Artificial Intelligence

INTRODUCTION TO NEURAL NETWORKS

Nguyễn Ngọc Thảo – Nguyễn Hải Minh
{nnthao, nhminh}@fit.hcmus.edu.vn

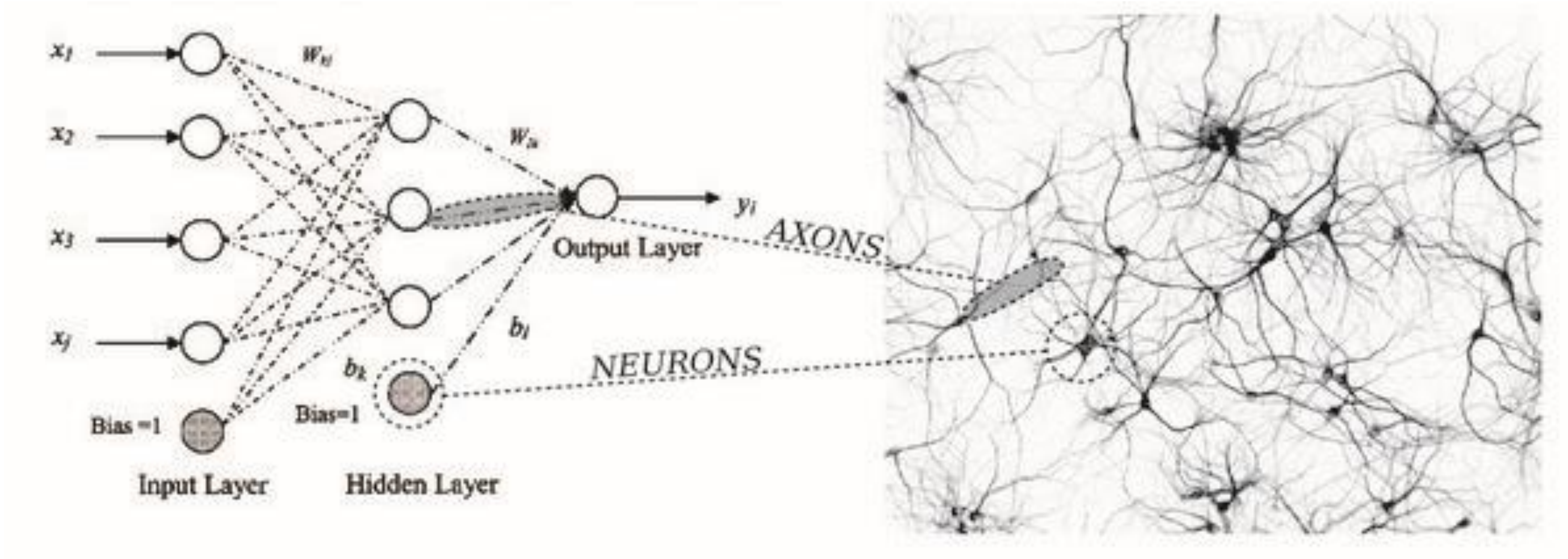
Outline

- Introduction to Artificial neural networks
- Perceptron and Learning
- Multi-layer neural networks

Artificial neural network

What is a neural network?

- A reasoning model based on the human brain, including billions of neurons and trillion connections between them



Biological neural network

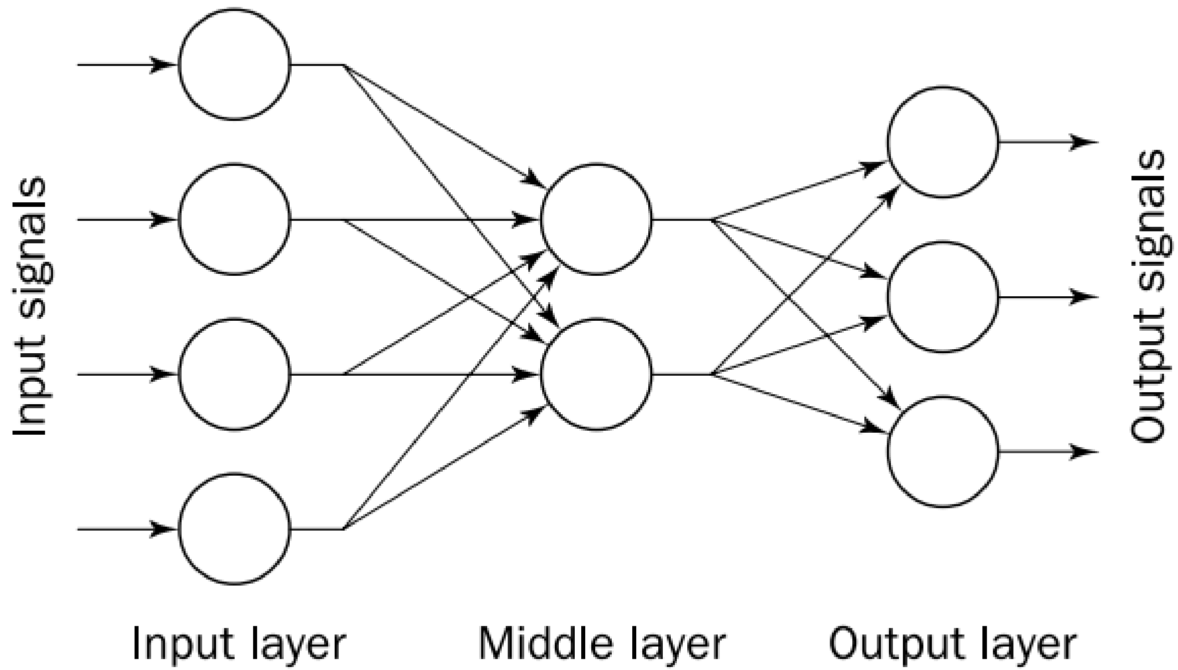
- A system that is highly complex, nonlinear and parallel information-processing
- Learning through experience is an essential characteristic.
- **Plasticity**: connections between neurons leading to the "right answer" are strengthened while those leading to the "wrong answer" are weakened.

Artificial neural networks (ANN)

- Resemble the human brain in terms of learning mechanisms
- Improve performance through experience and generalization

How does an ANN model the brain?

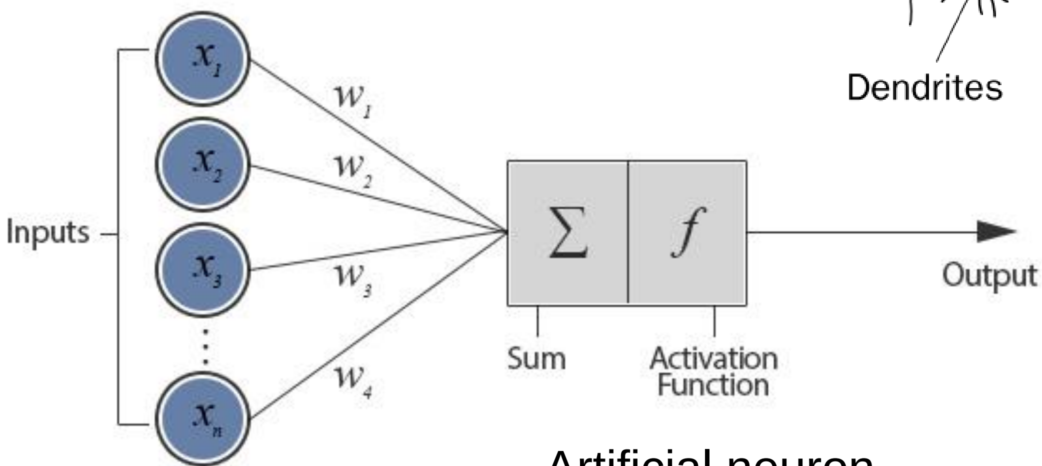
- An ANN includes many neurons, which are simple and highly interconnected processors arranging in a hierarchy of layers.



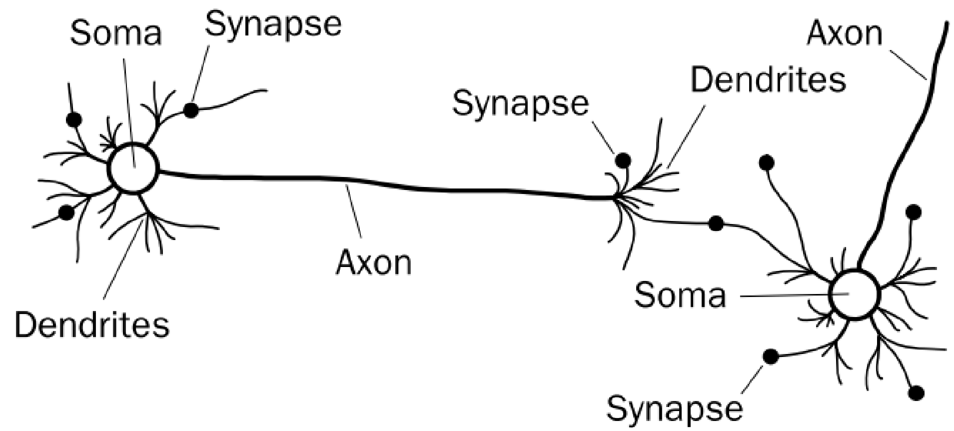
- Each neuron is an elementary information-processing unit.

How does an ANN model the brain?

- Each neuron receives several input signals through its connections and produces at most a single output signal.
- The neurons are connected by links, which pass signals from one neuron to another.
    - Each link associates with a numerical weight expressing the strength of the neuron input.
    - The set of weights is the basic mean of long-term memory in ANNs.
- ANNs "learn" through iterative adjustments of weights.



Artificial neuron



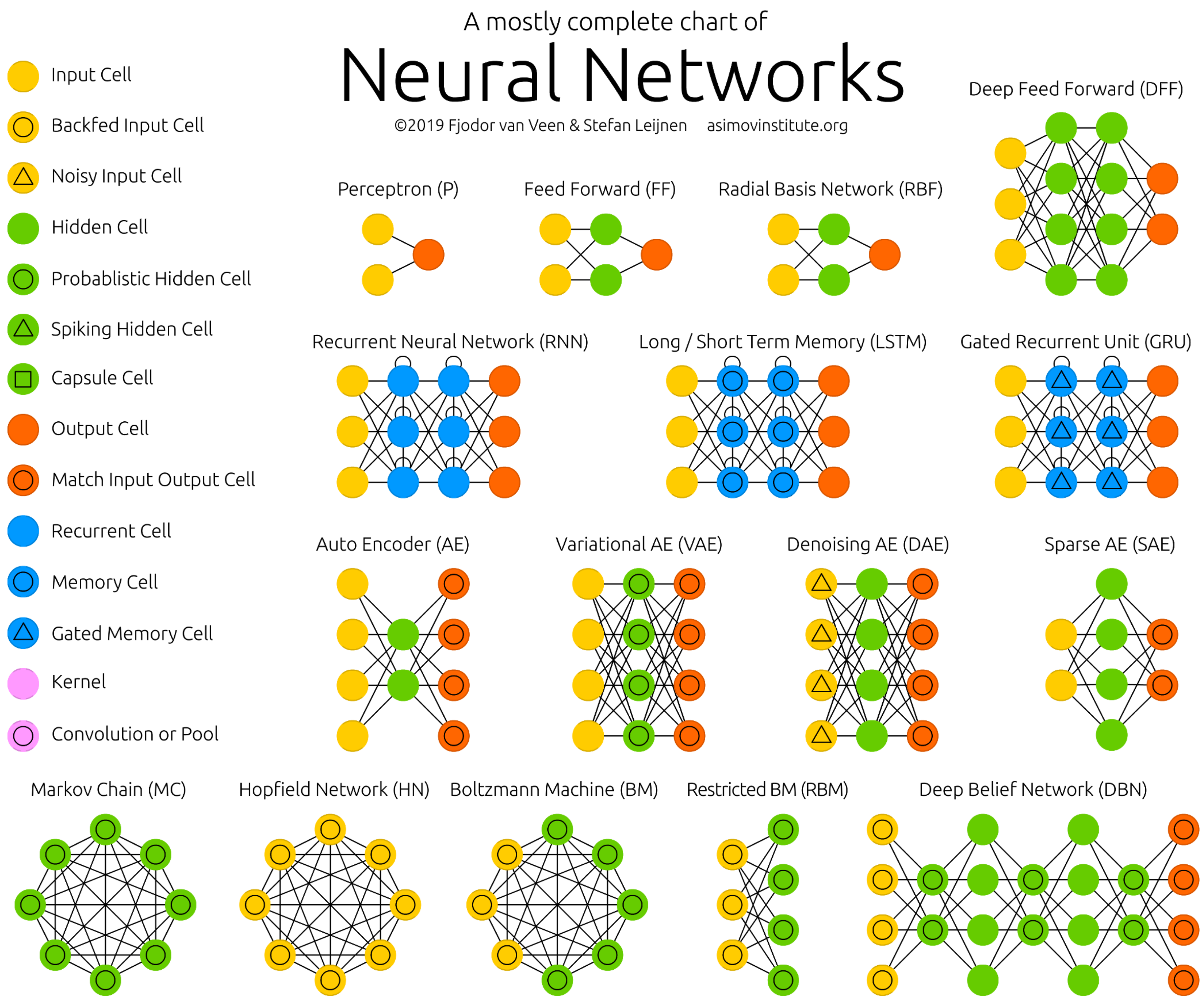
Biological neuron

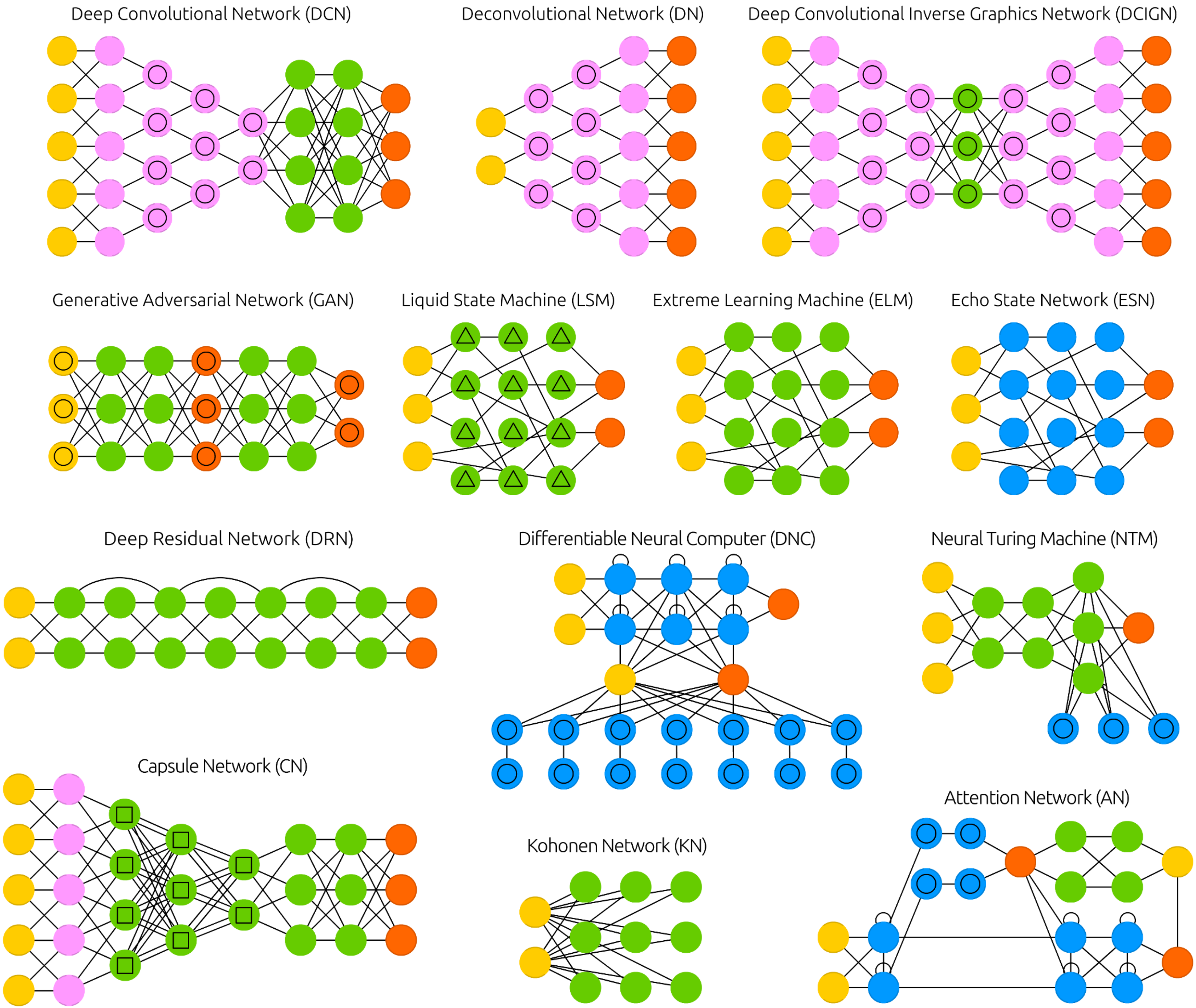
Analogy between biological and artificial neural networks

Biological neural network	Artificial neural network
Soma	Neuron
Dendrite	Input
Axon	Output
Synapse	Weight

How to build an ANN?

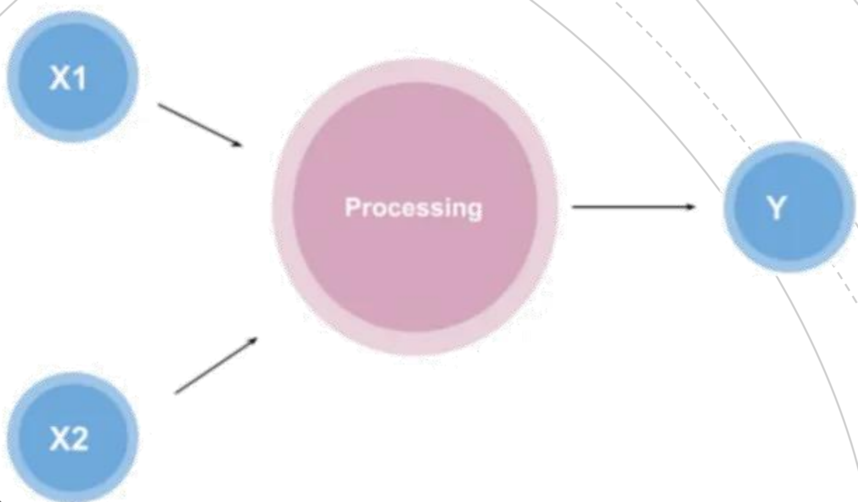
- The network architecture must be decided first,
  - How many neurons are to be used?
  - How the neurons are to be connected to form a network?
- Then determine which learning algorithm to use,
  - Supervised /semi-supervised / unsupervised / reinforcement learning
- And finally train the neural network
  - How to initialize the weights of the network?
  - How to update them from a set of training examples.


A mostly complete chart of
Neural Networks
©2019 Fjodor van Veen & Stefan Leijnen
asimovinstitute.org
Input Cell
Backfed Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Capsule Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Gated Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)



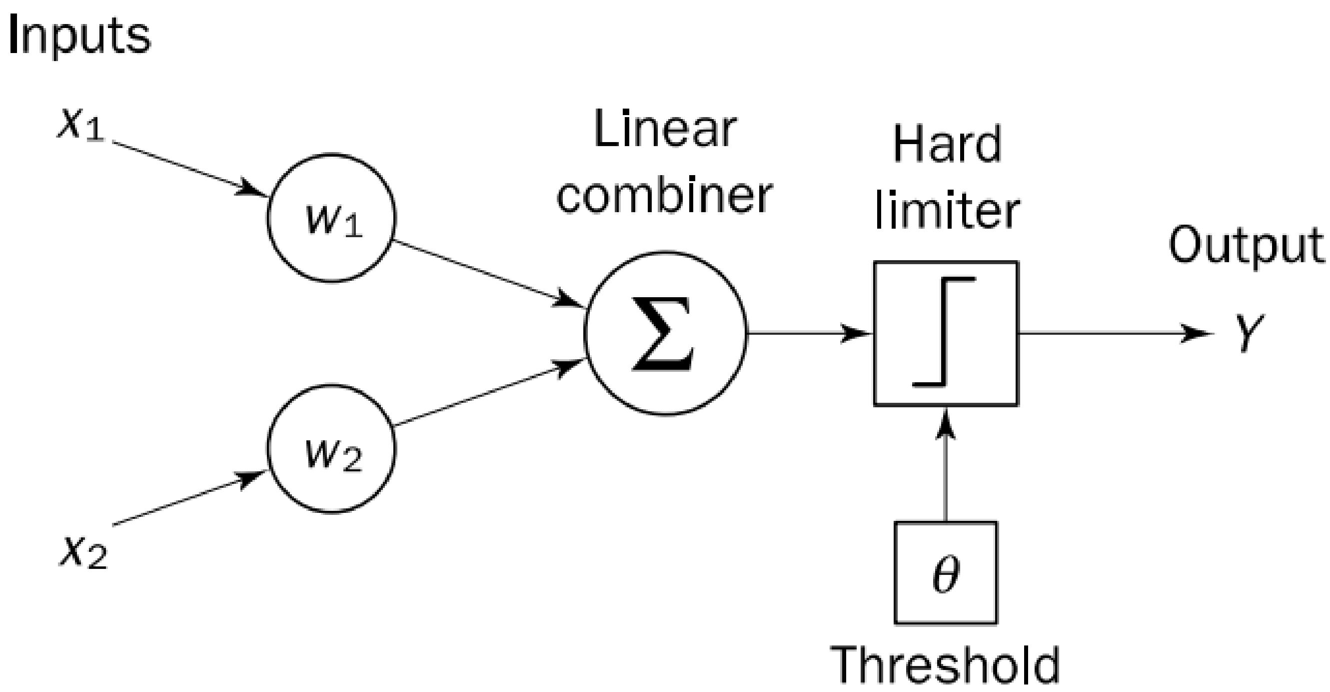
Source: http://www.asimovinstitute.org/neural-network-zoo/

Perceptron and Learning



Perceptron (Frank Rosenblatt, 1958)

- A perceptron has a single neuron with adjustable synaptic weights and a hard limiter.

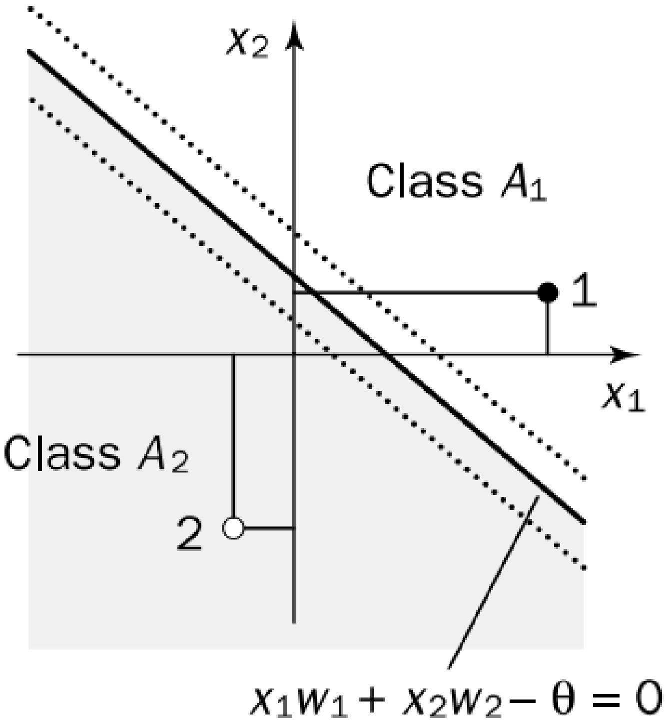


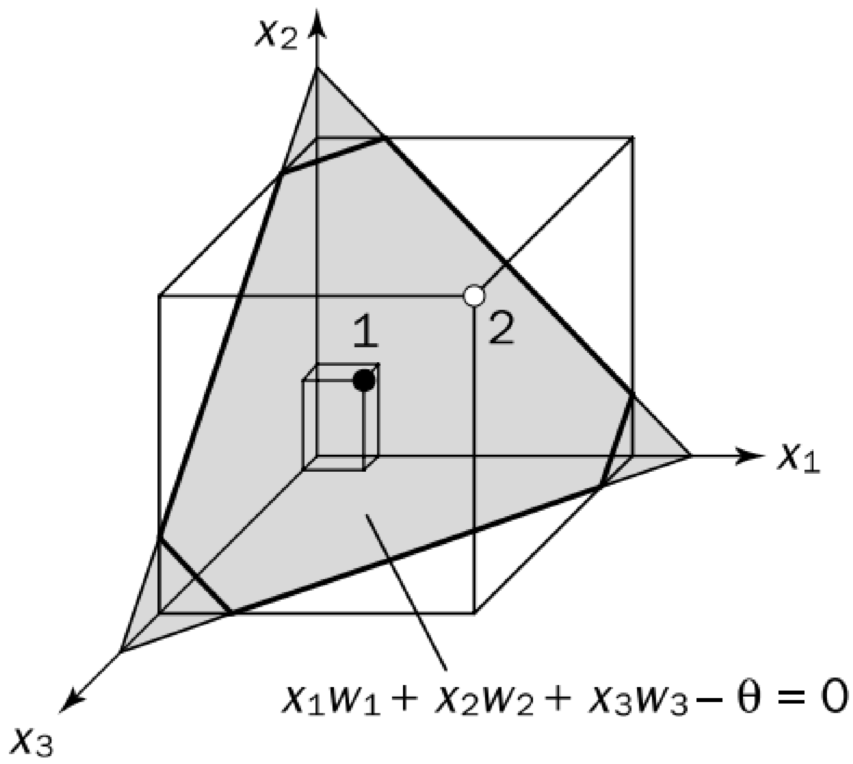
A single-layer two-input perceptron

How does a perceptron work?

- Divide the n-dimensional space into two decision regions by a hyperplane defined by the linearly separable function

$$\sum_{i=1}^{n} x_i w_i - \theta$$





Perceptron learning rule

- Step 1 – Initialization: Initial weights $\boldsymbol{w_1}, \boldsymbol{w_2}, \dots, \boldsymbol{w_n}$ and threshold $\boldsymbol{\theta}$ are randomly assigned to small numbers (usually in $[-0.5, 0.5]$, but not restricted to).
- Step 2 – Activation: At iteration $\boldsymbol{p}$, apply the $\boldsymbol{p}^{th}$ example, which has inputs $\boldsymbol{x_1(p)}, \boldsymbol{x_2(p)}, \dots, \boldsymbol{x_n(p)}$ and desired output $\boldsymbol{Y_d(p)}$, and calculate the actual output

$$\boldsymbol{Y(p) = \sigma\left(\sum_{i=1}^{n} x_i(p) w_i(p) - \theta\right)}$$

$$\sigma(x) = \begin{cases} 1 & if\ x \geq 0 \\ 0 & if\ x < 0 \end{cases}$$

  where n is the number of perceptron inputs and $step$ is the activation function

- Step 3 – Weight training
  - Update the weights $\boldsymbol{w_i}$: $\boldsymbol{w_i(p+1) = w_i(p) + \Delta w_i(p)}$

    where $\Delta \boldsymbol{w_i(p)}$ is the weight correction at iteration $\boldsymbol{p}$
  - The **delta rule** determines how to adjust the weights: $\boldsymbol{\Delta w_i(p) = \alpha \times x_i(p) \times e(p)}$

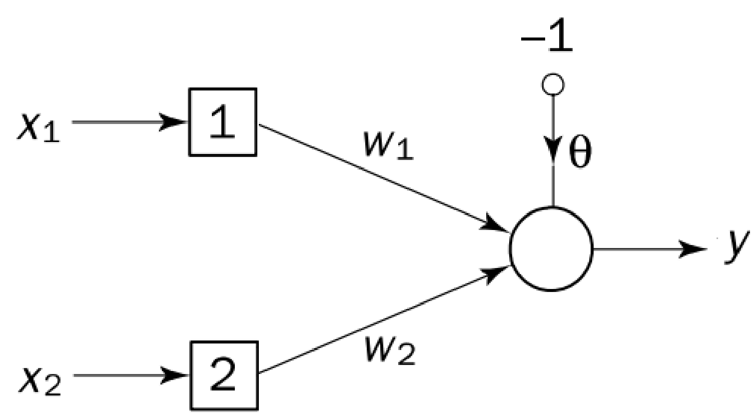
    where $\boldsymbol{\alpha}$ is the learning rate $(0 < \alpha < 1)$ and $\boldsymbol{e(p) = Y_d(p) - Y(p)}$
- Step 4 – Iteration: Increase iteration $\boldsymbol{p}$ by one, go back to Step 2 and repeat the process until convergence.

Perceptron for the logical AND/OR

- A single-layer perceptron **can** learn the **AND/OR** operations.

Epoch	Inputs x_1	x_2	Desired output Y_d	Initial weights w_1	w_2	Actual output Y	Error e	Final weights w_1	w_2
1	0	0	0	0.3	−0.1	0	0	0.3	−0.1
	0	1	0	0.3	−0.1	0	0	0.3	−0.1
	1	0	0	0.3	−0.1	1	−1	0.2	−0.1
	1	1	1	0.2	−0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	−1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0
3	0	0	0	0.2	0.0	0	0	0.2	0.0
	0	1	0	0.2	0.0	0	0	0.2	0.0
	1	0	0	0.2	0.0	1	−1	0.1	0.0
	1	1	1	0.1	0.0	0	1	0.2	0.1
4	0	0	0	0.2	0.1	0	0	0.2	0.1
	0	1	0	0.2	0.1	0	0	0.2	0.1
	1	0	0	0.2	0.1	1	−1	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1
5	0	0	0	0.1	0.1	0	0	0.1	0.1
	0	1	0	0.1	0.1	0	0	0.1	0.1
	1	0	0	0.1	0.1	0	0	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1

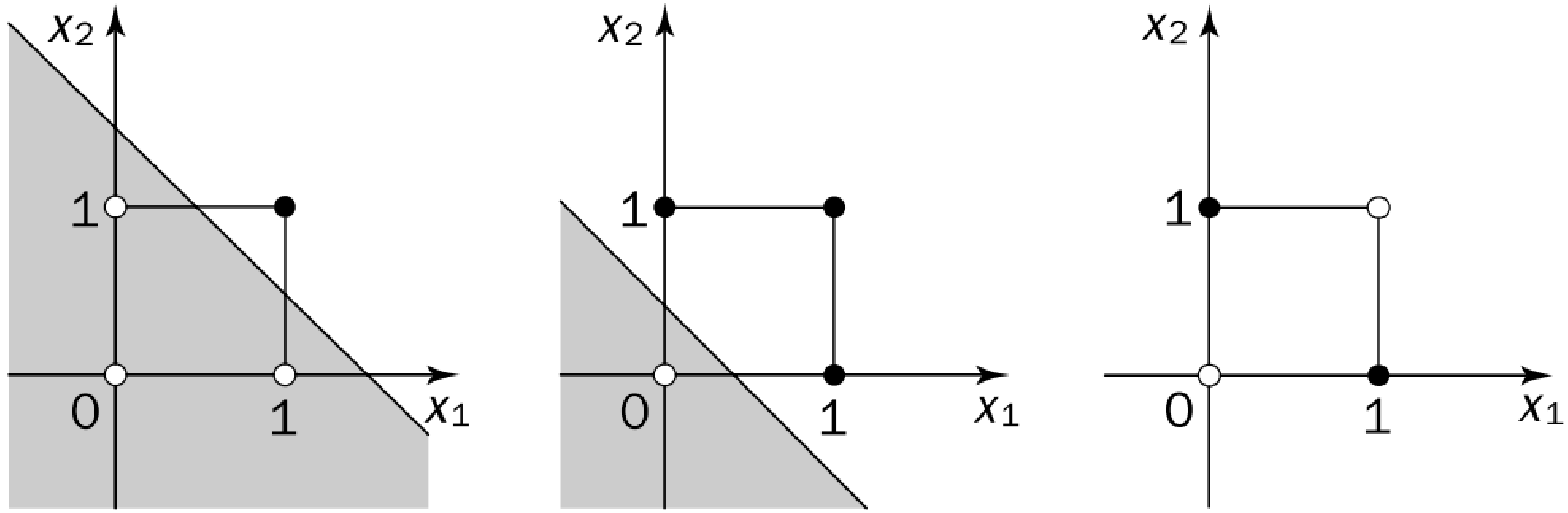
Threshold $\theta = 0.2$, learning rate $\alpha = 0.1$



The learning of logical AND converged after several iterations

Perceptron for the logical XOR

- It **cannot** be trained to perform the **Exclusive-OR**.



(a) AND ($x_1 \cap x_2$) (b) OR ($x_1 \cup x_2$) (c) Exclusive-OR ($x_1 \oplus x_2$)

Will a sigmoidal element do better?

- Perceptron can classify only linearly separable patterns regardless of the activation function used (Shynk, 1990; Shynk and Bershad, 1992)
- **Solution:** advanced forms of neural networks (e.g., multi-layer perceptrons trained with back-propagation algorithm)

An example of perceptron

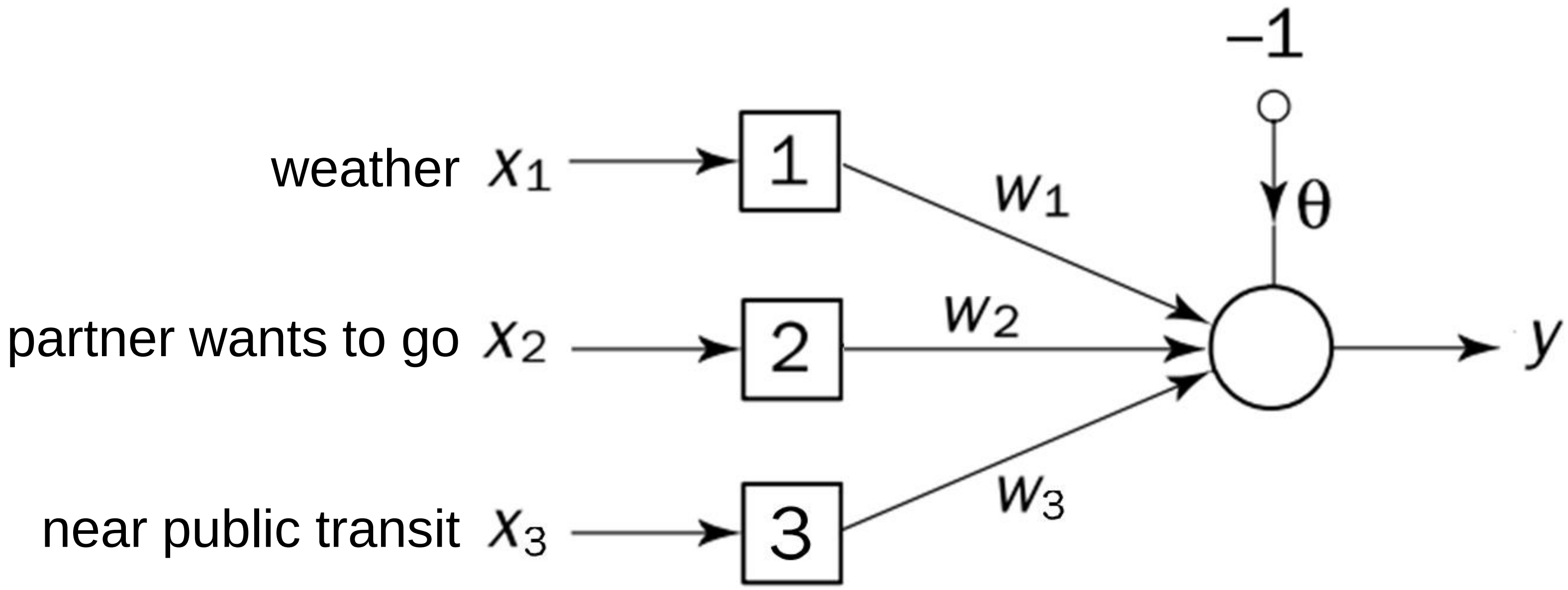
Is the weather good?

Does your partner want to accompany you?

Is the festival near public transit? (You don't own a car)

go to the festival?

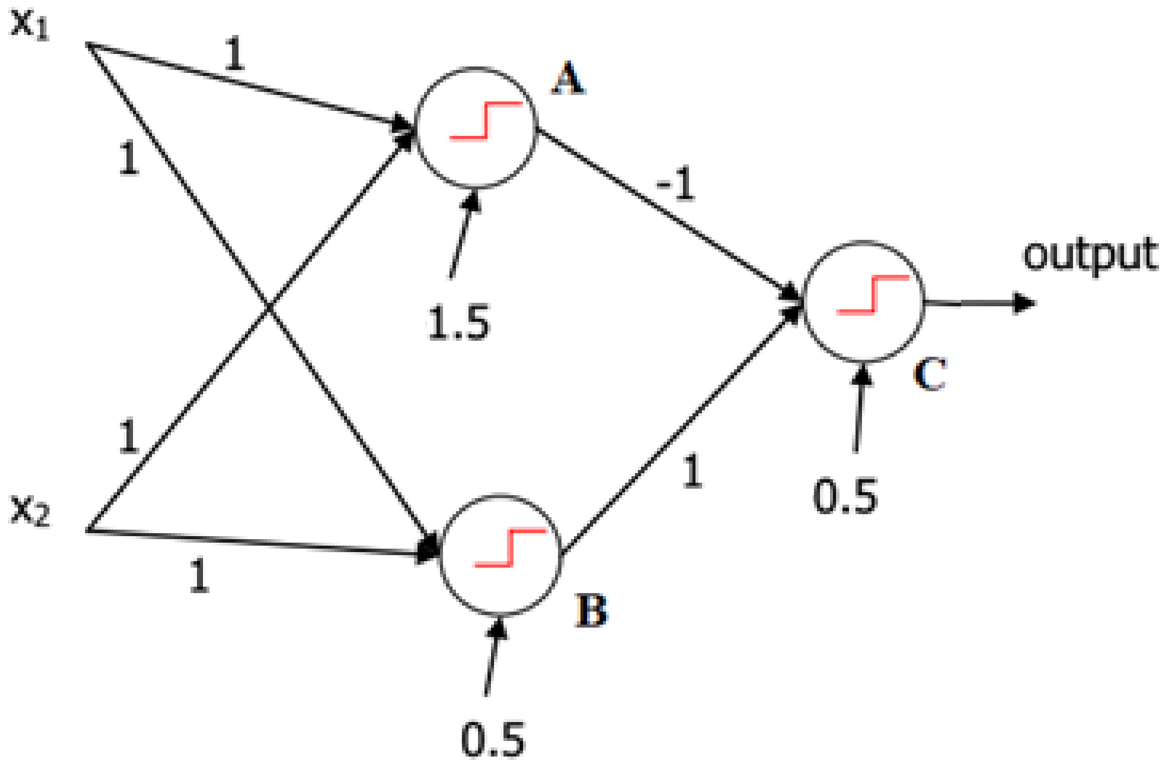
An example of perceptron



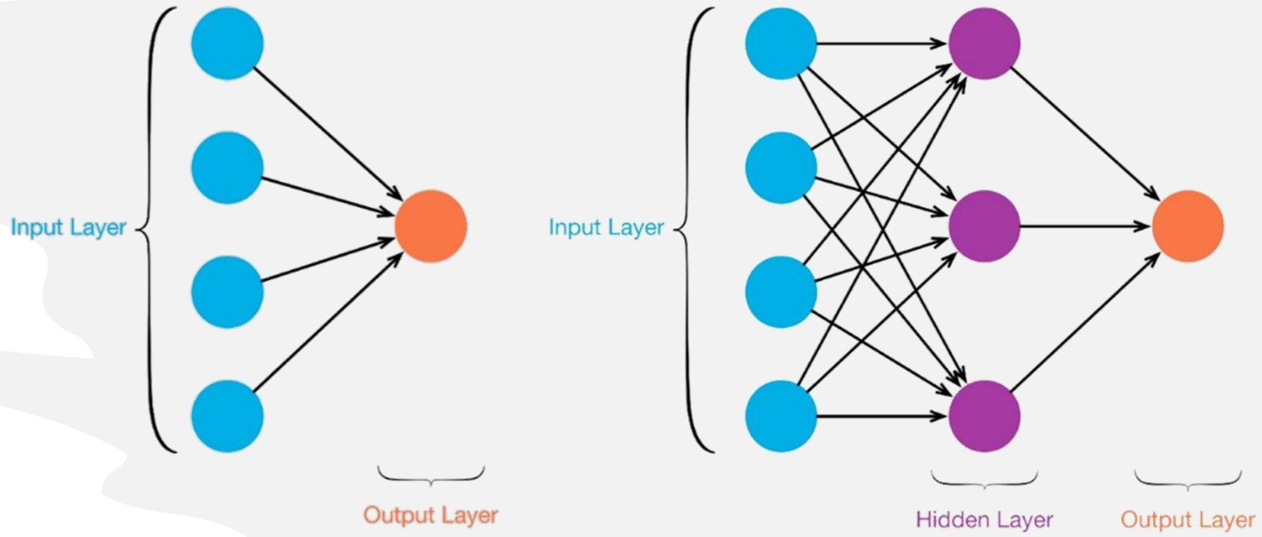
- $w_1 = 6, w_2 = 2, w_3 = 2 \rightarrow$ the weather matters to you much more than whether your partner joins you, or the nearness of public transit
- $\theta = 5 \rightarrow$ decisions are made based on the weather only
- $\theta = 3 \rightarrow$ you go to the festival whenever the weather is good or when both the festival is near public transit and your partner wants to join you.

Quiz 01: Perceptron

- Consider the following neural network which receives binary input values, x_1 and $x_{2,}$ and produces a single binary value.



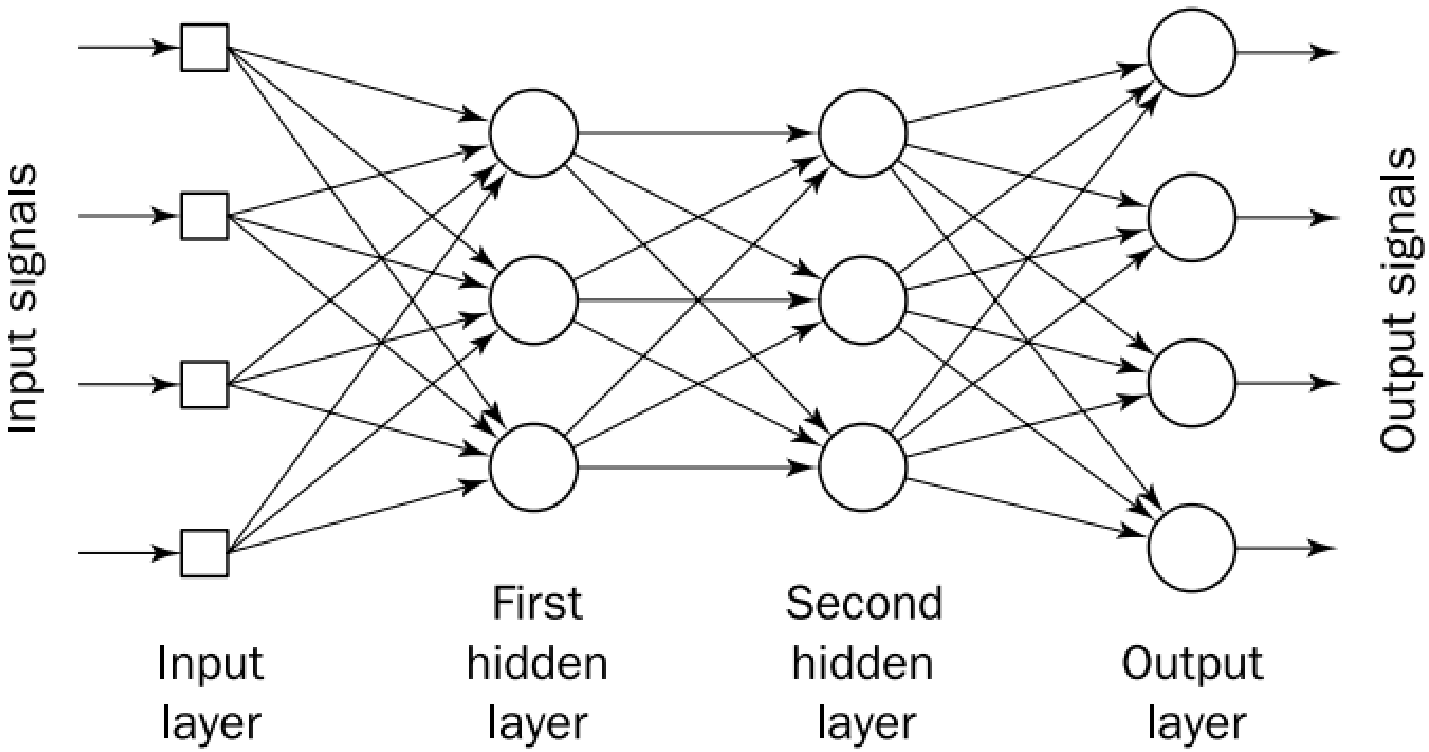
- For every combination (x_1, x_2), what are the output values at neurons, A, B and C?



Multi-layer neural networks

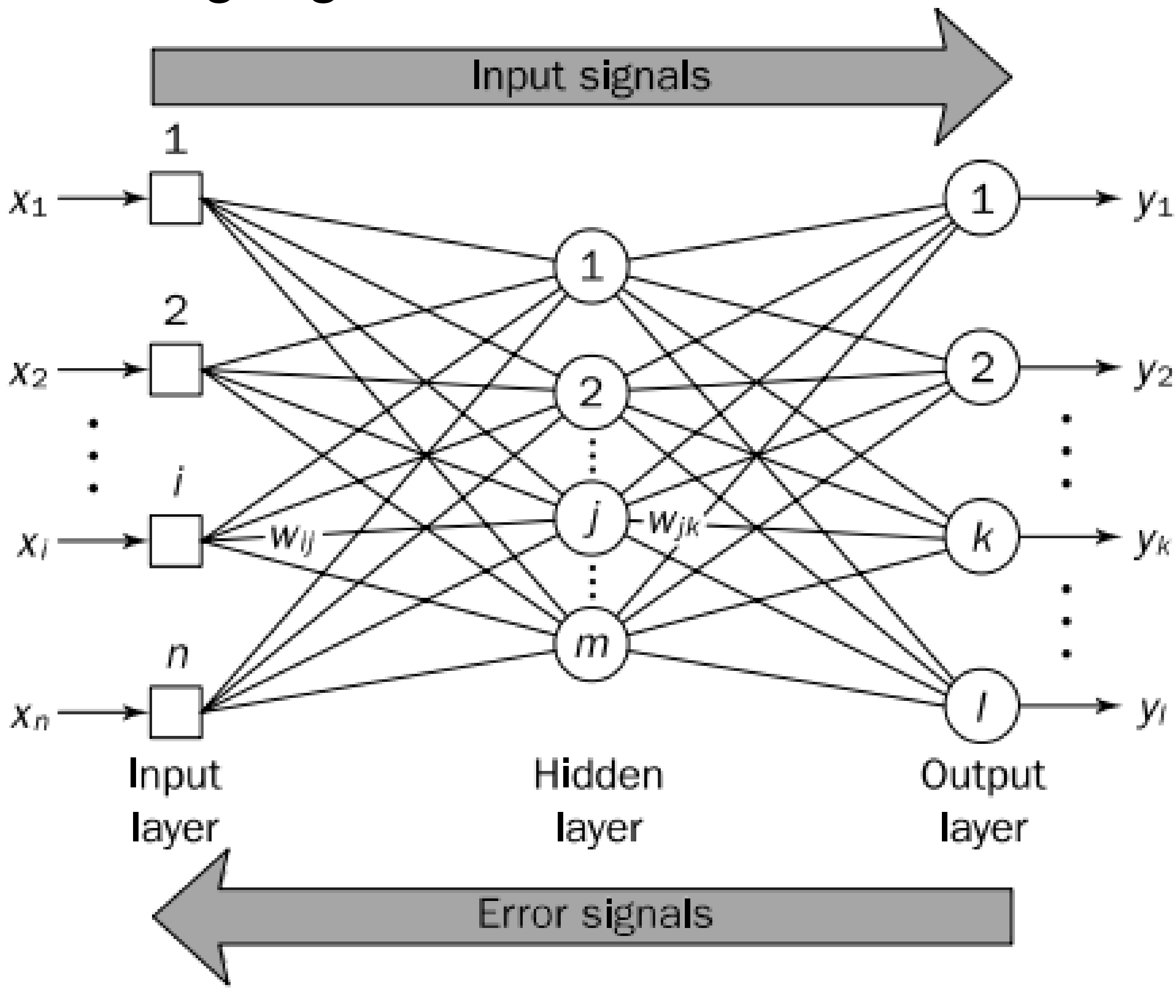
Multi-layer neural network

- A feedforward network with one or more hidden layers.
- The input signals are propagated forwardly on a layer-by-layer basis.



Back-propagation algorithm

- (Bryson and Ho, 1969), most popular among over a hundred different learning algorithms available



Back-propagation learning rule

- Step 1 – Initialization: Initial weights and thresholds are assigned to random numbers.
  - The numbers may be uniformly distributed in the range $\left(-\frac{2.4}{F_i}, +\frac{2.4}{F_i}\right)$ (Haykin, 1999), where F_i is the total number of inputs of neuron
  - The weight initialization is done on a neuron-by-neuron basis
- Step 2 – Activation: At iteration p, apply the p^{th} example, which has inputs $x_1(p), x_2(p), \dots, x_n(p)$ and desired outputs $y_{d,1}(p), y_{d,2}(p), \dots, y_{d,l}(p)$.
  - (a) Calculate the actual output, from n inputs, of neuron j in the hidden layer.

$$y_j(p) = \sigma\left(\sum_{i=1}^{n} x_i(p) w_{ij}(p) - \theta_j\right)$$

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

  - (b) Calculate the actual output, from k inputs, of neuron m in the hidden layer.

$$y_k(p) = \sigma\left(\sum_{j=1}^{m} y_j(p) w_{jk}(p) - \theta_k\right)$$

Back-propagation learning rule

- Step 3 – Weight training: Update the weights in the back-propagation network and propagate backward the errors associated with output neurons.
  - (a) Calculate the error gradient for neuron $\boldsymbol{k}$ in the output layer

$$\delta_k(p) = y_k(p) \times [1 - y_k(p)] \times \underbrace{\boxed{[y_{d,k}(p) - y_k(p)]}}_{e_k(p)}$$

    Calculate the weight corrections: $\Delta w_{jk}(p) = \alpha \times y_j(p) \times \delta_k(p)$

    Update the weights at the output neurons: $w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p)$

  - (b) Calculate the error gradient for neuron $\boldsymbol{j}$ in the hidden layer

$$\delta_j(p) = y_j(p) \times [1 - y_j(p)] \times \sum_{k=1}^{l} \delta_k(p)\, w_{jk}(p)$$

    Calculate the weight corrections: $\Delta w_{ij}(p) = \alpha \times x_i(p) \times \delta_j(p)$

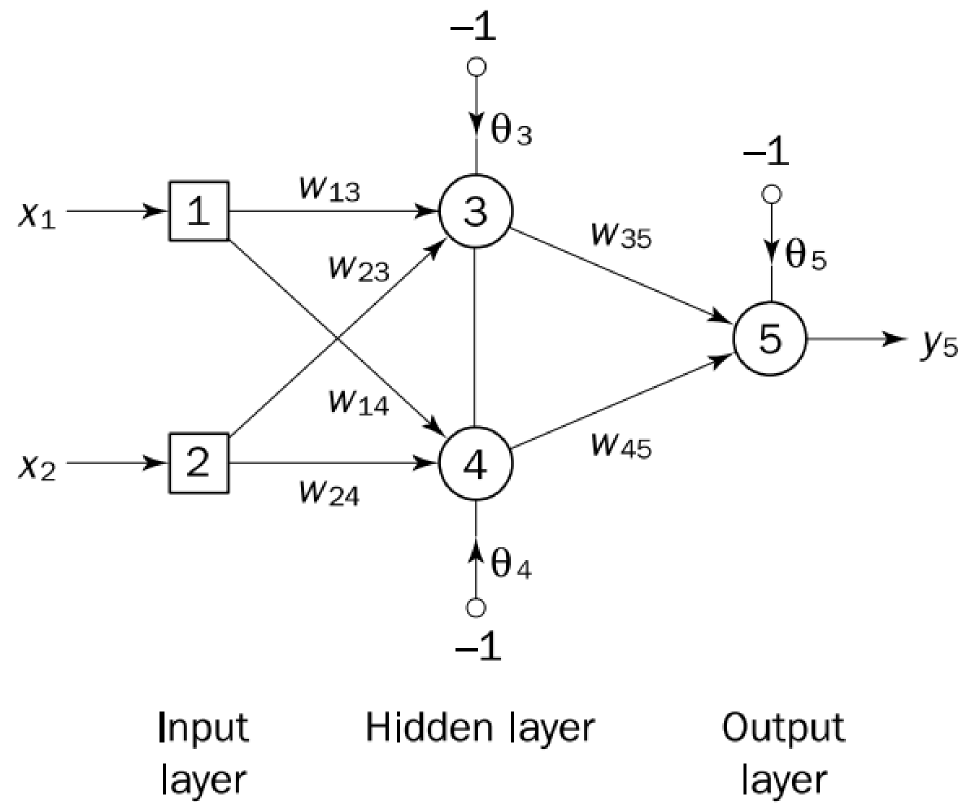
    Update the weights at the hidden neurons: $w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$

- Step 4: Iteration: Increase iteration $\boldsymbol{p}$ by one, go back to Step 2 and repeat the process until the selected error criterion is satisfied.

- A mathematical explanation can be found **here**.

Back-propagation network for XOR

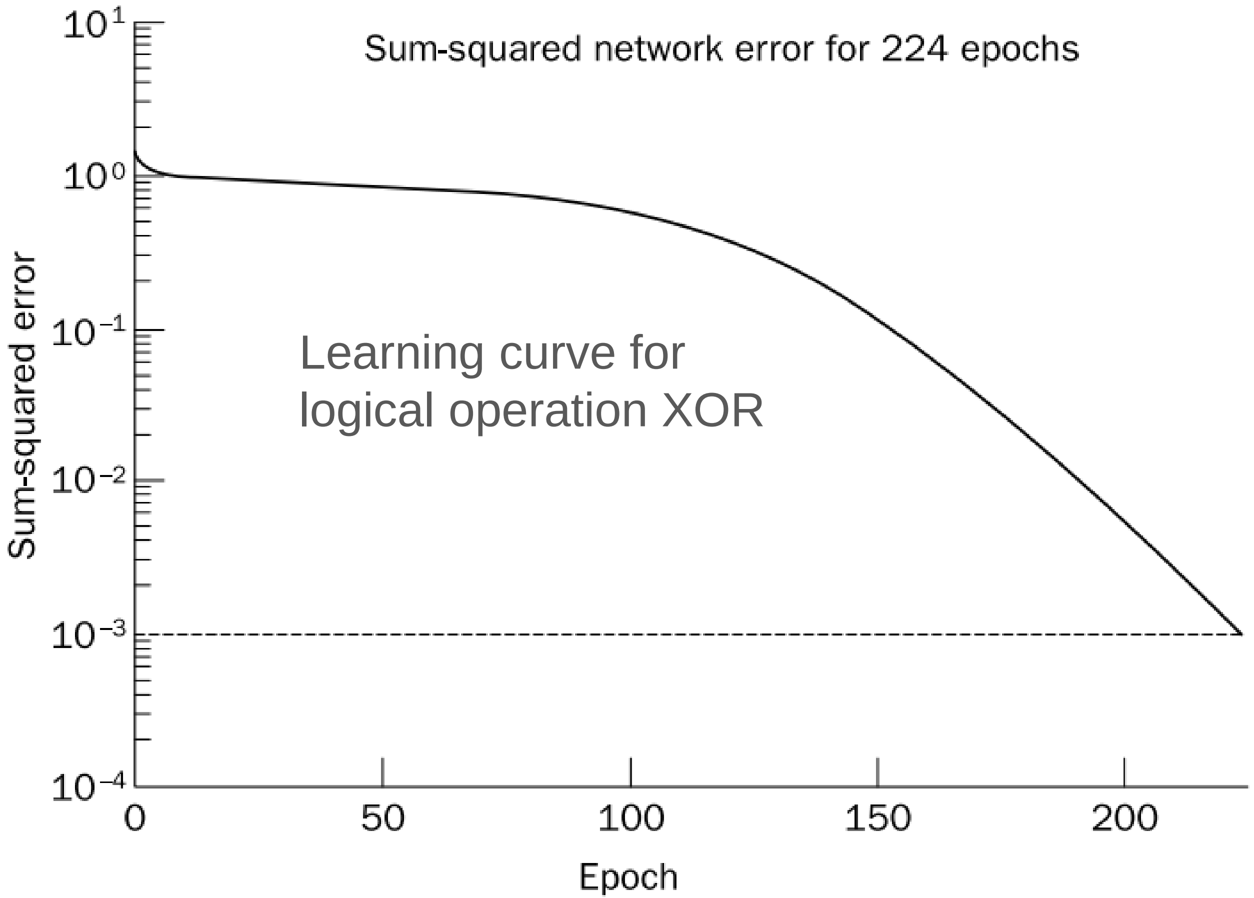
- The logical XOR problem took 224 epochs or 896 iterations for network training.



Inputs		Desired output	Actual output	Error	Sum of squared errors
x_1	x_2	y_d	y_5	e	
1	1	0	0.0155	–0.0155	0.0010
0	1	1	0.9849	0.0151	
1	0	1	0.9849	0.0151	
0	0	0	0.0175	–0.0175	

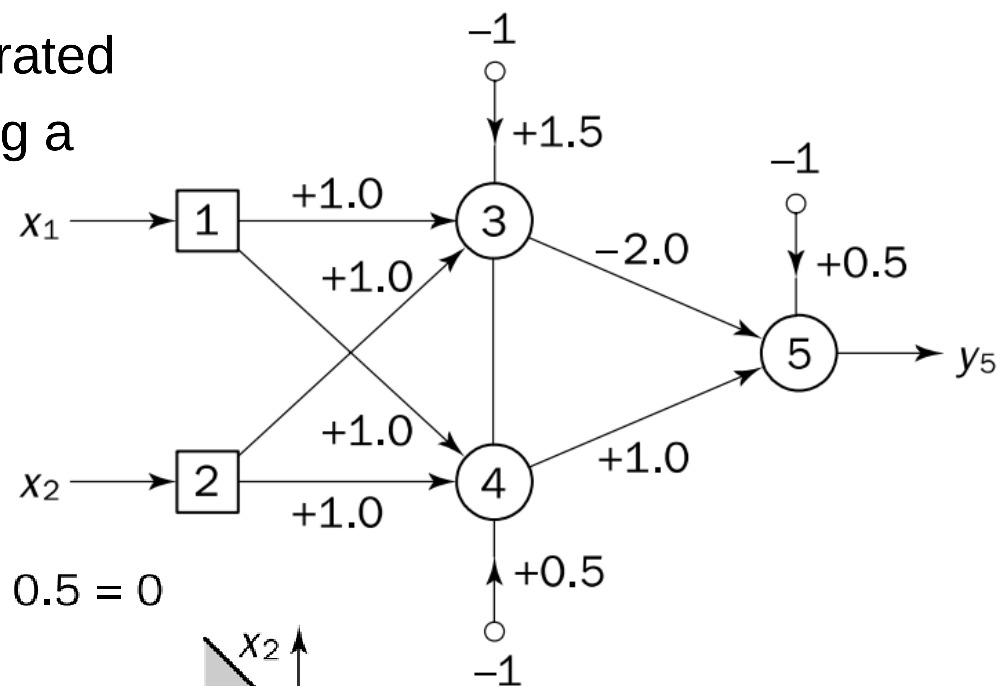
Sum of the squared errors (SSE)

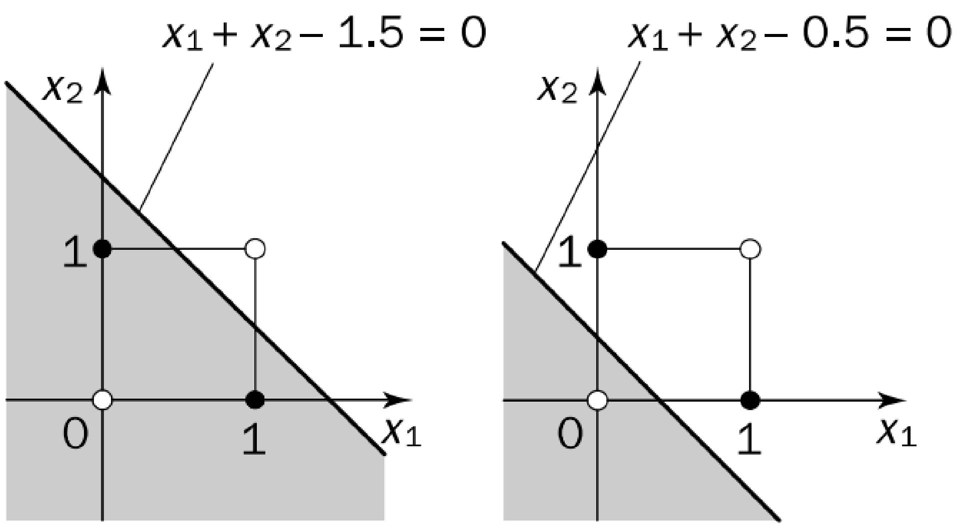
- When the SSE in an entire pass through all training sets is sufficiently small, a network is deemed to have converged.



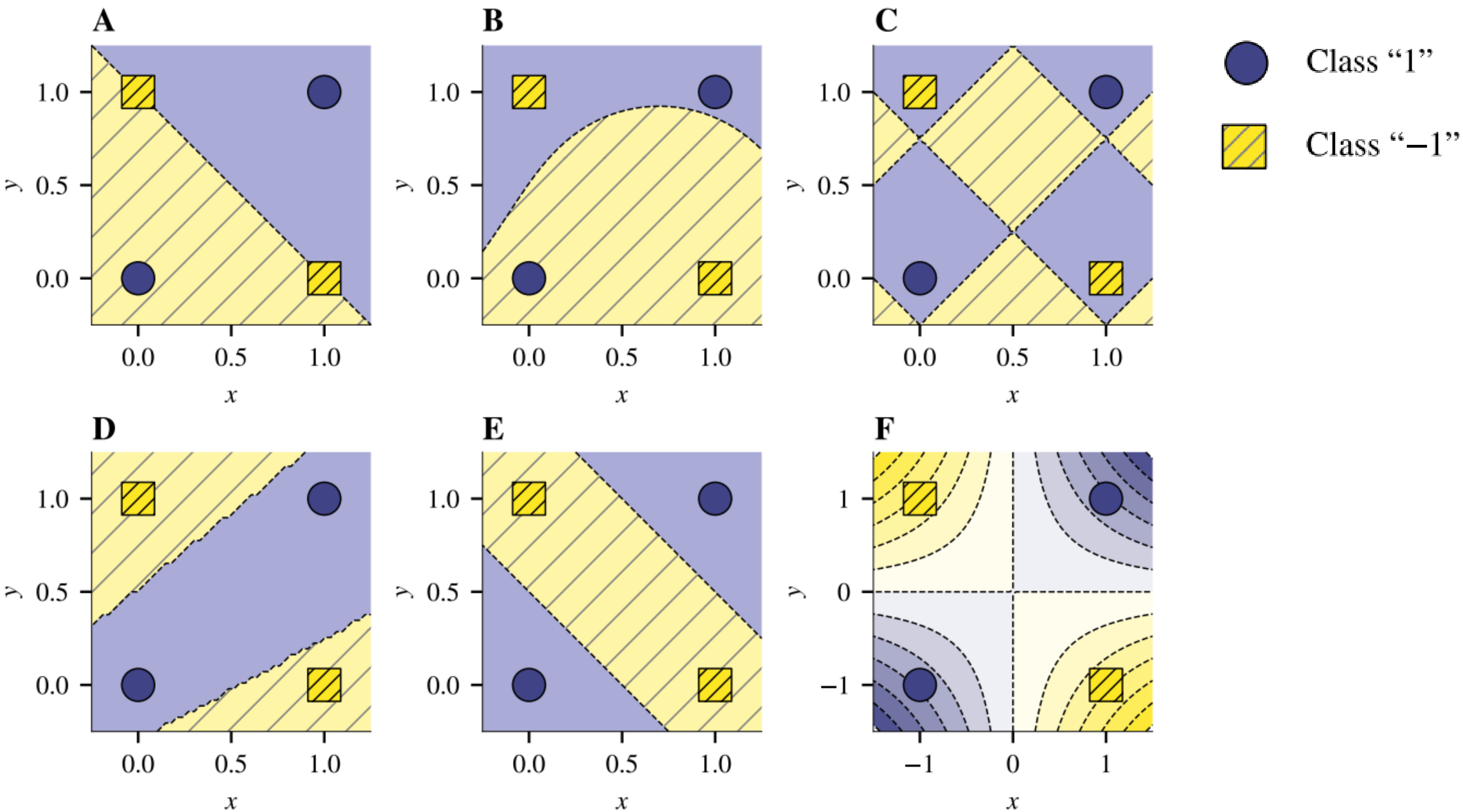
Decision boundaries for XOR

Decision boundaries are demonstrated with McCulloch-Pitts neurons using a sign function.





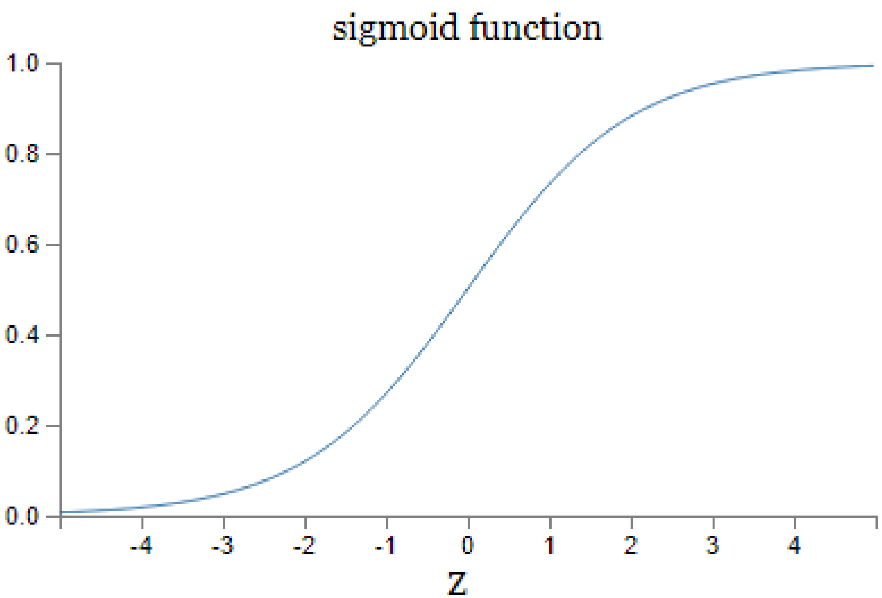
(a) (b) (c)

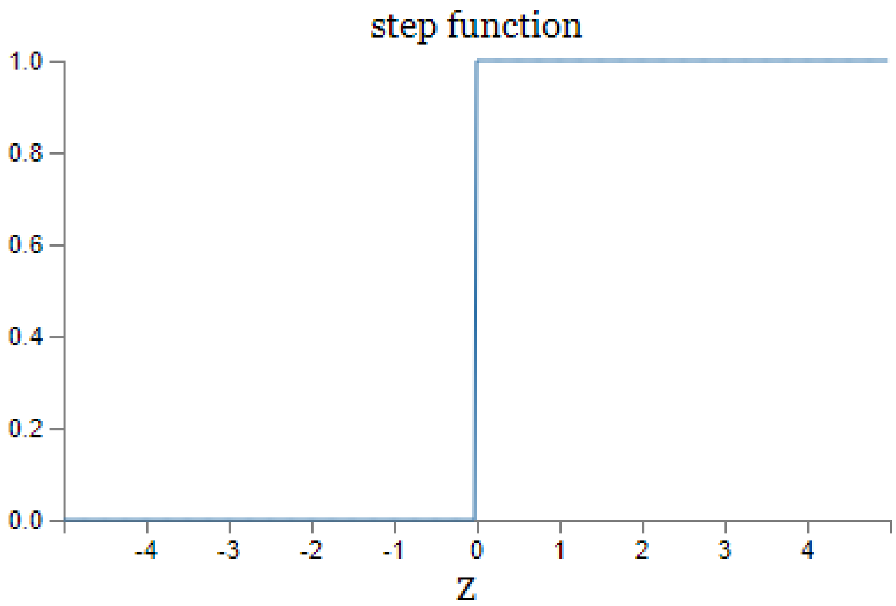


Visualization of the XOR decision problem for different types of classifiers. Markers correspond to the four data points to be classified. The colored/hatched background corresponds to the output of one exemplary decision function. **(A)** The linear decision boundary of a single-layer Perceptron cannot solve the problem. **(B, C)** This still holds for the generalization $\sigma(f(x) + g(y))$. **(D)** A multi-layer Perceptron (MLP) of the form $\sigma(\sum_i w_i \sigma(u_i x + v_i y + b_i))$ can be optimized using gradient descent to solve the problem correctly. **(E)** An alternative solution using a non-monotonic nonlinearity $\sigma'(\xi) = \sigma'(\xi^2 - 1)$. **(F)** Multiplication of two real-valued variables x, y can be seen as a superset of the XOR problem.

Sigmoid neuron vs. Perceptron

- Sigmoid neuron better reflects the fact that small changes in weights and bias cause only a small change in output.





A sigmoidal function is a smoothed-out version of a step function.

About back-propagation learning

- *Are randomly initialized weights and thresholds leading to different solutions?*
  - Starting with different initial conditions will obtain different weights and threshold values. The problem will always be solved within different numbers of iterations.
- Back-propagation learning cannot be viewed as emulation of brain-like learning.
  - Biological neurons do not work backward to adjust the strengths of their interconnections, synapses.
- The training is slow due to extensive calculations.
  - Improvements: Caudill, 1991; Jacobs, 1988; Stubbs, 1990

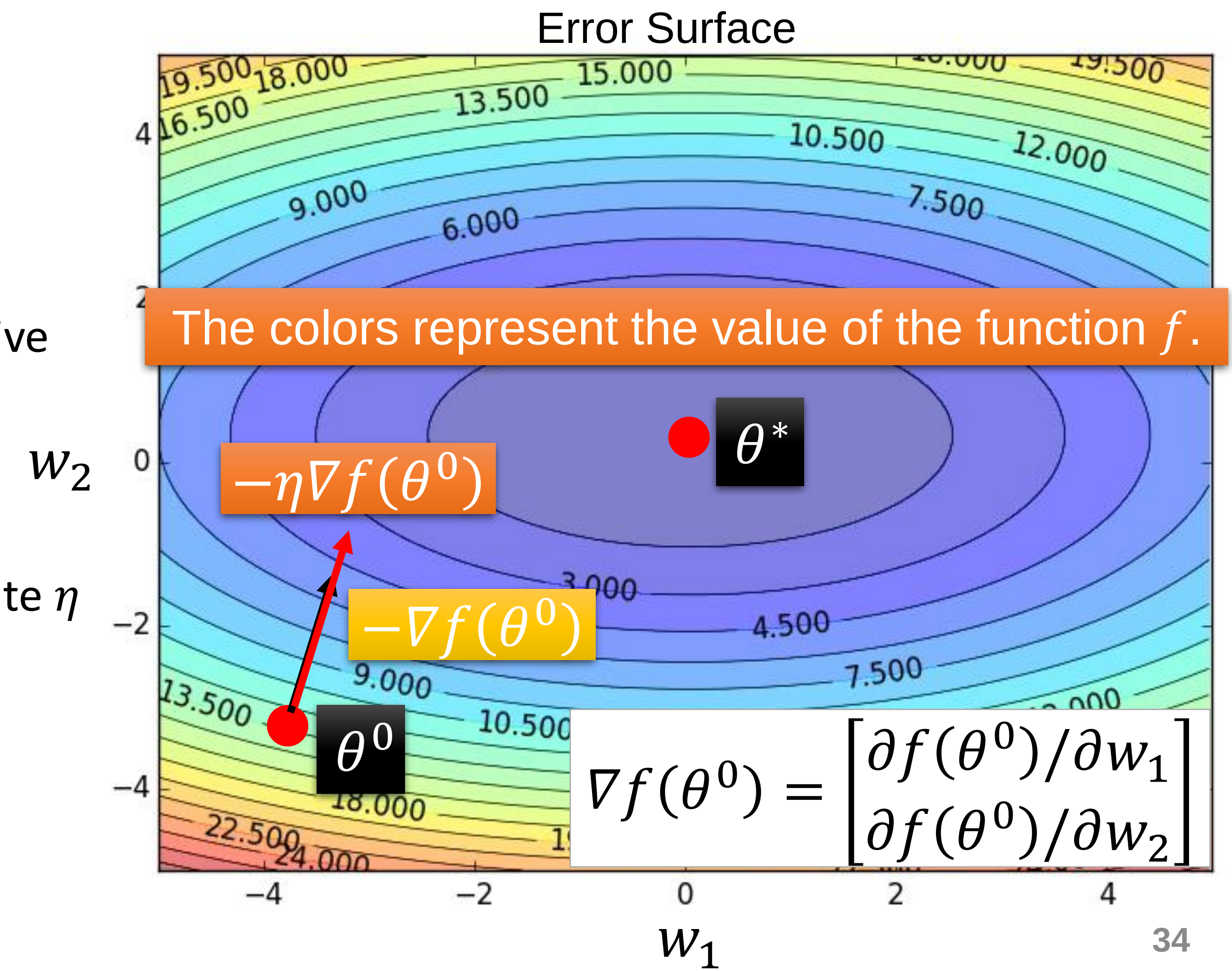
Gradient descent method

- Consider two parameters, w_1 and w_2, in a network.

Randomly pick a starting point θ^0

Compute the negative gradient at θ^0
$\rightarrow -\nabla f(\theta^0)$

Time the learning rate η
$\rightarrow -\eta \nabla f(\theta^0)$



Gradient descent method

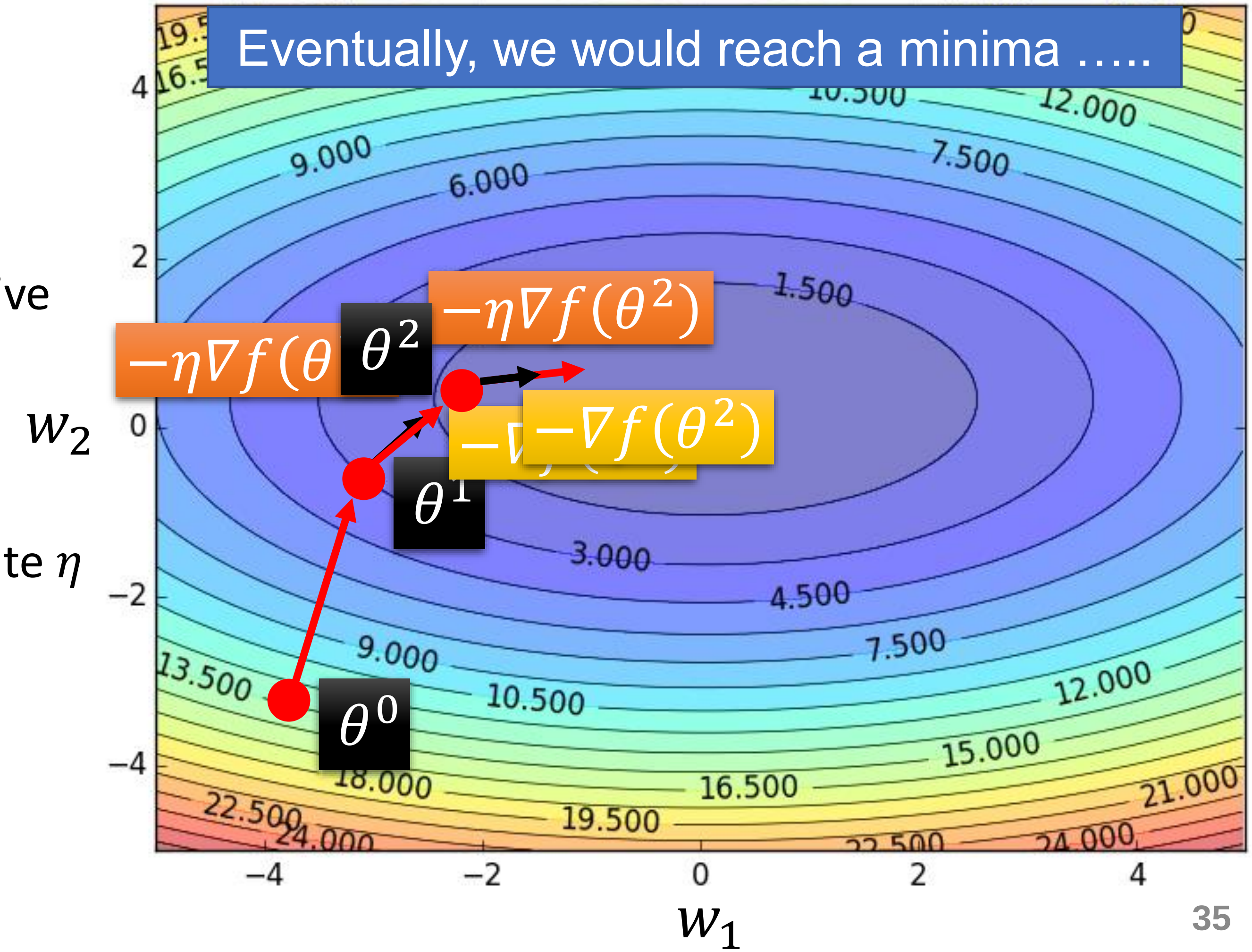
- Consider two parameters, w_1 and w_2, in a network.

Randomly pick a starting point θ^0

Compute the negative gradient at θ^0
$\rightarrow -\nabla f(\theta^0)$

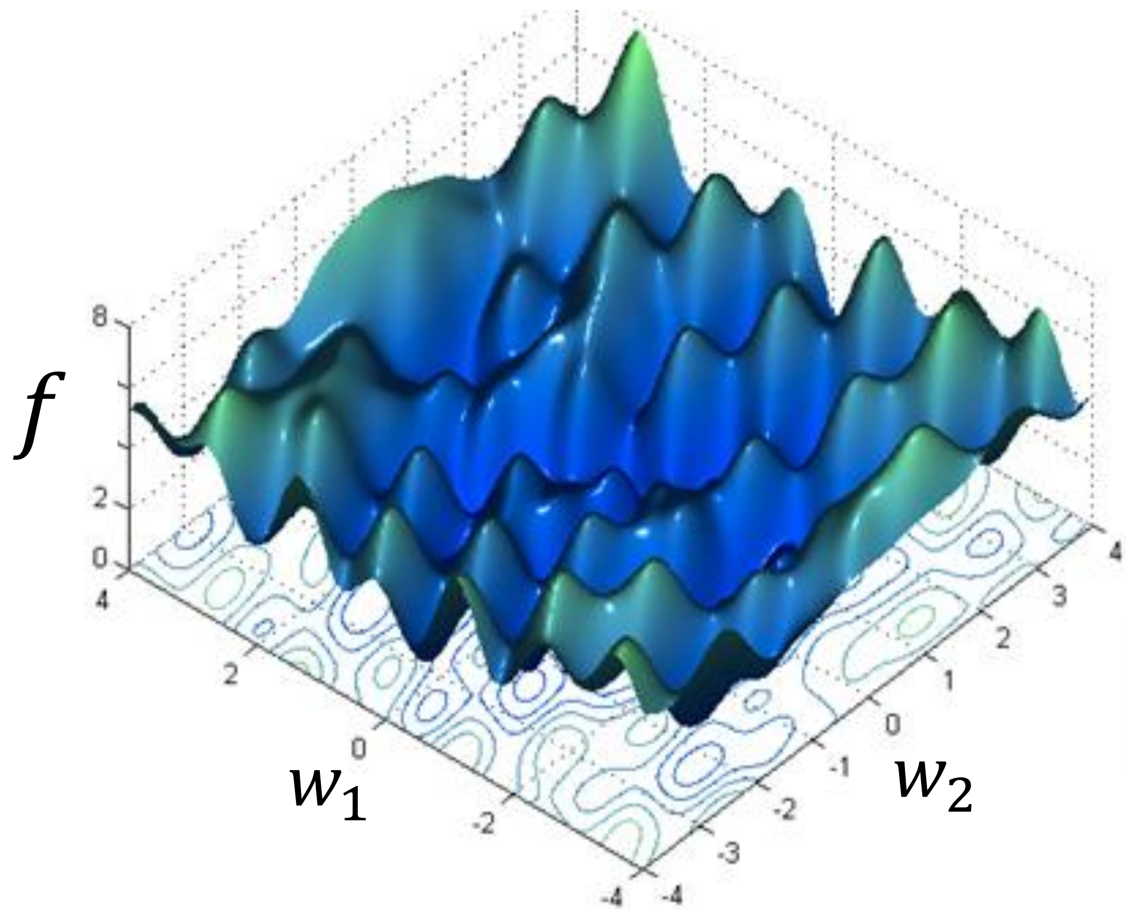
Time the learning rate η
$\rightarrow -\eta\nabla f(\theta^0)$

Error Surface



Gradient descent method

- Gradient descent never guarantees global minima.

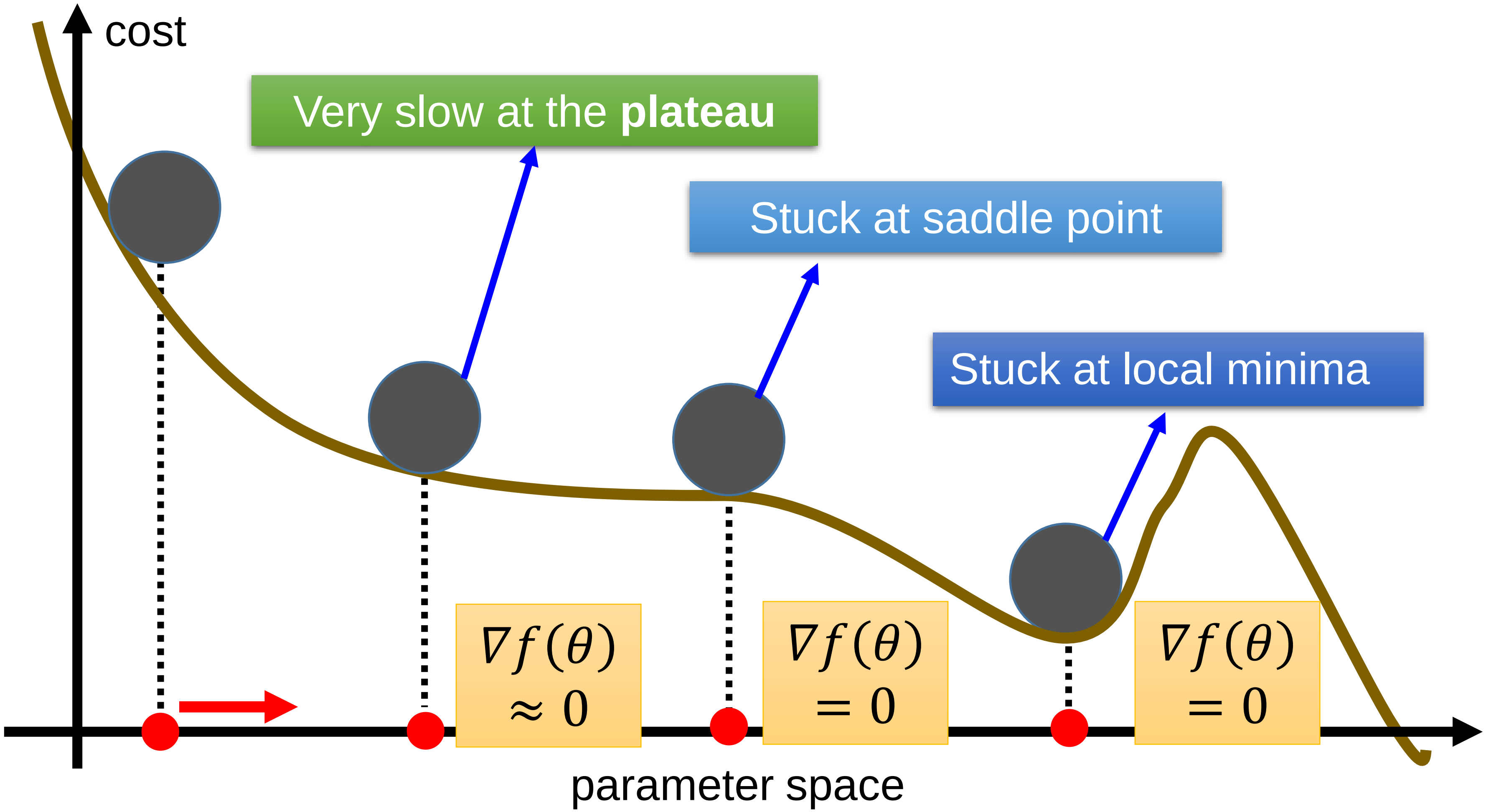


Different initial point θ^0

Reach different minima, so different results

Gradient descent method

- It also has issues at plateau and saddle point.

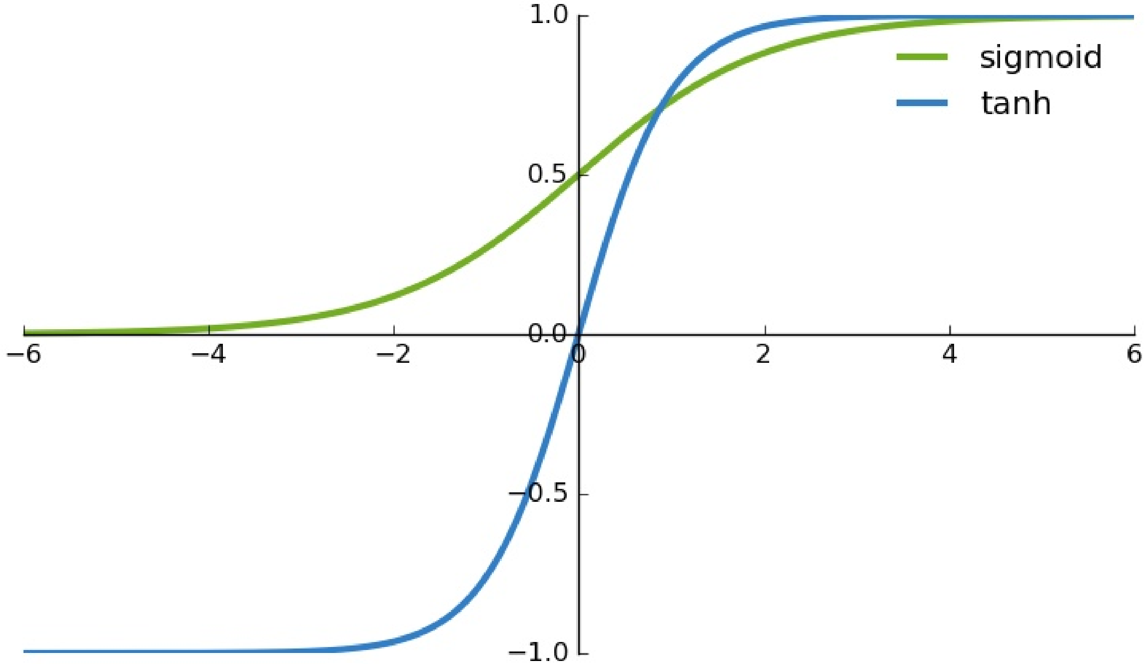


Accelerated learning in ANNs

- Use tanh instead of sigmoid: represent the sigmoidal function by a hyperbolic tangent

$$Y^{\tan h} = \frac{2a}{1 - e^{-bX}} - a$$

where $a = 1.716$ and $b = 0.667$ (Guyon, 1991)

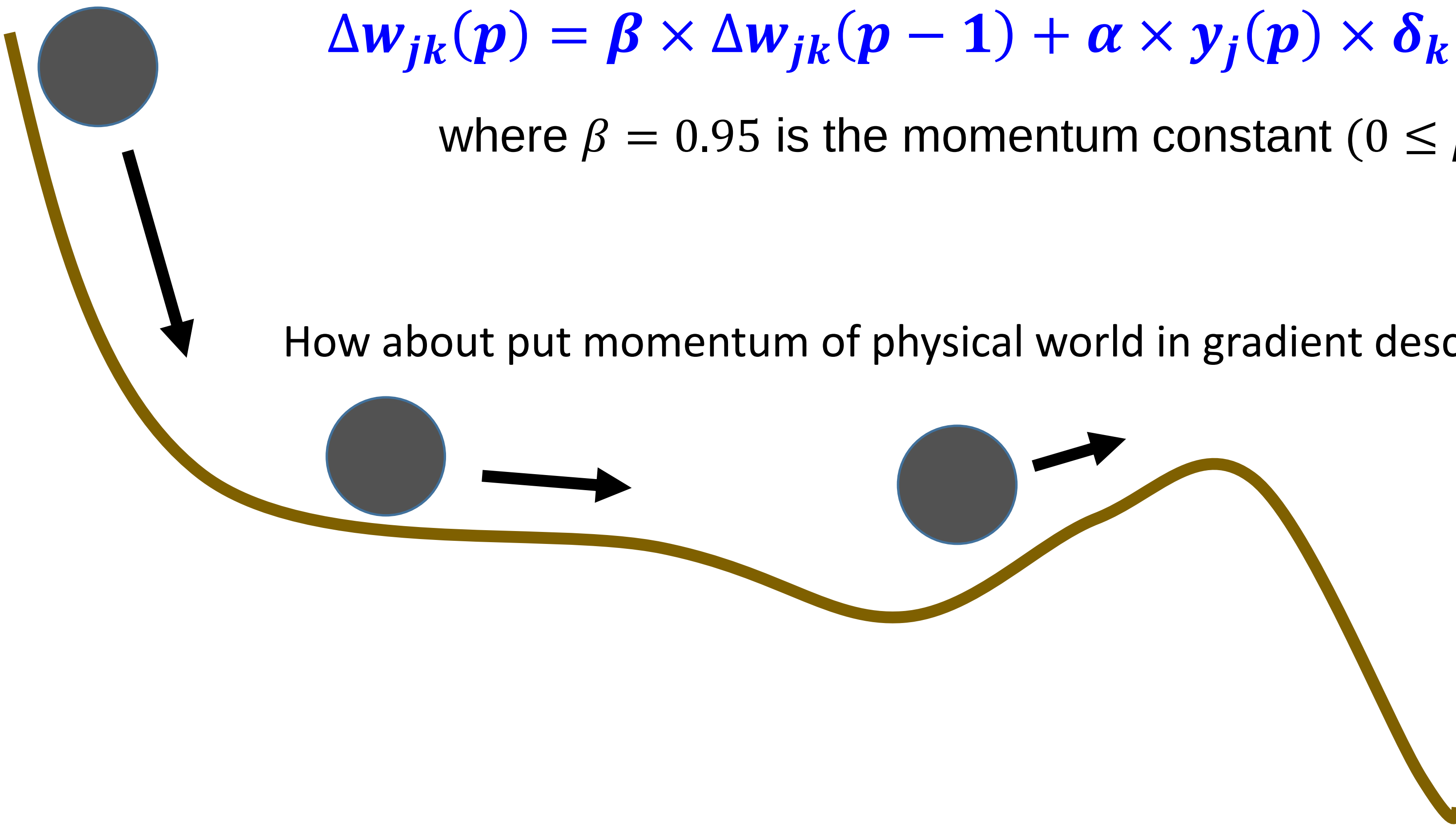


Accelerated learning in ANNs

- Generalized delta rule: A momentum term is included in the delta rule (Rumelhart et al., 1986)

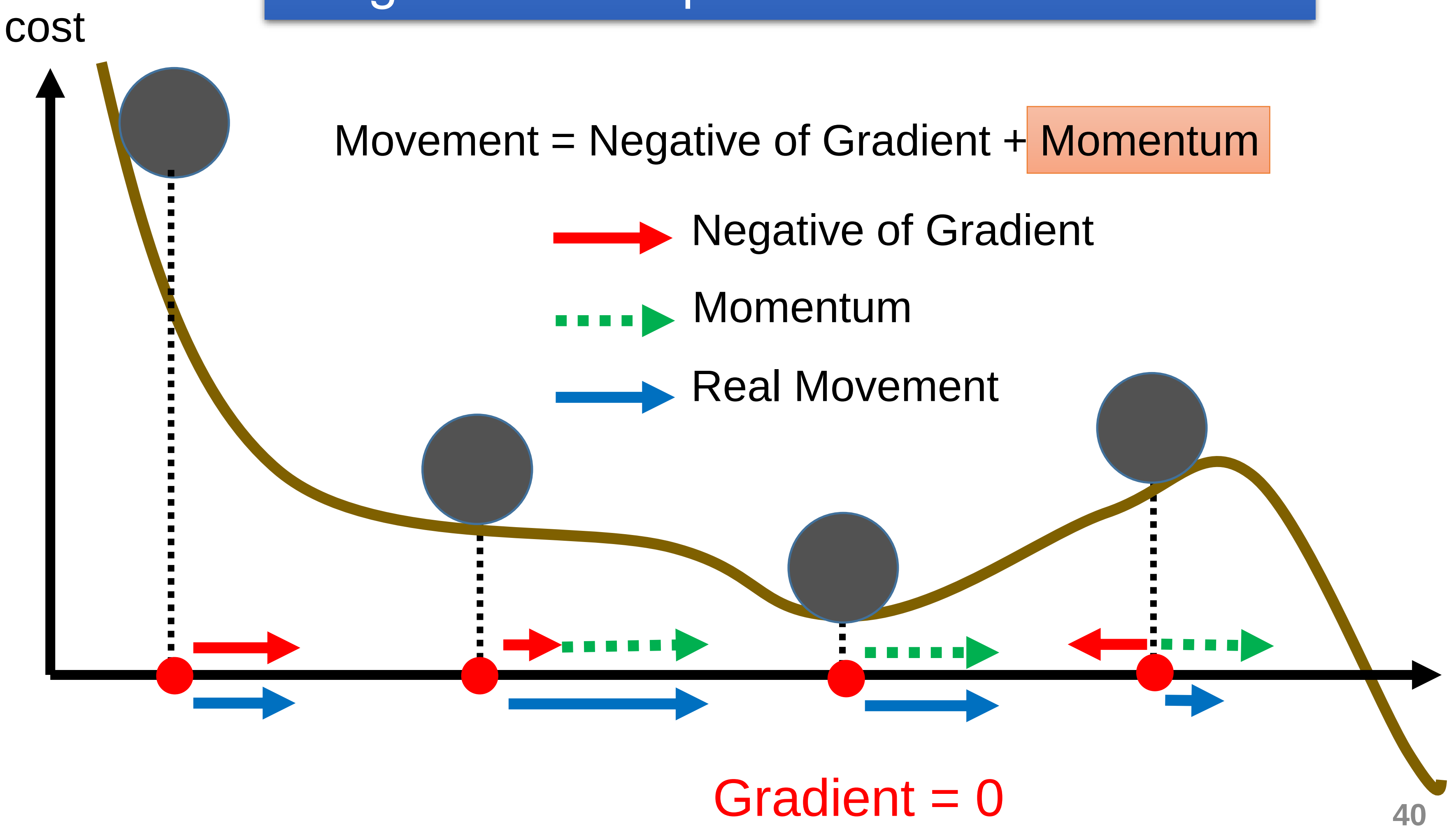
$$\Delta w_{jk}(p) = \beta \times \Delta w_{jk}(p-1) + \alpha \times y_j(p) \times \delta_k(p)$$

where $\beta = 0.95$ is the momentum constant $(0 \leq \beta \leq 1)$



Accelerated learning in ANNs

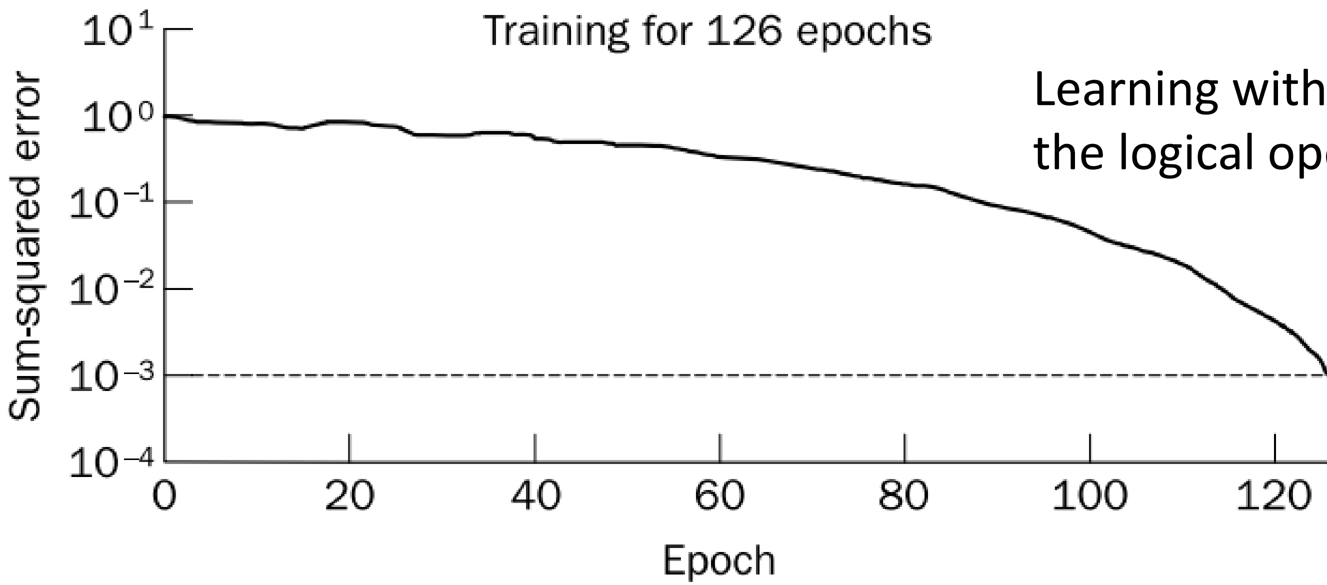
Still not guarantee reaching global minima, but give some hope ……



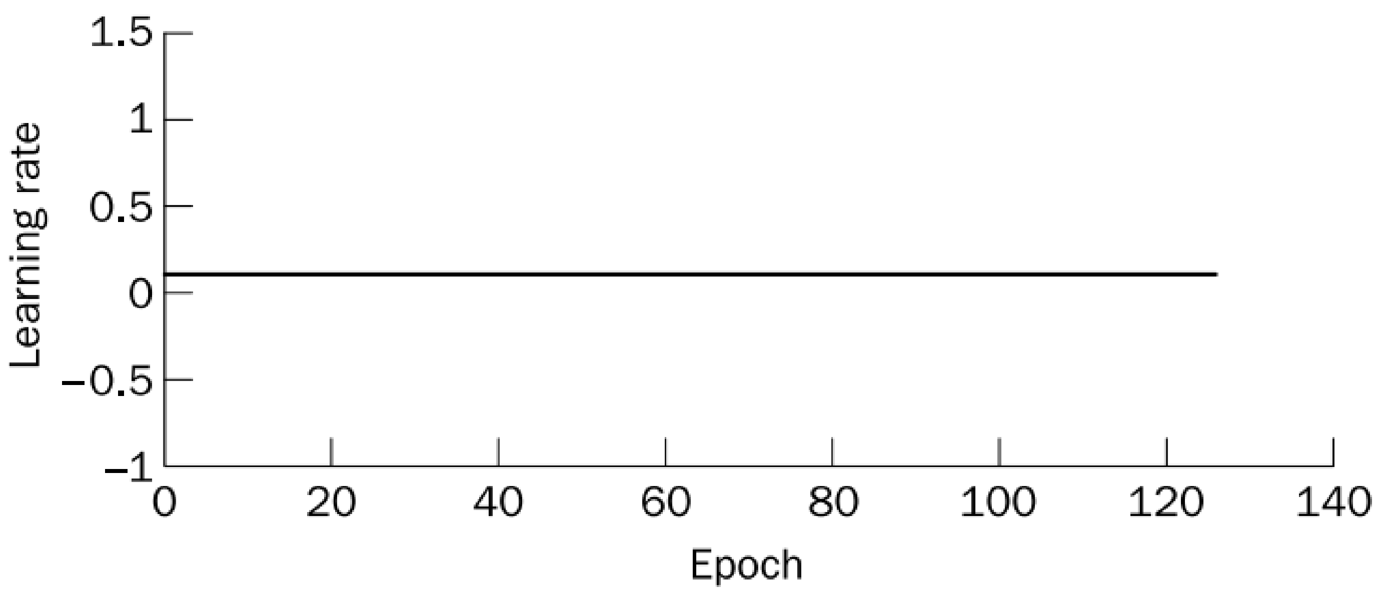
Accelerated learning in ANNs

- Adaptive learning rate: Adjust the learning rate parameter α during training
  - Small $\alpha \rightarrow$ small weight changes through iterations $\rightarrow$ smooth learning curve
  - Large $\alpha \rightarrow$ speed up the training process with larger weight changes $\rightarrow$ possible instability and oscillatory
- Heuristic-like approaches for adjusting α
  1. The algebraic sign of the SSE change remains for several consequent epochs $\rightarrow$ increase α.
  2. The algebraic sign of the SSE change alternates for several consequent epochs $\rightarrow$ decrease α
- One of the most effective acceleration means

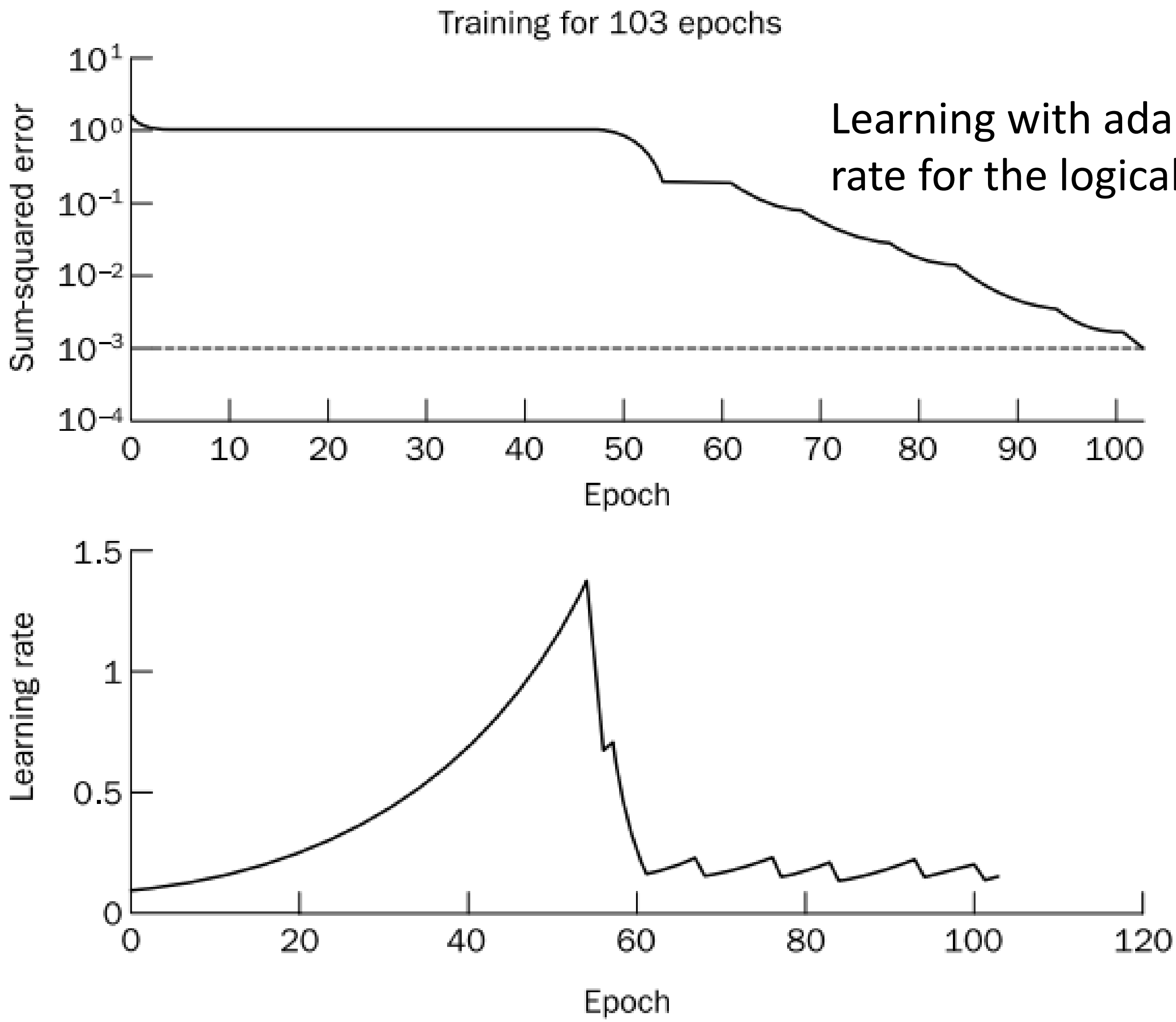
Learning with momentum only



Learning with momentum for the logical operation XOR.

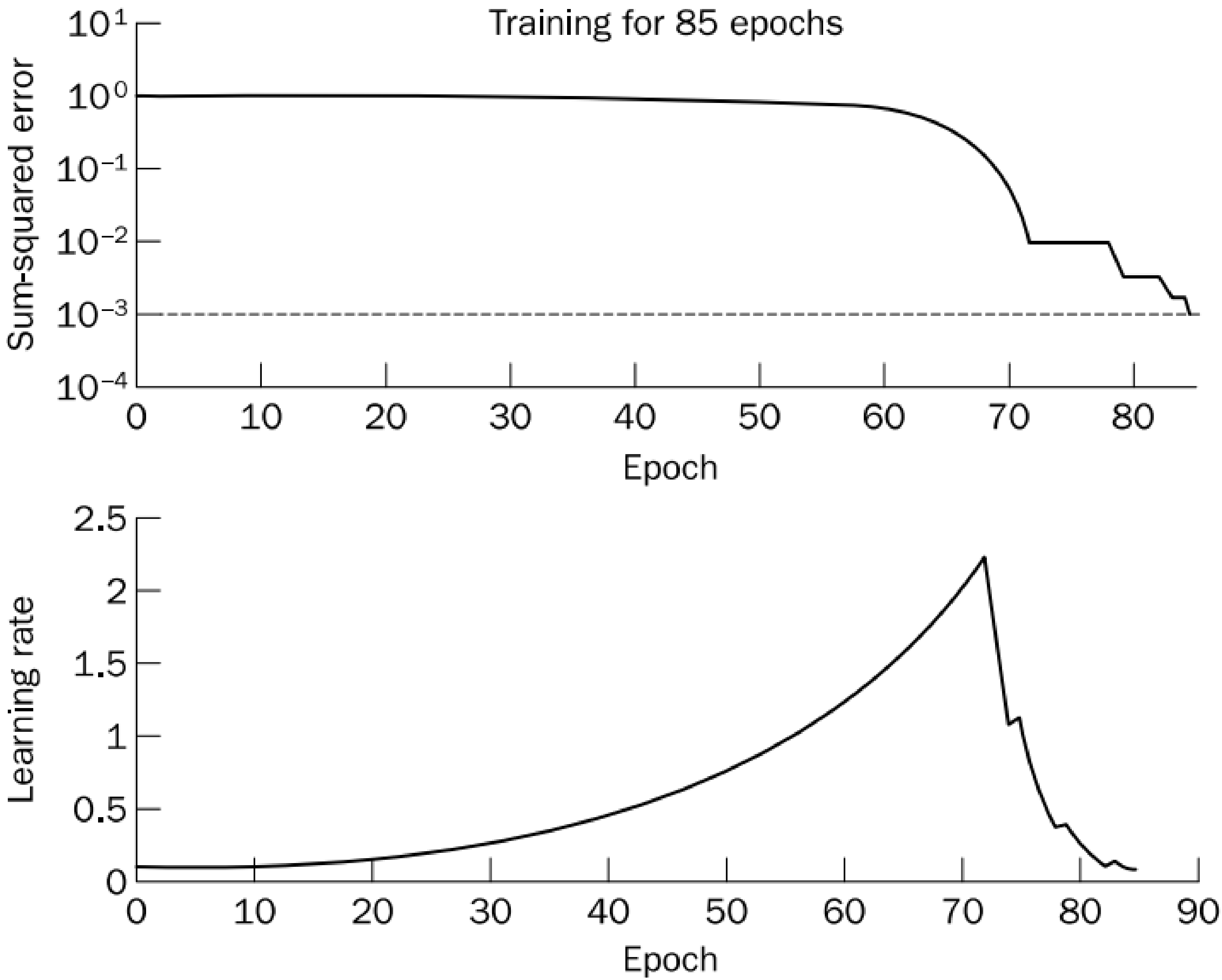


Learning with adaptive α only



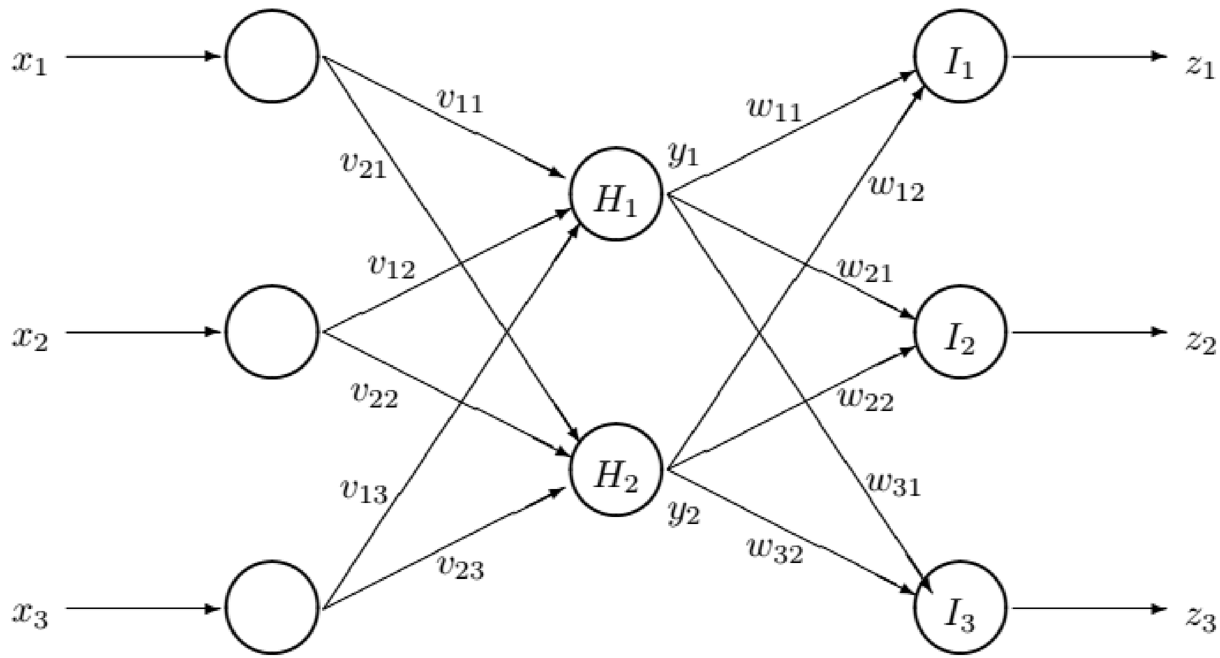
Learning with adaptive learning rate for the logical operation XOR.

Learning with adaptive α and momentum



Quiz 02: Multi-layer neural networks

- Consider the below feedforward network with one hidden layer of units.



- If the network is tested with an input vector $x = [1.0, 2.0, 3.0]$ then what are *the activation* H_1 *of the first hidden neuron* and the *activation* I_3 *of the third output neuron*?

Quiz 02: Multi-layer neural networks

- The input vector to the network is $x = [x_1, x_2, x_3]^T$
- The vector of hidden layer outputs is $y = [y_1, y_2]^T$
- The vector of actual outputs is $z = [z_1, z_2, z_3]^T$
- The vector of desired outputs is $t = [t_1, t_2, t_3]^T$
- The network has the following weight vectors

$$v_1 = \begin{bmatrix} -2.0 \\ 2.0 \\ -2.0 \end{bmatrix} \qquad v_2 = \begin{bmatrix} 1.0 \\ 1.0 \\ -1.0 \end{bmatrix} \qquad w_1 = \begin{bmatrix} 1.0 \\ -3.5 \end{bmatrix} \qquad w_2 = \begin{bmatrix} 0.5 \\ -1.2 \end{bmatrix} \qquad w_3 = \begin{bmatrix} 0.3 \\ 0.6 \end{bmatrix}$$

- Assume that all units have sigmoid activation function given by

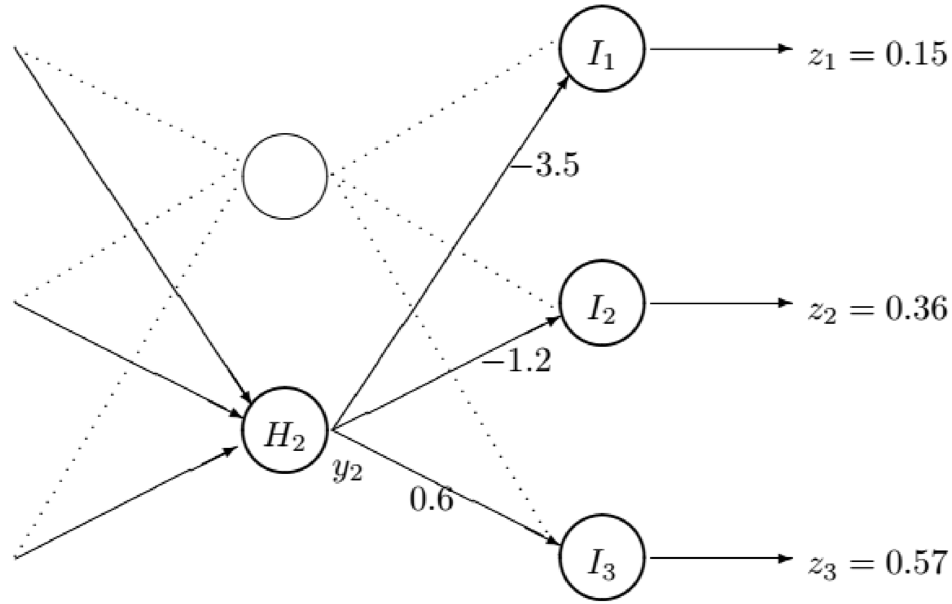
$$f(x) = \frac{1}{1 + \exp(-x)}$$

and that each unit has $\theta = 0$ (zero).

- (Hint: on some calculators, $\exp(x) = e^x$ where $e = 2.7182818$)

Quiz 03: Backpropagation

- The figure shows part of the network described in Slide 48.
- Use the same weights, activation functions and bias values as described.



- A new input pattern is presented to the network and training proceeds as follows. The actual outputs are given by $z = [0.15, 0.36, 0.57]^T$ and the corresponding target outputs are given by $t = [1.0, 1.0, 1.0]^T$.
- The weights w_{12}, w_{22} and w_{32} are also shown.
- What is the error for each of the output units?

THE END