


Introduction to Neural Networks

Marco Morik

Technische Universität Berlin - Machine Learning Group

2024 Aqtivate Workshop

Contents

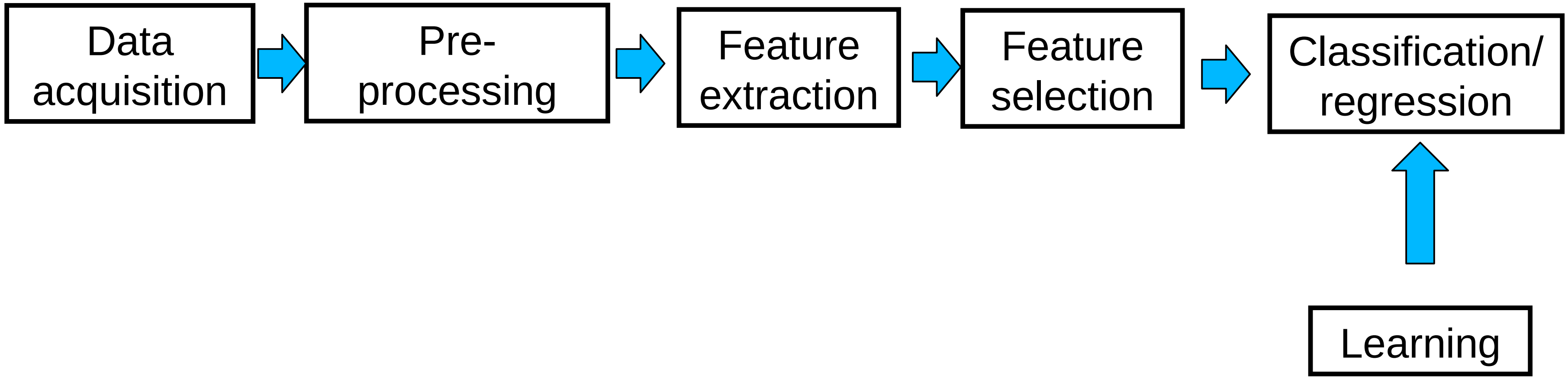
- History of NN
- Rosenblatt’s Perceptron
- Multi-Layer Perceptron (MLP)
- Gradient descent algorithm
- Backpropagation

History of Neural Network

- Progression (1943-1960)
  - First Mathematical model of neurons, Pitts & McCulloch (1943)
  - Beginning of artificial neural networks–Perceptron, Rosenblatt (1957)
- Degression (1960-1980)
  - Perceptron can't even learn the XOR function
  - We don't know how to train MLP
  - 1963 Backpropagation (Bryson et al.)
- Progression (1980-)
  - 1986 Backpropagation reinvented
- Degression (1993-)
  - SVM: Support Vector Machine is developed by Vapnik et al. (1995)
  - Graphical models are becoming more and more popular
  - Training deeper networks consistently yields poor results.
  - However, Yann LeCun (1998) developed deep convolutional neural networks
- Progression (2006-)
  - Deep Belief Networks (DBN) by Hinton et al. (2006)
  - Deep Autoencoder based networks by Greedy Layer-Wise Training of Deep Networks. Bengio et al.
  - Convolutional neural networks running on GPUs
  - AlexNet (2012). Krizhevsky et al.

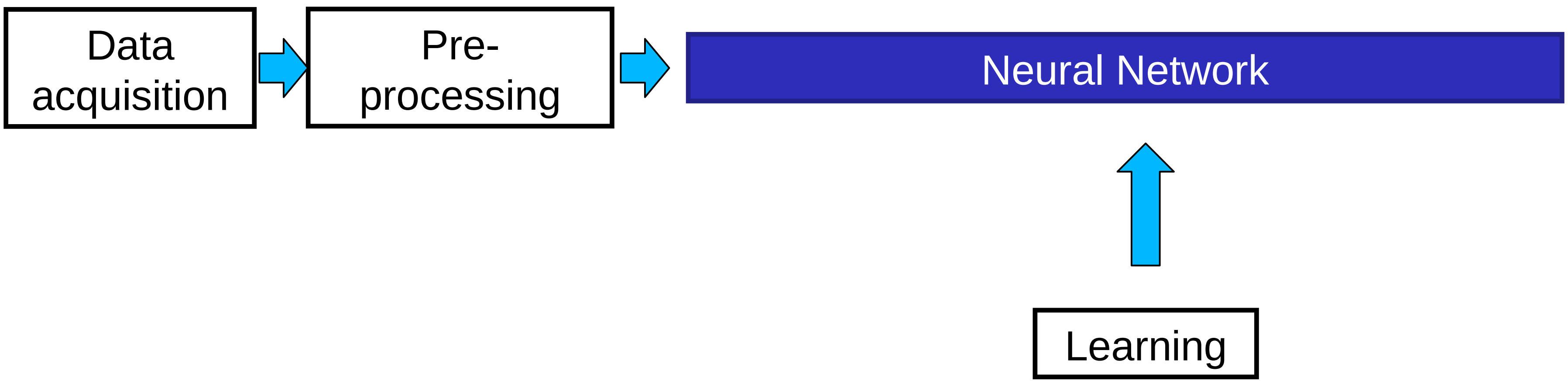
Core Idea: Feature Learning

Classical pattern recognition pipeline



Core Idea: Feature Learning

Neural networks pipeline



Deep Neural Network in action

- Learning representations with increasing level of abstraction
- By passing it with several layers hierarchically, we can classify the images in the output layer

Low-level features → More complex features

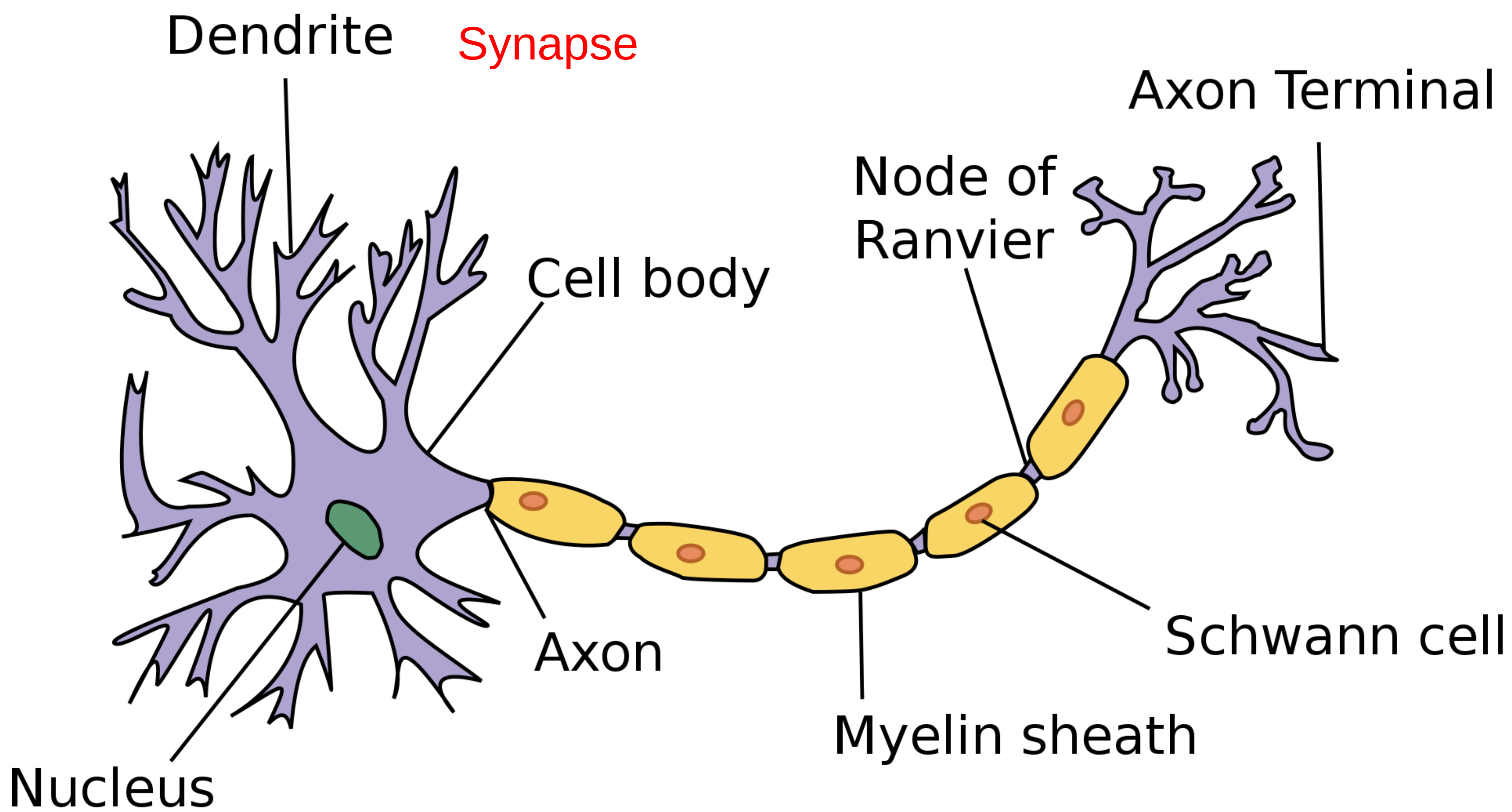
- Image recognition
  - pixel → edge → pattern → motif → part → object
- Text
  - Character → word → word group → clause → sentence → story
- Speech
  - sample → spectral band → sound → … → phone → phoneme → word

Goodfellow, 2016

Perceptron

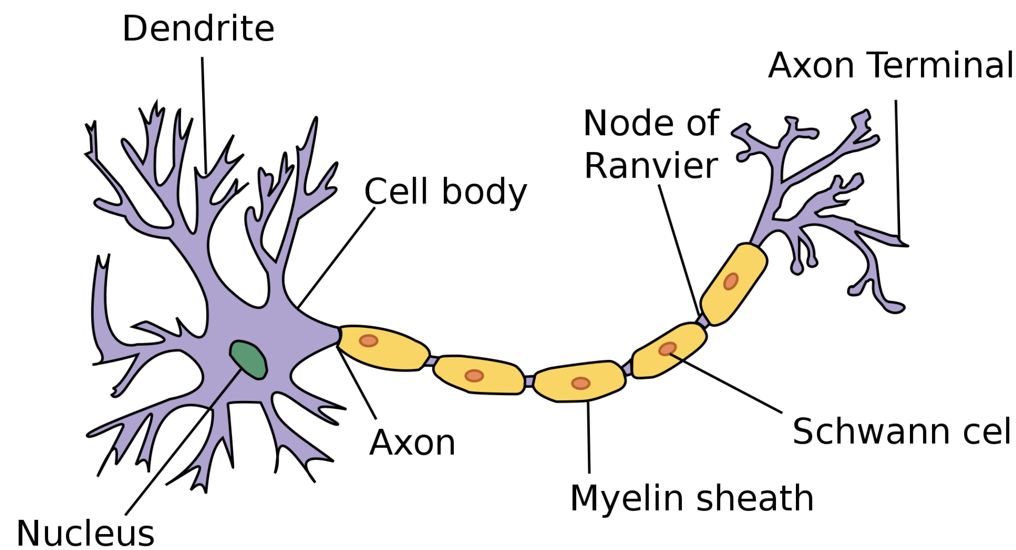
Neuronal Activity in the Brain

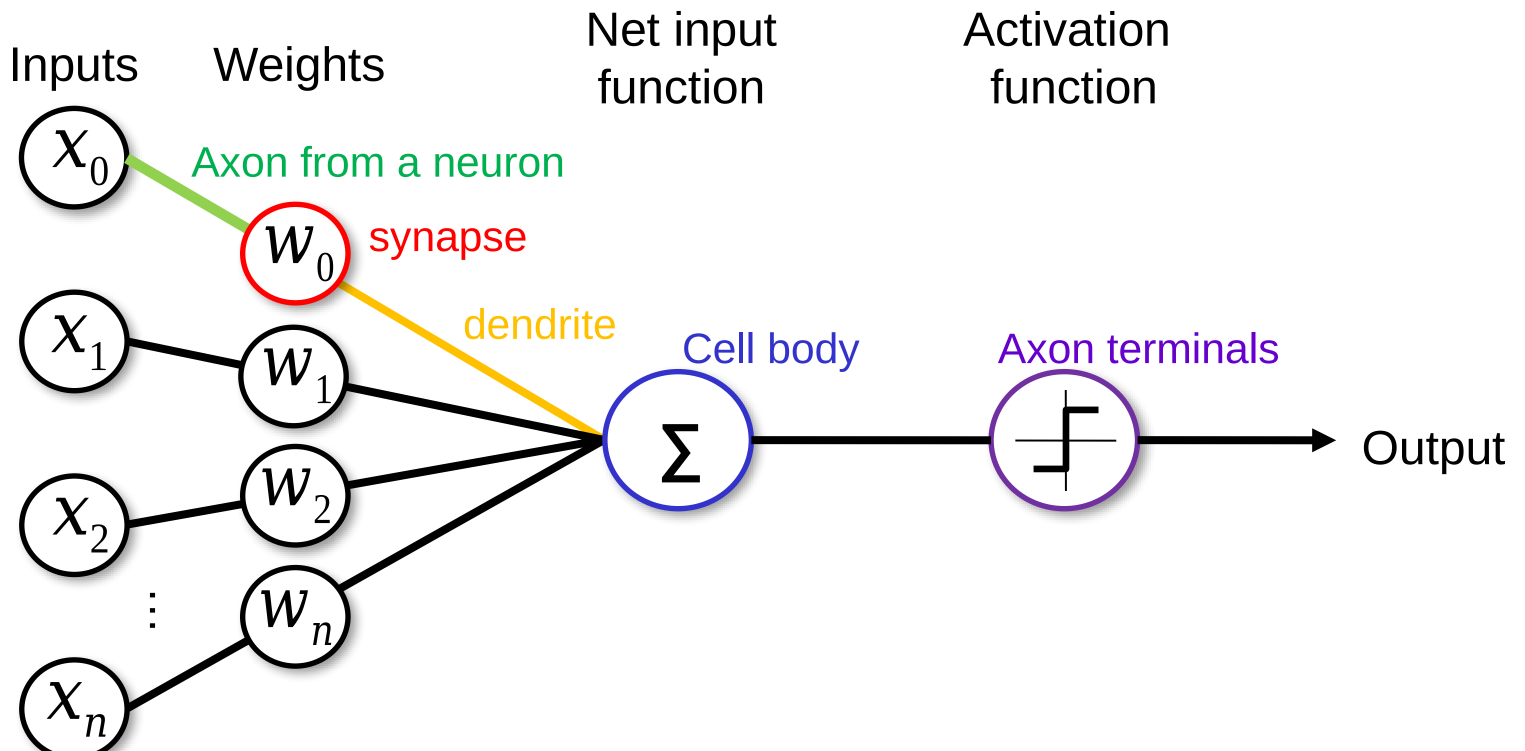
10^{11} neurons of > 20 types, 10^{14} synapses with very complex connections, 1ms–10ms cycle time Signals are noisy "spike trains" of electrical potential



https://commons.wikimedia.org/wiki/Neuron#/media/File:Neuron.svg

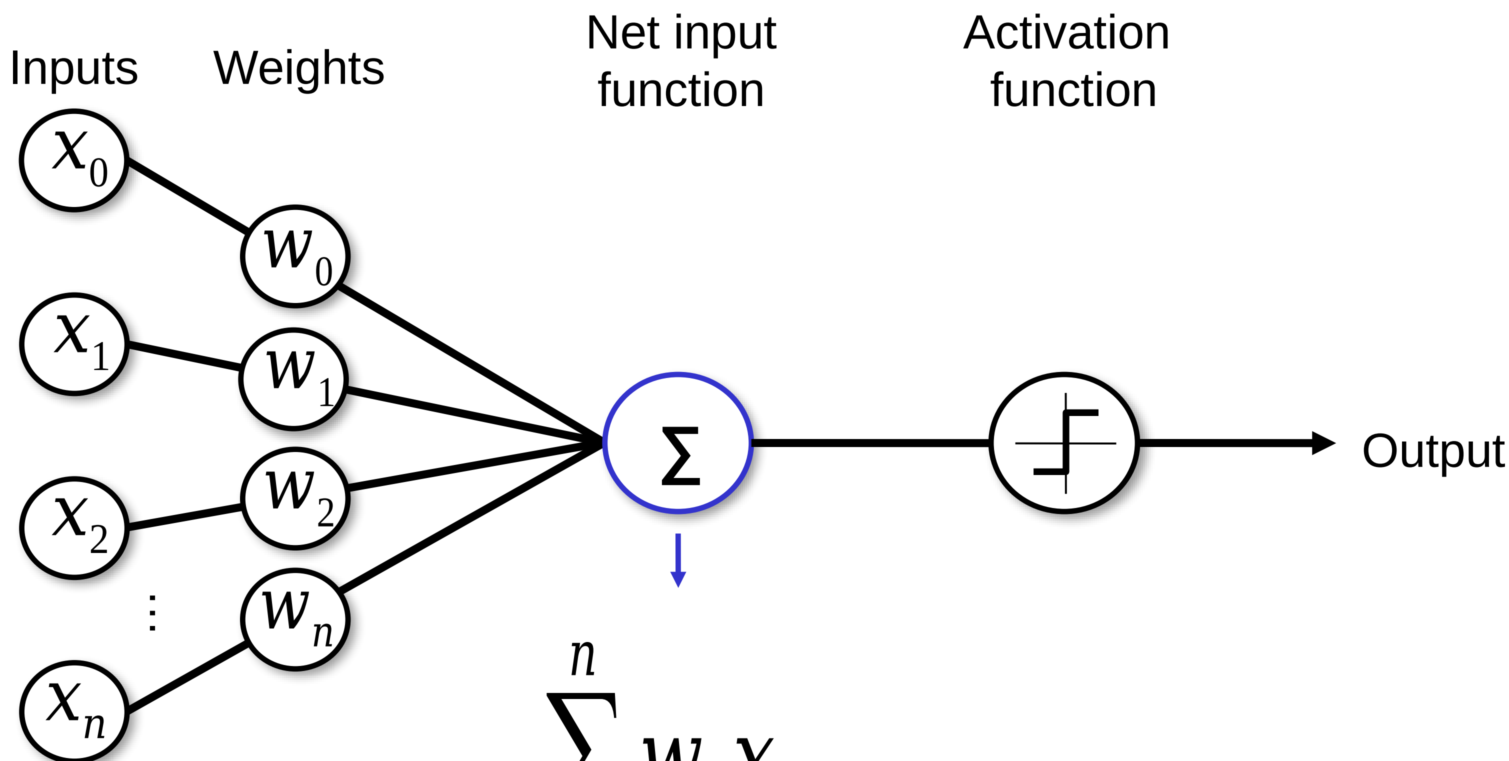
Rosenblatt's Perceptron





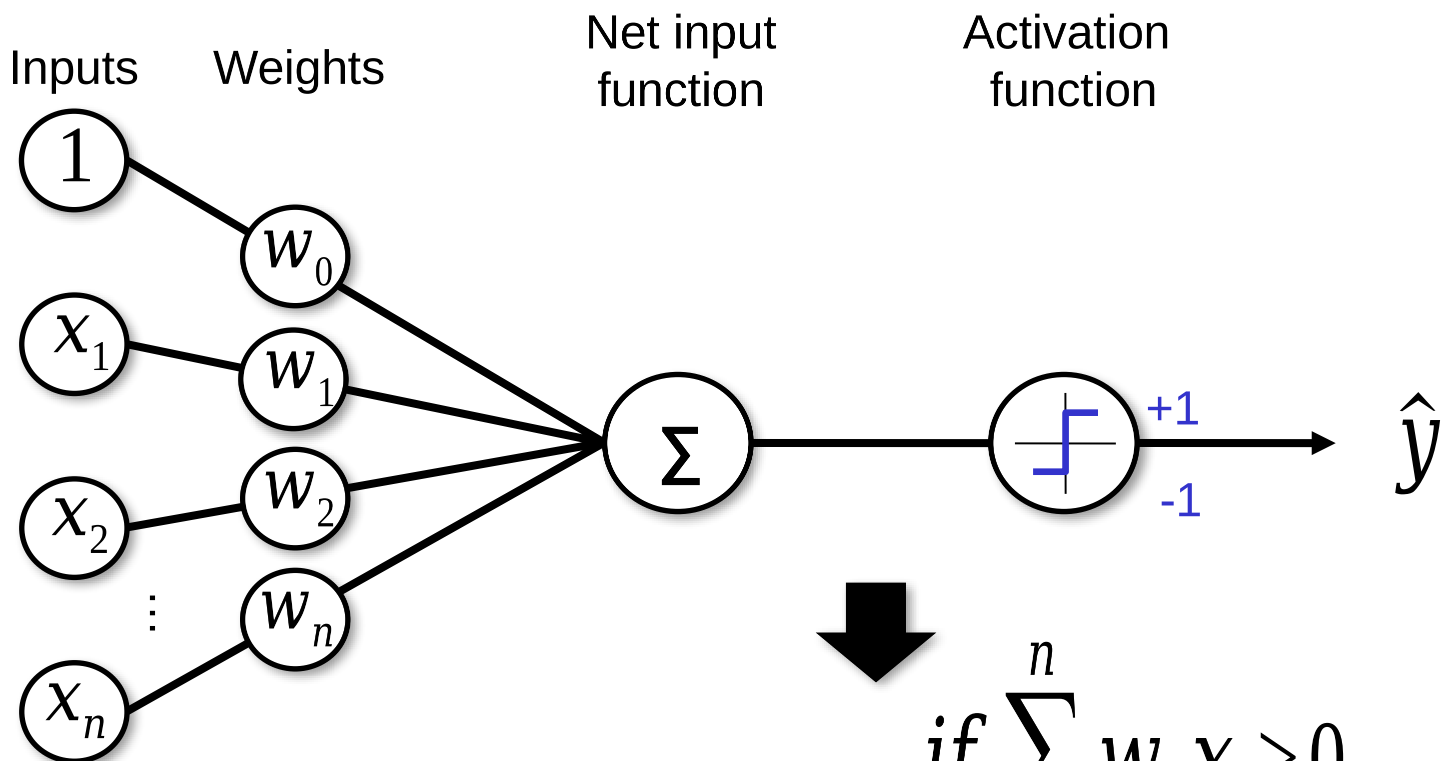
Schematic of Rosenblatt's perceptron

Rosenblatt’s Perceptron: Cell Body



$$\sum_{i=0}^{n} w_i x_i$$

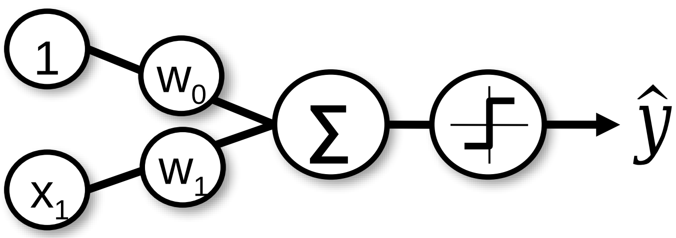
Rosenblatt's Perceptron: Activation Function

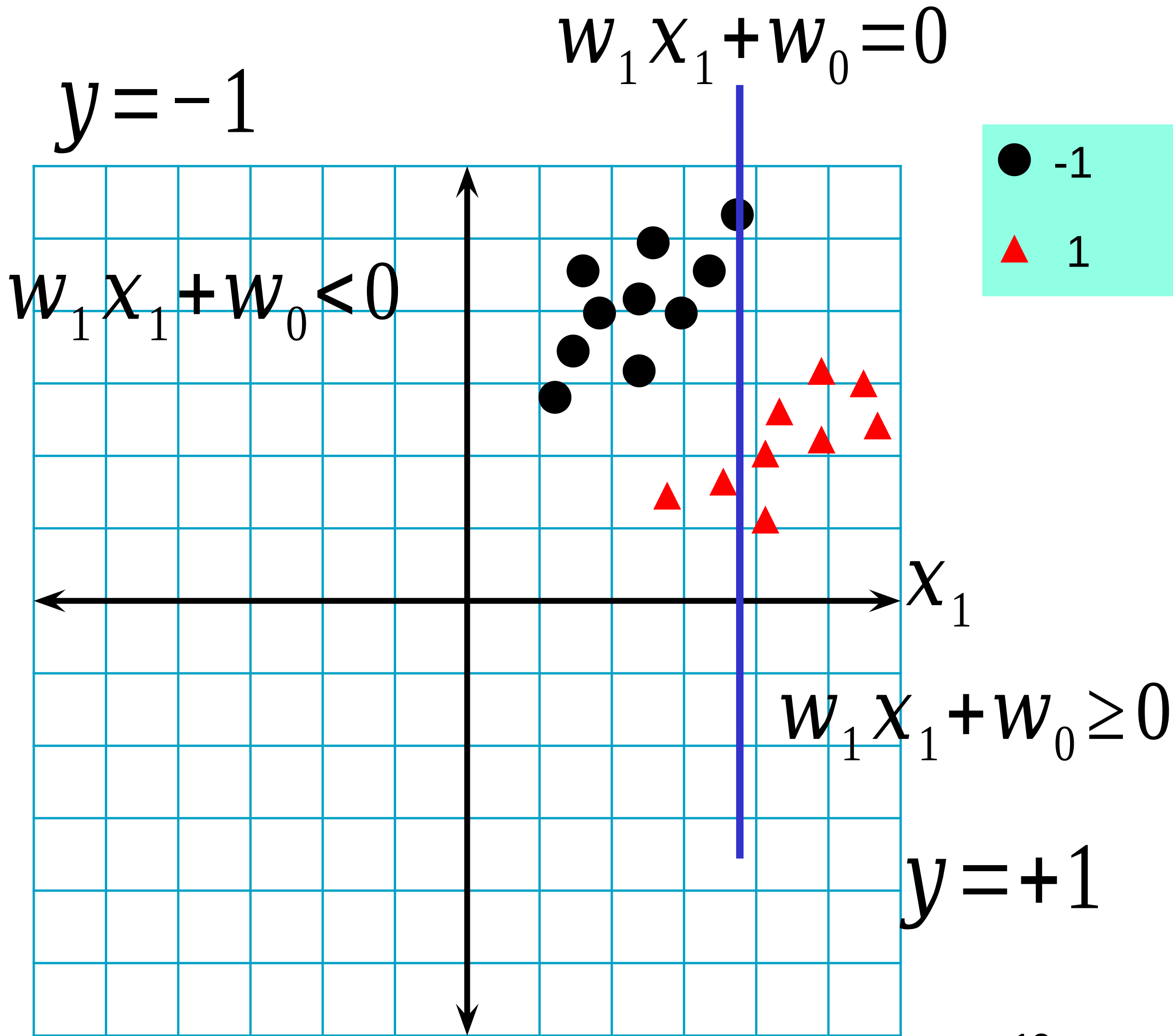


$$\hat{y} = \begin{cases} +1 & \text{if } \sum_{i=0}^{n} w_i x_i \geq 0 \\ -1 & \text{otherwise.} \end{cases}$$

Perceptron on 1-D coordinate

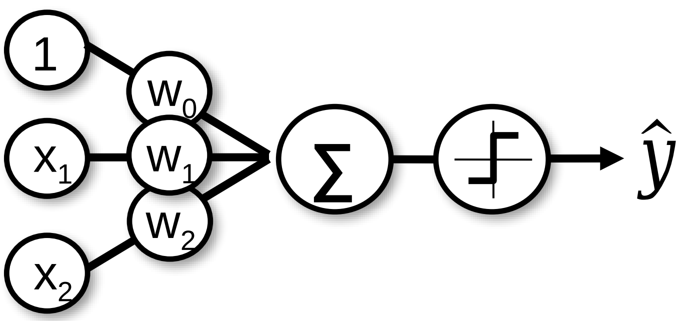
In case if

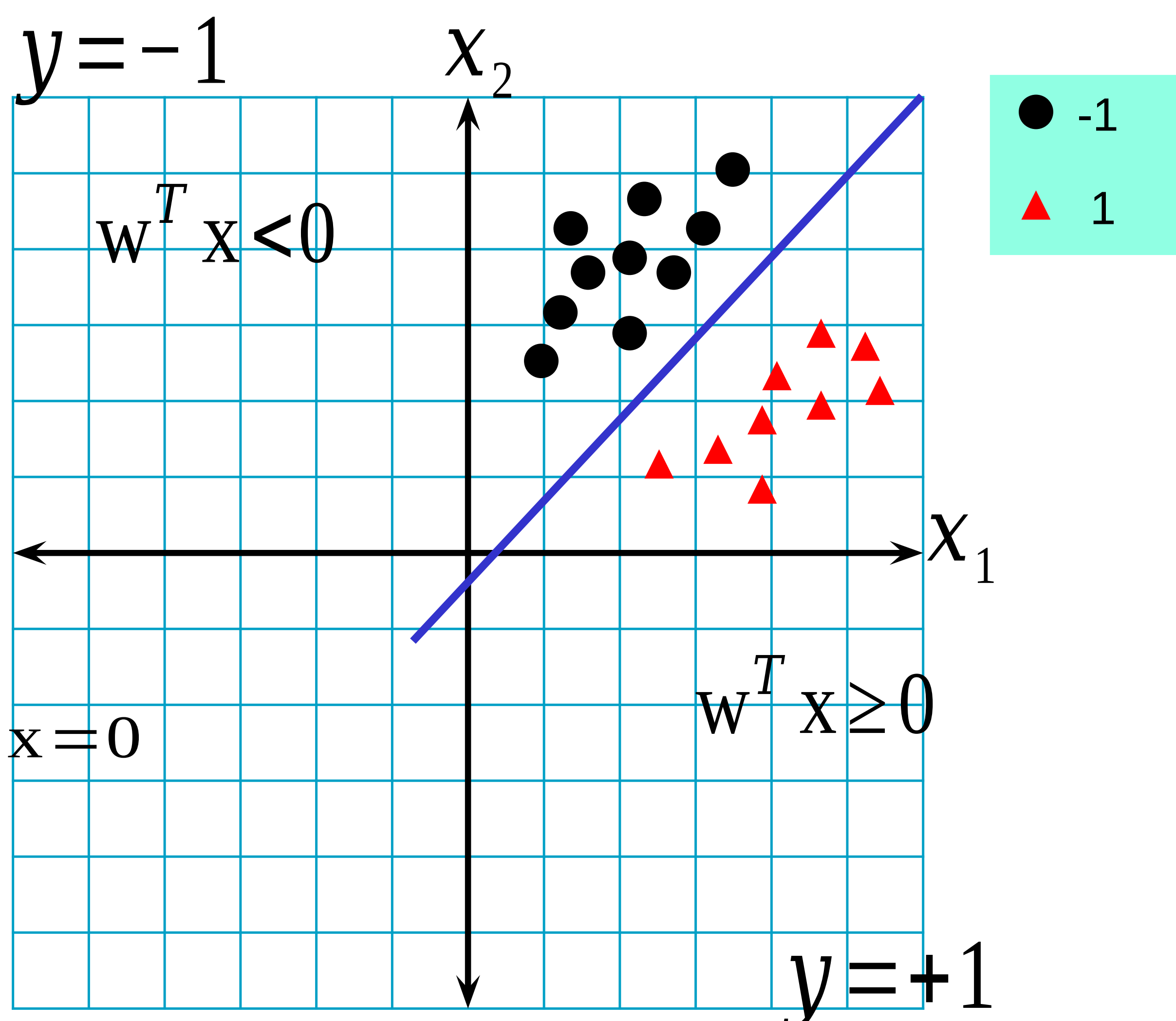




Perceptron on 2-D coordinate

In case if

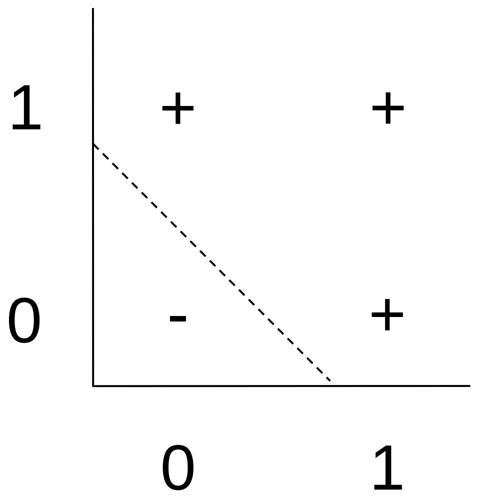




$$\left(w_0\, w_1\, w_2\right)\begin{pmatrix}1\\ x_1\\ x_2\end{pmatrix}=\mathrm{w}^T\mathrm{x}=0$$

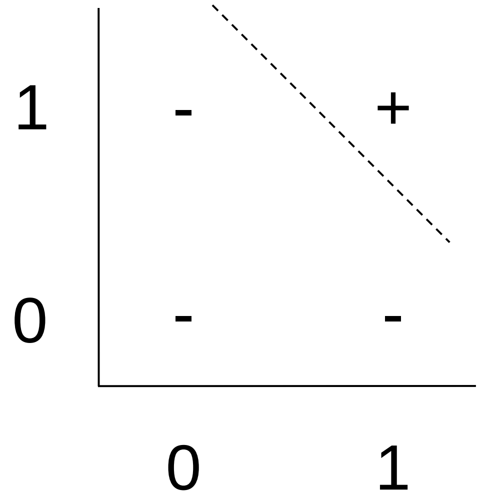
(Simple) AND/OR problem: linearly separable?

or



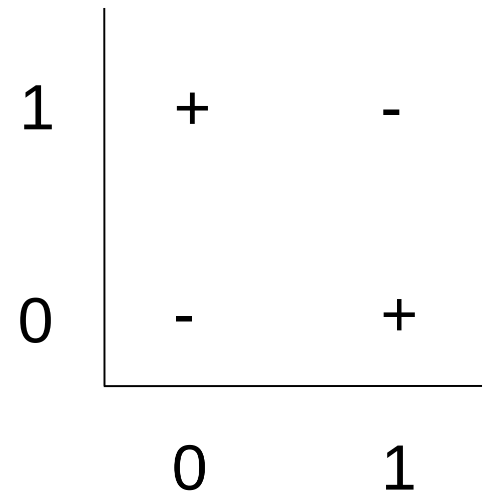
Yep

and



Yep

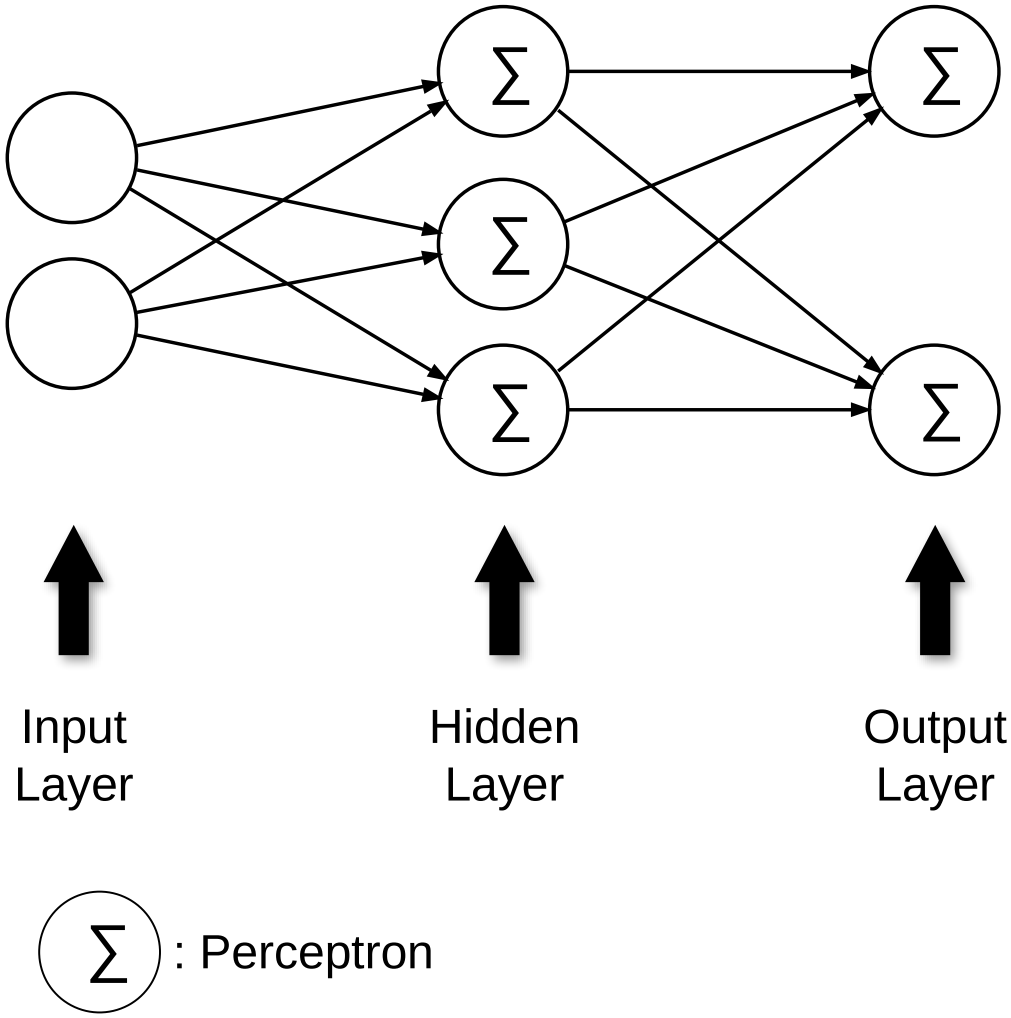
xor



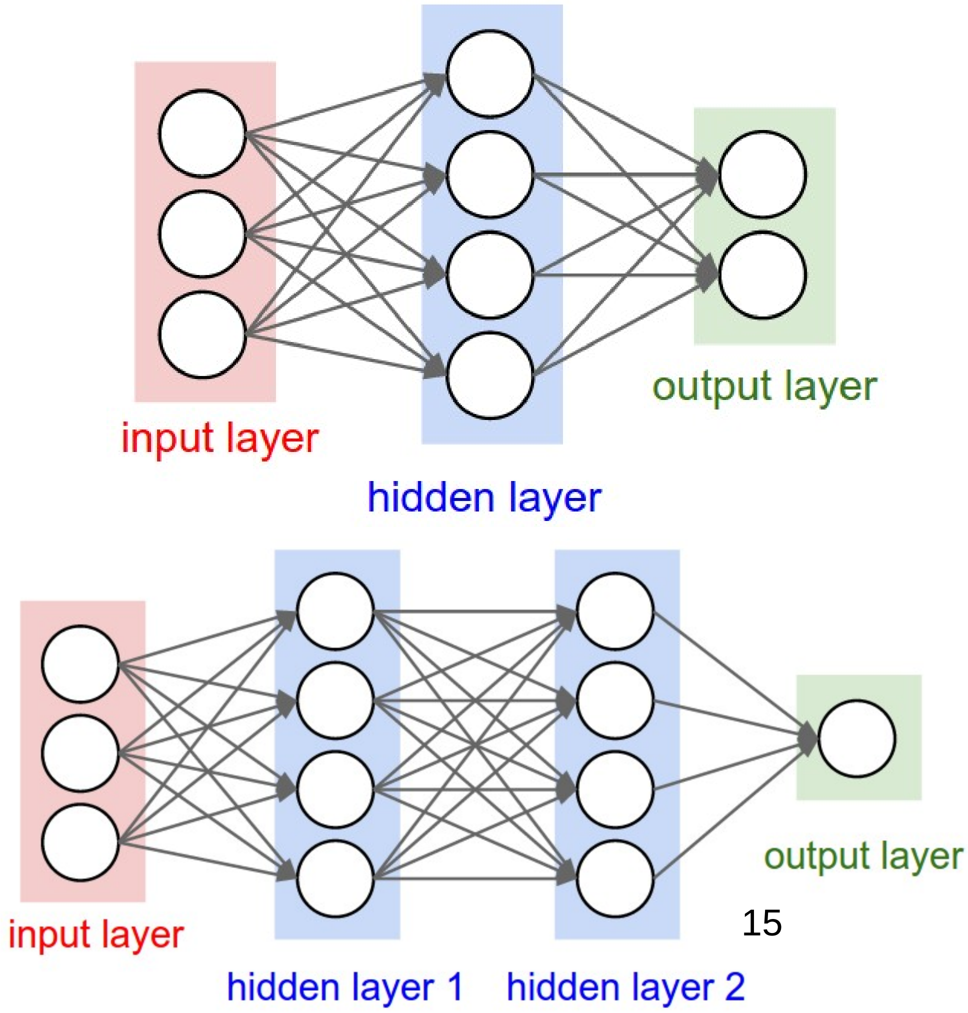
Nope

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

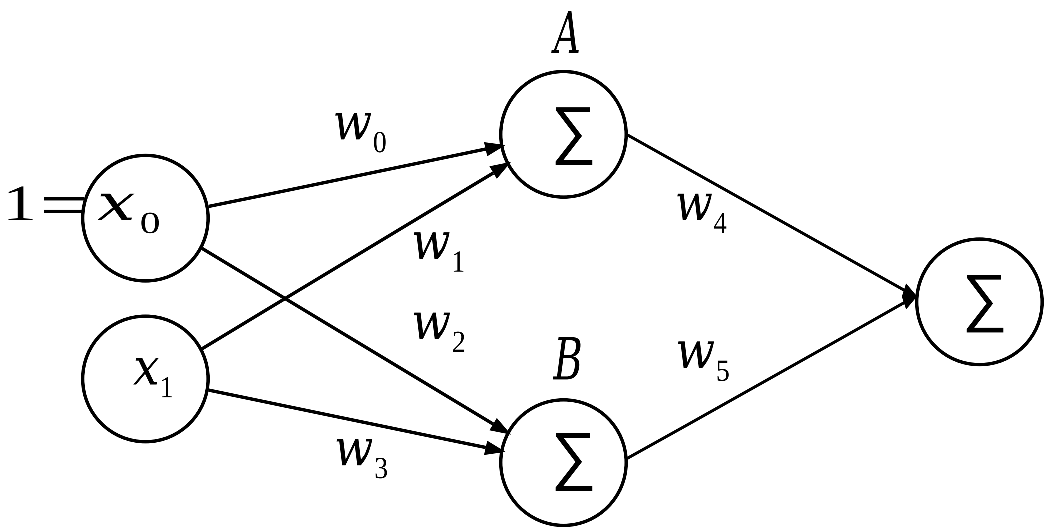
Multi-Layer Perceptron



2-Layer Neural Network



Multi-Layer Perceptron: Limitation



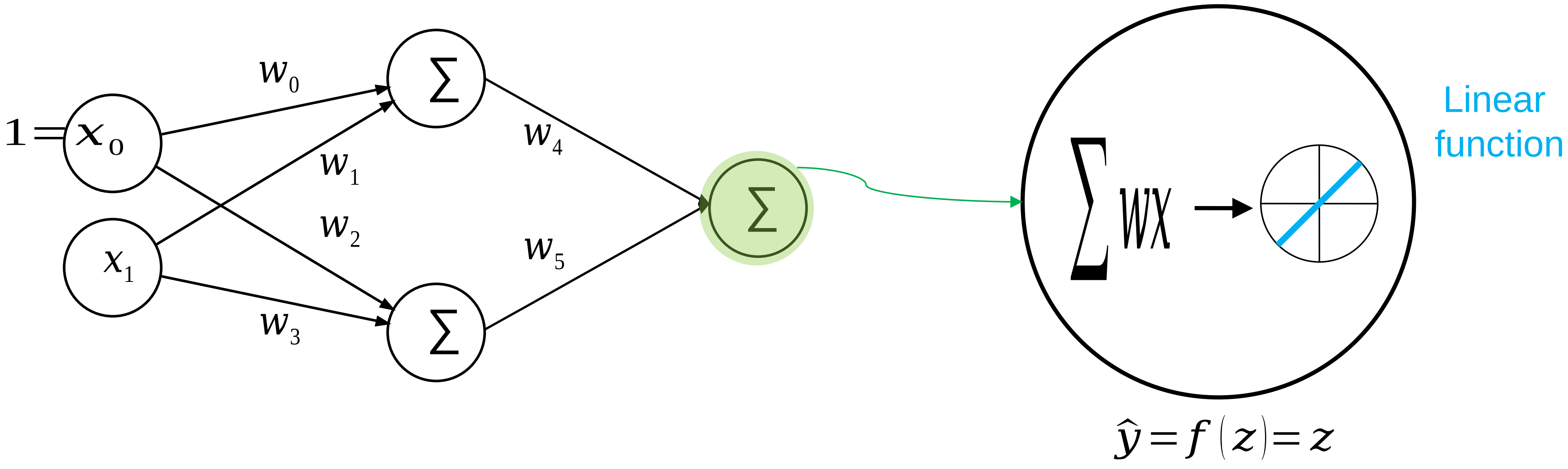
$$w_4 w_0 x_0 + w_5 w_2 x_0 + w_4 w_1 x_1 + w_5 w_3 x_1$$

$$(w_4 w_0 + w_5 w_2) x_0 + (w_4 w_1 + w_5 w_3) x_1$$

$$\hat{y} = w_a + w_b x_1$$

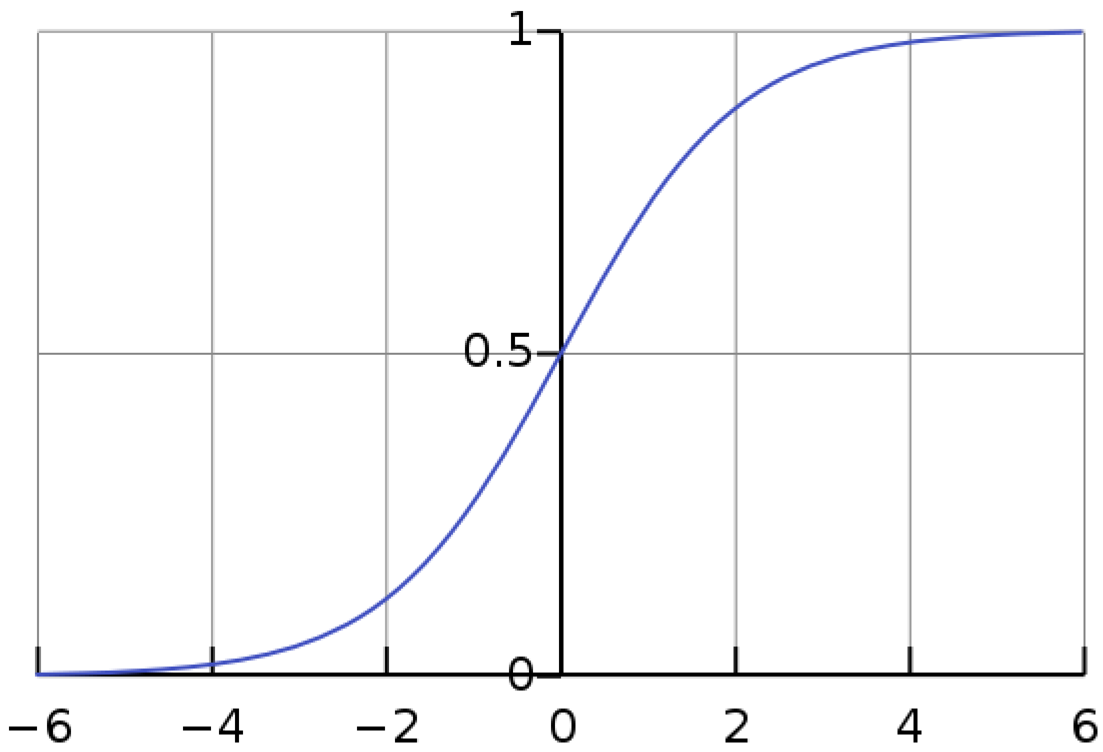
Still Linear equation
(Line, plane, or hyper-plane)

Multi-Layer Perceptron: Limitation



Multi-Layer Perceptron: Activation Function

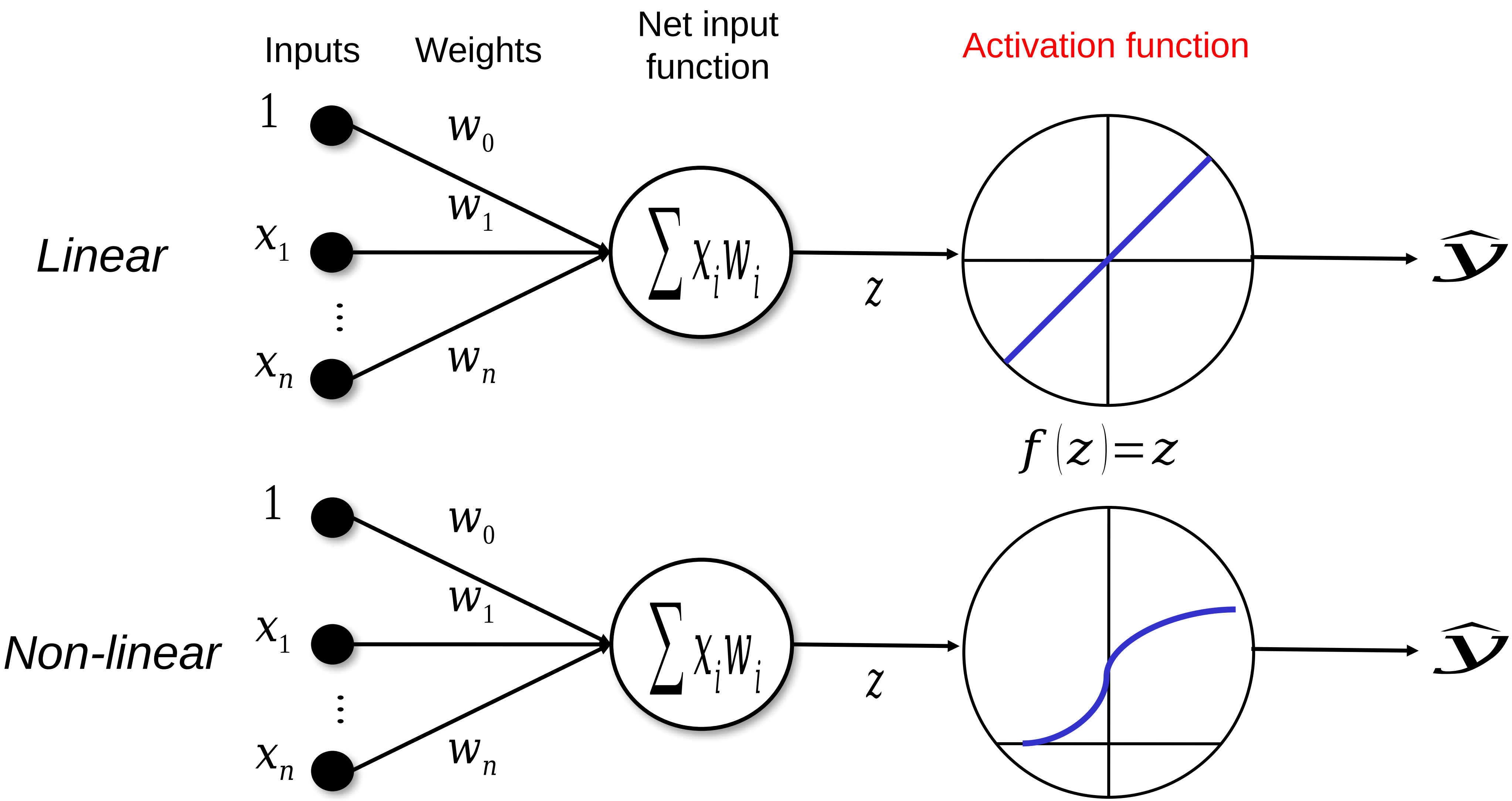
Non-linear function



Sigmoid function

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

Activation Functions

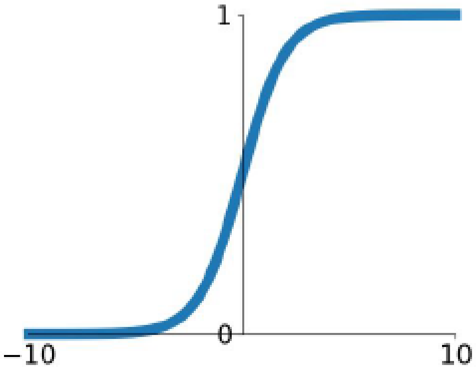


$$f(z) = \frac{1}{1 + e^{-z}}$$

Common Activation Functions

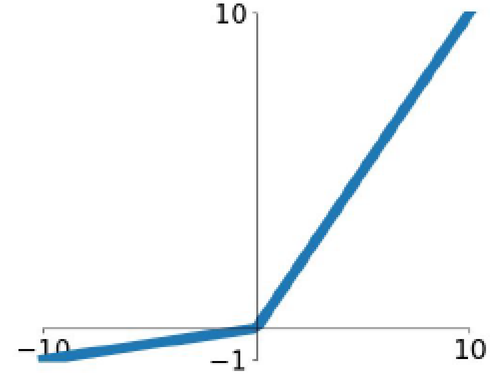
Sigmoid

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



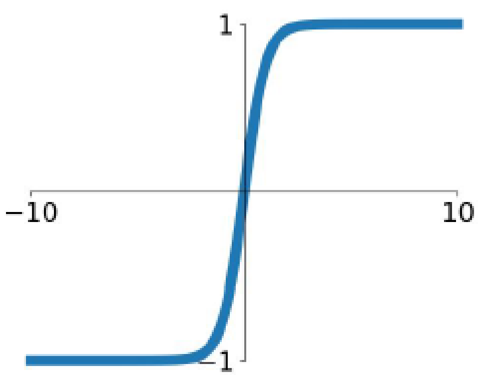
Leaky ReLU

$$max(0.1x, x)$$



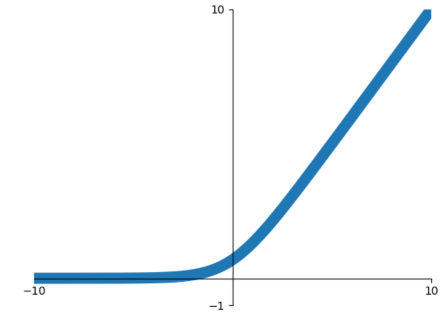
tanh

$$\tanh(x)$$



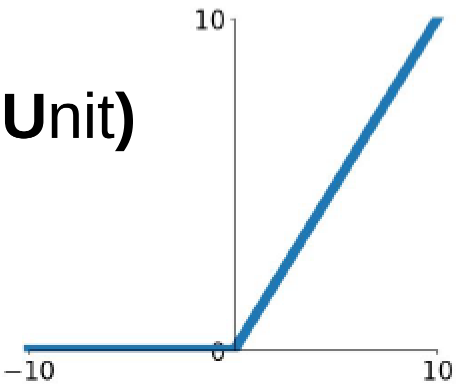
Softplus

$$\ln(1+e^{x})$$



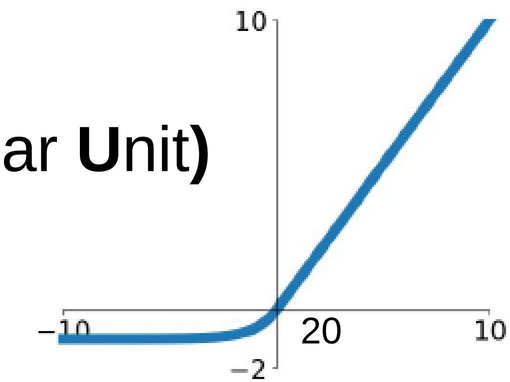
ReLU
(Rectified **L**inear **U**nit**)**

$$max(0, x)$$



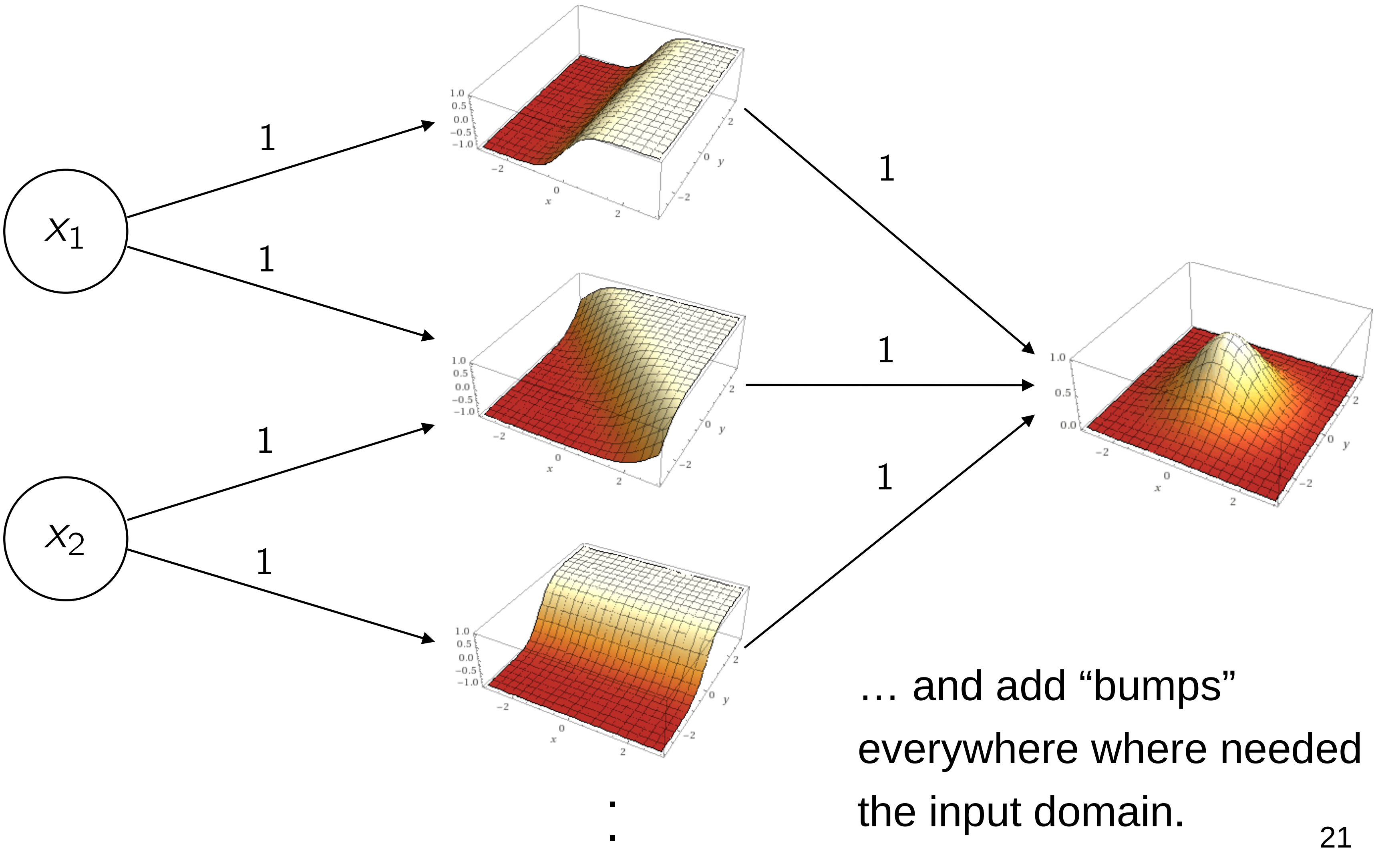
ELU
(Exponential **L**inear **U**nit**)**

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



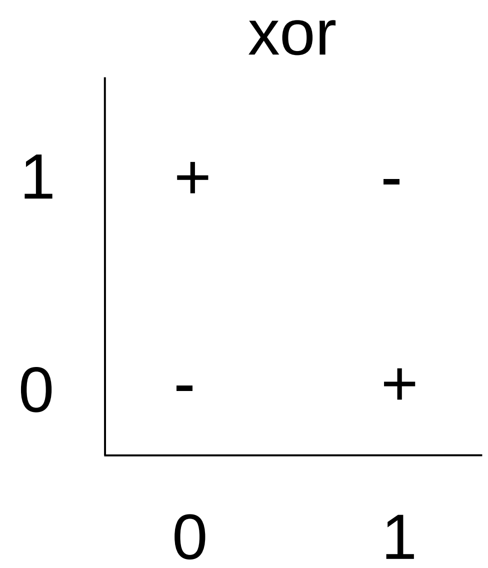
Multi-layer Perceptron is Universal Approximator

"Proof" by construction:

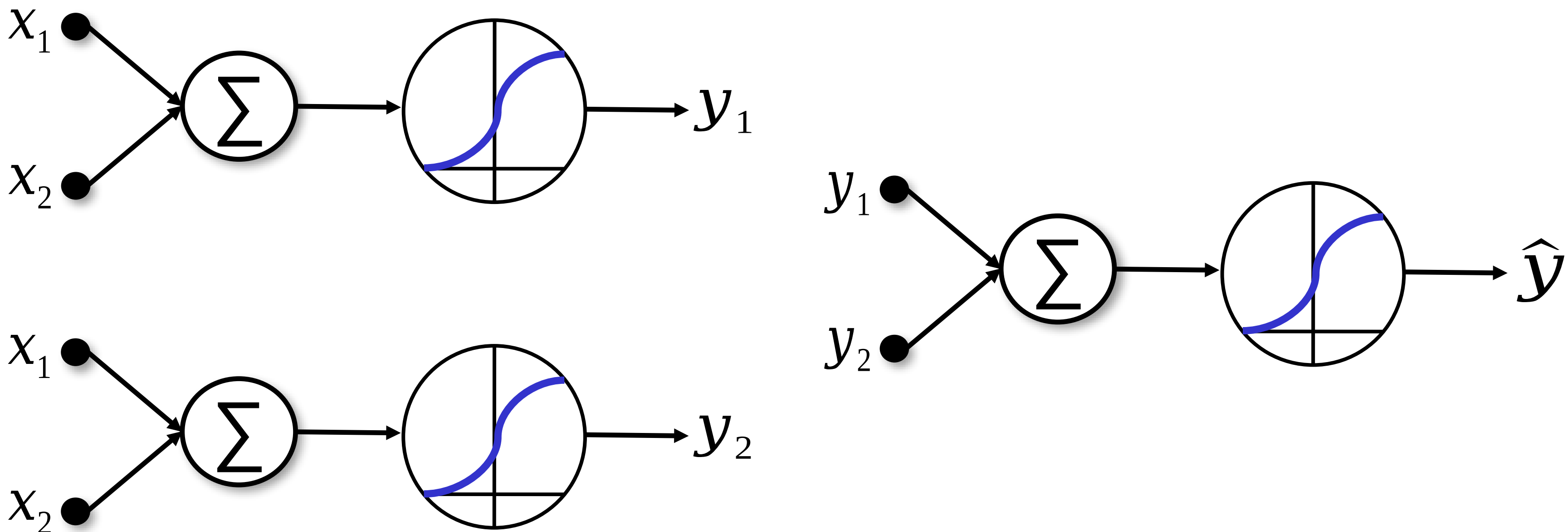


… and add "bumps" everywhere where needed in the input domain.

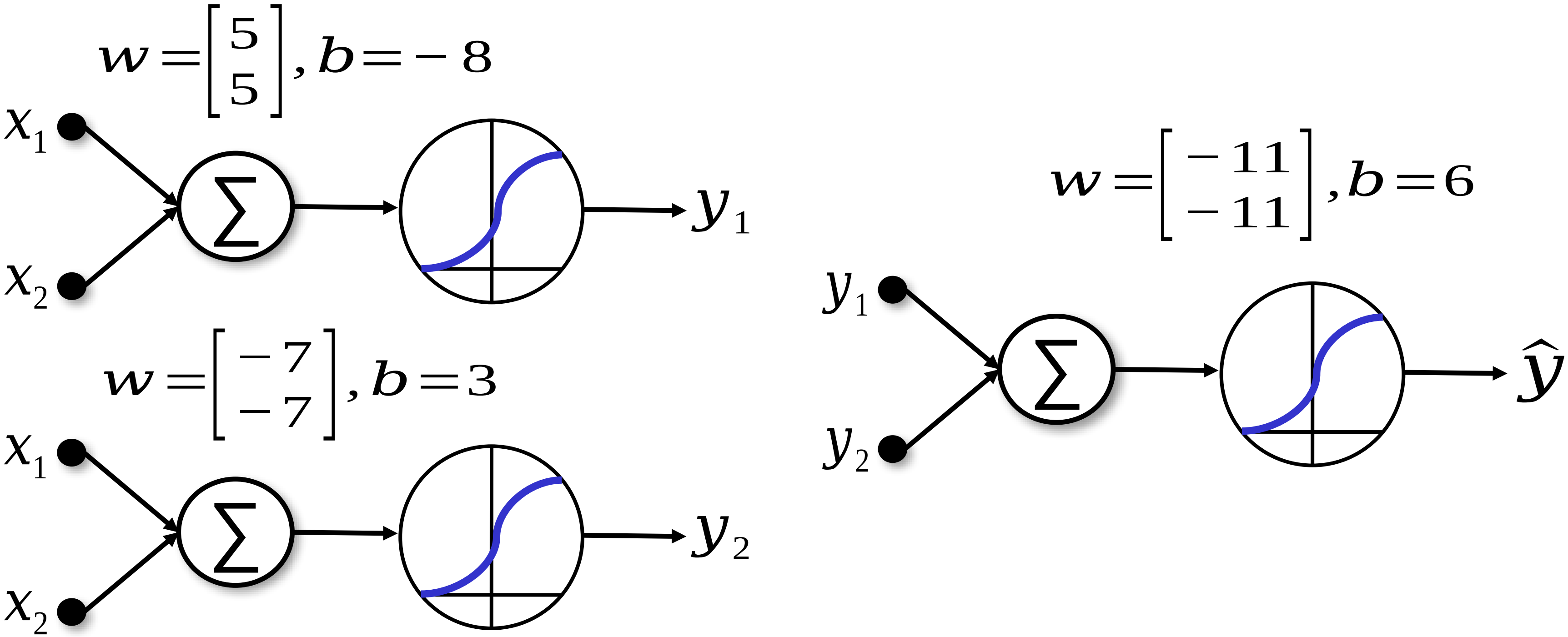
Multi-Layer Perceptron: Simple XOR Example



x_1	x_2	XOR
0	0	0 (-)
0	1	1 (+)
1	0	1 (+)
1	1	0 (-)



Multi-Layer Perceptron: Simple XOR Example



1) In case if $x_1=0$ and $x_2=0$

$$y_1=f\left(\begin{bmatrix}0 & 0\end{bmatrix}\begin{bmatrix}5\\5\end{bmatrix}-8\right)=f(-8)\approx 0$$

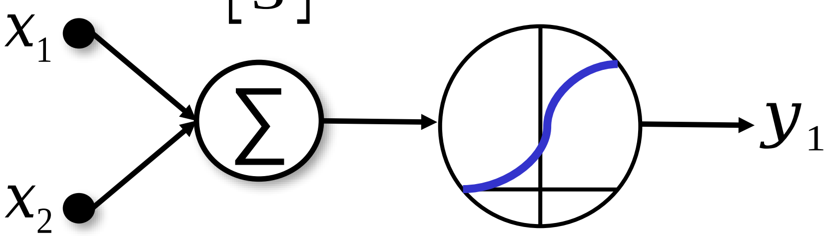
$$y_2=f\left(\begin{bmatrix}0 & 0\end{bmatrix}\begin{bmatrix}-7\\-7\end{bmatrix}+3\right)=f(3)\approx 1$$

$$\hat{y}=f\left(\begin{bmatrix}0 & 1\end{bmatrix}\begin{bmatrix}-11\\-11\end{bmatrix}+6\right)=f(-5)\approx 0$$

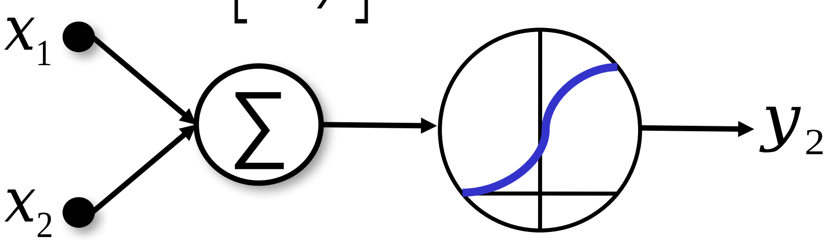
x_1	x_2	y_1	y_2	$\hat{y}$	XOR
0	0	0	1	0	0 (-)
0	1				1 (+)
1	0				1 (+)
1	1				0 (-)

Multi-Layer Perceptron: Simple XOR Example

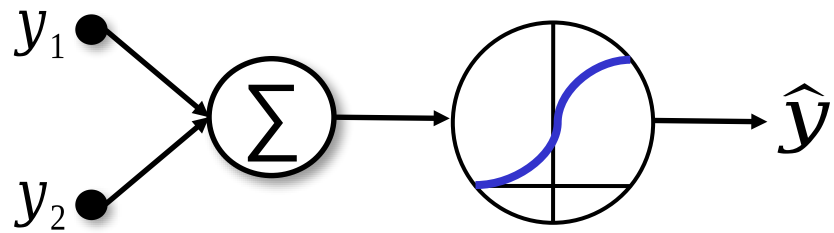
$$w=\begin{bmatrix}5\\5\end{bmatrix}, b=-8$$



$$w=\begin{bmatrix}-7\\-7\end{bmatrix}, b=3$$



$$w=\begin{bmatrix}-11\\-11\end{bmatrix}, b=6$$



1) In case if x_1=0 and x_2=1

$$y_1=f\left([0\ 1]\begin{bmatrix}5\\5\end{bmatrix}-8\right)=f(-3)\approx 0$$

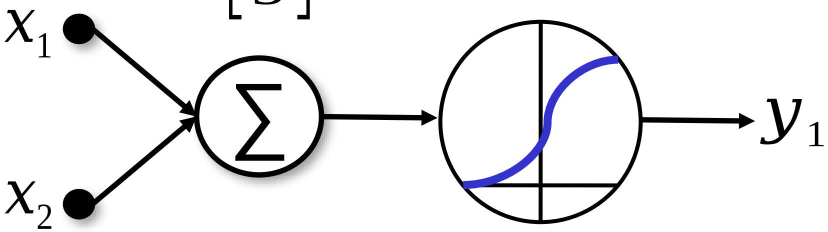
$$y_2=f\left([0\ 1]\begin{bmatrix}-7\\-7\end{bmatrix}+3\right)=f(-4)\approx 0$$

$$\hat{y}=f\left([0\ 0]\begin{bmatrix}-11\\-11\end{bmatrix}+6\right)=f(6)\approx 1$$

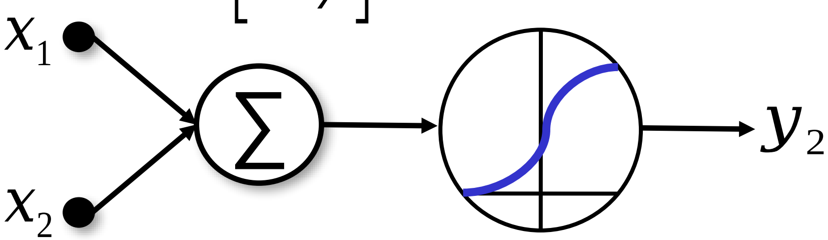
x_1	x_2	y_1	y_2	$\hat{y}$	XOR
0	0	0	1	0	0 (-)
0	1	0	0	1	1 (+)
1	0				1 (+)
1	1				0 (-)

Multi-Layer Perceptron: Simple XOR Example

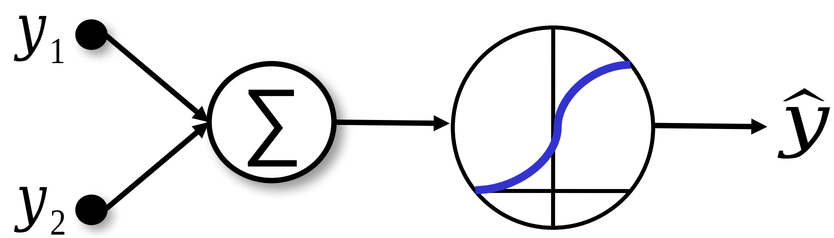
$$w=\begin{bmatrix}5\\5\end{bmatrix}, b=-8$$



$$w=\begin{bmatrix}-7\\-7\end{bmatrix}, b=3$$



$$w=\begin{bmatrix}-11\\-11\end{bmatrix}, b=6$$



1) In case if x_1=1 and x_2=0

$$y_1=f\left(\left[1\,0\right]\begin{bmatrix}5\\5\end{bmatrix}-8\right)=f(-3)\approx 0$$

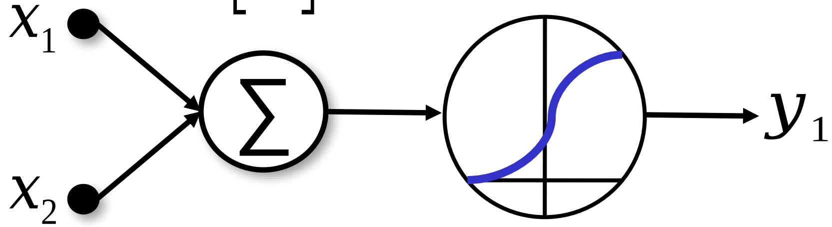
$$y_2=f\left(\left[1\,0\right]\begin{bmatrix}-7\\-7\end{bmatrix}+3\right)=f(-4)\approx 0$$

$$\hat{y}=f\left(\left[0\,0\right]\begin{bmatrix}-11\\-11\end{bmatrix}+6\right)=f(6)\approx 1$$

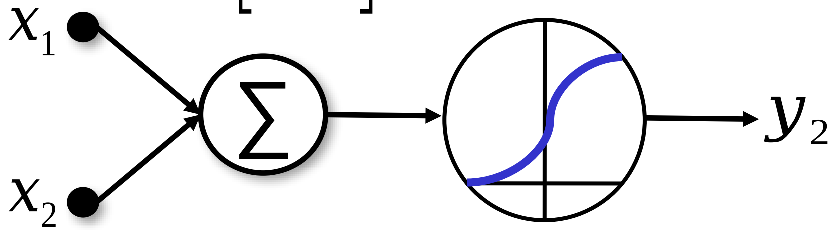
x_1	x_2	y_1	y_2	$\hat{y}$	XOR
0	0	0	1	0	0 (-)
0	1	0	0	1	1 (+)
1	0	0	0	1	1 (+)
1	1				0 (-)

Multi-Layer Perceptron: Simple XOR Example

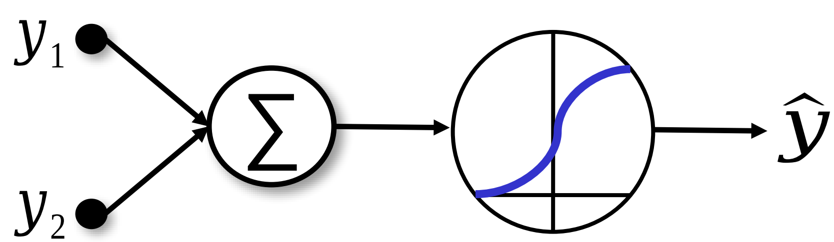
$$w=\begin{bmatrix}5\\5\end{bmatrix}, b=-8$$



$$w=\begin{bmatrix}-7\\-7\end{bmatrix}, b=3$$



$$w=\begin{bmatrix}-11\\-11\end{bmatrix}, b=6$$



1) In case if x_1=1 and x_2=1

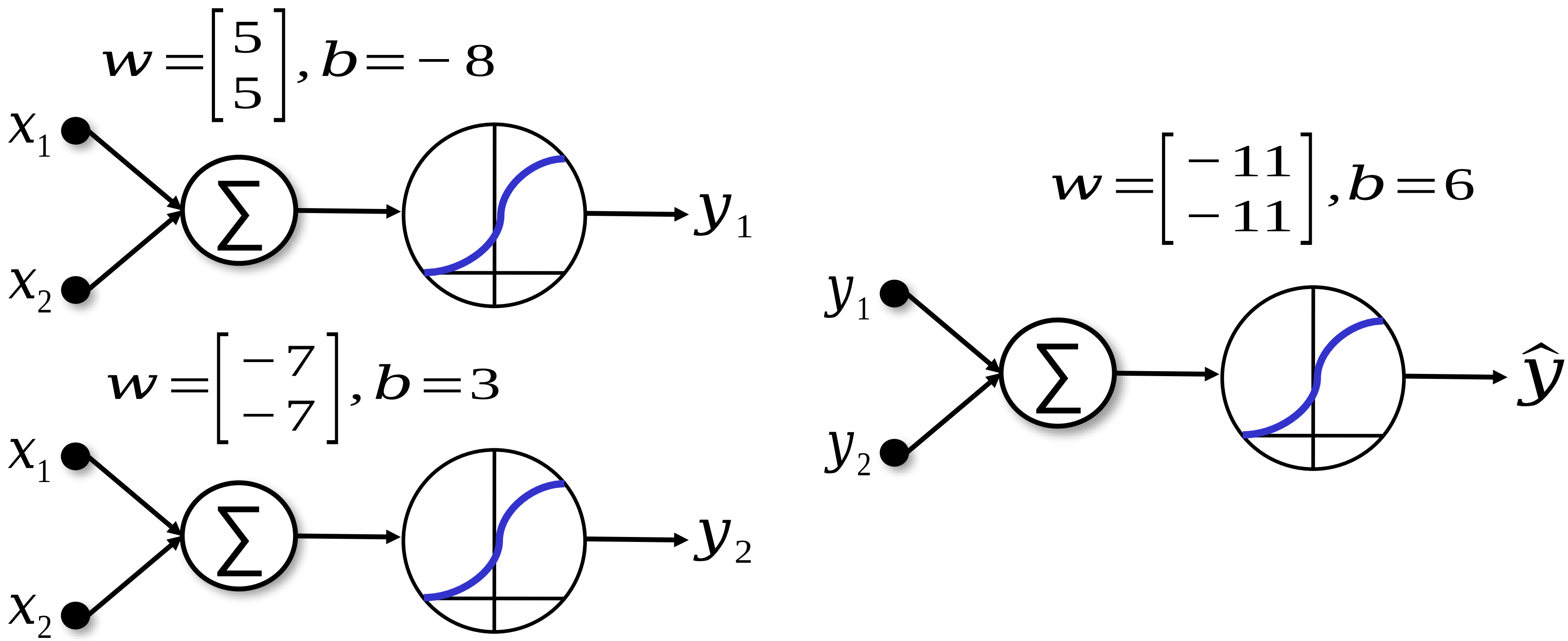
$$y_1=f\left([1\,1]\begin{bmatrix}5\\5\end{bmatrix}-8\right)=f(2)\approx 1$$

$$y_2=f\left([1\,1]\begin{bmatrix}-7\\-7\end{bmatrix}+3\right)=f(-11)\approx 0$$

$$\hat{y}=f\left([1\,0]\begin{bmatrix}-11\\-11\end{bmatrix}+6\right)=f(-5)\approx 0$$

x_1	x_2	y_1	y_2	$\hat{y}$	XOR
0	0	0	1	0	0 (-)
0	1	0	0	1	1 (+)
1	0	0	0	1	1 (+)
1	1	1	0	0	0 (-)

Multi-Layer Perceptron: Simple XOR Example



Question: How can we learn W and B from training data?

Cost(= Loss) Function

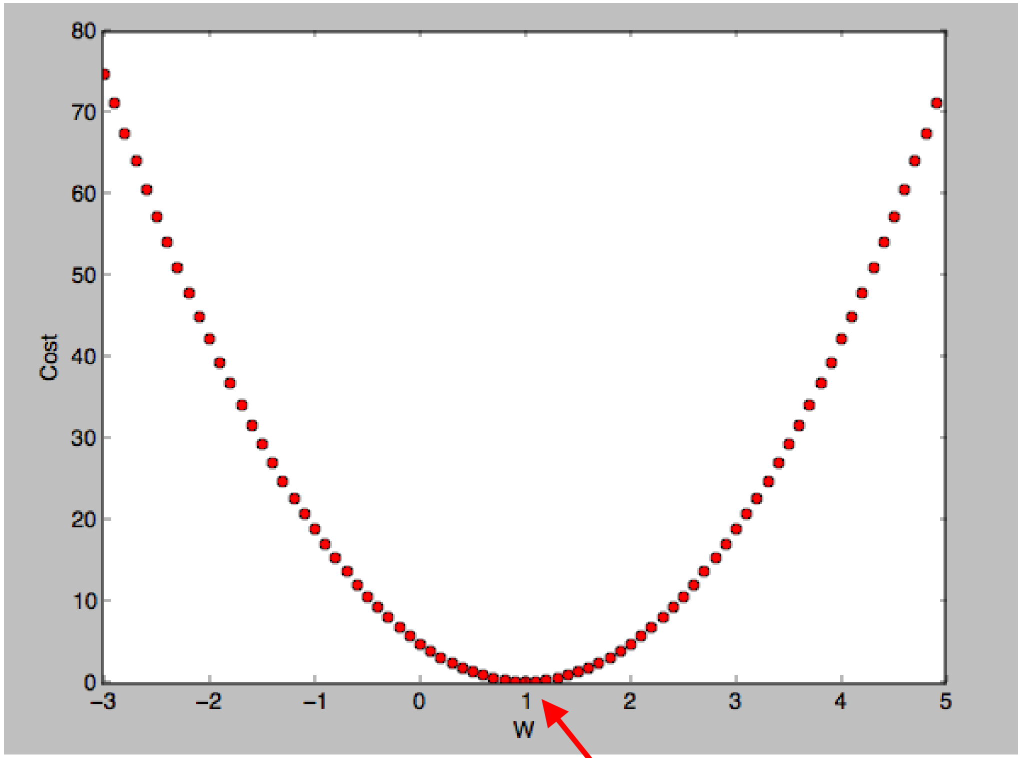
Loss: $$E(\mathrm{w}) \equiv \frac{1}{|D|} \sum_{d \in D} \underbrace{\left(y^{(d)} - \hat{y}^{(d)}\right)^2}_{\text{Difference between target value and output value for training sample}}$$

Our objective is to find w which minimizes cost function

$$\underset{\mathrm{w}}{minimize}\, E(\mathrm{w})$$

How cost(W) looks like?

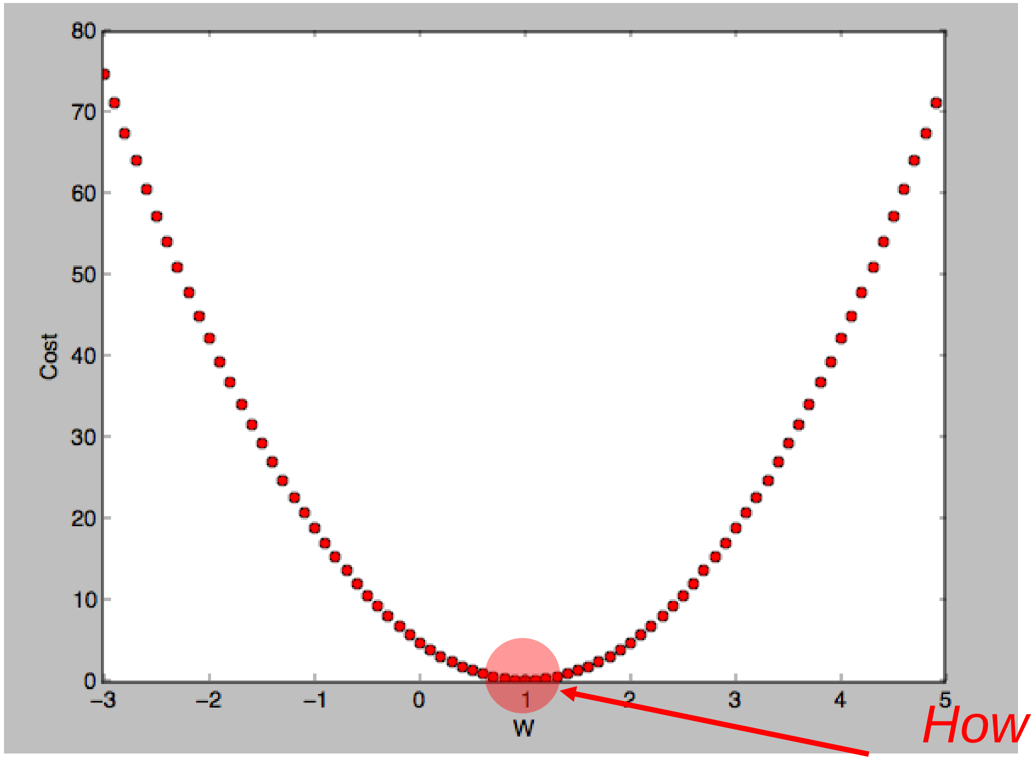
$$E(\mathrm{w}) \equiv \frac{1}{2} \sum_{d \in D} \left(y^{(d)} - \hat{y}^{(d)} \right)^2$$

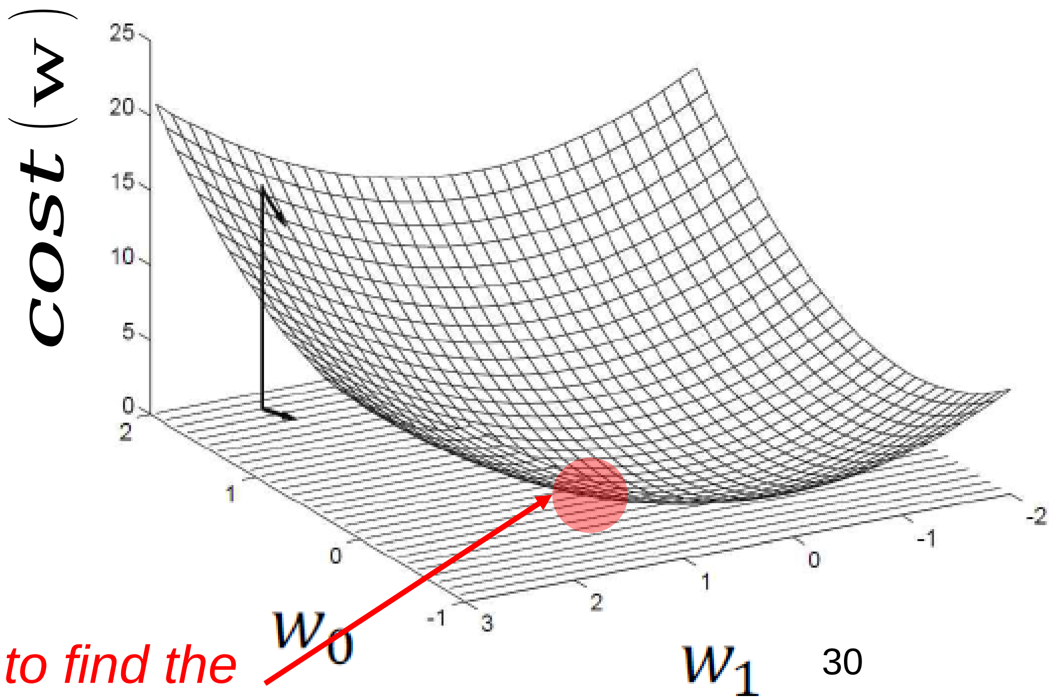


Minimum point = 1

How to minimize cost?

$$E(\mathrm{w}) \equiv \frac{1}{2} \sum_{d \in D} \left(y^{(d)} - \hat{y}^{(d)} \right)^2$$

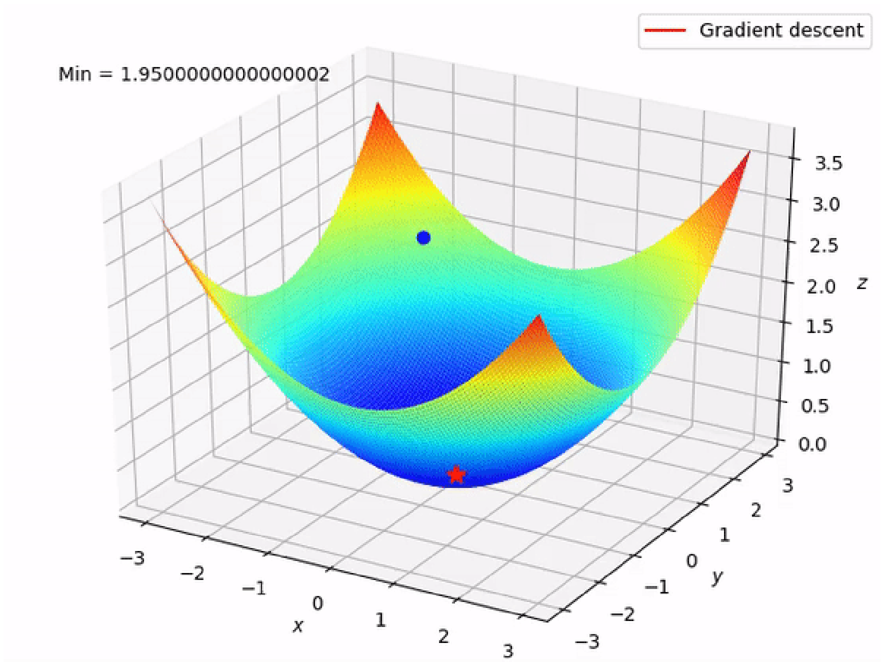




How to find the minimum point?

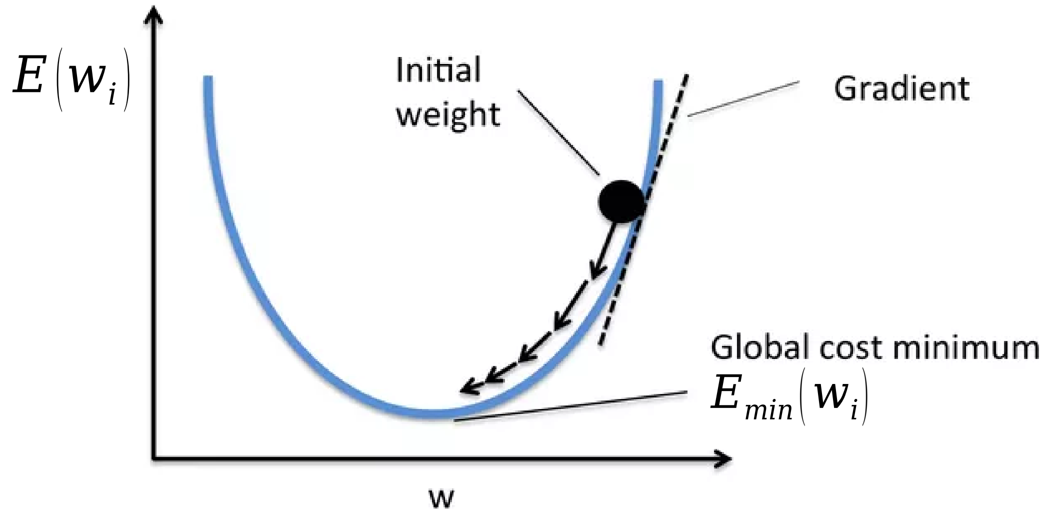
Gradient Descent Algorithm

- Minimize cost function
- Gradient descent is used for many minimization problems
- For a given cost function, it will find w to minimize cost
- Repeat until you converge to a local minimum



Gradient Descent Algorithm

How it works?

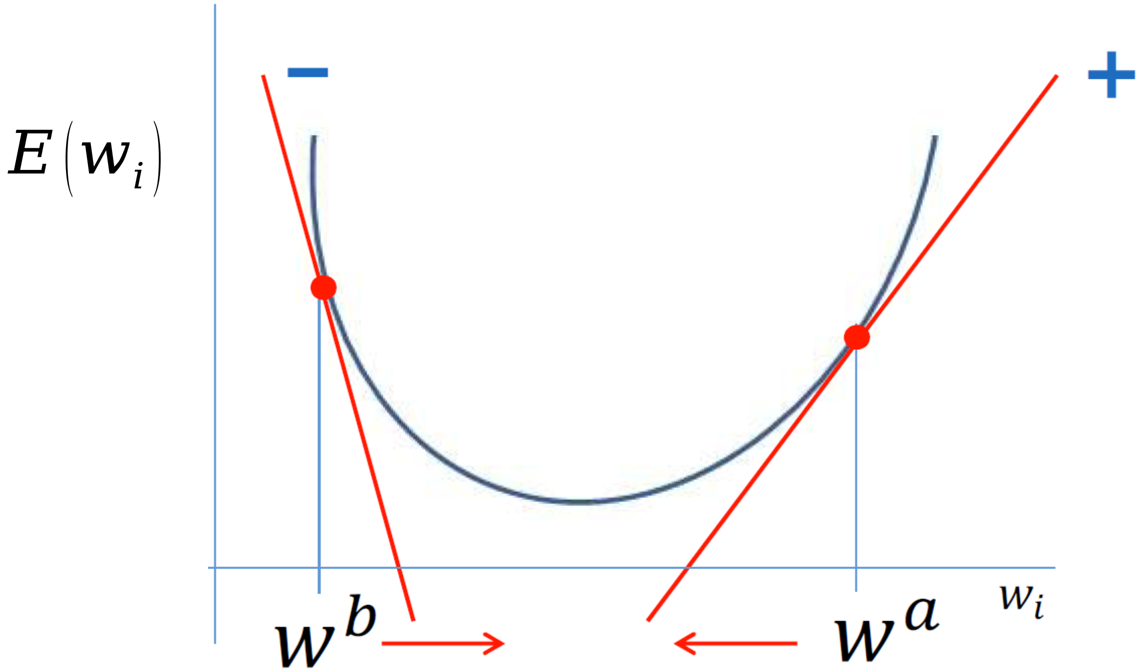


1. Start with initial guesses
   - Start at random value

2. Each weight is updated by taking a step into the opposite direction of the gradient
   - Compute the partial derivative of the cost function for each weight

3. Repeat until you converge to a local minimum

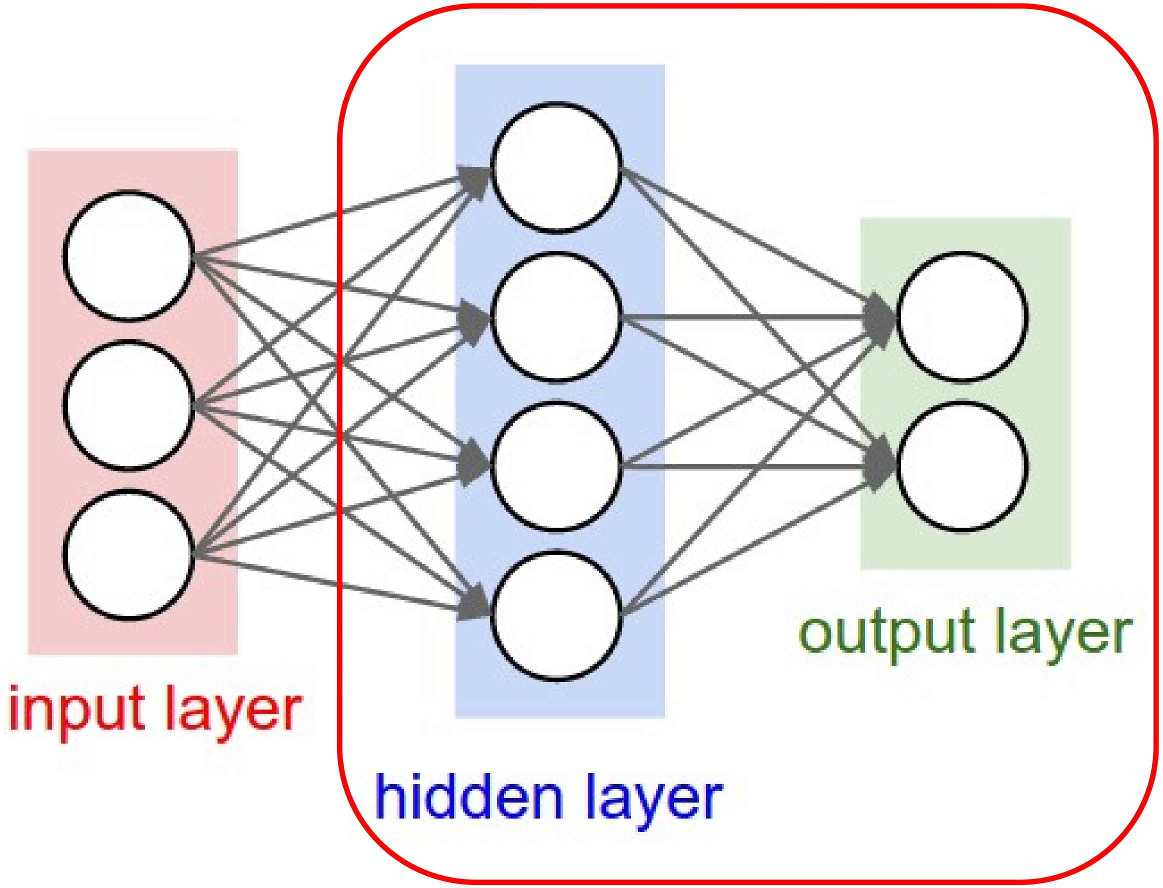
Gradient Descent Algorithm



$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = -\eta \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} \left(y^d - \hat{y}^d \right)^2 = -\eta \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} \left(y^d - \hat{y}^d \right)^2$$

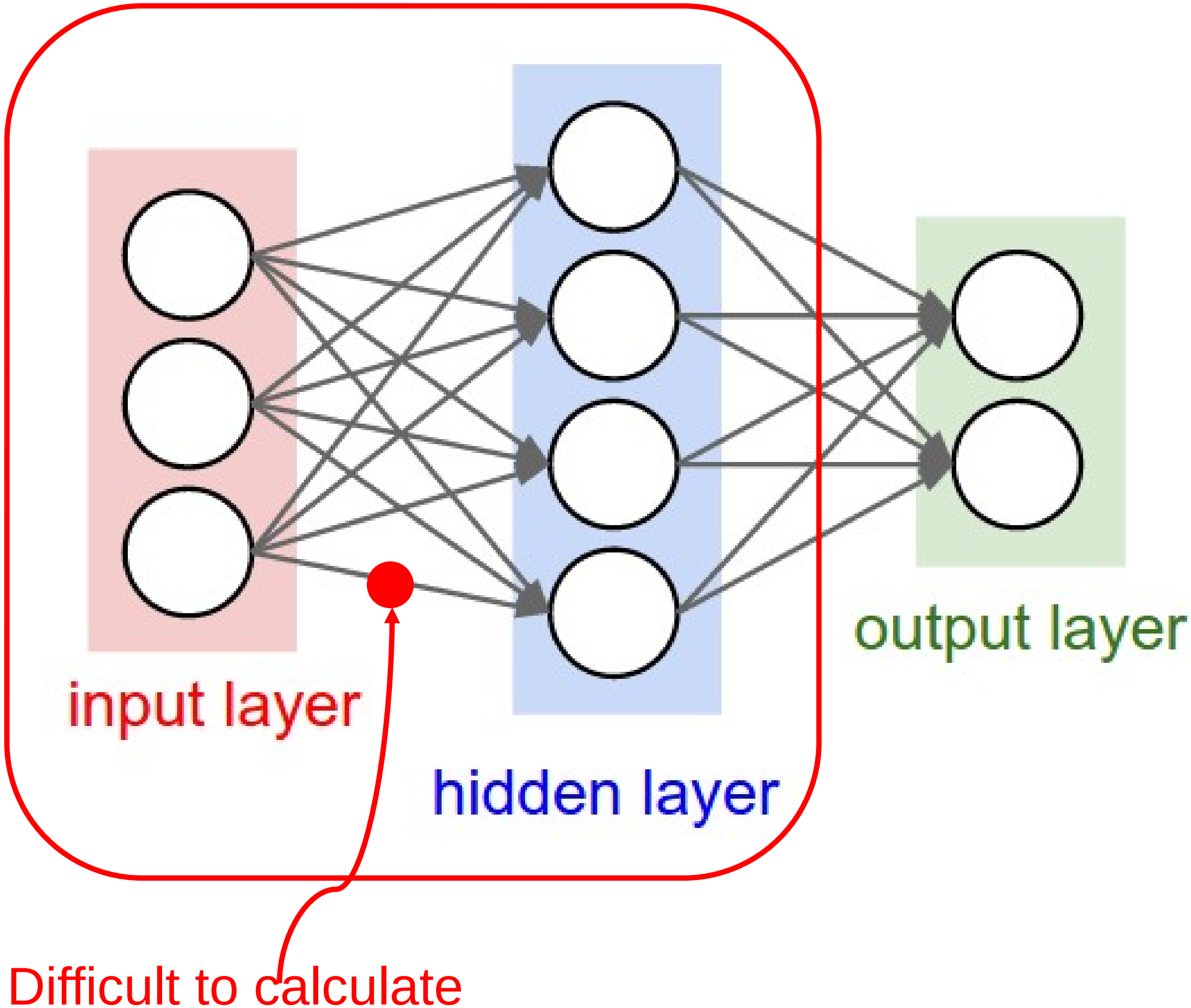
$$\Delta w_i = \eta \times \sum_{d \in D} \left(y^{(d)} - \hat{y}^{(d)} \right) \times \left(-x_i \right)$$

Learning on Multi-Layer Perceptron

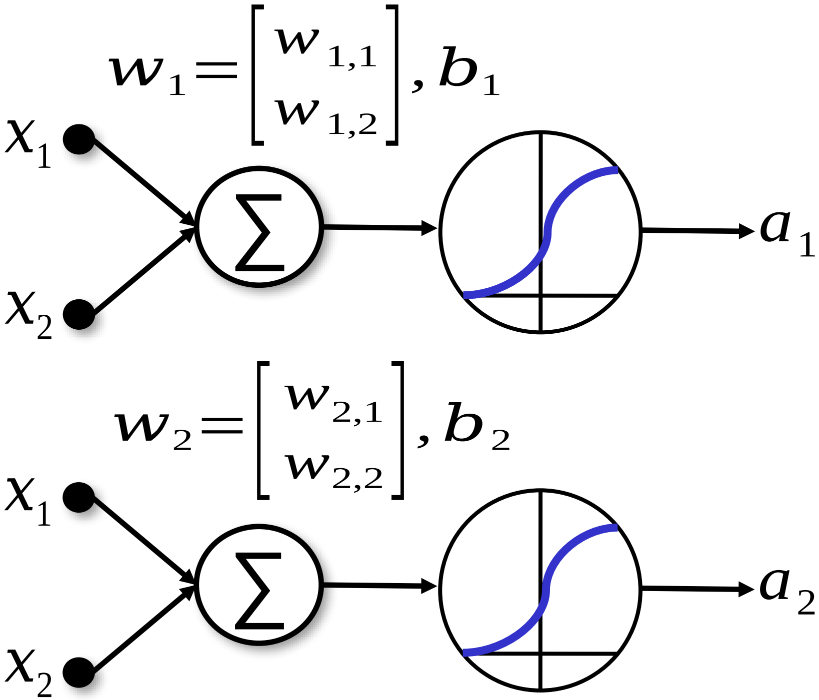


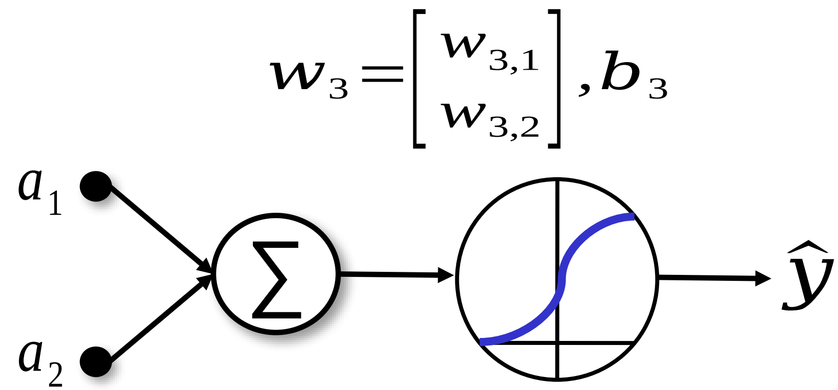
$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d \left(y^{(d)} - \hat{y}^{(d)} \right)^2$$

Learning on Multi-Layer Perceptron



Backpropagation with Multi-Layer Perceptron





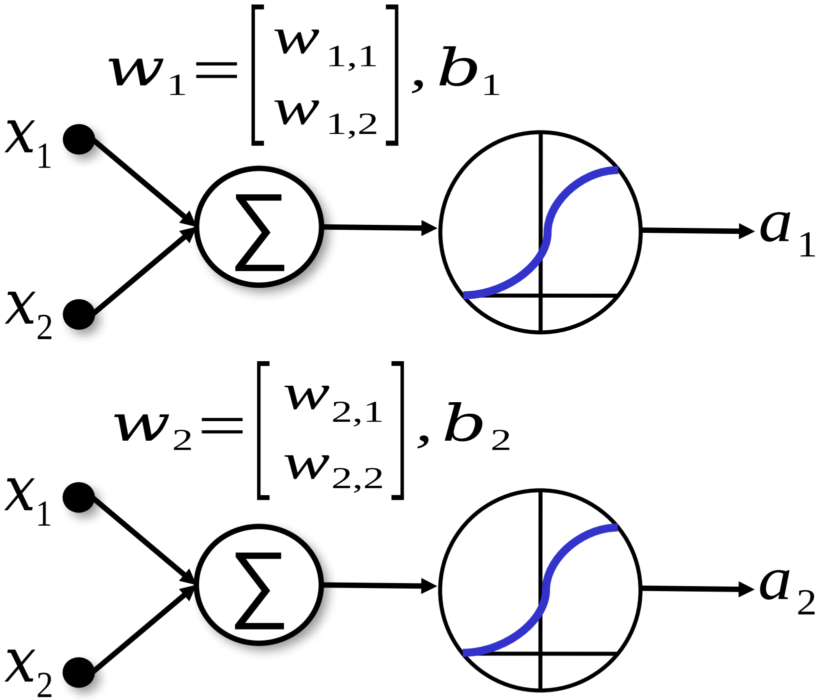
Gradient Descent:

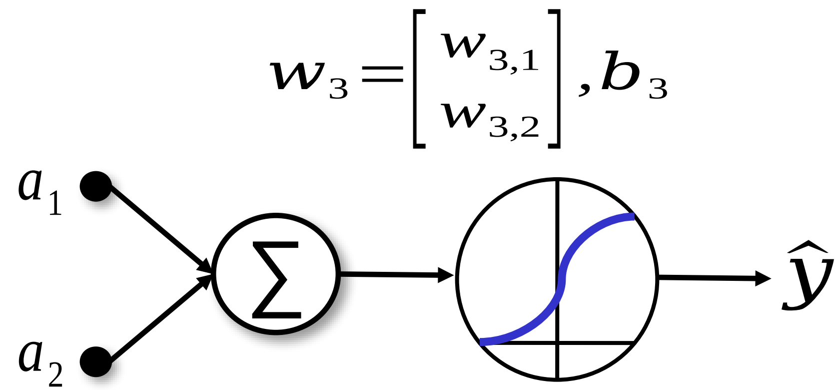
$$\frac{\delta E}{\delta w_i} = \frac{\delta}{\delta w_i} \frac{1}{2} \left(y - \hat{y}\right)^2$$

Chain Rule:

$$\frac{\delta z}{\delta x} = \frac{\delta z}{\delta y} * \frac{\delta y}{\delta x}$$

Backpropagation with Multi-Layer Perceptron





Chain Rule:

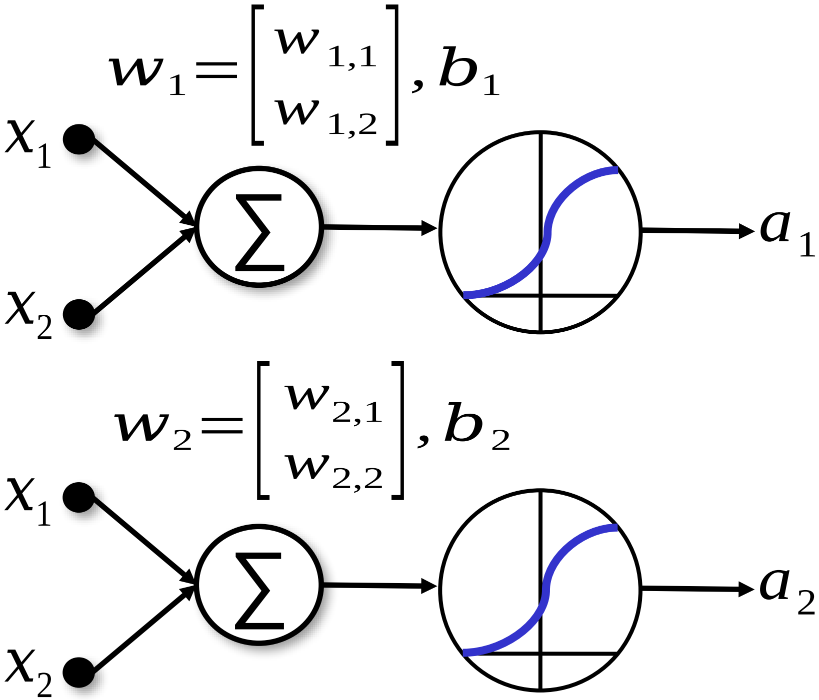
$$\frac{\delta z}{\delta x} = \frac{\delta z}{\delta y} * \frac{\delta y}{\delta x}$$

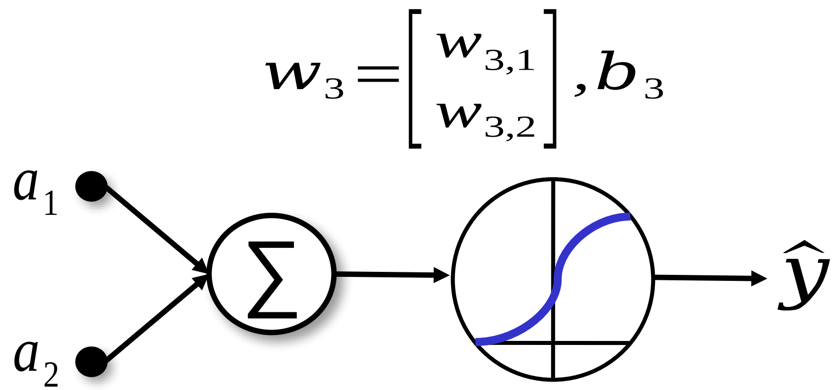
Gradient Descent:

$$\frac{\delta E}{\delta w_i} = \frac{\delta}{\delta w_i} \frac{1}{2} \left(y - \hat{y} \right)^2$$

$$\frac{\delta E}{\delta w_i} = \frac{\delta E}{\delta \hat{y}} \frac{\delta \hat{y}}{\delta w_i} = -\left(y - \hat{y} \right) \frac{\delta \hat{y}}{\delta w_i}$$

Backpropagation with Multi-Layer Perceptron





Chain Rule:

$$\frac{\delta z}{\delta x} = \frac{\delta z}{\delta y} * \frac{\delta y}{\delta x}$$

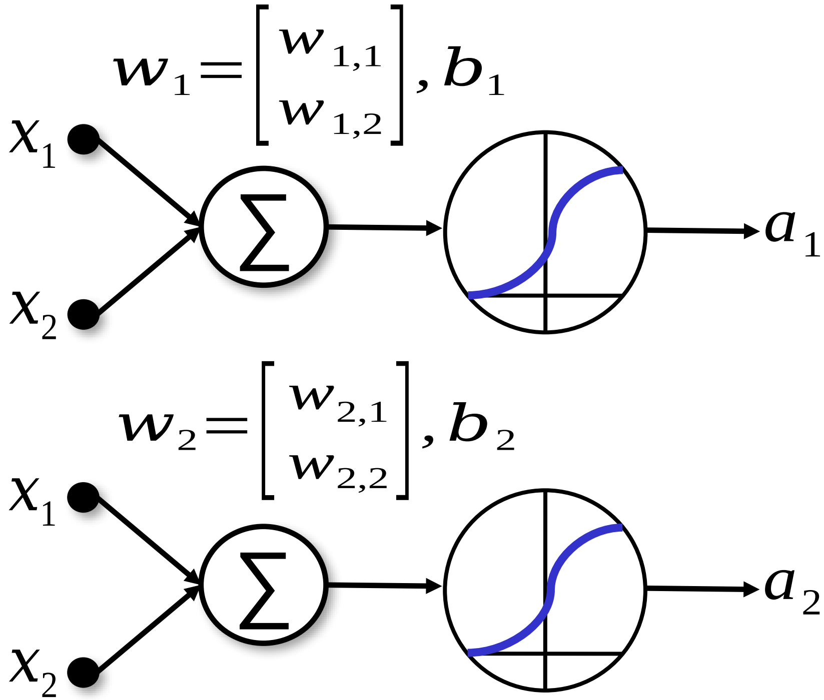
Gradient Descent:

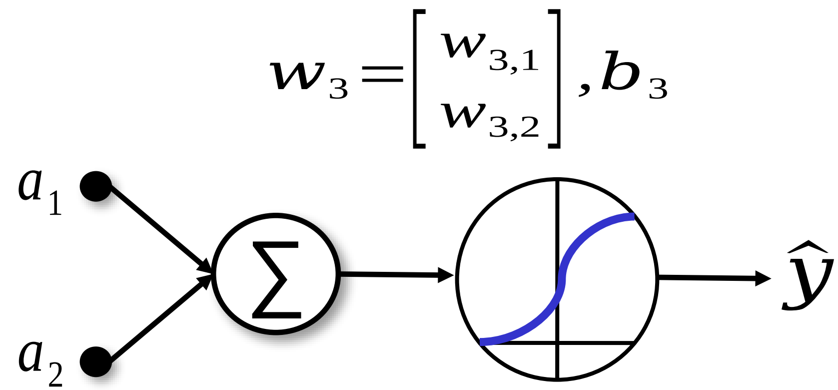
$$\frac{\delta E}{\delta w_i} = \frac{\delta}{\delta w_i} \frac{1}{2} (y - \hat{y})^2$$

$$\frac{\delta E}{\delta w_i} = \frac{\delta E}{\delta \hat{y}} \frac{\delta \hat{y}}{\delta w_i} = -(y - \hat{y}) \frac{\delta \hat{y}}{\delta w_i}$$

$$\frac{\delta \hat{y}}{\delta w_i} = \frac{\delta}{\delta w_i} sigmoid\left(w_3^T y + b_3\right)$$

Backpropagation with Multi-Layer Perceptron





Chain Rule:

$$\frac{\delta z}{\delta x} = \frac{\delta z}{\delta y} * \frac{\delta y}{\delta x}$$

$$\frac{\delta}{\delta x} sigmoid(x) = sigmoid(x) * (1 - sigmoid(x))$$

Gradient Descent:

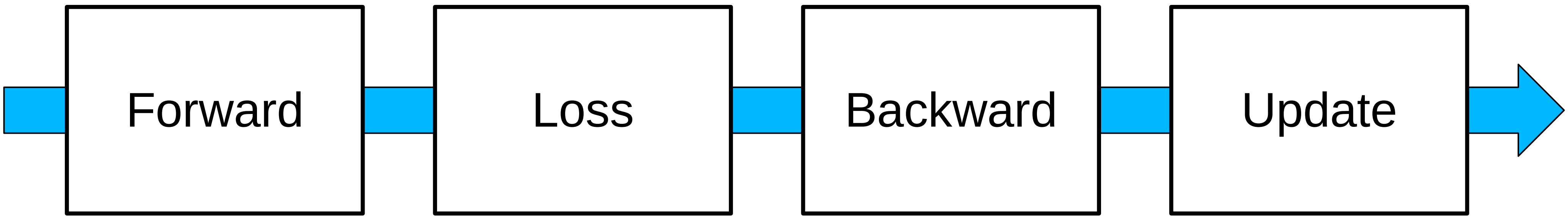
$$\frac{\delta E}{\delta w_i} = \frac{\delta E}{\delta \hat{y}} \frac{\delta \hat{y}}{\delta w_i} = -(y - \hat{y}) \frac{\delta \hat{y}}{\delta w_i}$$

$$\frac{\delta \hat{y}}{\delta w_i} = \frac{\delta}{\delta w_i} sigmoid(w_3^T y + b_3)$$

$$\frac{\delta \hat{y}}{\delta w_i} = sigmoid(w_3^T y + b_3) * (1 - sigmoid(w_3^T y + b_3))$$

$$* \frac{\delta}{\delta w_i}(w_3^T y + b_3)$$

Learning Process



Easy using modern Deep Learning Frameworks, e.g. Pytorch

```
y_pred = model(x_data)  # 1. forward
loss = criterion(y_pred, y_data)  # 2. loss
loss.backward()  # 3. backward
optimizer.step()  # 4. update
```

Summary

- We learned what a perceptron and multilayer perceptron is
- We have some intuition about using gradient descent on an error function
- We know a learning delta rule for updating weights in order to minimize the error:
- We know activation function for non-linearity
- We can use this rule to learn an MLP using the backpropagation algorithm