_t} \right)$$

$$= \mathbf{w}^{(t)} + y_t \mathbf{x}_t \mathbb{I}_{\mathrm{sign}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_t) \neq y_t}$$

# Logistic regression



# Logistic regression with SGD

$$\hat{L}(\mathbf{w}) = -\frac{1}{n}\sum_{i=1}^{n} \log \sigma\left(y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i\right) \rightarrow \min$$

- Computing the gradients

$$\nabla_{\mathbf{w}} \log \sigma\left(y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i\right) = \frac{\nabla_{\mathbf{w}} \sigma\left(y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i\right)}{\sigma(y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i)}$$

$$= \frac{\sigma\left(y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i\right)\left(1 - \sigma\left(y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i\right)\right)}{\sigma(y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i)} y_i \mathbf{x}_i$$

* Note the class labels here are +1 and -1. Since $\sigma(-x) = 1 - \sigma(x)$, we can write the log-likelihood function as

$$\hat{L}(\mathbf{w}) = -\frac{1}{n}\sum_{i:y_i=1} \log \sigma\left(\mathbf{w}^{\mathrm{T}} \mathbf{x}_i\right) - \frac{1}{n}\sum_{i:y_i=-1} \log\left(1 - \sigma\left(\mathbf{w}^{\mathrm{T}} \mathbf{x}_i\right)\right) = -\frac{1}{n}\sum_{i=1}^{n} \log \sigma\left(y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i\right)$$

** Derivative of sigmoid $\sigma'(x) = \sigma(x)\left(1 - \sigma(x)\right)$

# Logistic regression with SGD

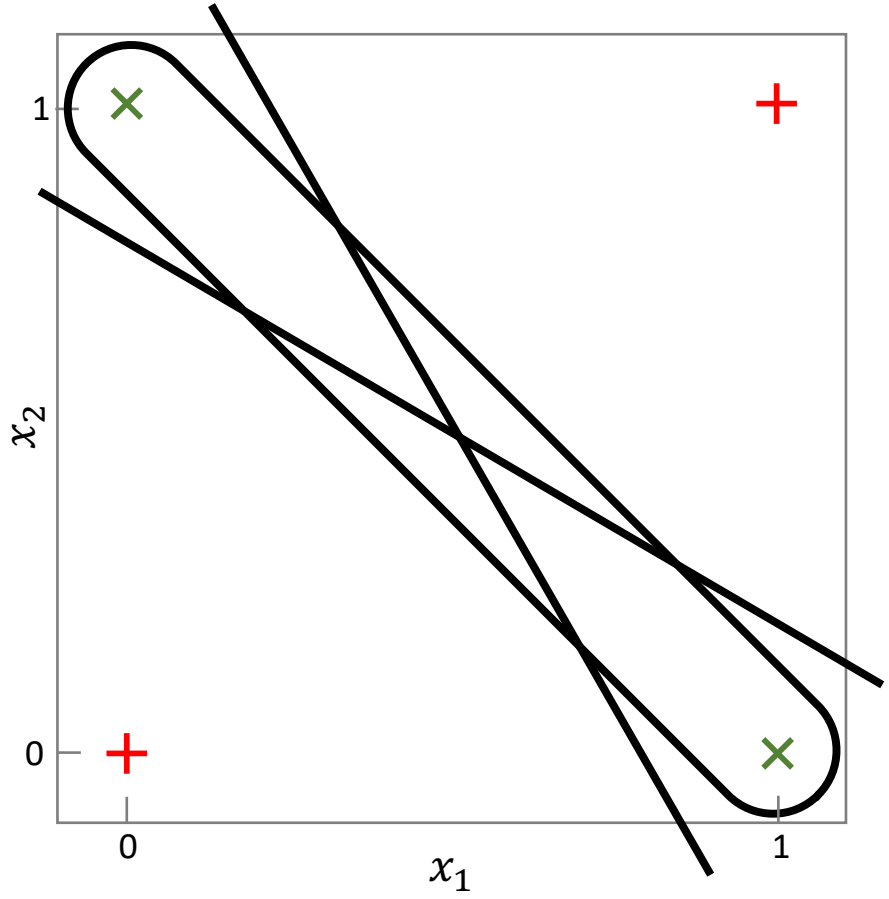
$$\hat{L}(\mathbf{w}) = -\frac{1}{n}\sum_{i=1}^{n} \log \sigma\left(y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i\right) \to \min$$

- SGD step

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \frac{\alpha^{(t)}}{n} \frac{\sigma\left(y_t \mathbf{w}^{\mathrm{T}} \mathbf{x}_t\right)\left(1 - \sigma\left(y_t \mathbf{w}^{\mathrm{T}} \mathbf{x}_t\right)\right)}{\sigma(y_t \mathbf{w}^{\mathrm{T}} \mathbf{x}_t)} y_t \mathbf{x}_t$$

gradient $\approx 0$
when $|\mathbf{w}^{\mathrm{T}} \mathbf{x}_t| \gg 0$

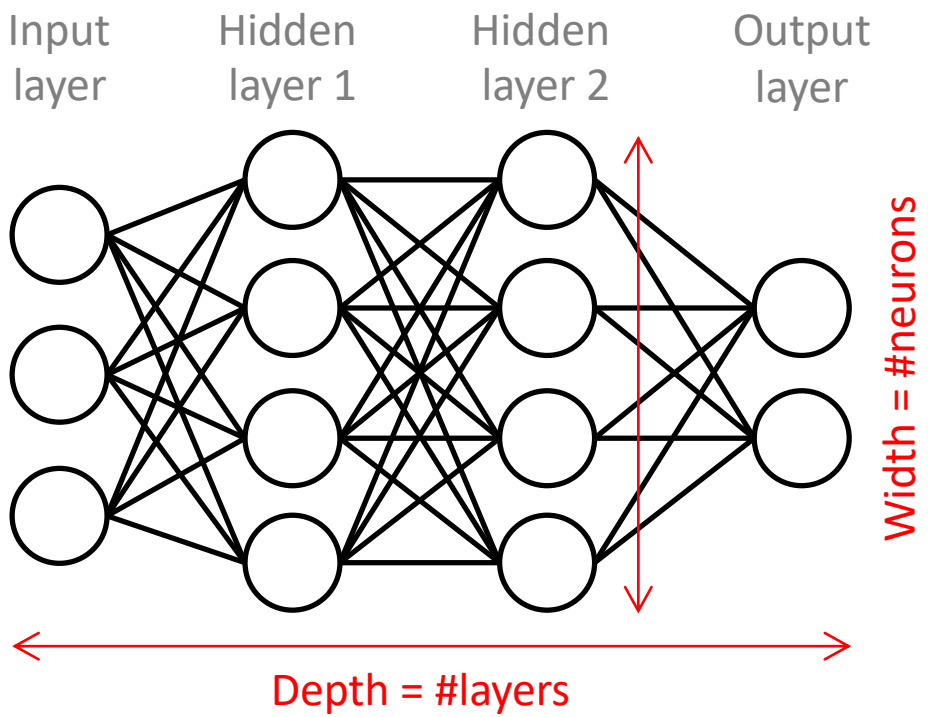
# The XOR problem



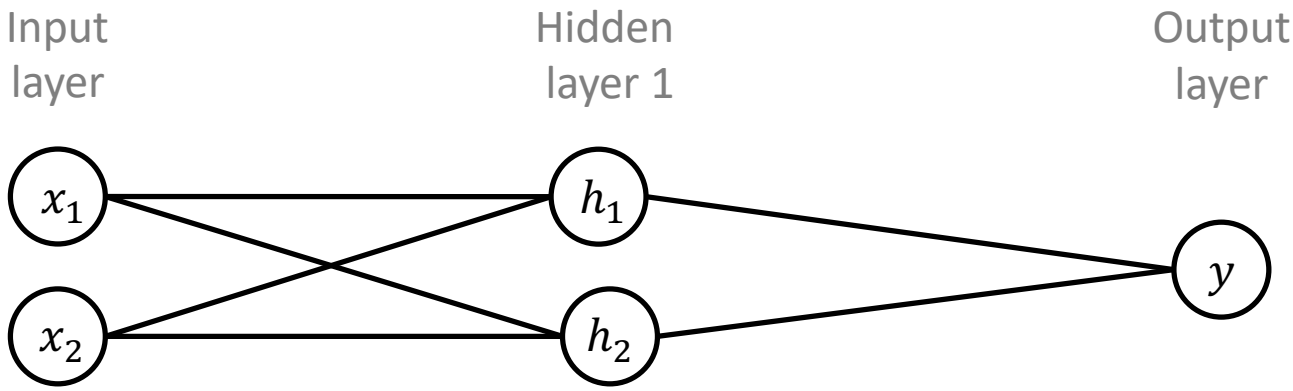

Not linearly separable!

Minsky, Papert 1969

# Multi-layer perceptron



# Solving XOR with one hidden layer




$$\boldsymbol{h} = \max\{0, \boldsymbol{W}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{c}\} \qquad y = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{h} + b$$

Input:

$$\boldsymbol{X} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

Hidden layer:

$$\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \qquad \boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

$$\boldsymbol{H} = \max\left\{0, \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}\right\} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

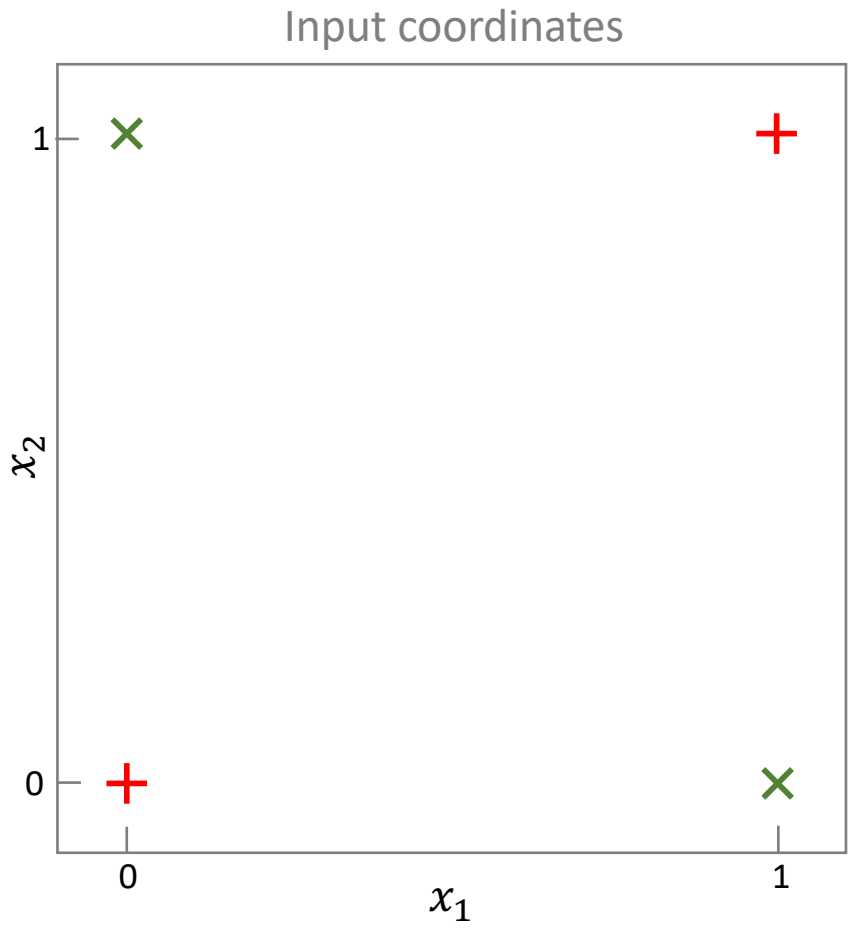
Output layer:

$$\boldsymbol{W} = \begin{bmatrix} 1 \\ -2 \end{bmatrix} \qquad b = 0$$

$$\boldsymbol{Y} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

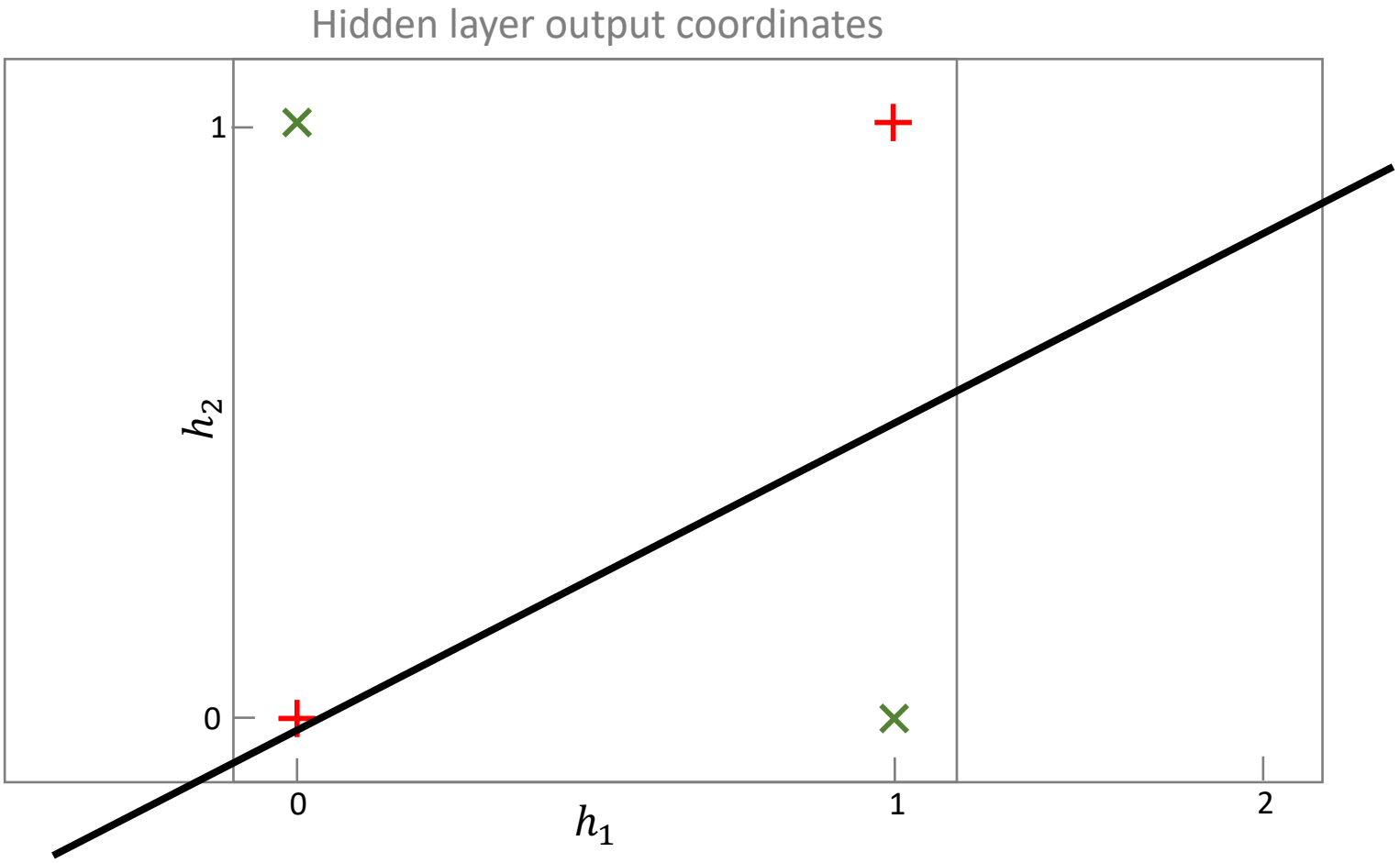
**(Learned XOR)**

# Solving XOR with one hidden layer




Example: Goodfellow et al.

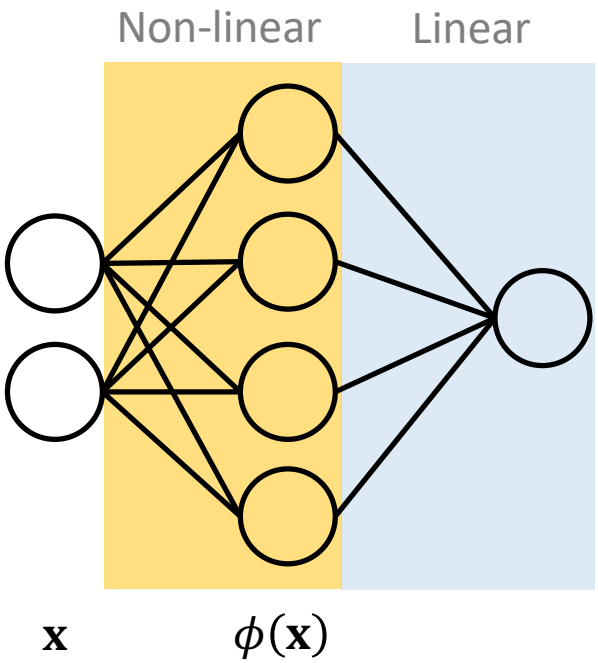
# Solving XOR with one hidden layer

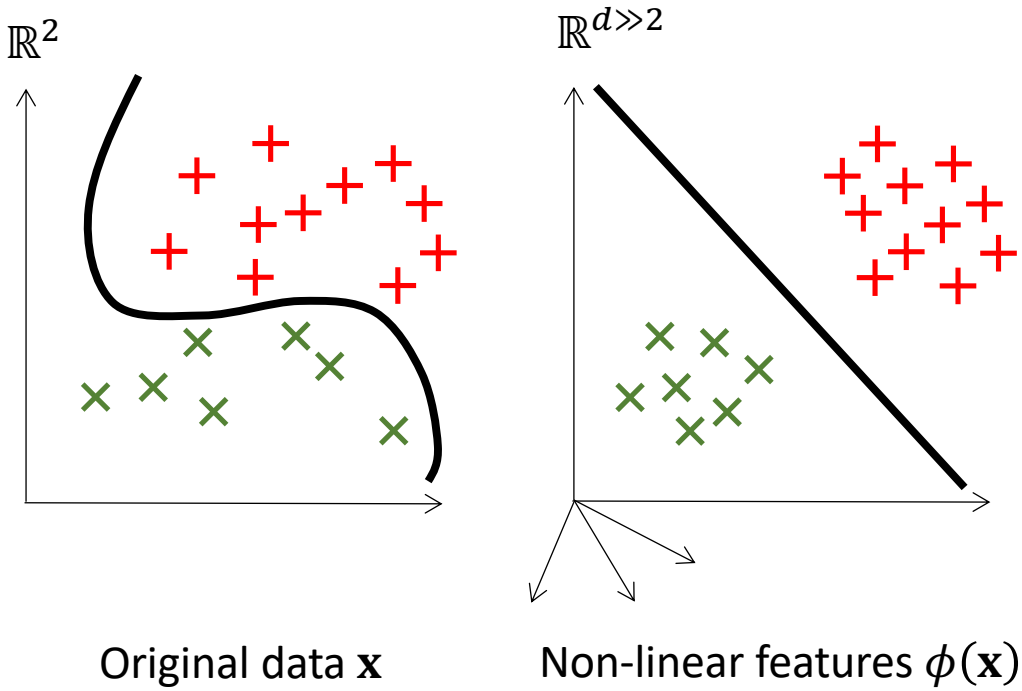



linearly separable!

Example: Goodfellow et al.

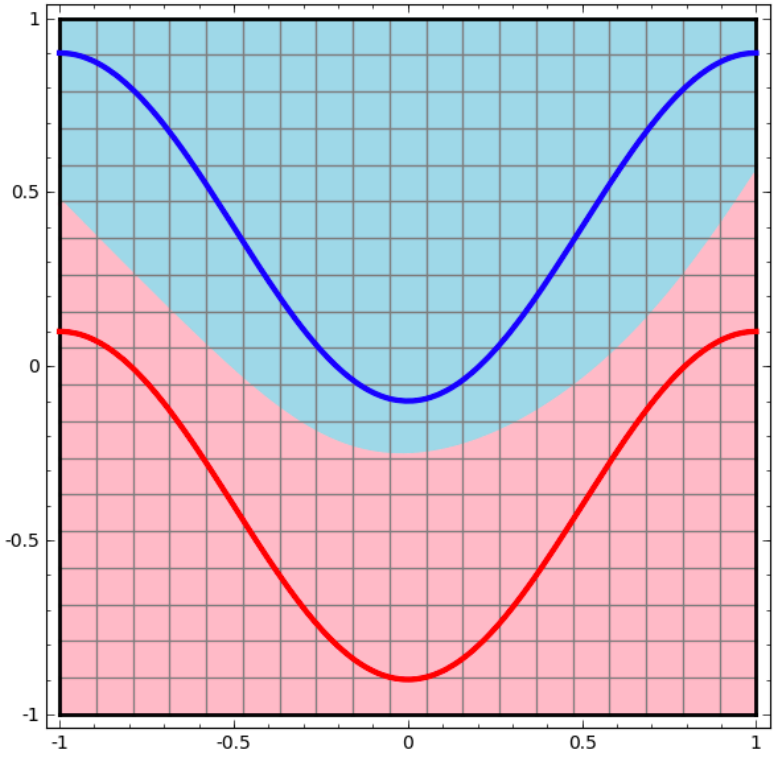
# Representation learning



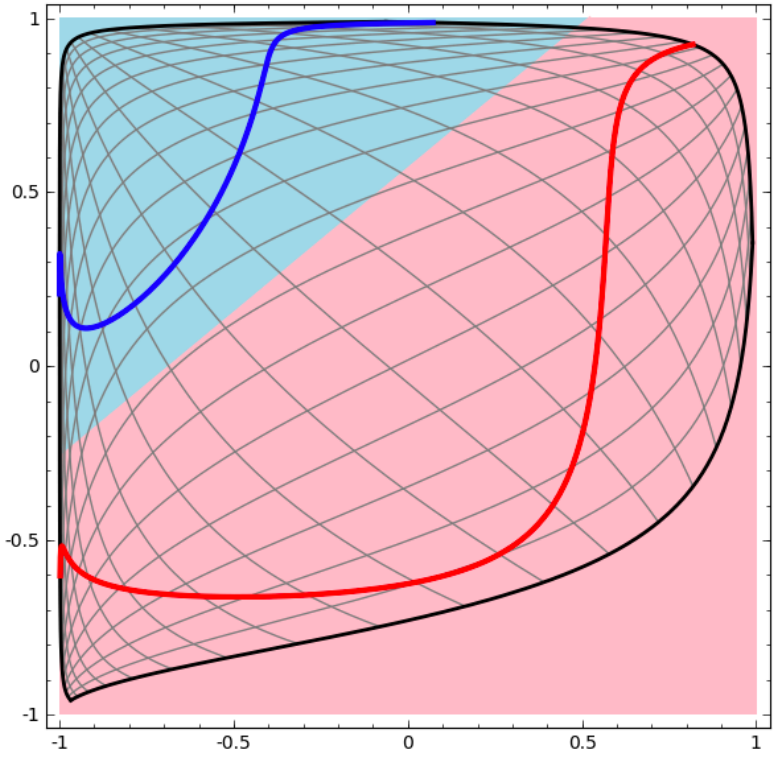



# Representation learning

Input coordinates




Hidden layer coordinates




Hidden layer learns a representation that is linearly separable

Image credit: Christopher Olah

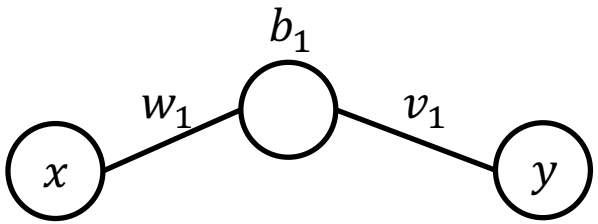
# Universal approximation

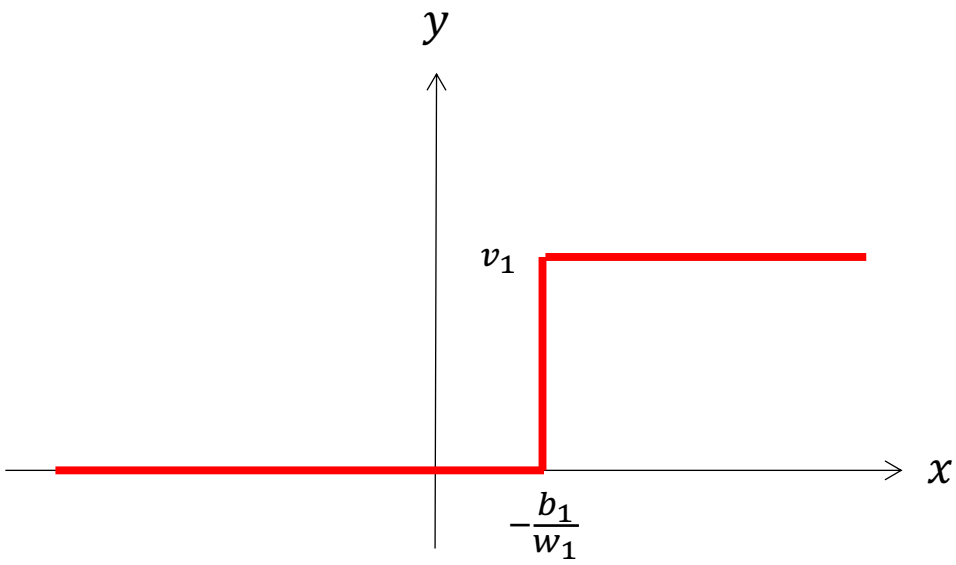
A perceptron with one hidden layer of finite width can **arbitrarily accurately approximate any continuous function** on a compact subset of $\mathbb{R}^n$ (under mild assumptions on the activation function)

- 🙂 one hidden layer is enough to represent a reasonable function
- ☹️ how to find the optimal weights?
- ☹️ how wide should the layer be?
- 😡 width grows exponentially with dimensions

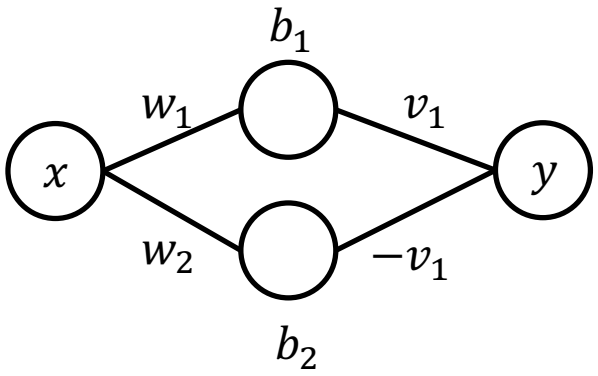
*More generally, a Borel-measurable function
Cybenko 1989; Hornik 1991

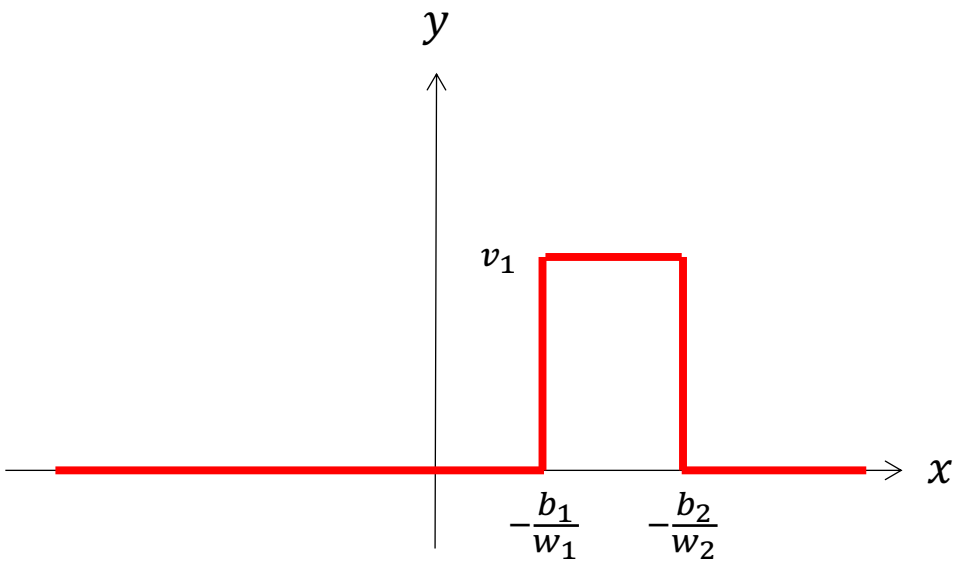
# Intuition



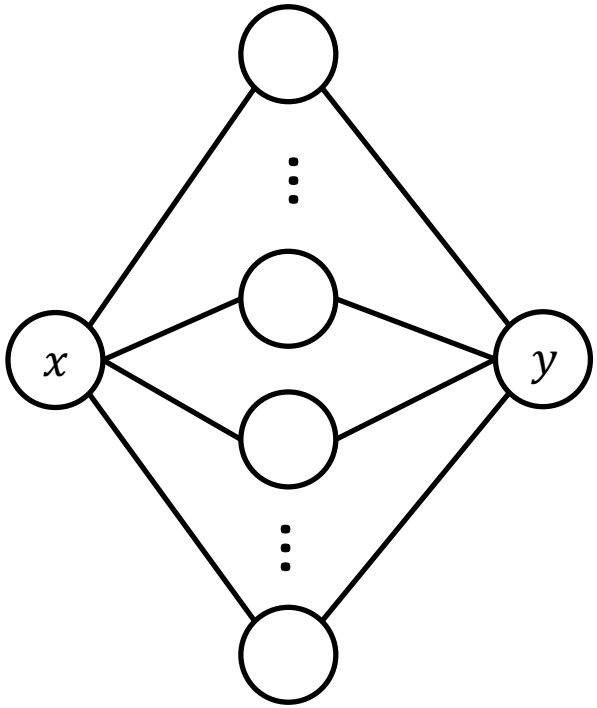



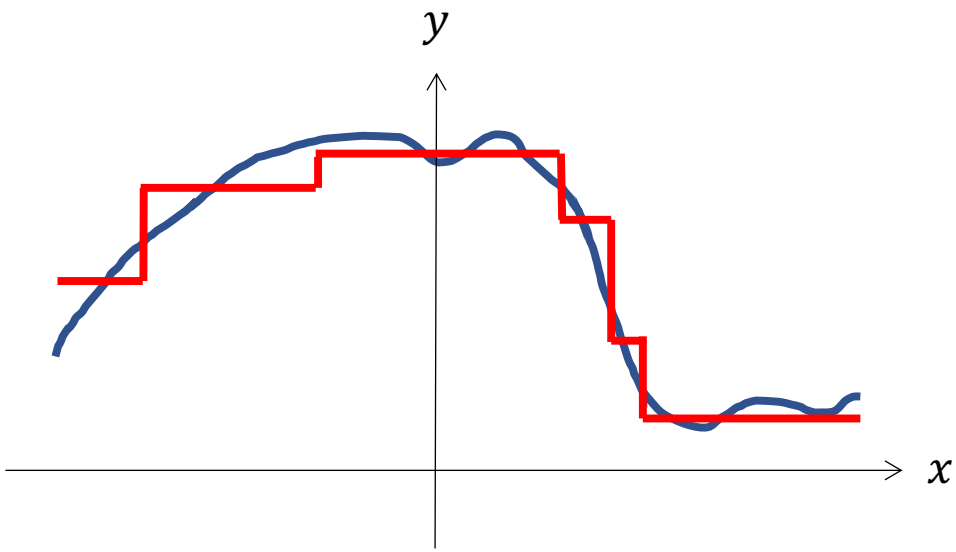
# Intuition



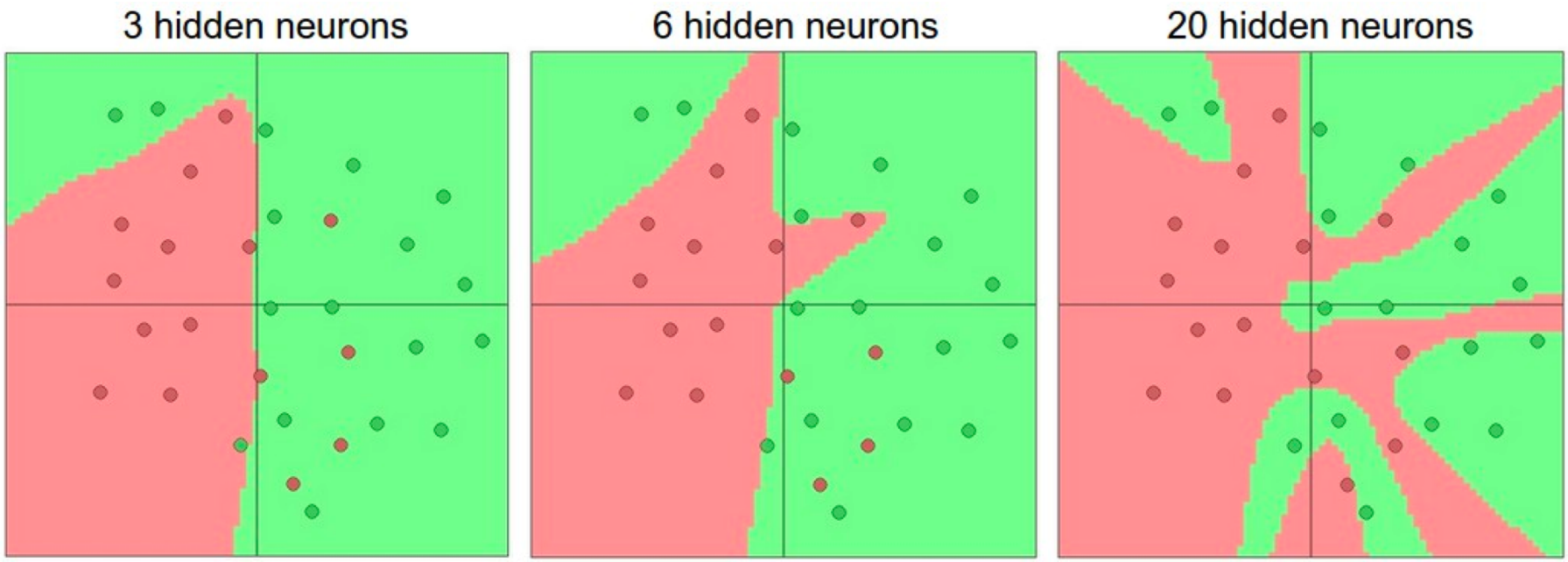



# Intuition

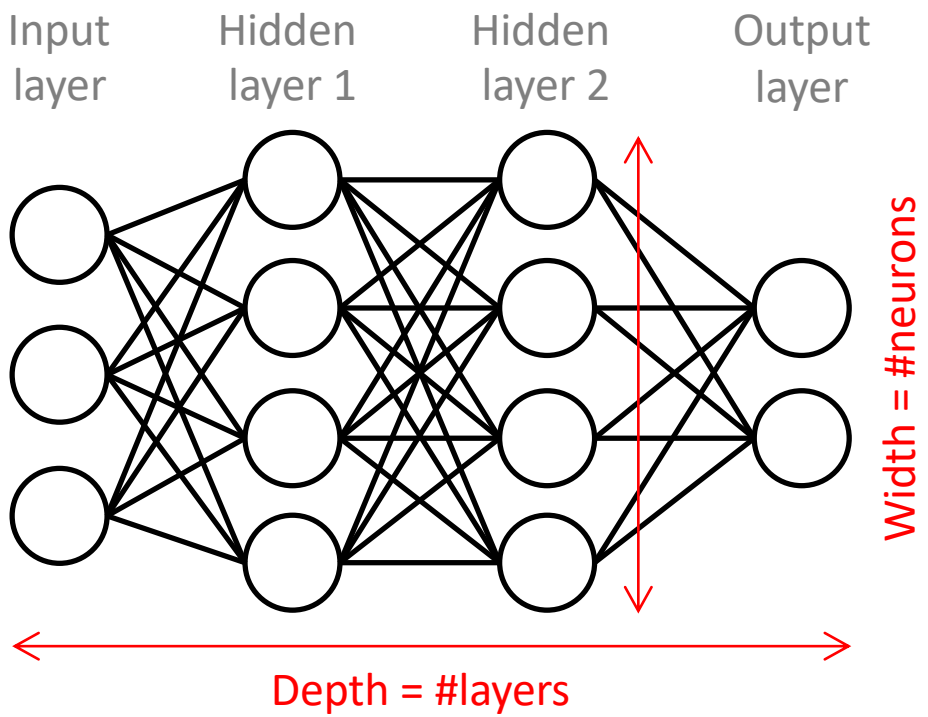


# Universal approximation

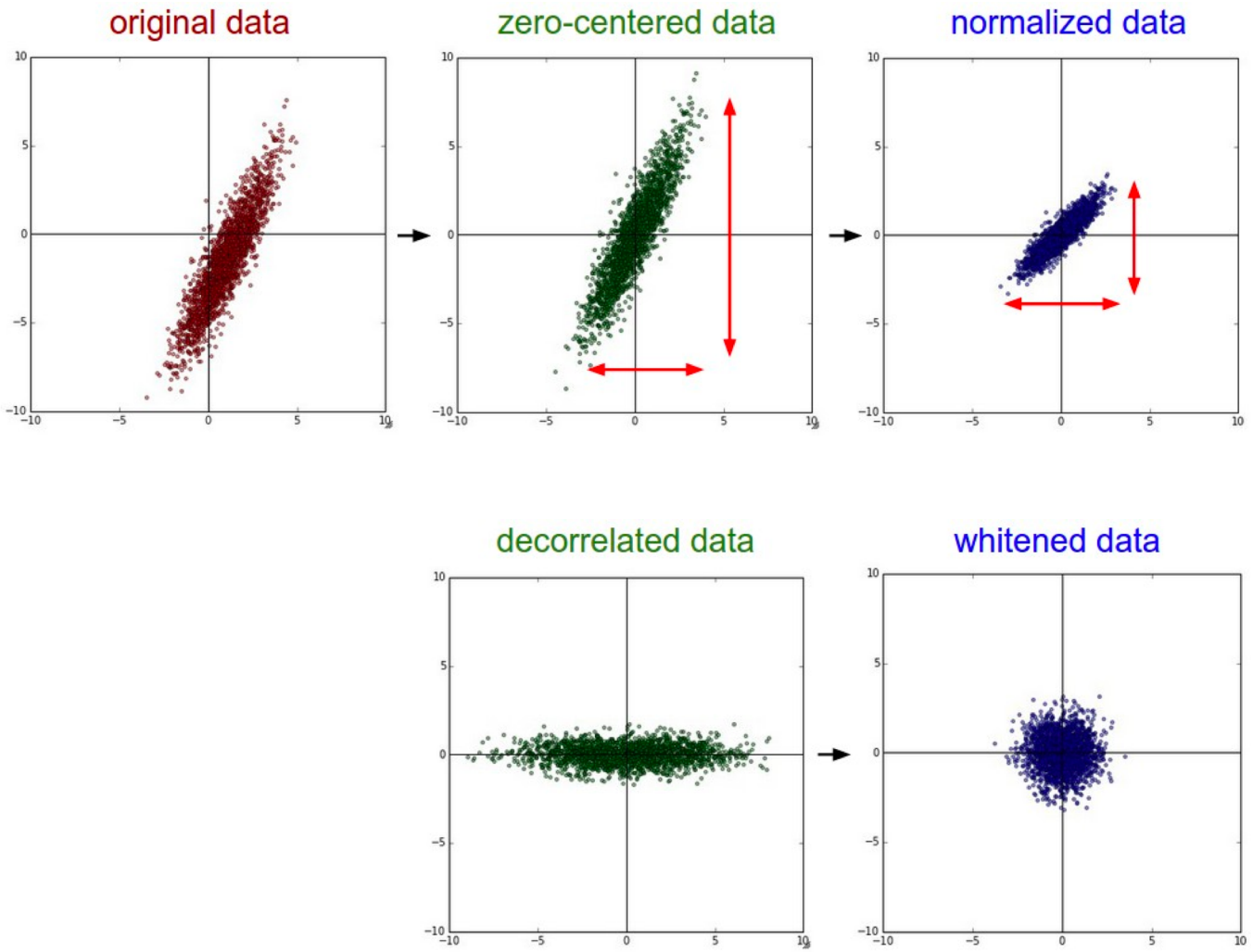


# Depth vs Width



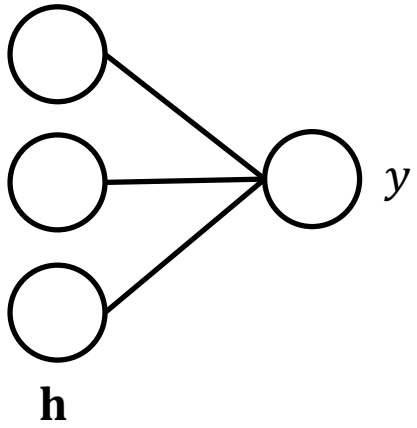

What is better: depth or width?

# Input pre-processing



# Popular output layer types

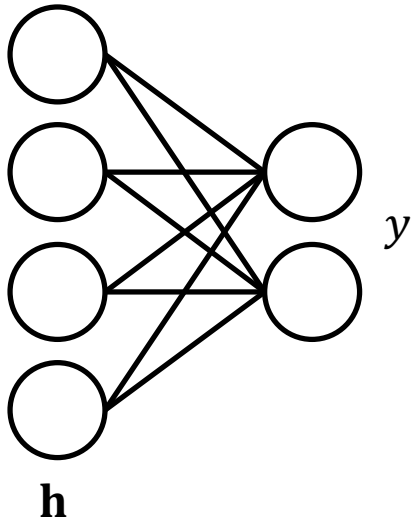
- 1D regression
- Binary classification




$$y = \mathbf{w}^{\mathrm{T}}\mathbf{h}$$

$$y = \sigma\left(\mathbf{w}^{\mathrm{T}}\mathbf{h}\right)$$

- nD regression
- Softmax (multi-class classification)




$$\mathbf{y} = \mathbf{W}^{\mathrm{T}}\mathbf{h}$$

$$\mathbf{y} = \frac{e^{\mathbf{w}_i^{\mathrm{T}}\mathbf{h}}}{\sum_i e^{\mathbf{w}_i^{\mathrm{T}}\mathbf{h}}}$$