Multi-layer Perceptron

Nguyen Ngoc Thao

Department of Computer Science, FIT
University of Science, VNU-HCM

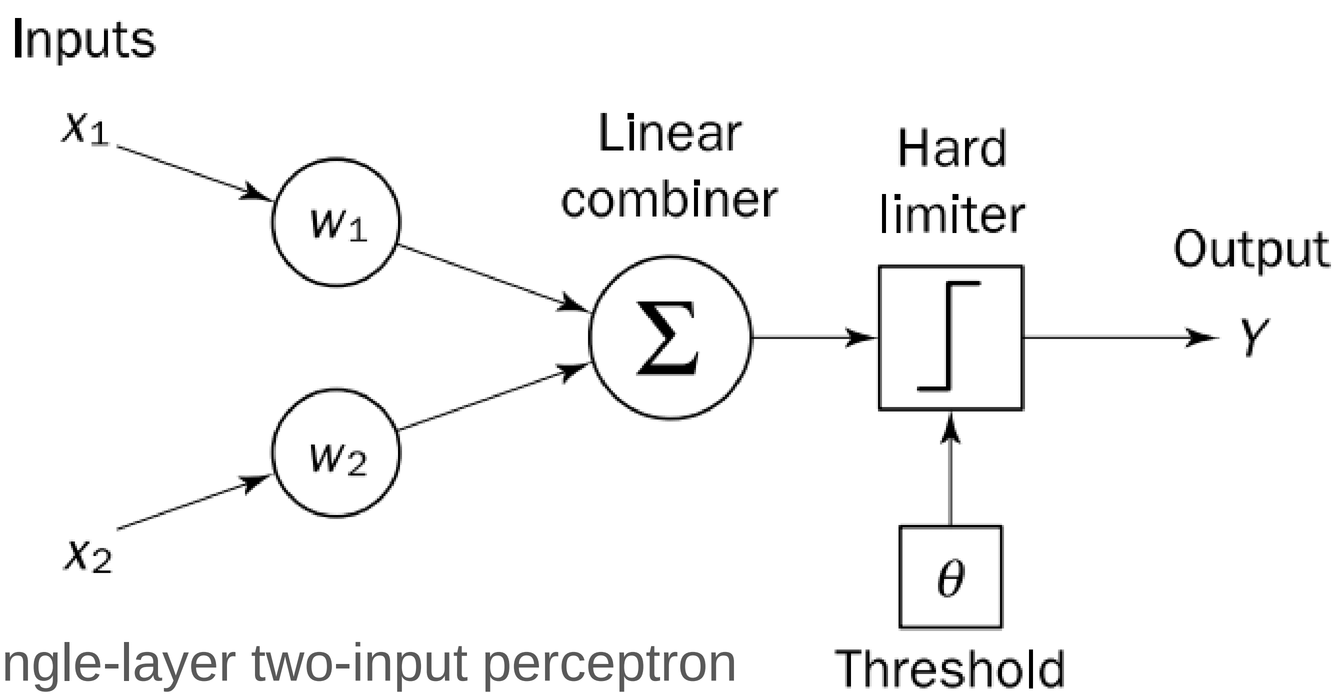
Content outline

- The perceptron
- Multi-layer Perceptron (MLP)
- Accelerated learning in MLP

Perceptron: The simplest ANN

Perceptron (Frank Rosenblatt, 1958)

- The perceptron consists a single neuron with adjustable synaptic weights and a hard limiter.

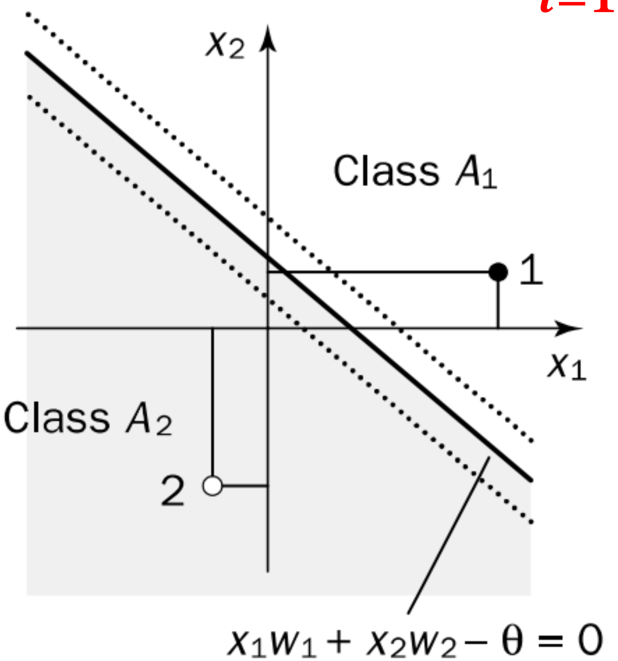


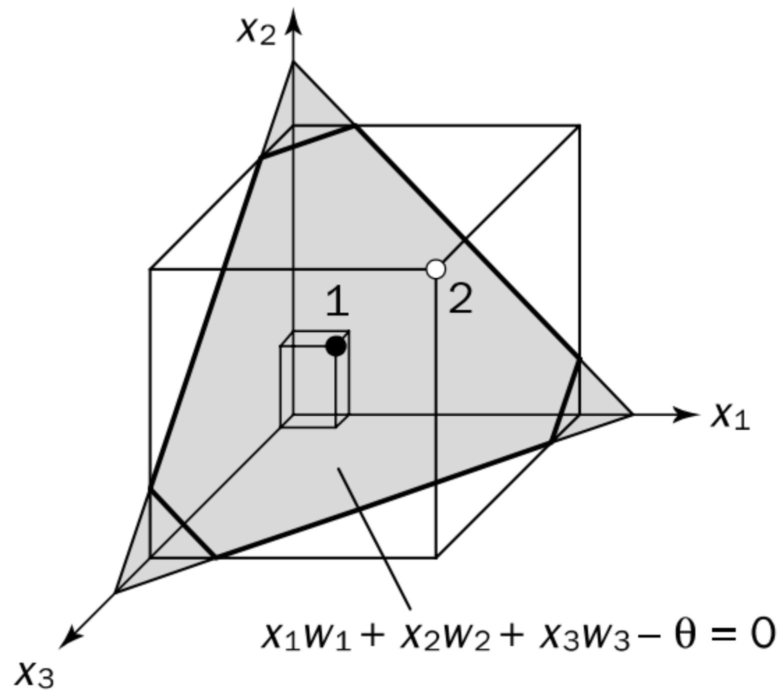
Single-layer two-input perceptron

How does a perceptron work?

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

$$\sum_{i=1}^{n} x_i w_i - \theta = 0$$





Perceptron learning rule

- Step 1: Initialization
  - Initial weights $w_1, w_2, \dots, w_n$ and threshold θ are randomly assigned to numbers $\in [-0.5, 0.5]$.
- Step 2: Activation
  - Activate the perceptron by applying inputs $\boldsymbol{x_1(p), x_2(p), \dots, x_n(p)}$ and desired output $\boldsymbol{Y_d(p)}$.
  - Calculate the actual output at iteration $\boldsymbol{p}$

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

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

* *Iteration p refers to the p^{th} training example presented to the perceptron.*

Perceptron learning rule

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

    where $\Delta w_i(p)$ is the weight correction at iteration p
  - The delta rule determines how to adjust the weights by $\Delta w_i(p)$

$$\boldsymbol{\Delta w_i(p) = \alpha \times x_i(p) \times e(p)}$$

    where α is the learning rate $(0 < \alpha < 1)$ and $e(p) = Y_d(p) - Y(p)$

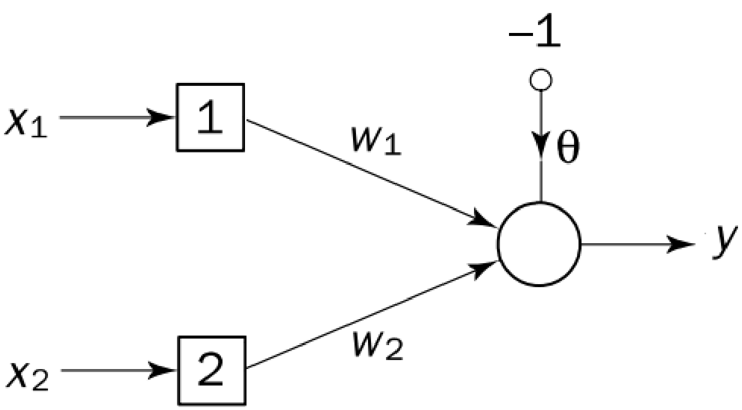
- Step 4: Iteration
  - Increase iteration $\boldsymbol{p}$ by one, go back to Step 2 and repeat the process until convergence.

Perceptron for the logical AND/OR

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

Epoch	Inputs		Desired output	Initial weights		Actual output	Error	Final weights	
	x_1	x_2	Y_d	w_1	w_2	Y	e	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

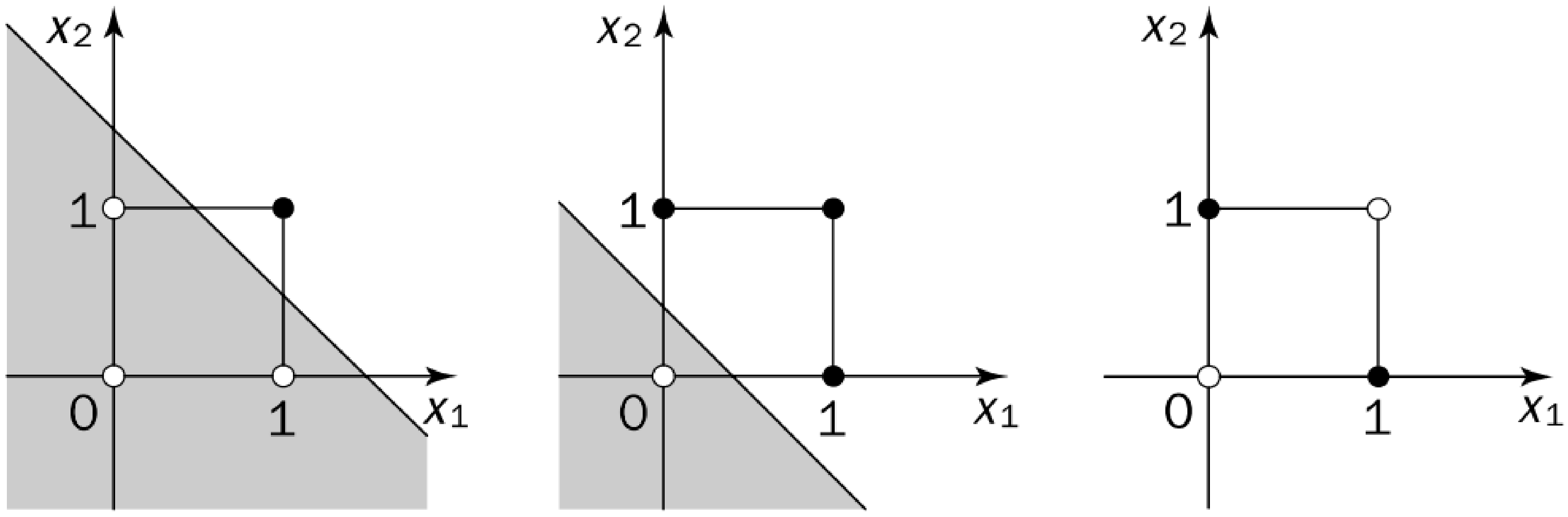
Threshold: $\theta = 0.2$; learning rate: $\alpha = 0.1$.



The learning of logical AND converged after several iterations

Perceptron for the logical XOR

- However, a single-layer perceptron 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?

- Perceptrons 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)

Perceptron: An example

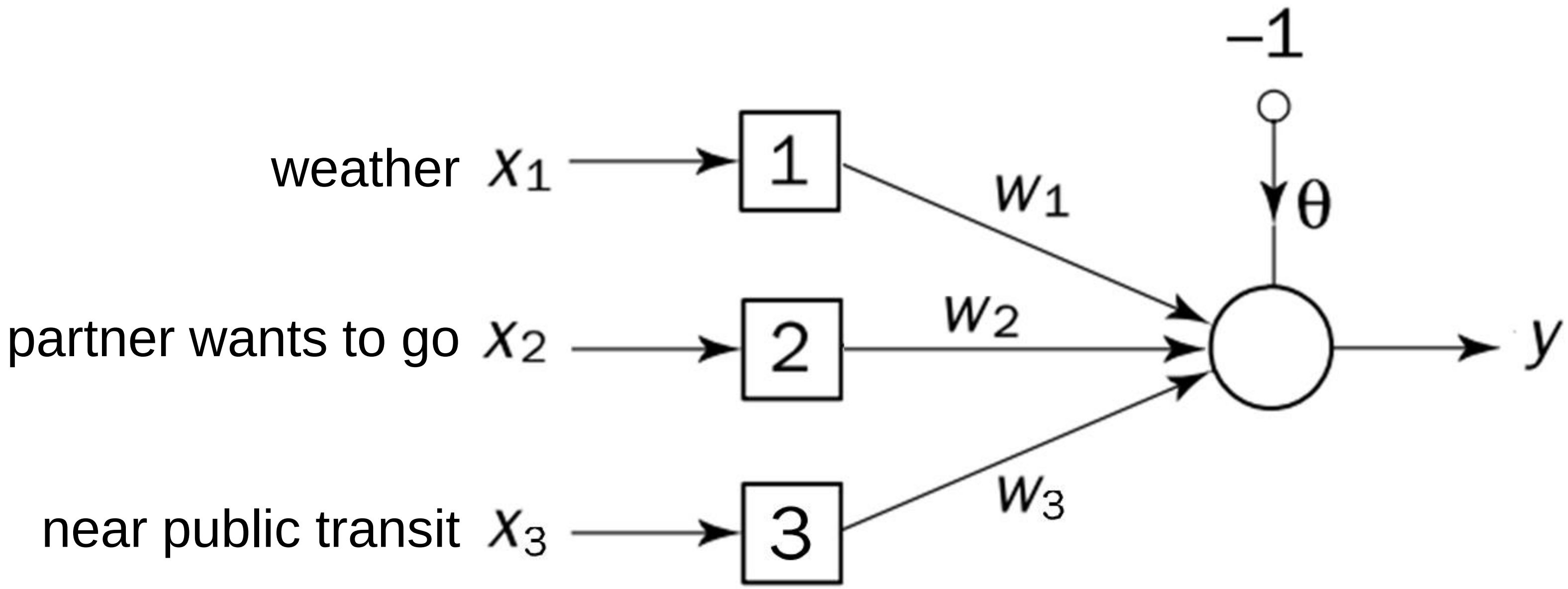
Is the weather good?

Does your partner want to accompany you?

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

go to the festival?

Perceptron: An example

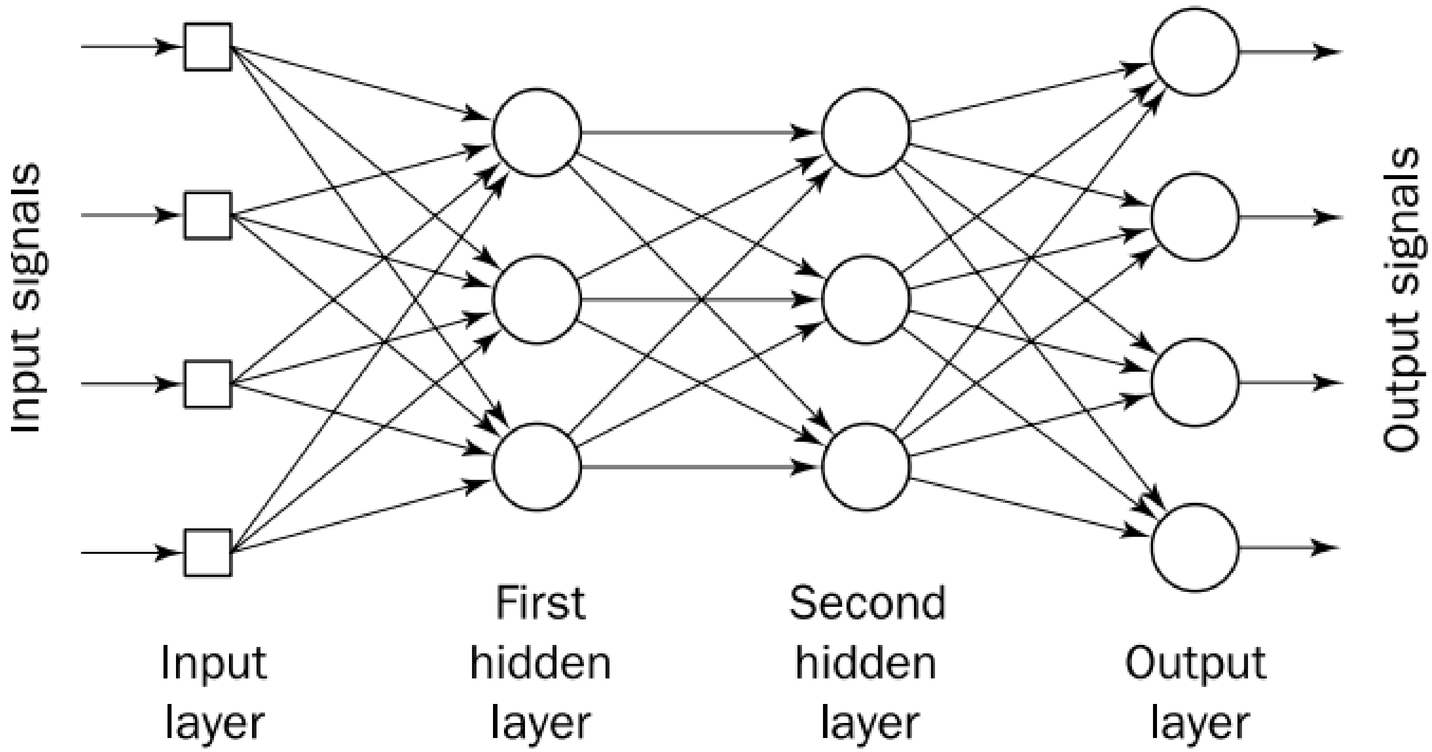


- $w_1 = 6, w_2 = 2, w_3 = 2$ $\rightarrow$ the weather matters to you much more than whether your partner joins you, or the nearness of public transit
- $\theta = 5$ $\rightarrow$ decisions are made based on the weather only
- $\theta = 3$ $\rightarrow$ you go to the festival whenever the weather is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

Multi-layer Neural Networks

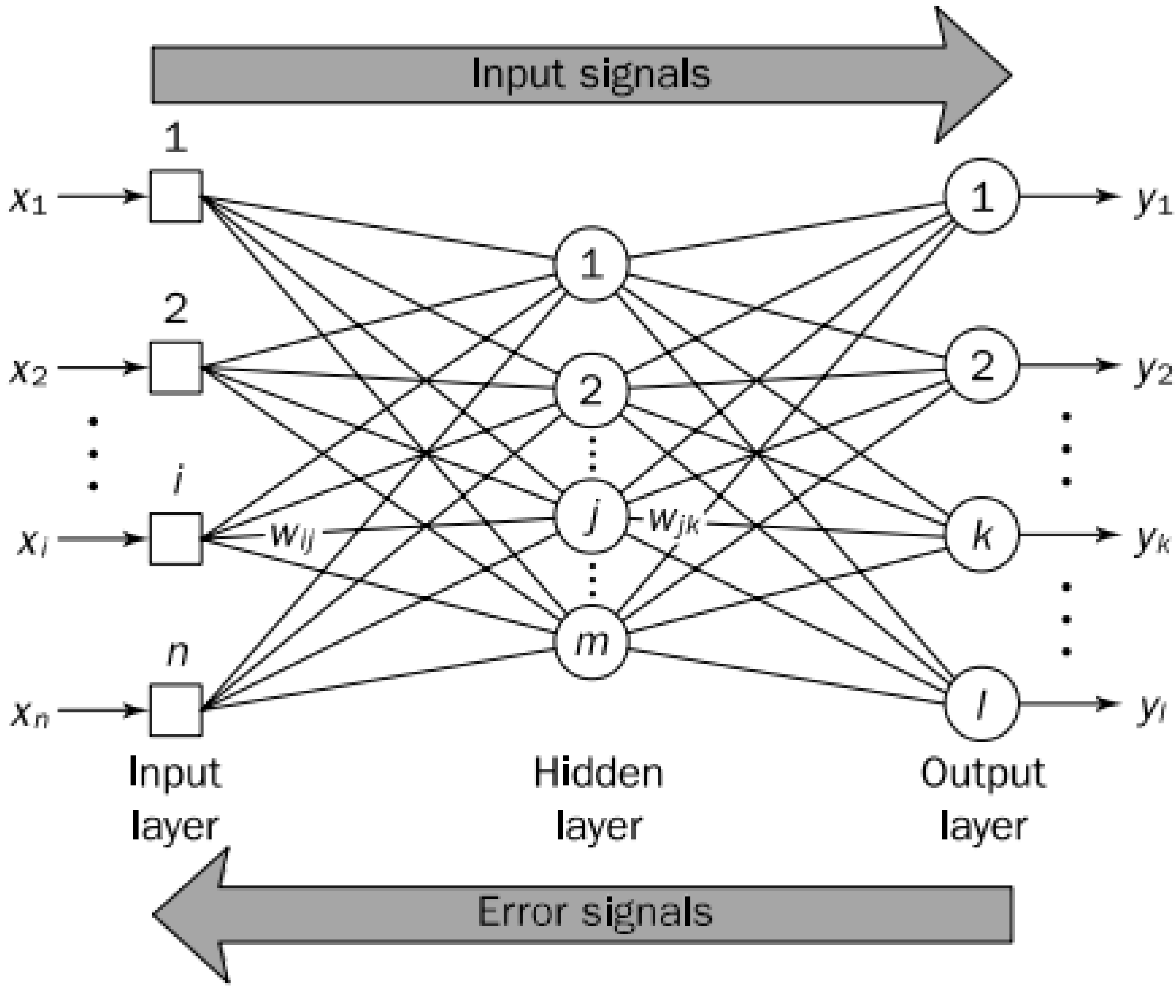
Multi-layer neural network

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



Back-propagation learning algorithm

- Proposed by 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.
  - Random numbers may uniformly distributed inside a small 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
  - Activate the network by applying inputs $\boldsymbol{x_1(p), x_2(p), \ldots, x_n(p)}$ and desired outputs $\boldsymbol{y_{d,1}(p), y_{d,2}(p), \ldots, y_{d,l}(p)}$.

Back-propagation learning rule

- Step 2: Activation (cont.)
  - (a) Calculate the actual outputs of the neurons in the hidden layer

$$\boldsymbol{y_j(p) = \mathbf{sigmoid}\left[\sum_{i=1}^{n} x_i(p) \times w_{ij}(p) - \theta_j\right]}$$

where $\boldsymbol{n}$ is the number of inputs of neuron $\boldsymbol{j}$ in the hidden layer.

  - (b) Calculate the actual outputs of the neurons in the output layer

$$\boldsymbol{y_k(p) = \mathbf{sigmoid}\left[\sum_{j=1}^{m} y_j(p) \times w_{jk}(p) - \theta_k\right]}$$

where $\boldsymbol{m}$ is the number of inputs of neuron $\boldsymbol{k}$ in the output layer

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 the neurons in the output layer

$$\boldsymbol{\delta_k(p) = y_j(p) \times y_k(p) \times [1 - y_k(p)] \times e_k(p)}$$

where $e_k(p) = y_{d,k}(p) - y_k(p)$

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

Update the weights at the output neurons

$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p)$$

Back-propagation learning rule

- Step 3: Weight training (cont.)
  - (b) Calculate the error gradient for the neurons in the hidden layer

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

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

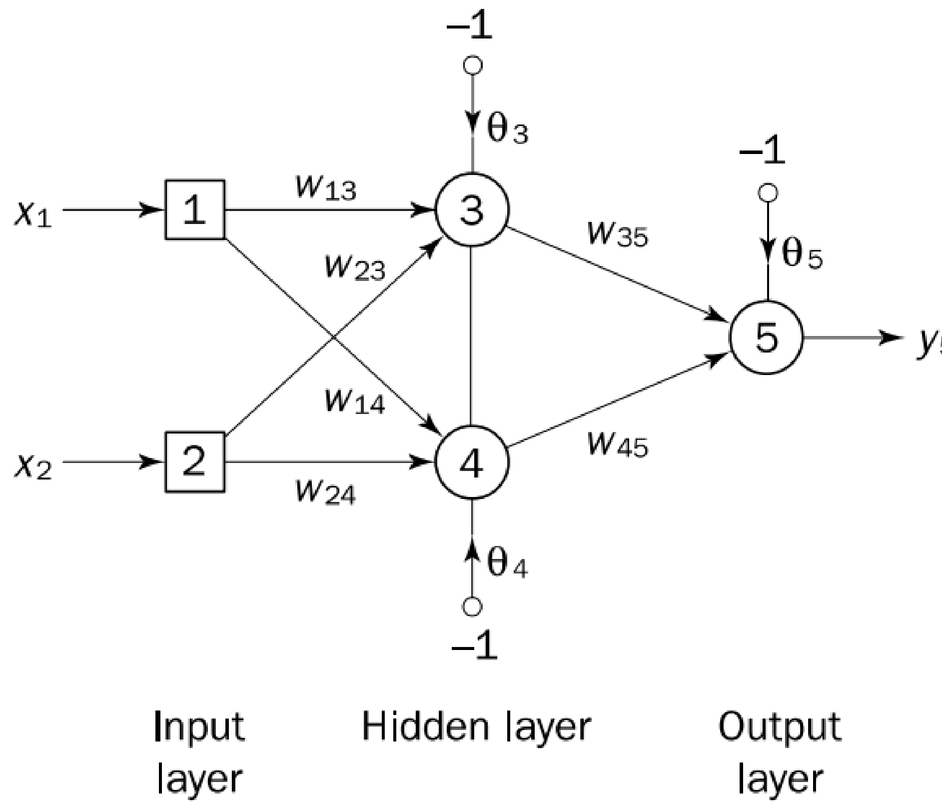
Update the weights at the hidden neurons:

$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$$

- 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.

Back-propagation network for XOR

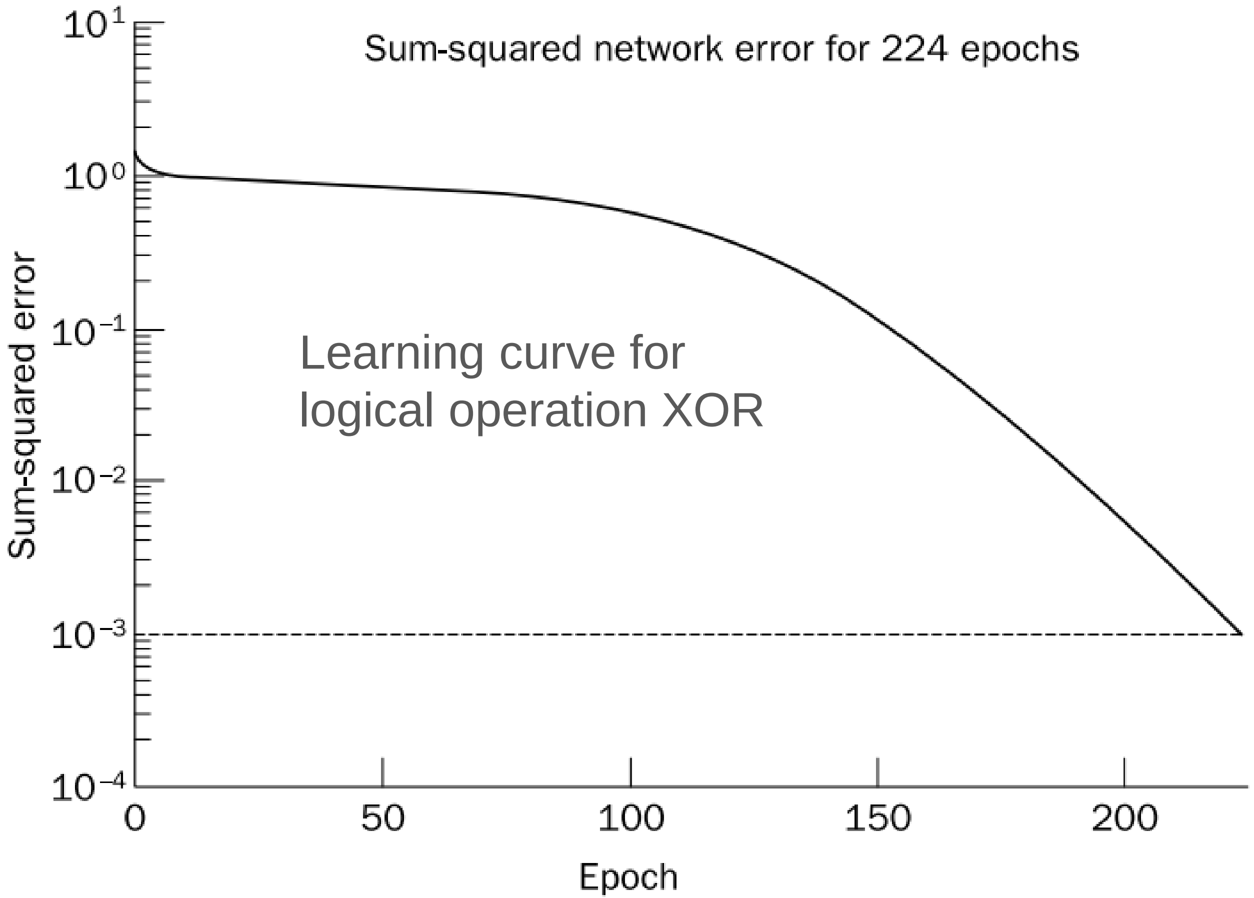
- The logical XOR problem took 224 epochs or 896 iterations for network training.



Inputs x_1	Inputs x_2	Desired output y_d	Actual output y_5	Error e	Sum of squared errors
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	

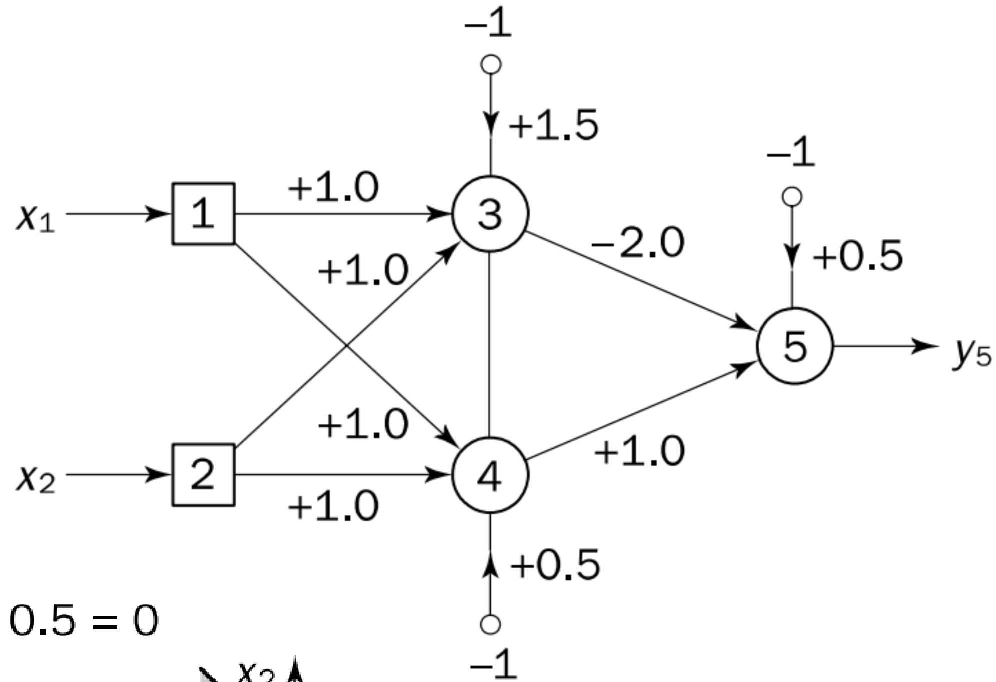
Sum of the squared errors (SSE)

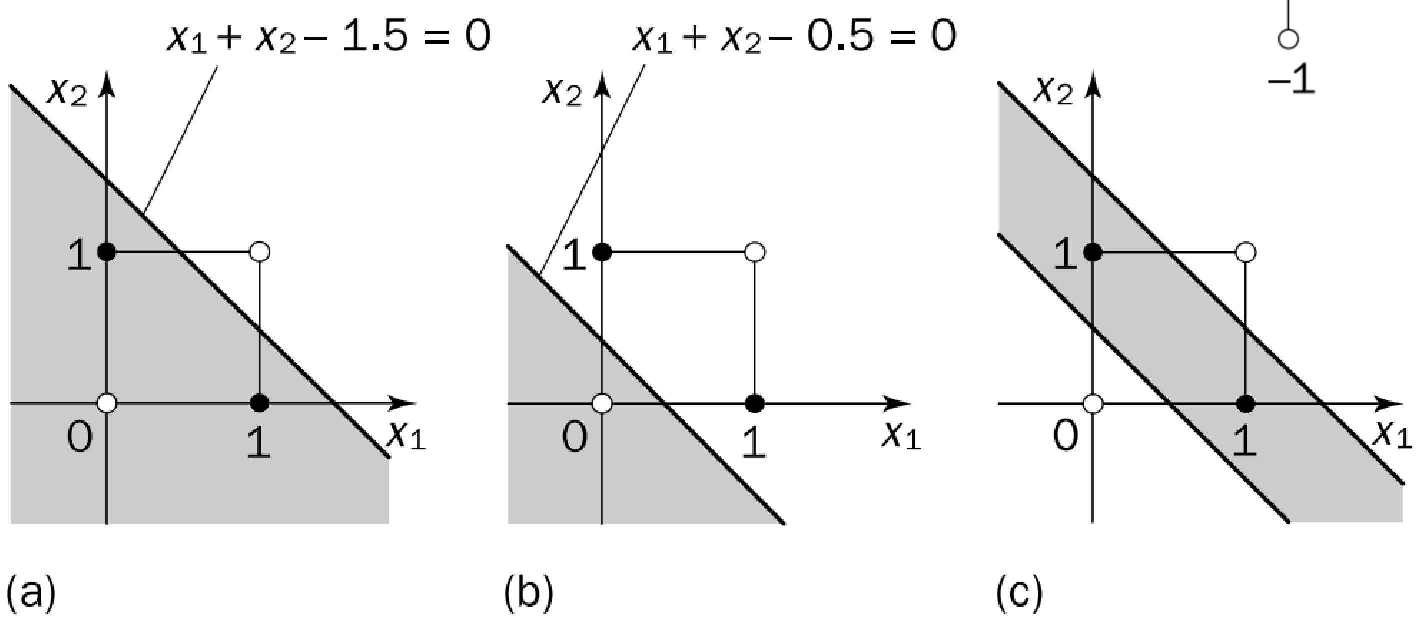
- When the SSE in an entire pass through all training sets is sufficiently small, a network is deemed to have converged.



Decision boundaries for XOR

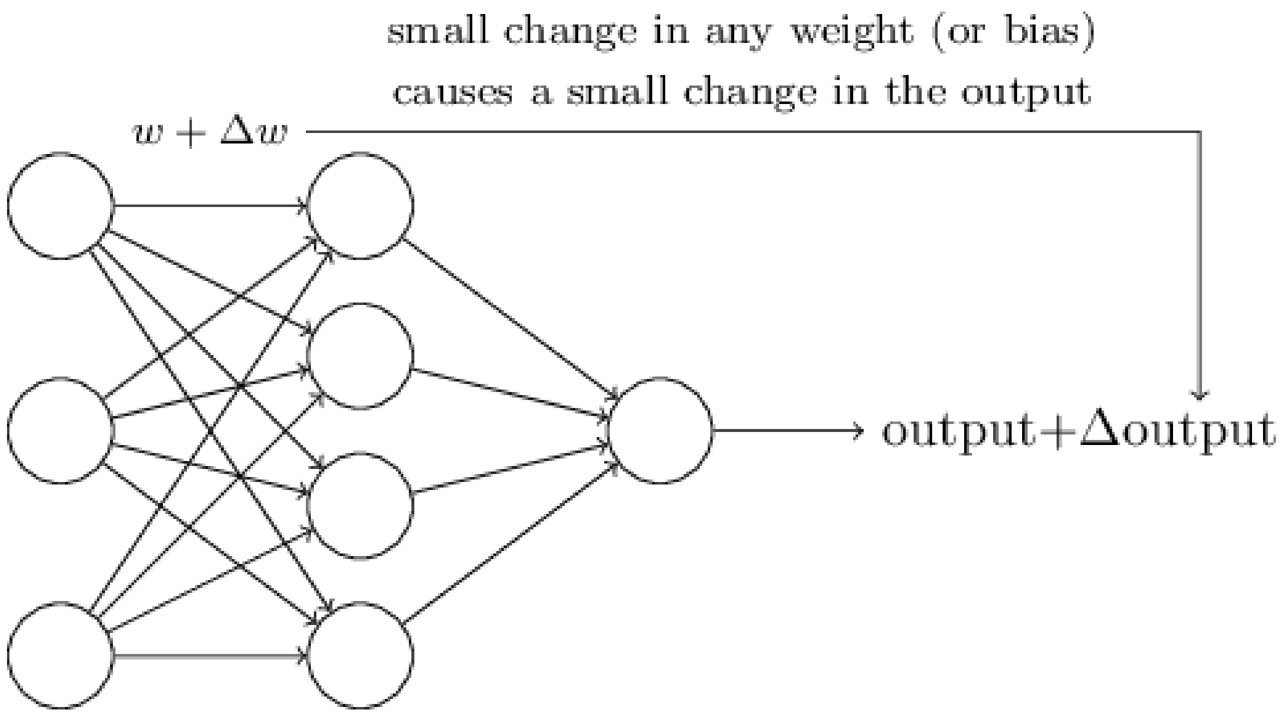
Decision boundaries are demonstrated with McCulloch-Pitts neurons using a sign function.





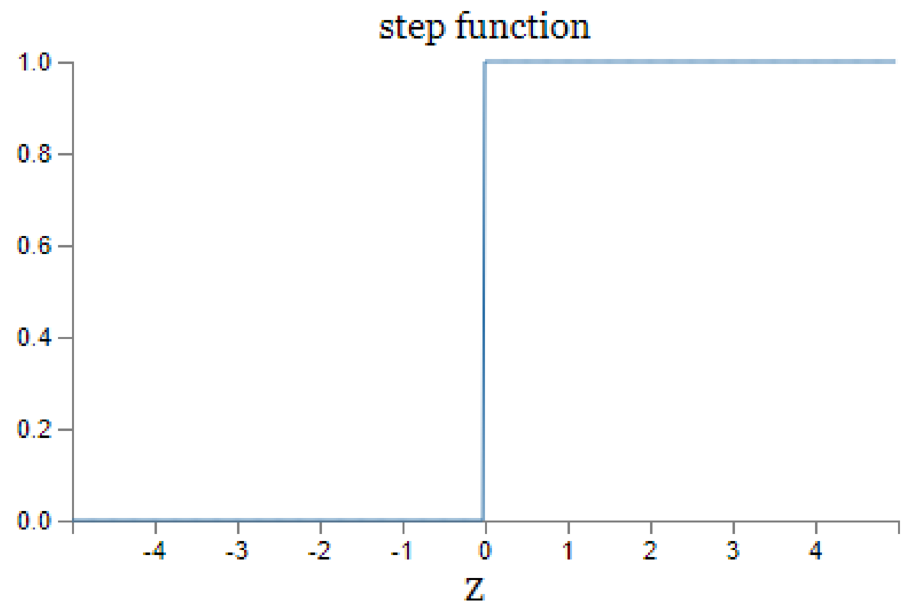
Sigmoid neuron vs. perceptron

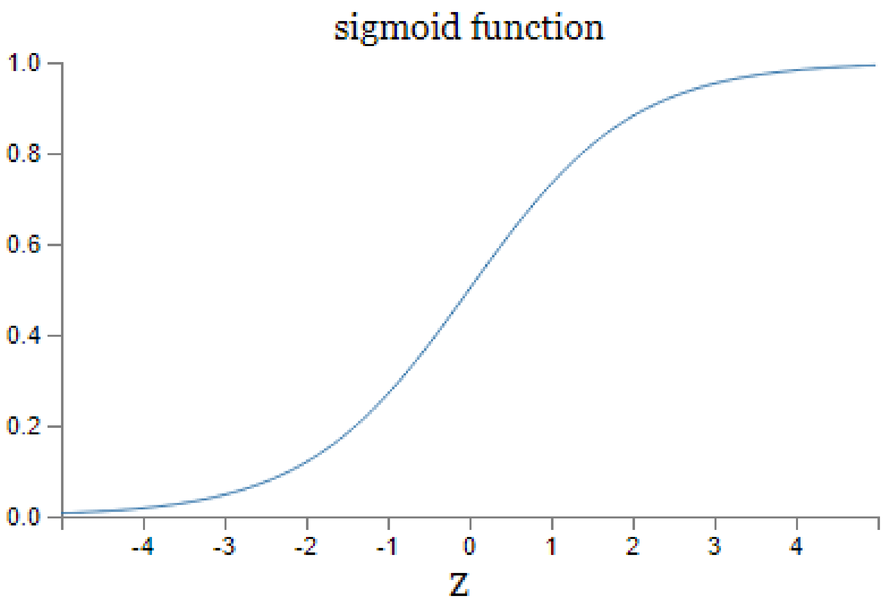
- Sigmoid neuron is similar to perceptron, but modified so that small changes in weights and bias cause only a small change in output.



Sigmoid neuron vs. perceptron

- The shape of 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. However, the problem will always be solved although using a different number 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

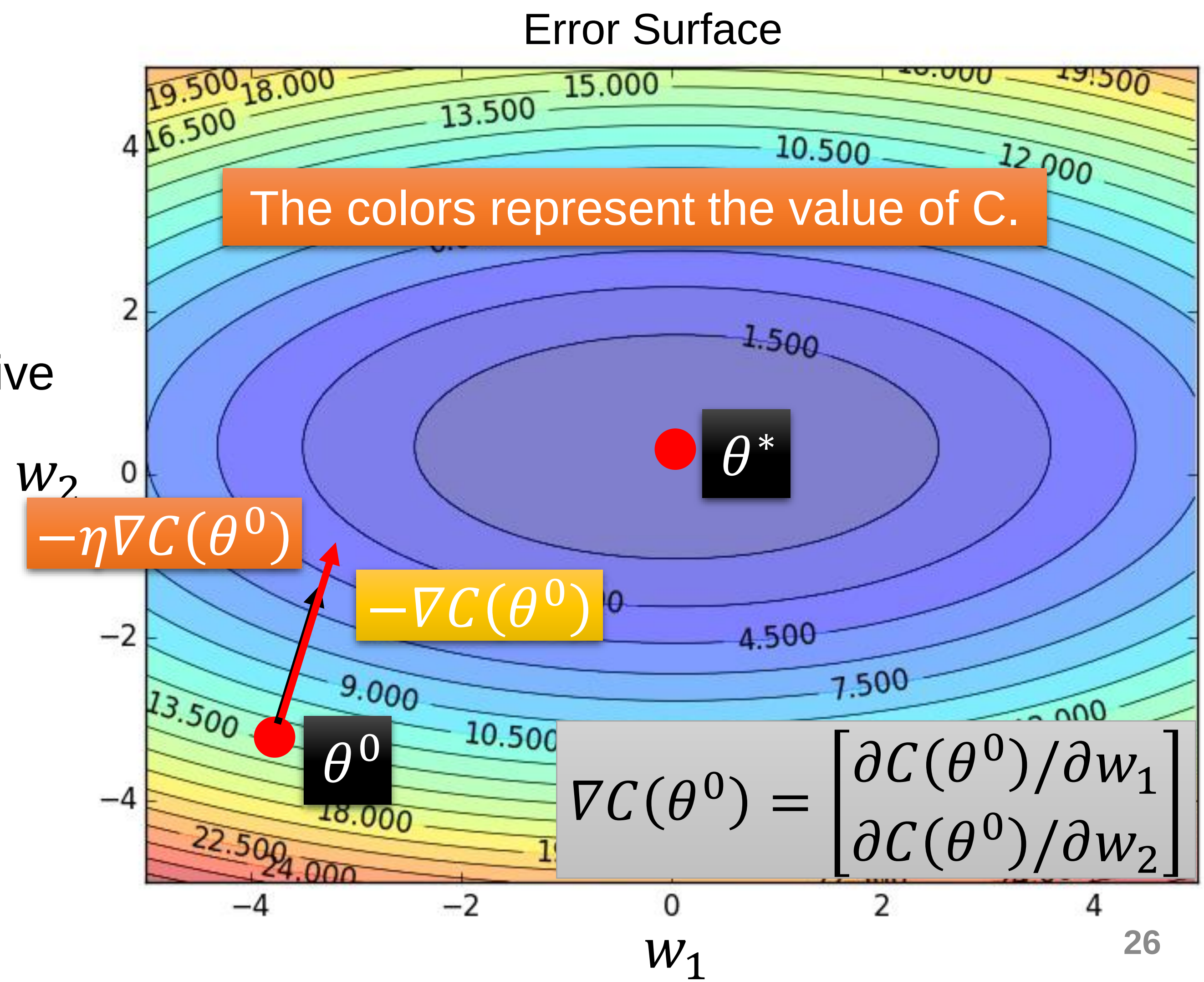
Gradient descent method

- Consider two parameters, w_1 and w_2 in a network

Randomly pick a starting point θ^0

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

Times the learning rate η $\rightarrow -\eta\nabla C(\theta^0)$



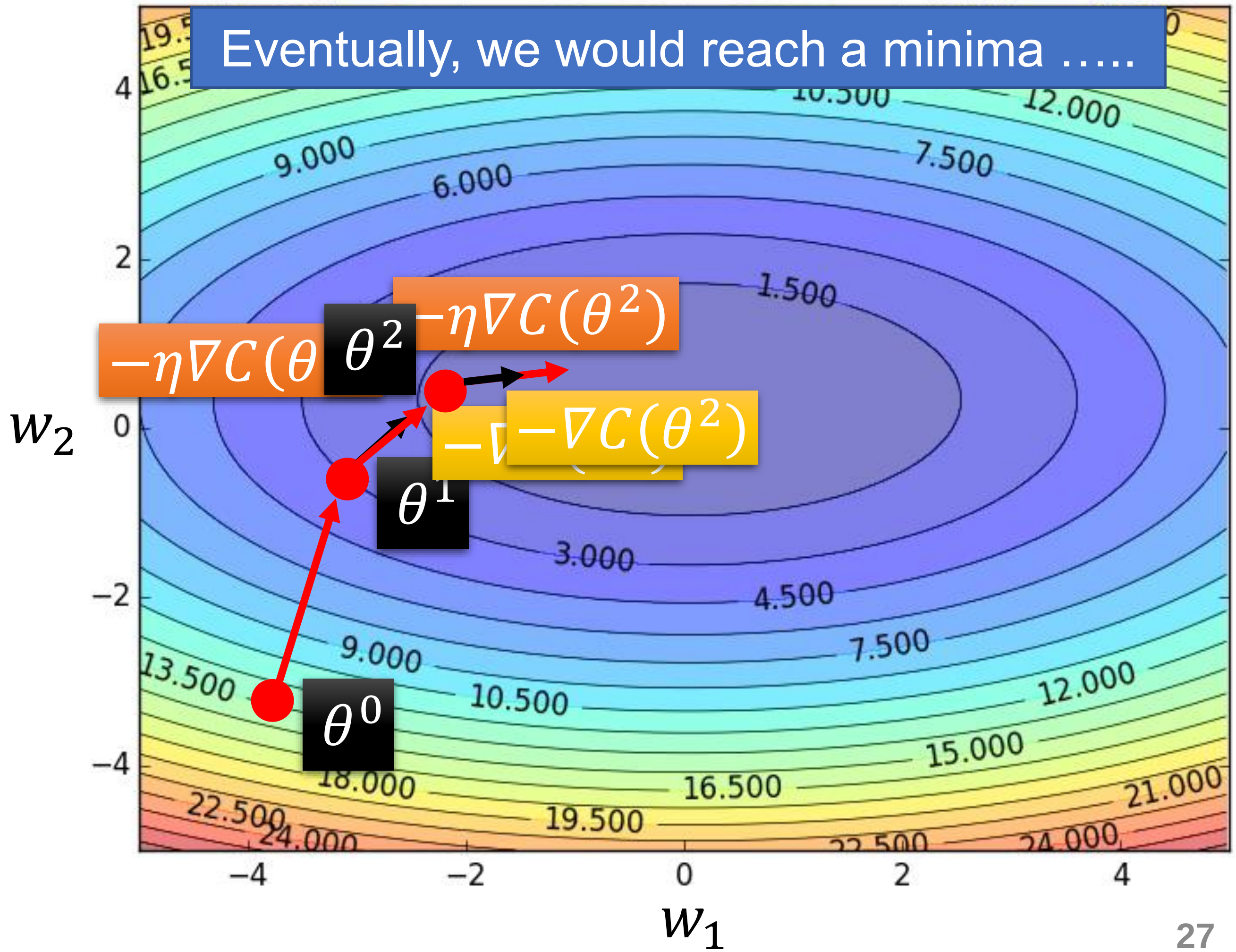
Gradient descent method

- Consider two parameters, w_1 and w_2 in a network

Randomly pick a starting point θ^0

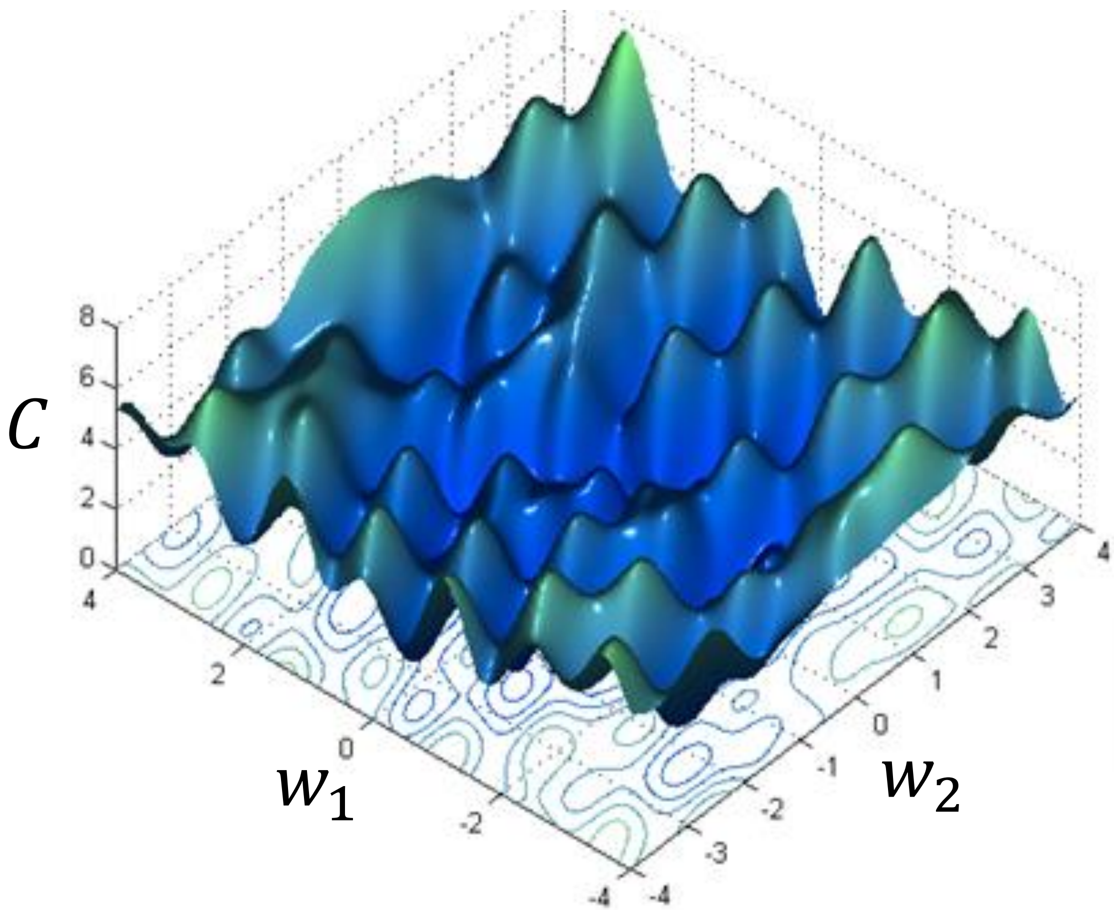
Compute the neg gradient at $\theta^0 \rightarrow -\nabla C(\theta^0)$

Times the learnin rate η $\rightarrow -\eta\nabla C(\theta^0)$



Gradient descent method

- Gradient descent never guarantees global minima.

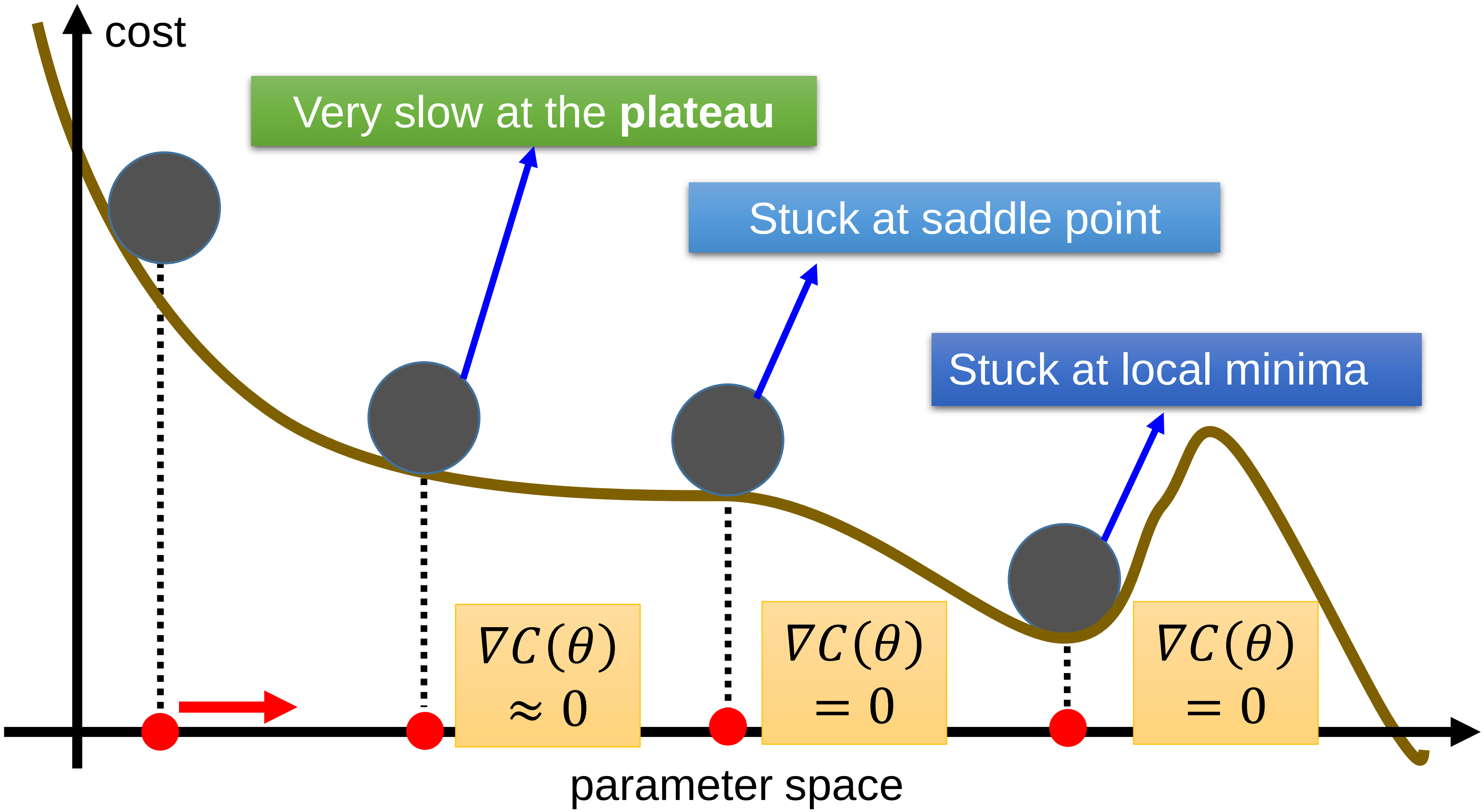


Different initial point θ^0

Reach different minima, so different results

Gradient descent method

- It also has issues at plateau and saddle point.



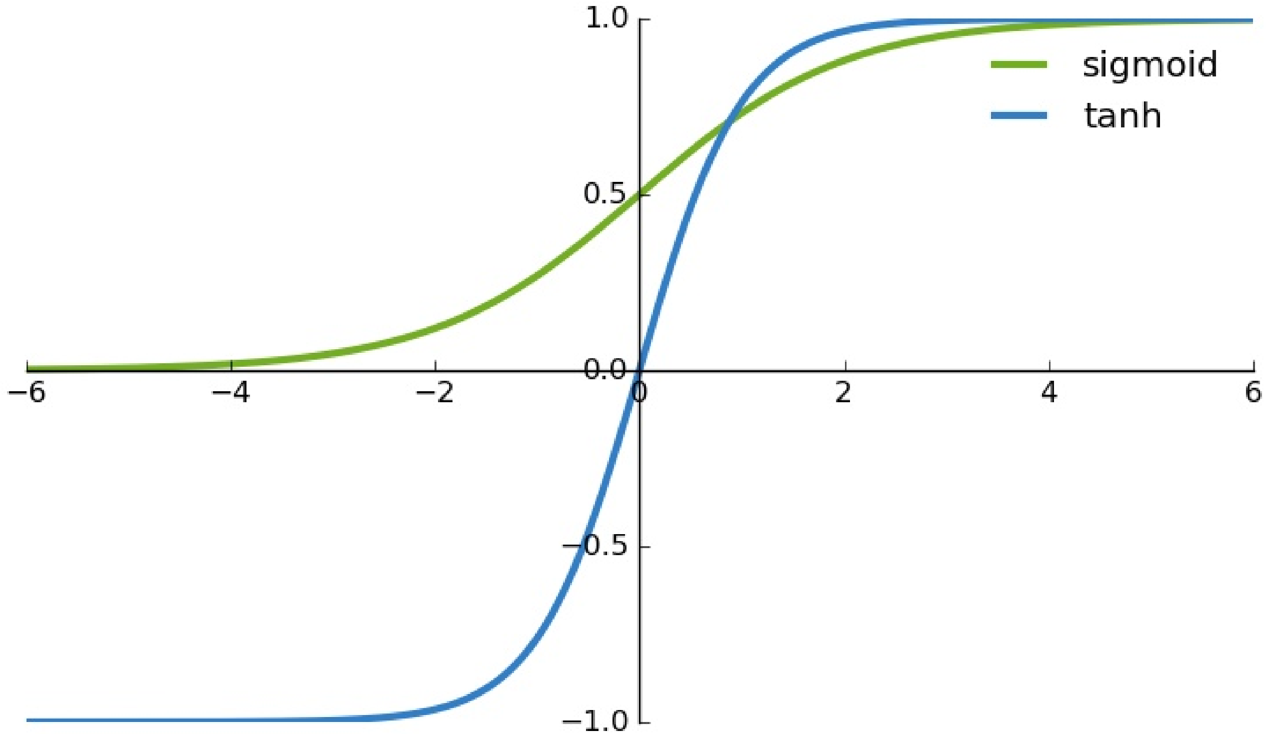
Accelerated Learning in Multi-layer Neural Networks

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)



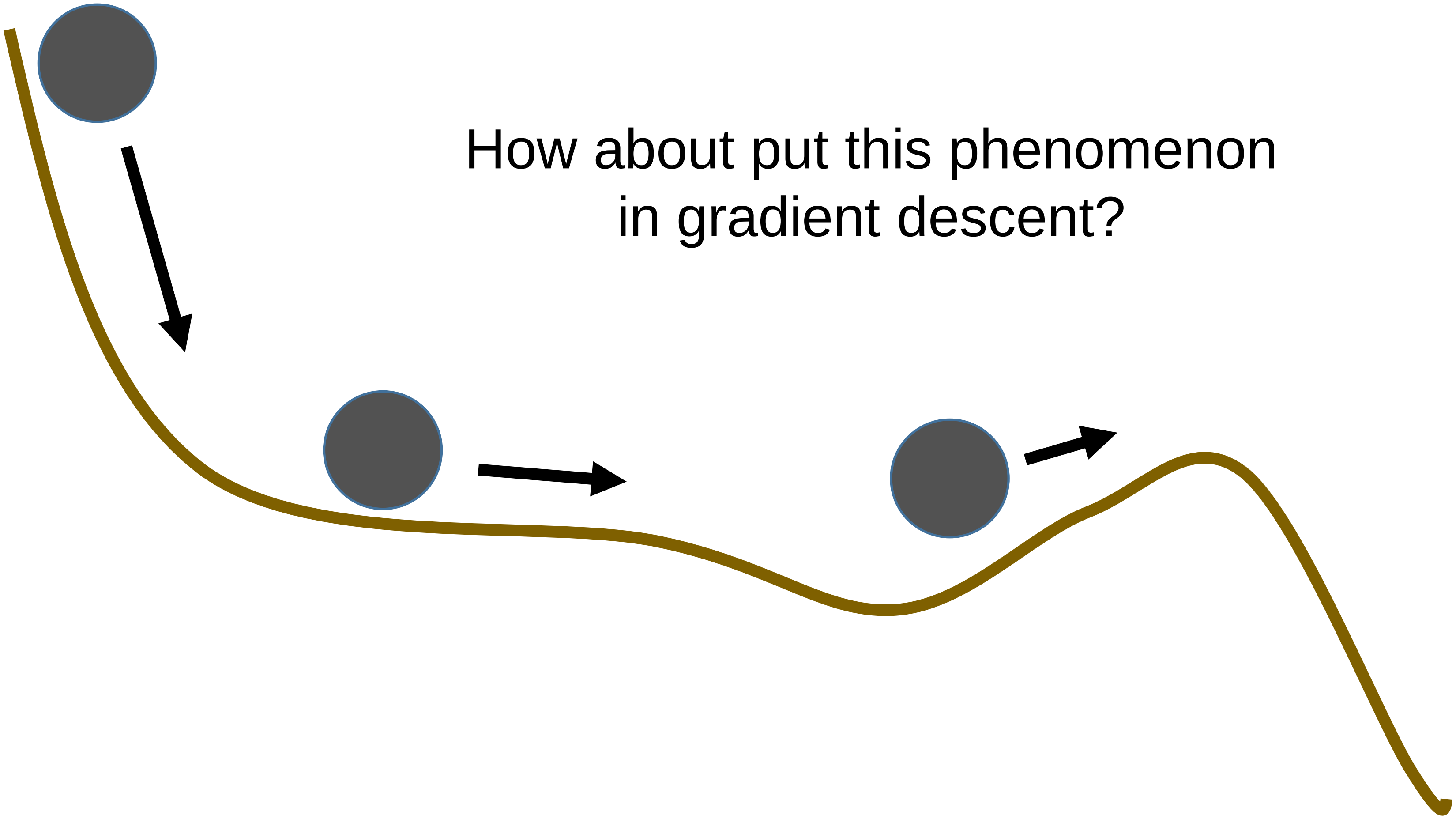
Generalized delta rule

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

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

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

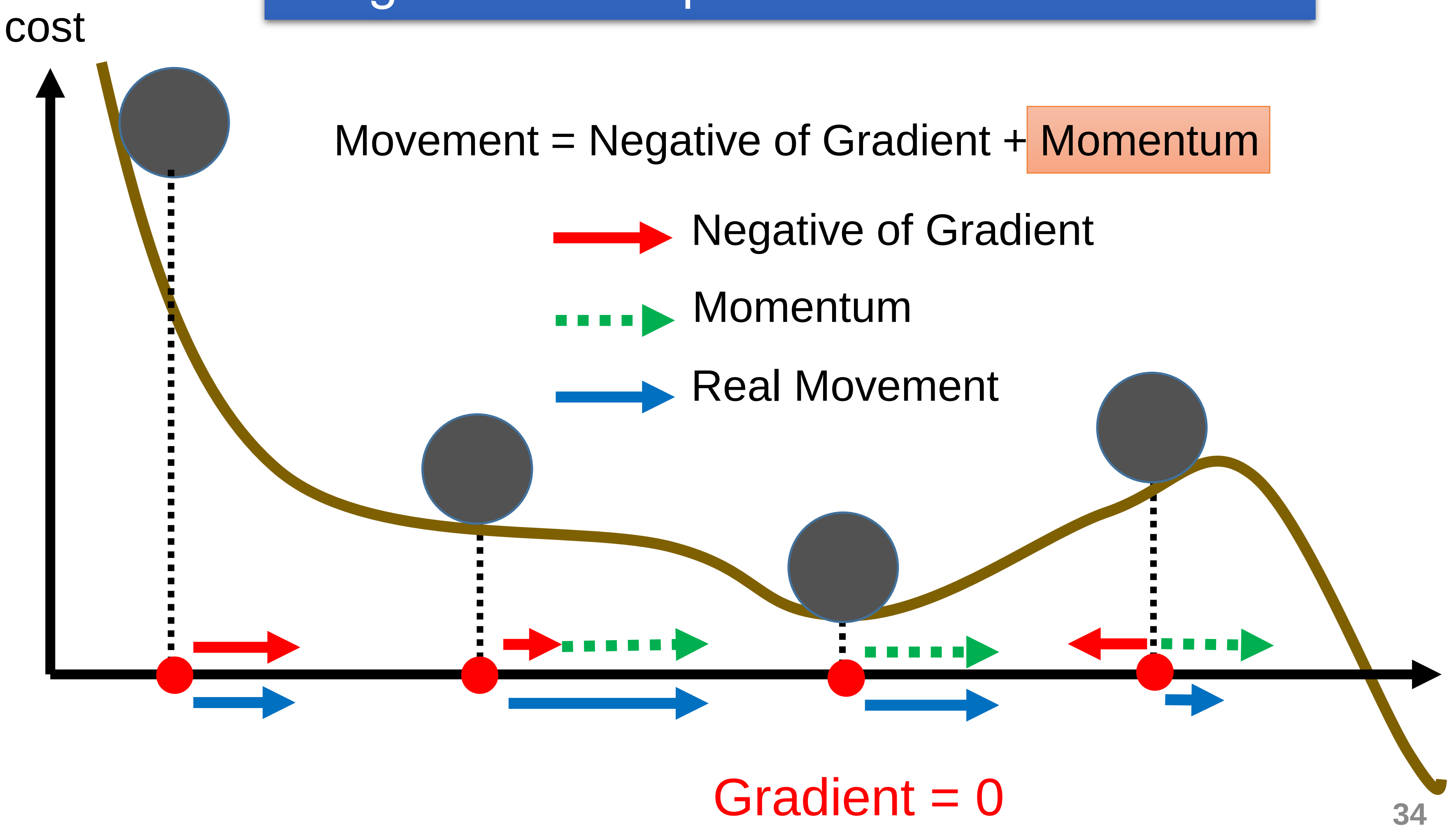
Momentum in physical world



How about put this phenomenon
in gradient descent?

Gradient descent with momentum

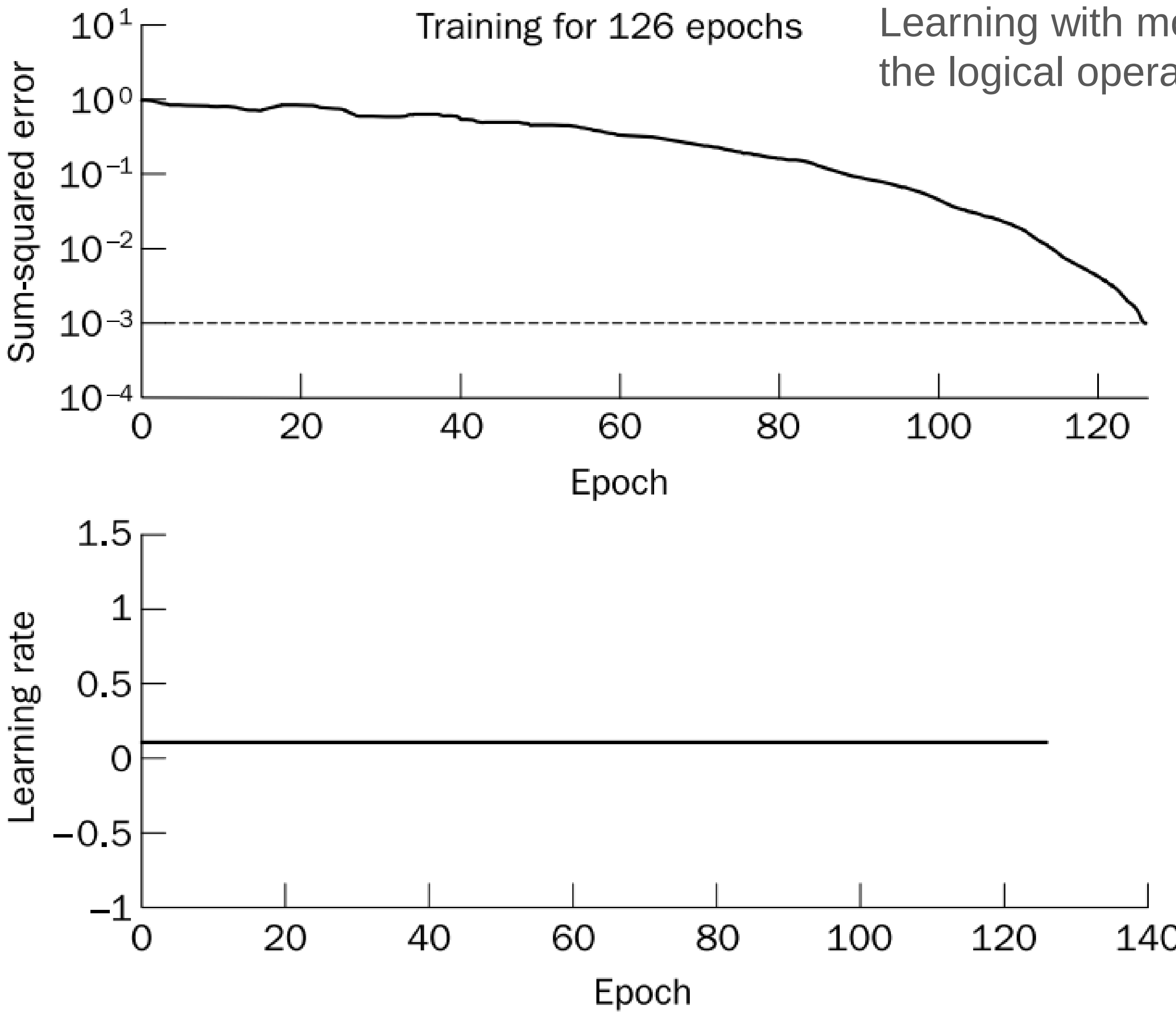
Still not guarantee reaching global minima, but give some hope ……



Adaptive learning rate

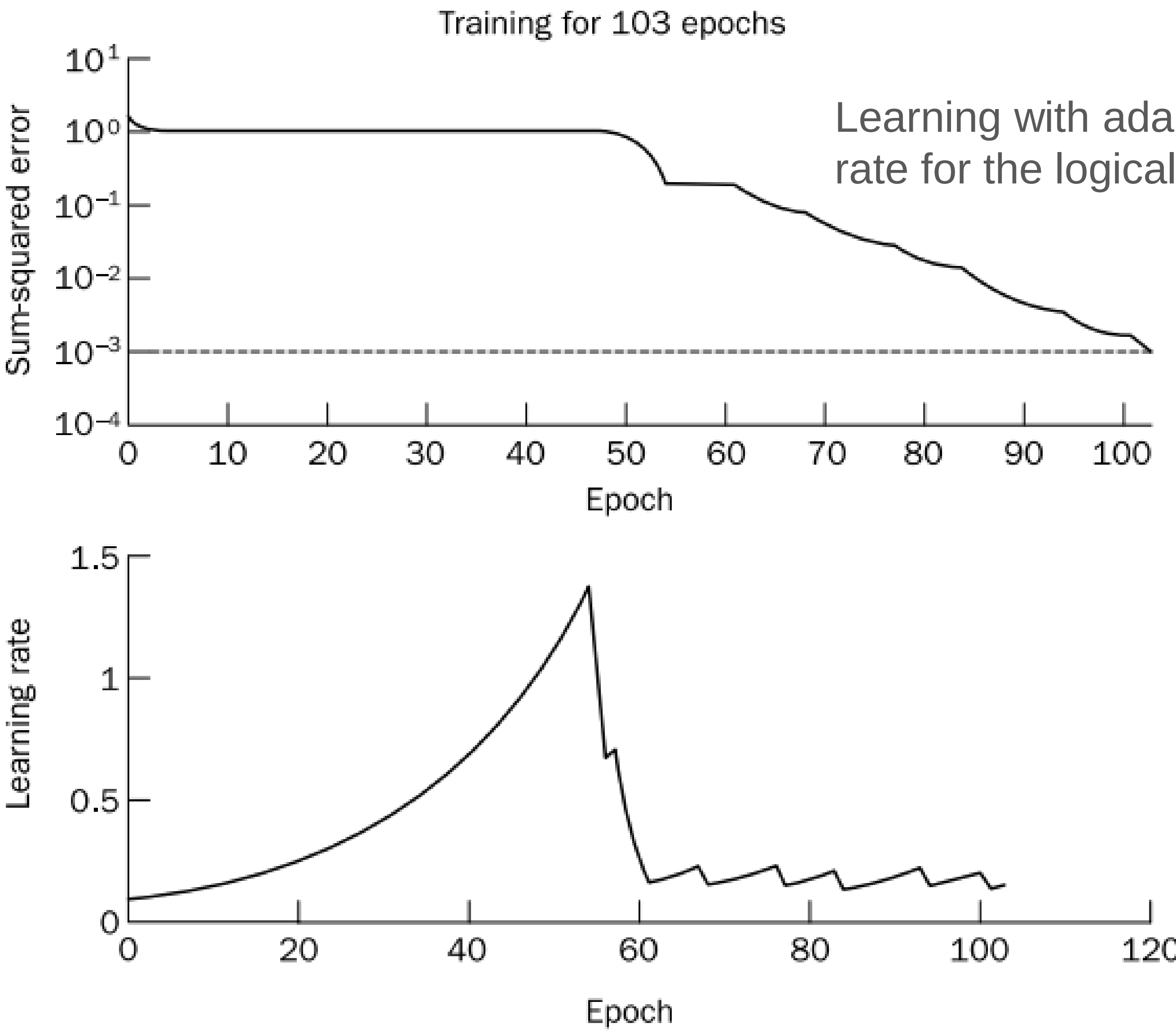
- One of the most effective acceleration means
- Adjust the learning rate parameter α during training
  - Small α $\rightarrow$ small weight changes through iterations $\rightarrow$ smooth learning curve
  - Large α $\rightarrow$ speed up the training process with larger weight changes $\rightarrow$ possible instability and oscillatory
- Heuristic-like approaches
  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

Learning with momentum only



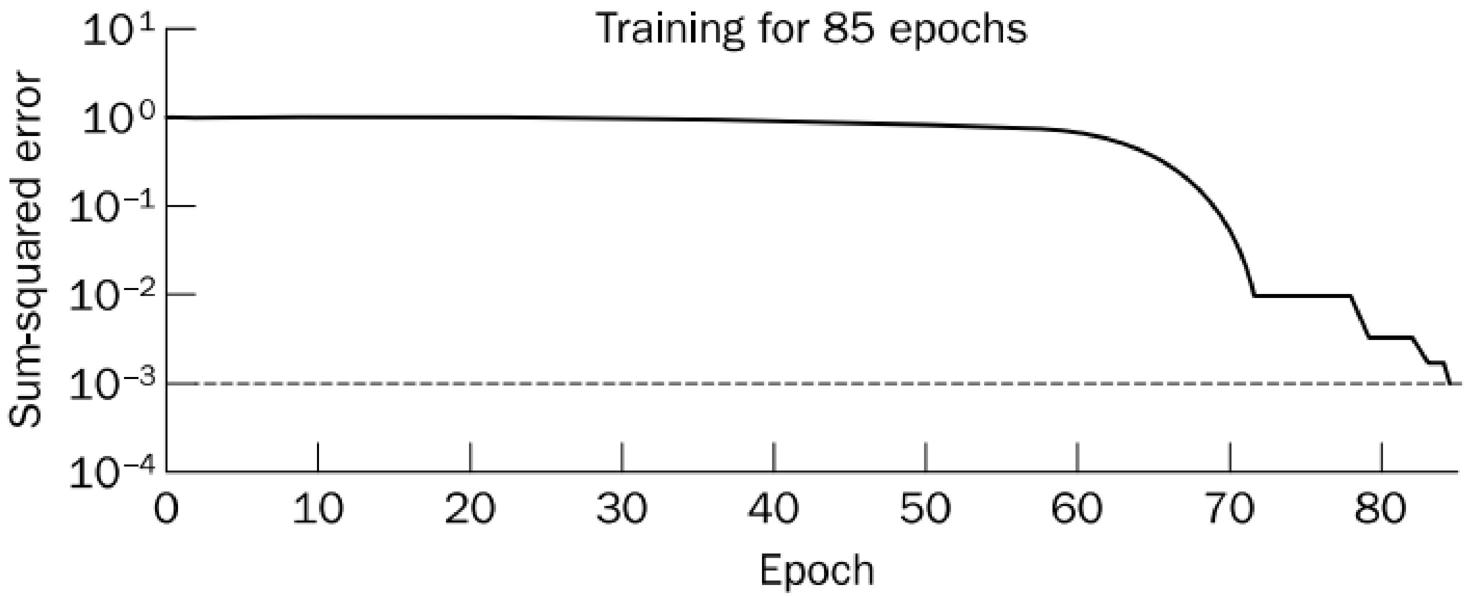
Learning with momentum for the logical operation XOR.

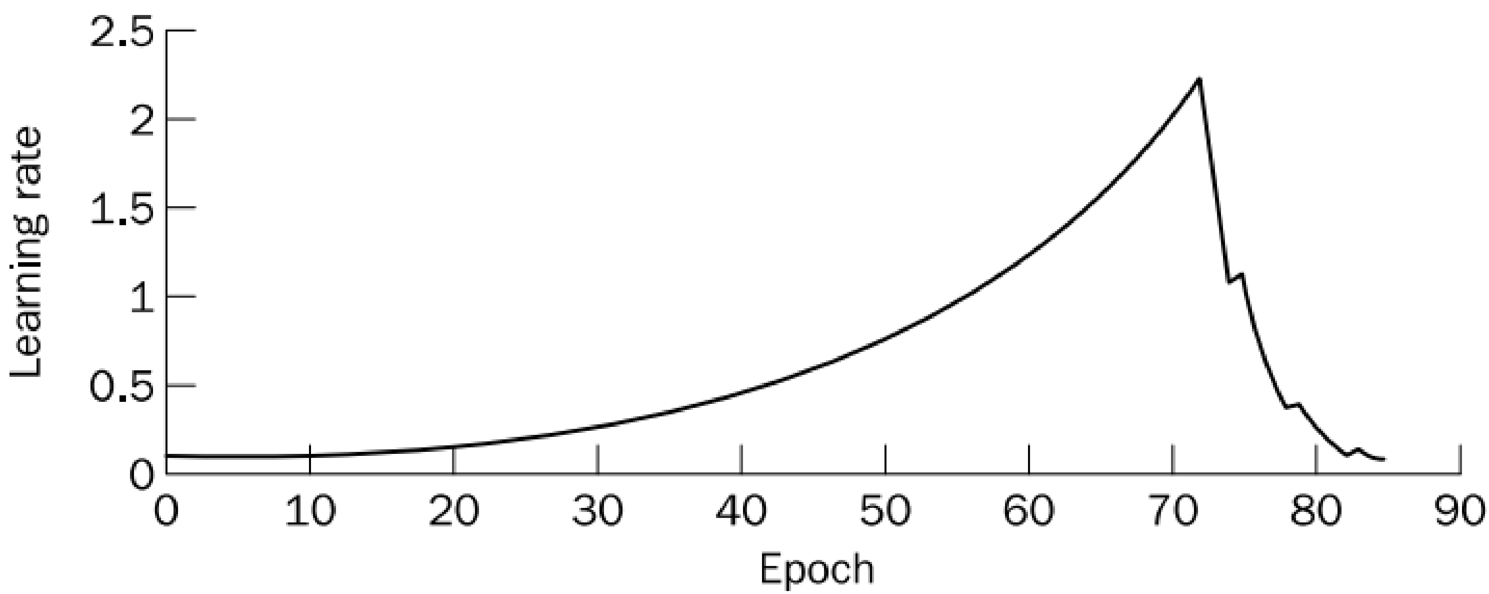
Learning with adaptive α only



Learning with adaptive learning rate for the logical operation XOR.

Learning with adaptive α and momentum






THE
END