HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Introduction to deep learning

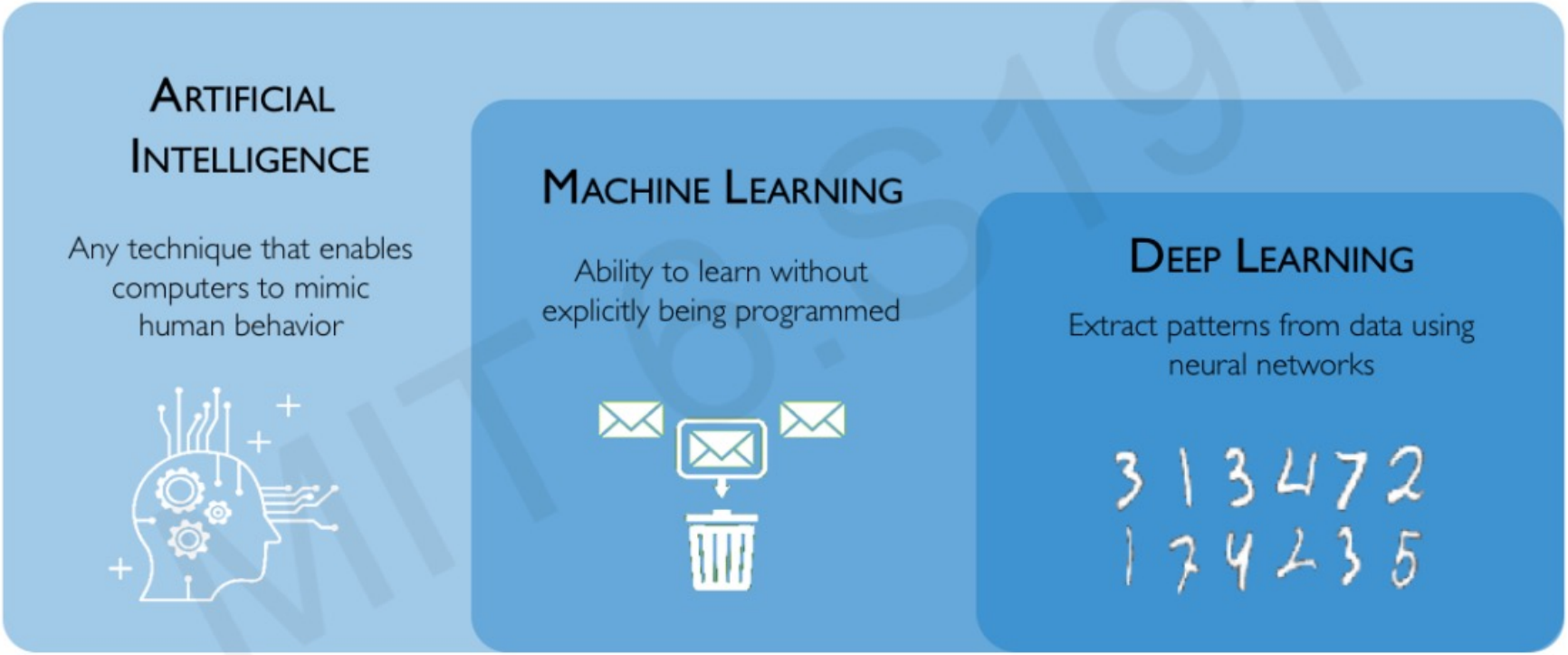
(IT3320E)

Lesson 1

Introduction

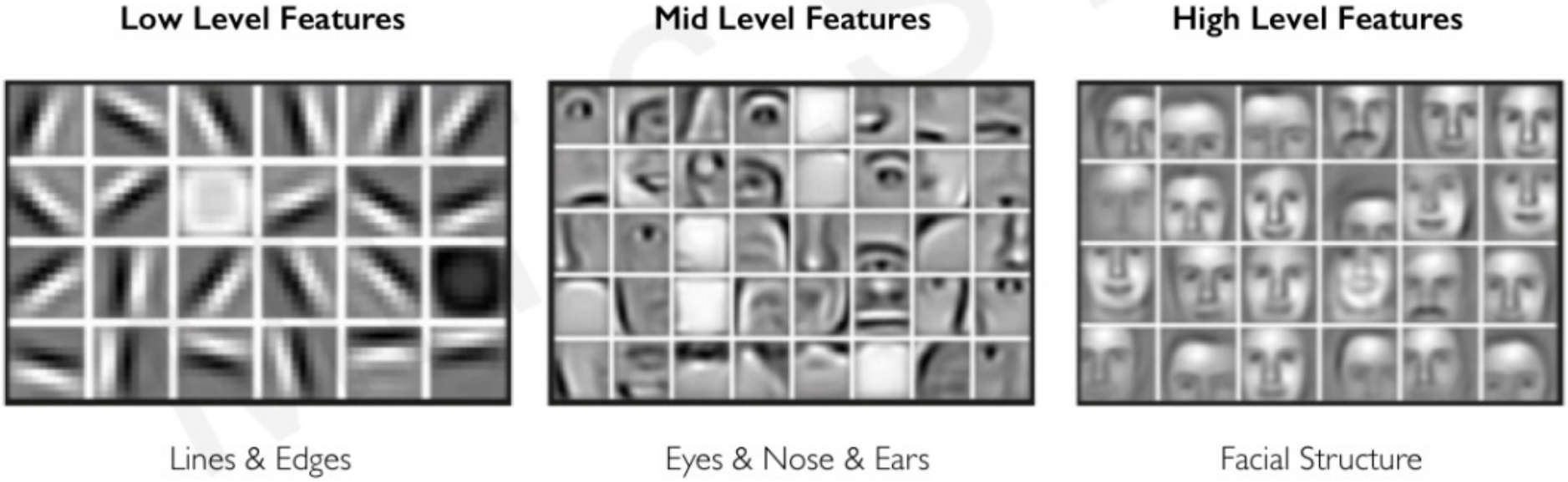
What is deep learning?

- Deep learning is a class of machine learning algorithms that uses multiple layers to progressively extract higher-level features from the raw input

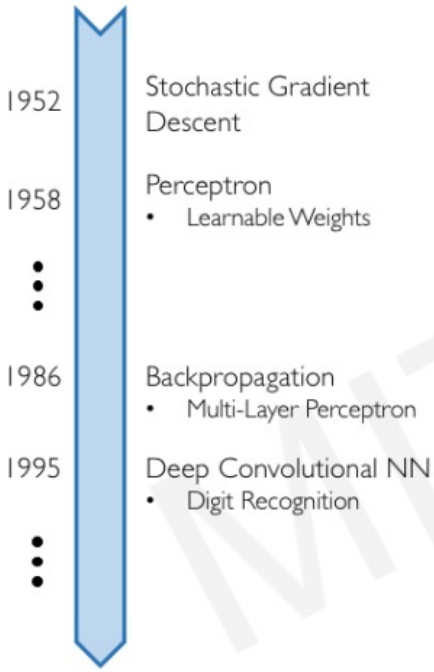


Why deep learning?

- Traditional machine learning methods require manual feature engineering, which depends on the domain expertise and experience.
- Deep learning allows learning feature representation from data



Why now?



Neural Networks date back decades, so why the resurgence?

1. Big Data

- Larger Datasets
- Easier Collection & Storage

IMAGENET

WIKIPEDIA
The Free Encyclopedia

2. Hardware

- Graphics Processing Units (GPUs)
- Massively Parallelizable

3. Software

- Improved Techniques
- New Models
- Toolboxes

TensorFlow

A few concepts on machine learning

Supervised learning

Functions $\mathcal{F}$

$f : \mathcal{X} \to \mathcal{Y}$

Training data

$\{(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}\}$

LEARNING

find $\hat{f} \in \mathcal{F}$
s.t. $y_i \approx \hat{f}(x_i)$

Learning machine

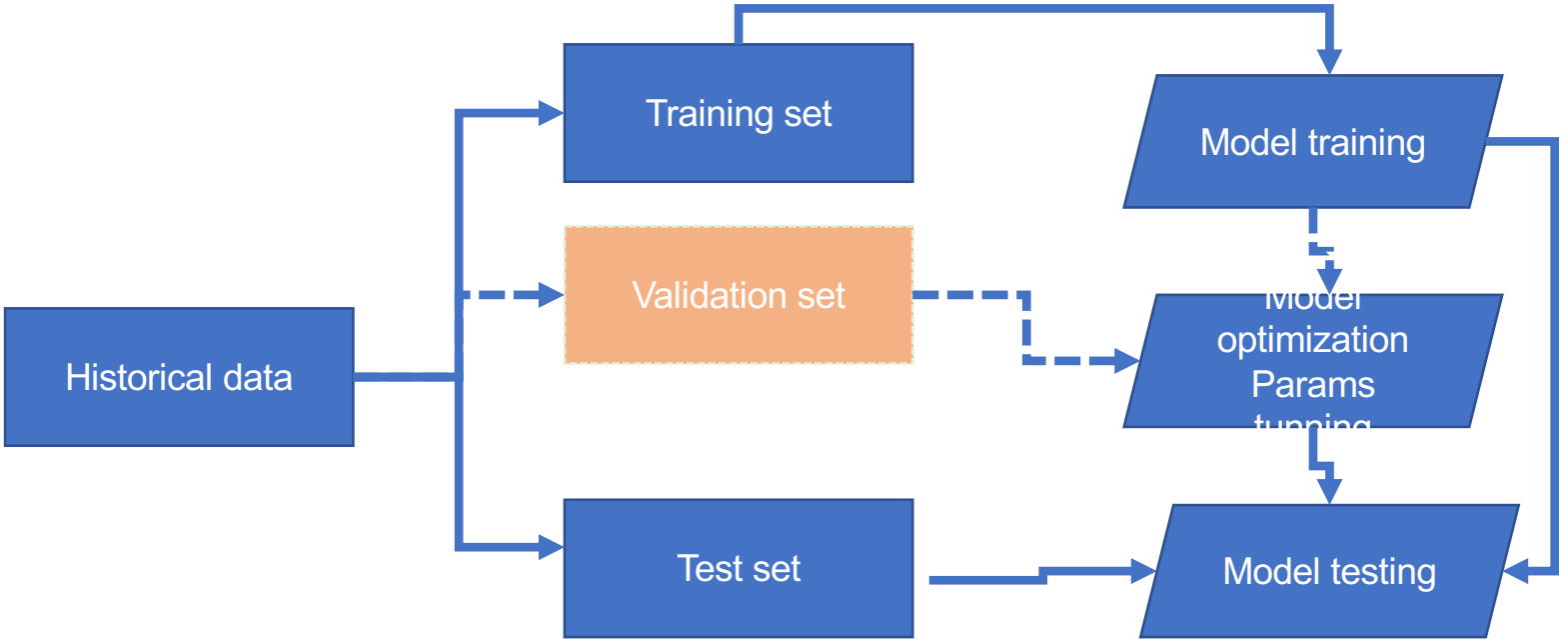
PREDICTION

$y = \hat{f}(x)$

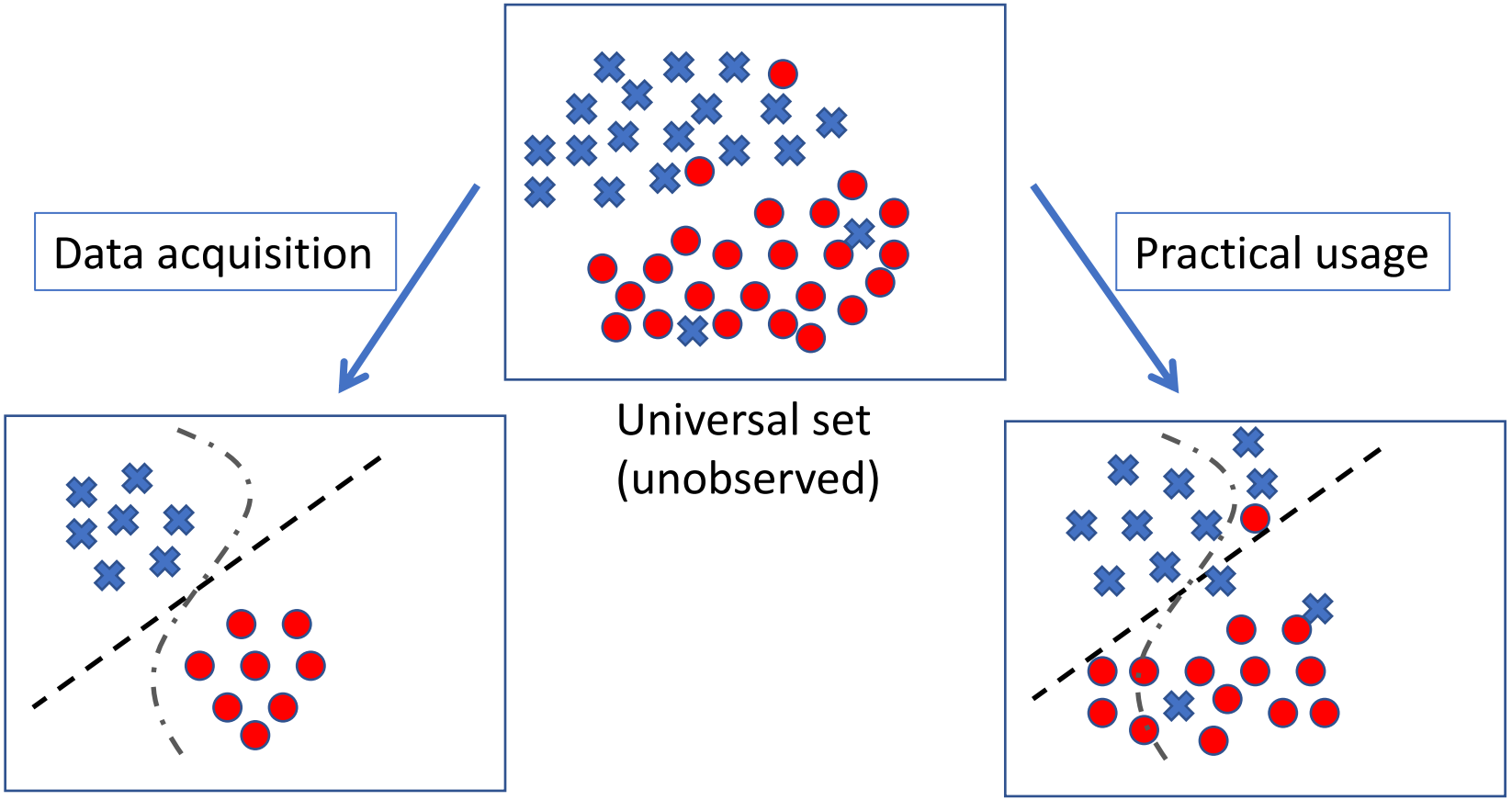
New data

x

Machine learning process

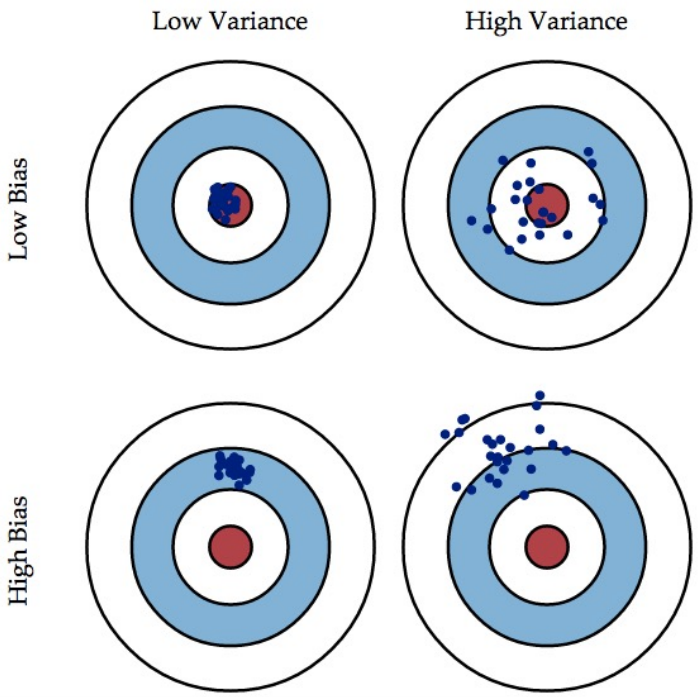


Train and test datasets



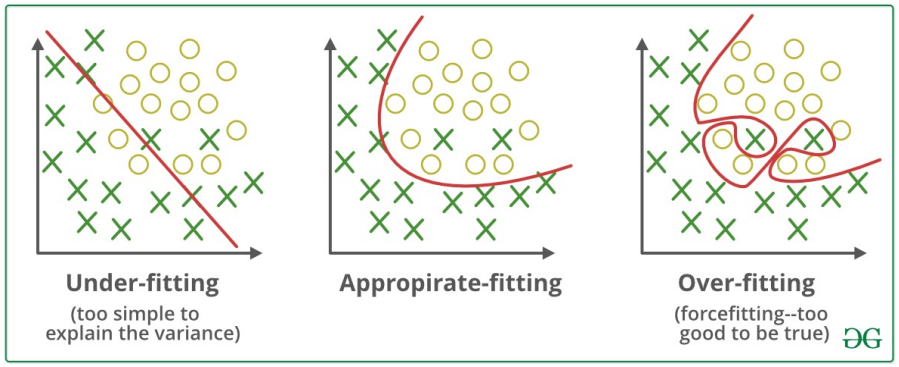
Bias and variance

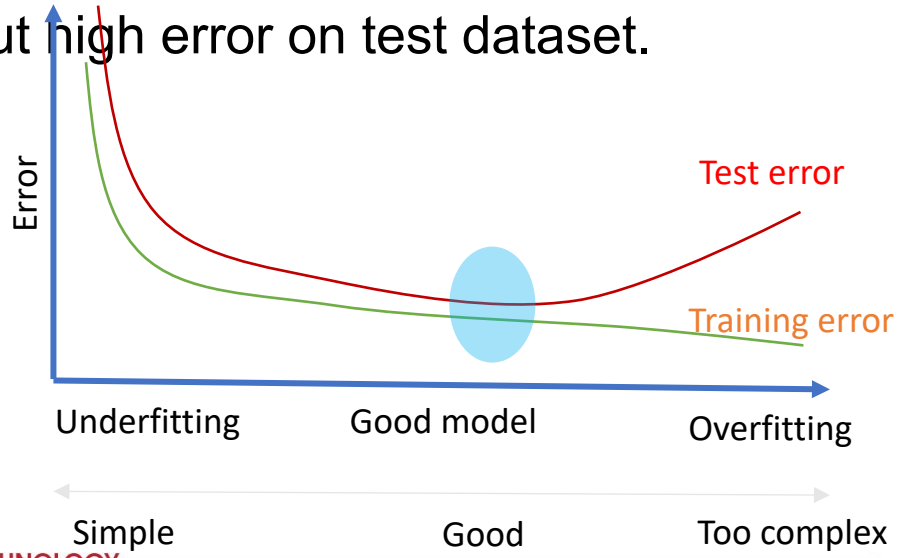
- **Bias:** The difference between the average prediction and the correct value of the data.
- **Variance:** The variability of model prediction for a given data point or a value which tells us spread of our data



Overfiting and underfiting

- **Underfitting:** the model is too simple to represent the properties of data.
  - Hight bias and low variance.
  - High error on training and test datasets.
- **Overfitting:** the model is too complex that it also learns noise in the data.
  - Low bias and high variance.
  - Low error on train dataset, but high error on test dataset.





Adapted from L. Lazebnik

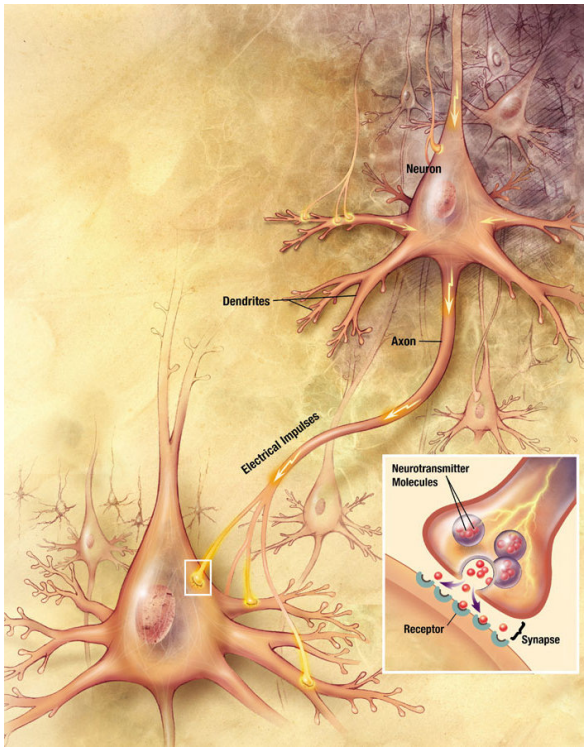
Artifacial neural networks (ANN)

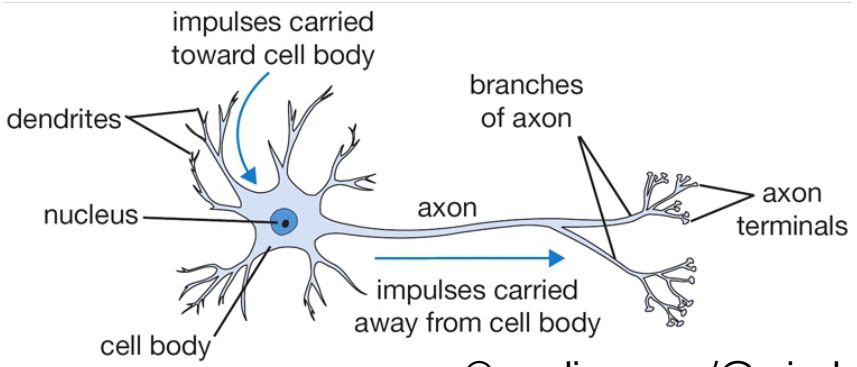
What is it?

- The human brain is made up of many interconnected neurons
- Neurons receive signals from other neurons through synapses.
- When a neuron receives enough signals (within a few ms), it fires a pulse that “fires” its signal

© made2591.github.io

Perceptron

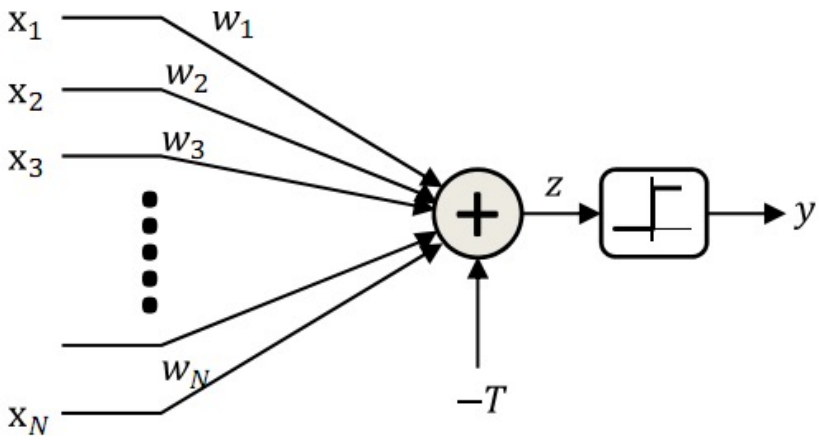




© medium.com/@rajachak127

Modeling perceptron

- Rosenblatt, Frank. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.
  - Input signals x1, x2, ...
  - Bias x0 = T
  - Net input = weighted sum = Net(w,x)
  - Fires a “fire” pulse, which activates if the weighted sum of the inputs with “bias” T is not negative



$$z = \sum_i w_i x_i - T$$

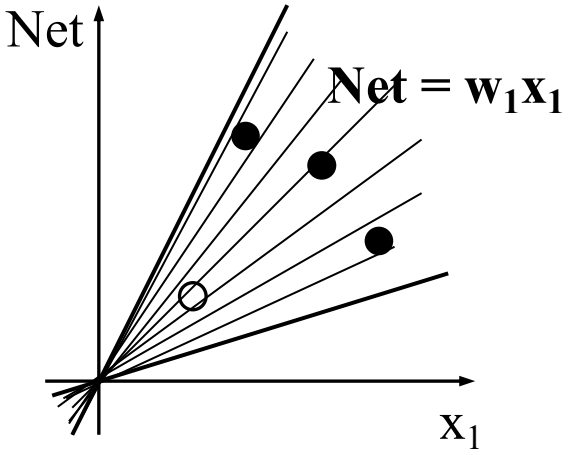
$$y = \begin{cases} 1 \; if \; z \geq 0 \\ 0 \; else \end{cases}$$

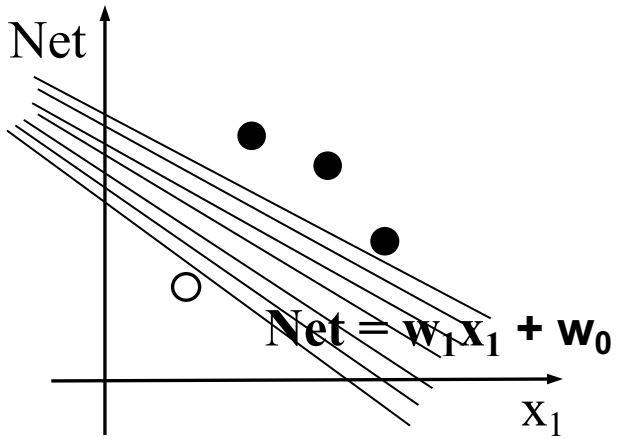
Net input and bias

- The net input is usually calculated by a linear function

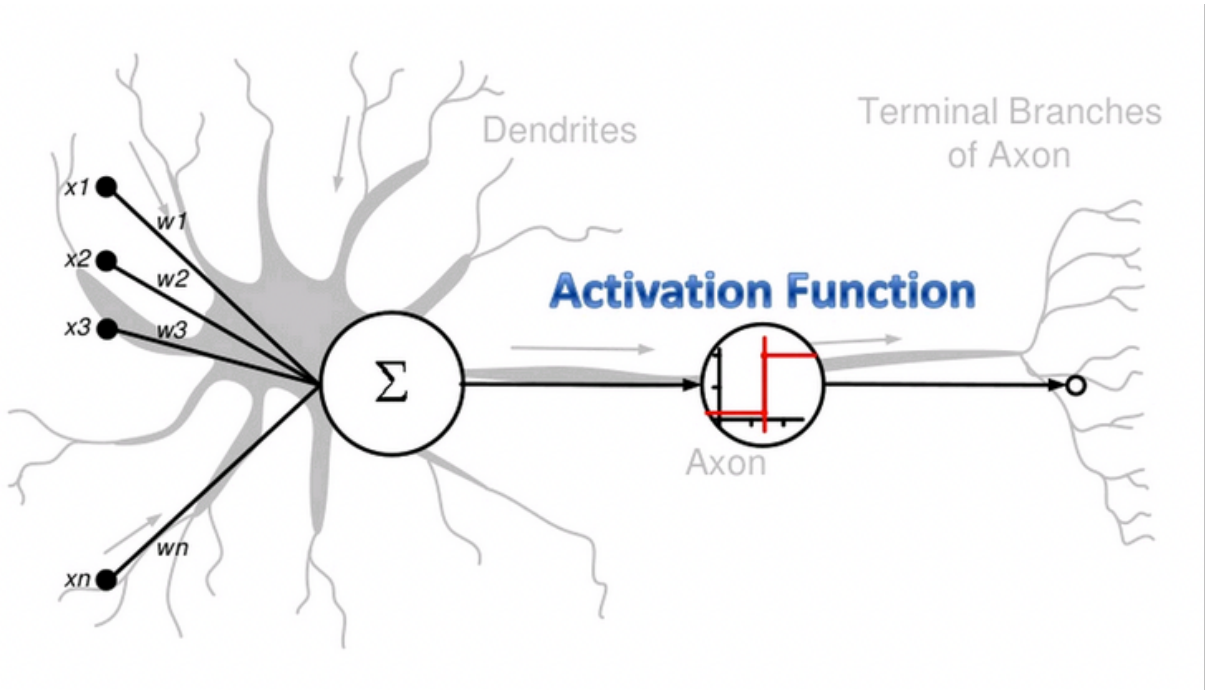
$$Net = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_m x_m = w_0 .1 + \sum_{i=1}^{m} w_i x_i = \sum_{i=0}^{m} w_i x_i$$

- The role of adjusted weight (bias) w_0
  - → The `Net=w1x1` family of functions cannot decompose examples into two classes.
  - → However: The `Net=w1x1+w0` family of functions can!





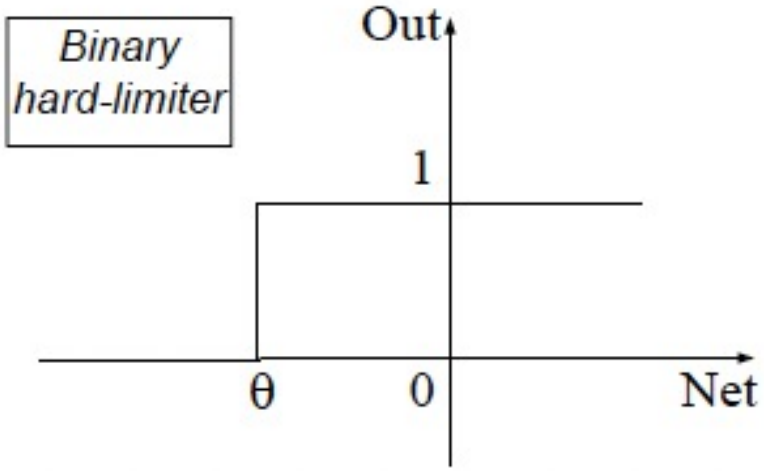
Activation function



Hard-limiter function

- Hard-limiter
  - Threshold
  - Discontinuous
  - Discontinuous derivative

$$\varphi^{\text{hlim}}(v) = \begin{cases} 1 & \text{for } v \geq 0 \\ 0 & \text{for } v < 0 \end{cases}$$

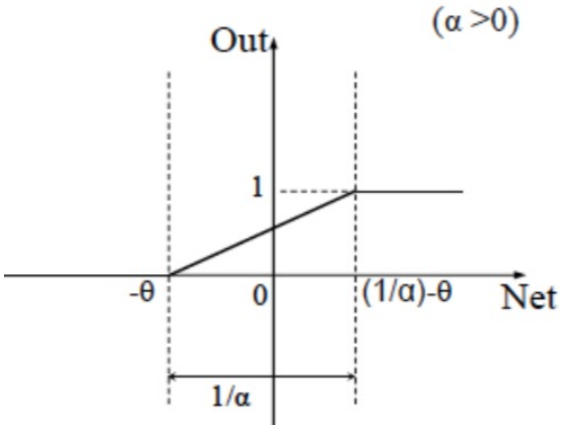


Threshold logic function

- Saturating linear
- Contiguous
- Discontinuous derivative

$$Out(Net) = tl(Net, \alpha, \theta) = \begin{cases} 0, & \text{if } Net < -\theta \\ \alpha(Net + \theta), & \text{if } -\theta \le Net \le \frac{1}{\alpha} - \theta \\ 1, & \text{if } Net > \frac{1}{\alpha} - \theta \end{cases}$$

$$= \max(0, \min(1, \alpha(Net + \theta)))$$

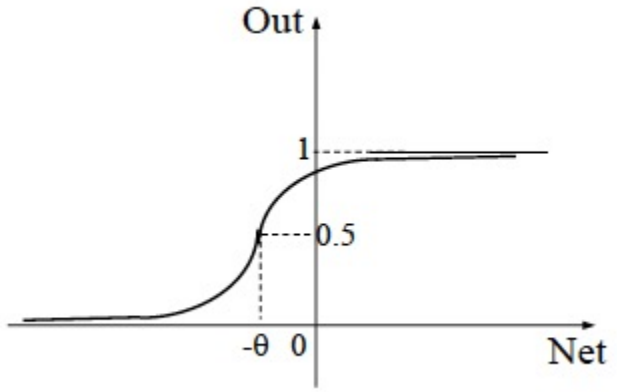


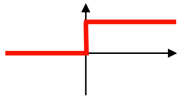
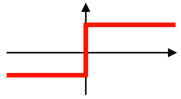
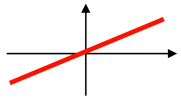
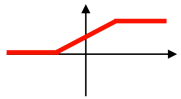
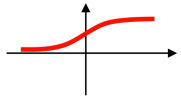
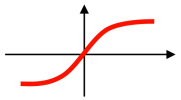
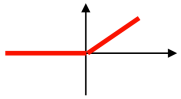
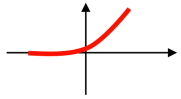
Sigmoid function

- Most popular
- Output (0,1)
- Continuous derivatives
- Easy to differentiate

$$Out(Net) = sf(Net, \alpha, \theta) = \frac{1}{1 + e^{-\alpha(Net+\theta)}}$$

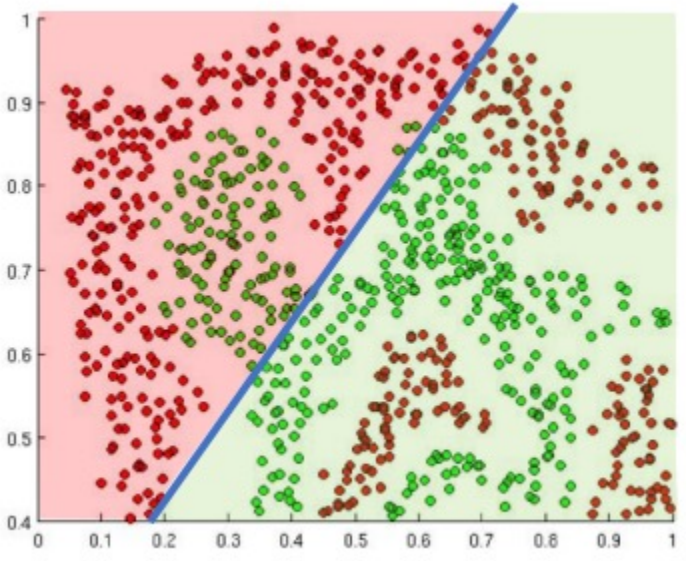
$$\frac{d\sigma(x)}{d(x)} = \sigma(x) \cdot (1 - \sigma(x))$$



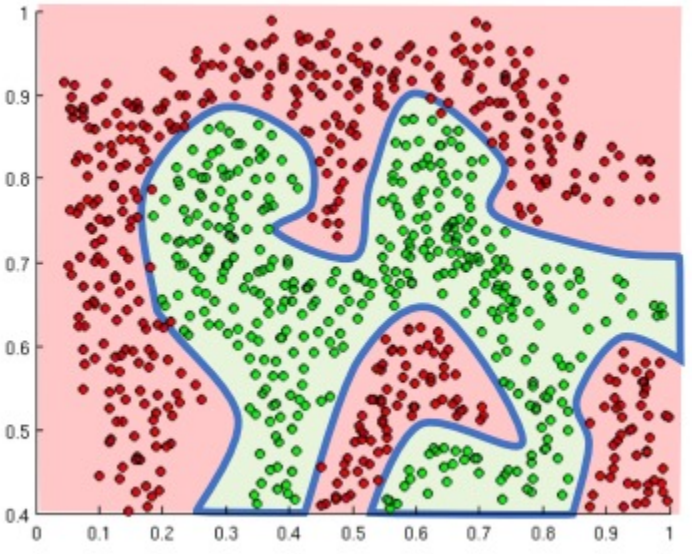
Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^{z})$	Multi-layer Neural Networks	

The importance of the activation functions

- The purpose of using the activation function is to introduce nonlinear layers into the neural network



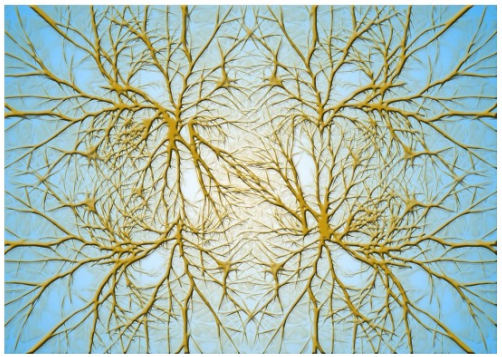
A linear activation function always produces a linear separator no matter how large the network is.

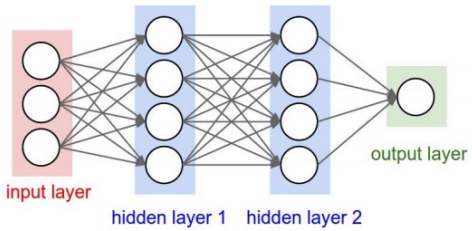


Nonlinear classes allow us to approximate complex functions.

Artificial Neural Networks and the Brains

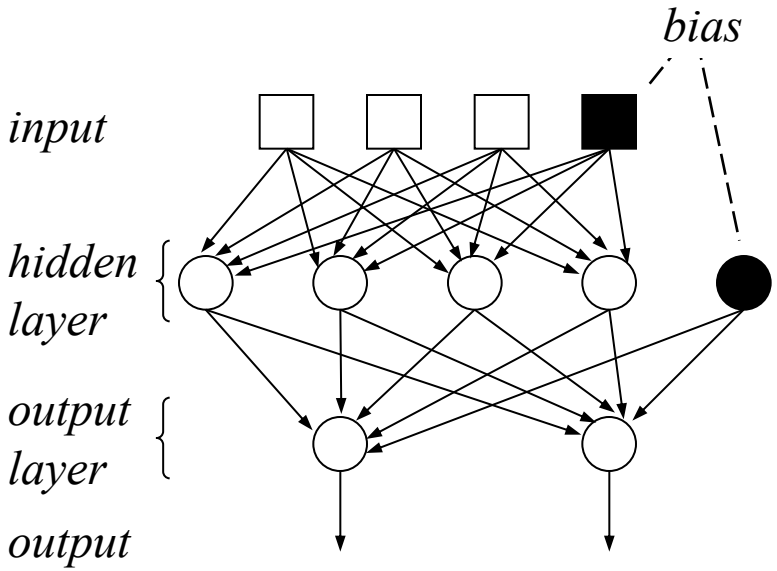
- Biological neural networks have complex connections
- Artificial Neural Network (ANN)
  - Neurons organize into layers to increase computational efficiency through parallelization.
- ANNs have the ability to learn, recall, and generalize from learning data.
- The capacity of an ANN depends on
  - The architecture (topology) of the neural network
  - Input/output characteristics of each neuron
  - Learning Algorithms (Training)
  - Learning data





ANN: network architecture (1)

- The architecture of an ANN is defined by:
  - Number of input and output signals
  - Number of layers
  - Number of neurons in each layer
  - Number of connections for each neuron
  - How neurons (in a layer, or between layers) connect with each other
- An ANN must have
  - An input layer
  - An output layer
  - No, one, or more hidden layer(s)



Example: An ANN with

1 hidden layer

Input: 3 signals

Output: 2 values

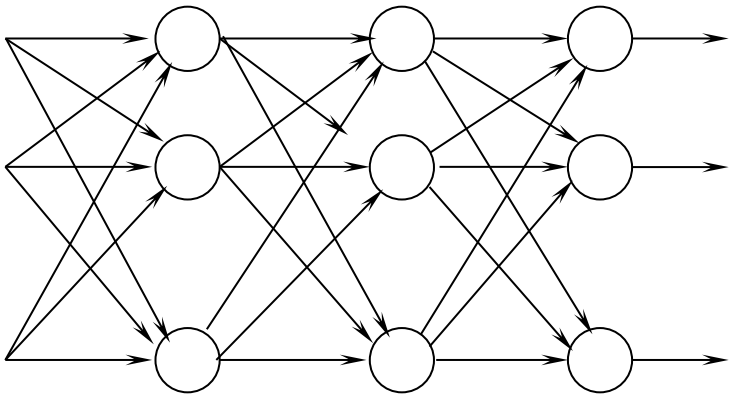
In total, there are 6 neurons

4 on the hidden layer

2 on the output layer

ANN: network architecture (2)

- A layer is a group of neurons
- A hidden layer is a layer between the input layer and the output layer.
- Hidden nodes do not interact directly with the external environment (of the neural network).
- An ANN is said to be fully connected if every output from one layer connects to every neuron of the next layer.

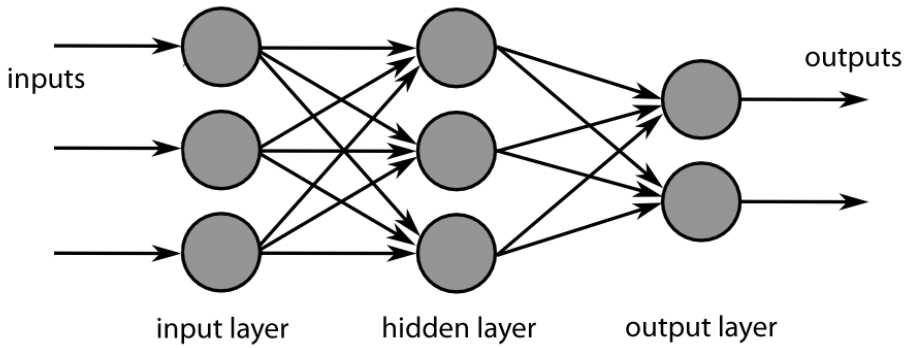


ANN: network architecture (3)

- An ANN is said to be a feed-forward network if none of the outputs of one node is the input of another node of the same (or a previous) layer.
- When a node's outputs are back-linked as inputs to a node in the same (or a previous) layer, it is a feedback network.
  - If the feedback is the input link for nodes in the same layer, then it is lateral feedback.
- Feedback networks with closed loops are called recurrent networks.

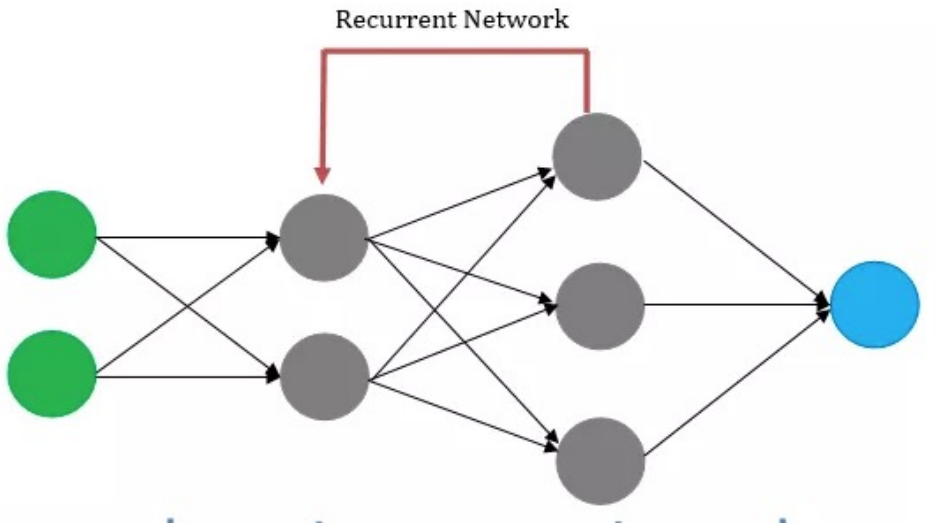
Feed-forward neural network

- The connections between the units do not form a directed cycle



Recurrent neural network

- The connections between units may form a directed cycle



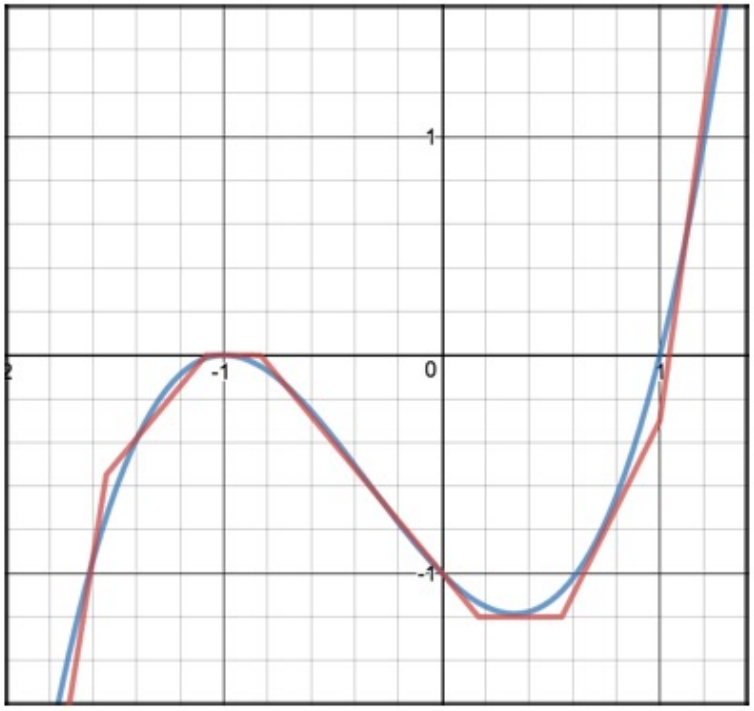
© learning.maxtech4u.com

Why multi-layer network?

- A neural network can represent any function if it is wide enough (the number of neurons in a layer is large enough) and deep enough (the number of layers is large enough).
  - If you want to reduce the depth of the network in many cases, you will have to compensate by increasing the width to a power of times!
  - A hidden single-layer neural network may need exponentially more neurons than a multi-layer network
- Multilayer networks need far fewer neurons than shallow networks to represent the same function.
  - Multilayer networks are more valuables

Universal approximation theorem

Any bounded continuous function can be learned (approximately) by an ANN using one hidden layer [Cybenko, 1989; Hornik et al., 1991]

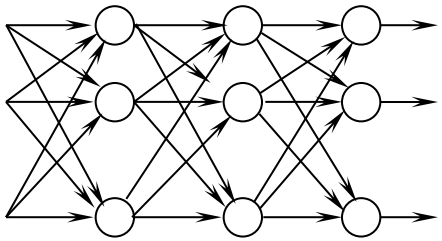


$$n_1(x) = Relu(-5x - 7.7)$$
$$n_2(x) = Relu(-1.2x - 1.3)$$
$$n_3(x) = Relu(1.2x + 1)$$
$$n_4(x) = Relu(1.2x - .2)$$
$$n_5(x) = Relu(2x - 1.1)$$
$$n_6(x) = Relu(5x - 5)$$

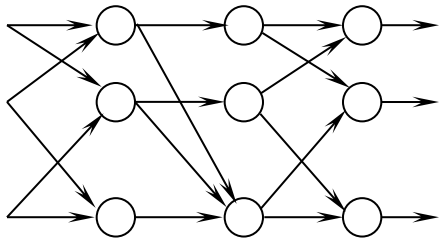
$$Z(x) = -n_1(x) - n_2(x) - n_3(x) + n_4(x) + n_5(x) + n_6(x)$$

Neural network learning

- 2 types of learning in artificial neural networks
  - Parameter learning
    - The goal is to adaptively change the weights of the connections in the neural network
  - Structure learning
    - The goal is to adaptively change the network structure, including the number of neurons and the types of connections between them



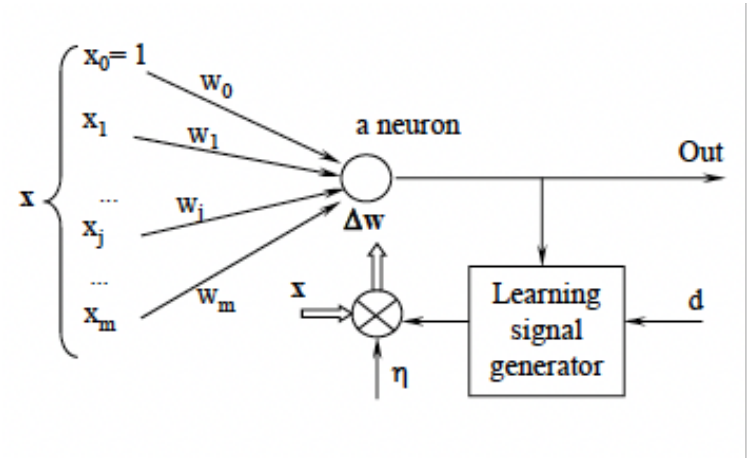
Or

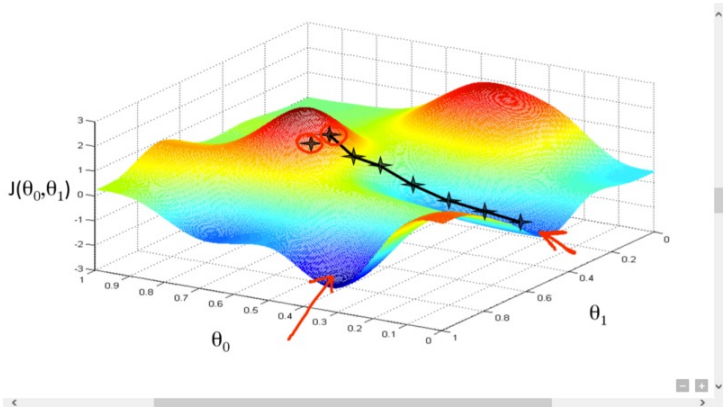


- These two types of learning can be done simultaneously or separately
- In most cases, we will only consider parameter learning

Neural network learning principle

- Define errror (loss) function
- Goal: find connection weights to minimize errors
- One of learning methods:
  - Gradient descent: back propagation of the errors





© hackernoon.com/gradient-descent-aynk-7cbe95a778da

Example of loss function

- Consider an ANN with n output neurons
- For a learning example (x,d), the value of the **training error** caused by the (current) weight vector w:

$$E_{\mathbf{x}}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}\left(d_i - Out_i\right)^2$$

- The learning error caused by the (current) weight vector w for the entire learning set D:

$$E_D(\mathbf{w}) = \frac{1}{|D|}\sum_{\mathbf{x}\in D} E_{\mathbf{x}}(\mathbf{w})$$

Minimize Errors with Gradient

- The gradient of E (denoted ∇E) is a vector

$$\nabla E(\mathbf{w}) = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_N} \right)$$

  - where N is the total number of weights (links) in the network

- The gradient ∇E determines the direction that causes the fastest increase in the error value E

- Therefore, the direction that causes the fastest decrease is the direction opposite to the gradient of E .

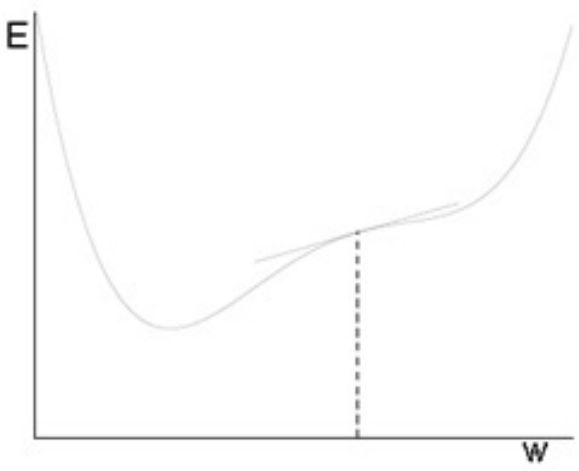
$$\Delta \boldsymbol{w} = -\eta \,.\, \nabla \mathrm{E}\,(\boldsymbol{w})\,; \qquad \Delta w_i = -\eta \frac{\partial E}{\partial w_i}, \quad \forall i = 1..N$$

- Requirement: The action functions used in the network must have continuous derivatives

Example of gradient descent

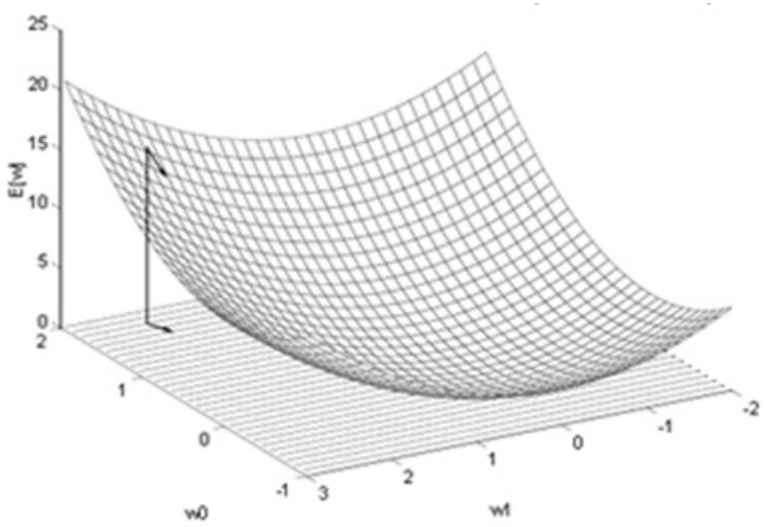
Không gian một chiều

E(w)



Không gian 2 chiều

$E(w_1,w_2)$



Back propagation algorithm

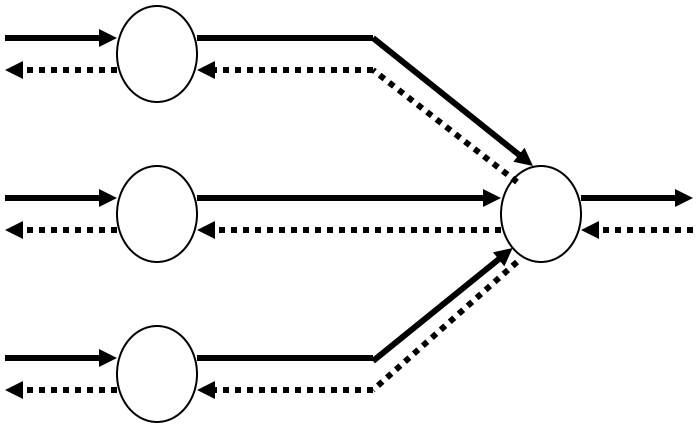
Multi-layer ANN and back-propagation algorithm

- A multi-layer neural network (NN) learned by the Back Propagation (BP) algorithm can represent a highly non-linear separation function.
- The BP learning algorithm is used to learn the weights of a multi-layer neural network
  - Fixed network structure (neurons and links between them are fixed)
  - For each neuron, the action function must have a continuous derivative
- The BP algorithm applies the gradient descent strategy in the weights update rule
  - To minimize the error (difference) between the actual output values and the desired output values, for the learning examples

Back-propagation (BP) learning algorithm (1)

- The back-propagation learning algorithm searches for a vector of weights that minimizes the overall system error for the training set
- BP algorithm consists of 2 stages
  - Signal forward stage
    - The input signals (vectors of input values) are propagated forward from the input layer to the output layer (passing through the hidden layers).
  - Backward propagation of error stage
    - Based on the desired output value of the input vector, the system calculates the error value
    - Starting from the output layer, the error value is **propagated back** through the network, from layer to layer (backward), until the input layer.
    - Error back-propagation is done by calculating (recursively) the **local gradient of each neuron**.

Back-propagation (BP) learning algorithm (2)



Signal propagation stage:

- Activate (transmit signal over) the network

Error back-propagation stage:

- Output error calculation
- Propagate (backward) error

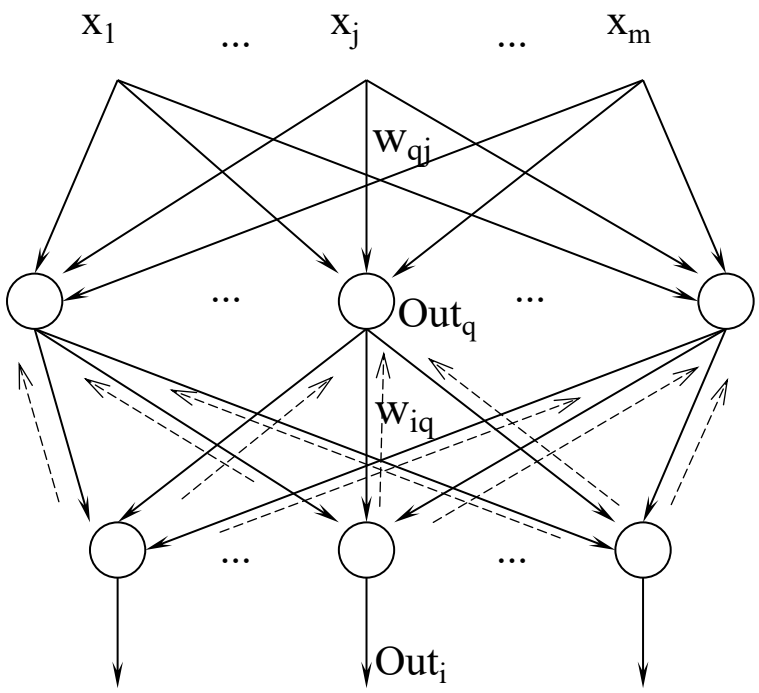
BP: a network architecture example

- Given a 3-layers network depicted in the figure
- m number of input signals x_j (j=1..m)
- l number of neurons in the hidden layer z_q (q=1..l)
- n number of neurons in the ouput layeer y_i (i=1..n)
- w_{qj} is the weight of the connection from the input signal x_j to the neuron in the hidden layer z_q
- w_{iq} is the weight of the connection from the neuron in the hidden layer z_q to the neuron in the output layer y_i
- Out_q is the output (local) of the neuron in the hidden layer z_q
- Out_i is the output of the network at the neuron of the output layer y_i

Input x_j (j=1..m)

Hidden neuron z_q (q=1..l)

Output neuron y_i (i=1..n)



BP Algorithm: Forward Propagation (1)

- For each training sample ***x***
  - Input vector ***x*** is propagate from the input to the output layer
  - The network will produce an actual output value ***Out*** (that is a vector of Out_i, i=1..n)

- For an input vector ***x***, a neuron z_q in the hidden layer receives the overall net input:

$$Net_q = \sum_{j=1}^{m} w_{qj} x_j$$

…and produces an output (locally) that is equal to:

$$Out_q = f(Net_q) = f\left(\sum_{j=1}^{m} w_{qj} x_j\right)$$

where f(.) is the activation function of the neuron z_q

BP Algorithm: Forward Propagation (2)

- The overall net input of the neuron y_i in the output layer

$$Net_i = \sum_{q=1}^{l} w_{iq} Out_q = \sum_{q=1}^{l} w_{iq} f\left(\sum_{j=1}^{m} w_{qj} x_j \right)$$

- The neuron y_i produces an output

$$Out_i = f(Net_i) = f\left(\sum_{q=1}^{l} w_{iq} Out_q \right) = f\left(\sum_{q=1}^{l} w_{iq} f\left(\sum_{j=1}^{m} w_{qj} x_j \right) \right)$$

- The vector of Out_i (i=1..n) is the actual output value of the network, given the input vector $\boldsymbol{x}$

BP algorithm: Back propagation (1)

- For a training sample ***x***
  - The error signals due to the difference between the expected output value **d** and the actual output value ***Out*** are calculated
  - These error signals are back-propagated from the output layer to the input layer, to update the weights.
- To consider error signals and their backpropagation, it is necessary to define an error function

$$E(w) = \frac{1}{2}\sum_{i=1}^{n}(d_i - Out_i)^2 = \frac{1}{2}\sum_{i=1}^{n}[d_i - f(Net_i)]^2$$

$$= \frac{1}{2}\sum_{i=1}^{n}\left[d_i - f\left(\sum_{q=1}^{l} w_{iq} Out_q\right)\right]^2$$

BP algorithm: Back propagation (2)

- According to the gradient-descent method, the weights of the links from the hidden layer to the output layer are updated by

$$\Delta w_{iq} = -\eta \frac{\partial E}{\partial w_{iq}}$$

- Using the derivative chain rule for $\partial E/\partial w_{iq}$,

$$\Delta w_{iq} = -\eta \left[\frac{\partial E}{\partial Out_i}\right]\left[\frac{\partial Out_i}{\partial Net_i}\right]\left[\frac{\partial Net_i}{\partial w_{iq}}\right] = \eta\left[d_i - Out_i\right]\left[f'\left(Net_i\right)\right]\left[Out_q\right] = \eta\delta_i Out_q$$

(Note: the "–" sign was included in $\partial E/\partial Out_i$)

- δ_i is the **error signal** of the neuron y_i in the output layer

$$\delta_i = -\frac{\partial E}{\partial Net_i} = -\left[\frac{\partial E}{\partial Out_i}\right]\left[\frac{\partial Out_i}{\partial Net_i}\right] = \left[d_i - Out_i\right]\left[f'\left(Net_i\right)\right]$$

where Net_i is the net input of the neuron y_i in the output layer, and $f'(\text{Net}_i)=\partial f(\text{Net}_i)/\partial \text{Net}_i$

BP algorithm: Back propagation (3)

- To update the weights of the links from the input layer to the hidden layer, we also apply the gradient-descent method and the derivative chain rule.

$$\Delta w_{qj} = -\eta \frac{\partial E}{\partial w_{qj}} = -\eta \left[\frac{\partial E}{\partial Out_q} \right] \left[\frac{\partial Out_q}{\partial Net_q} \right] \left[\frac{\partial Net_q}{\partial w_{qj}} \right]$$

- From the formula for calculating the error function $E(\mathbf{w})$, we see that each error component $(d_i - y_i)$ $(i=1..n)$ is a function of Out_q

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{n} \left[d_i - f\left(\sum_{q=1}^{l} w_{iq} Out_q \right) \right]^2$$

BP algorithm: Back propagation (4)

- Applying the derivatives chain rule, we have

$$\Delta w_{qj} = \eta \sum_{i=1}^{n} \left[\left(d_i - Out_i \right) f'\left(Net_i \right) w_{iq} \right] f'\left(Net_q \right) x_j$$

$$= \eta \sum_{i=1}^{n} \left[\delta_i w_{iq} \right] f'\left(Net_q \right) x_j = \eta \delta_q x_j$$

- δ_q is **the error signal** of the neuron z_q of the hidden layer

$$\delta_q = -\frac{\partial E}{\partial Net_q} = -\left[\frac{\partial E}{\partial Out_q} \right] \left[\frac{\partial Out_q}{\partial Net_q} \right] = f'\left(Net_q \right) \sum_{i=1}^{n} \delta_i w_{iq}$$

where Net_q is the net input of the neuron z_q of the hidden layer, and $f'(\text{Net}_q)=\partial f(\text{Net}_q)/\partial \text{Net}_q$

BP algorithm: Back propagation (5)

- According to the above formulas for calculating the error signals δ_i and δ_q, the error signal of a neuron in the hidden layer is different from the error signal of a neuron in the output layer.
- Because of this difference, the weight update procedure in the BP algorithm is also known as the generalized delta learning rule
- The error signal δ_q of the neuron z_q of the hidden layer is determined by
  - The error signal δ_i of the neuron y_i of hte output layer (that the neuron z_q connected to), and
  - The coefficients are the weights w_{iq}

BP algorithm: Back propagation (6)

- The process of calculating error signals as above can be easily extended (generalized) to neural networks with more than 1 hidden layer.
- The general form of the weight update rule in the BP algorithm is:

$$\Delta w_{ab} = \eta \delta_a x_b$$

  - ***b*** and ***a*** is the 2 indices corresponding to 2 end of the link ($b \rightarrow a$) (from a neuron (or input signal) ***b*** to neuron ***a***)
  - x_b is the output value of the hidden layer neuron (or input signal) ***b***
  - δ_a is the error signal of the neuron ***a***

Back_propagation_incremental(D, η)

The network consits of Q layers, $q = 1,2,...,Q$

${}^{q}Net_i$ and ${}^{q}Out_i$ are the net input and the output of the neuron i of the layer q

m is the number of inputs and n is the number of outputs of the network

${}^{q}w_{ij}$ is the weigh of the link from the neuron j of the layer $(q-1)$ to the neuron i of the layer q

Stage 0 (initialization)

Select the $E_{threshold}$ (the acceptable error value)

Initialize the weights with small random values

Set E=0

Stage 1 (Start a learning epoch)

Apply the input vector of the learning example k to the input layer (q=1)

${}^{q}Out_i = {}^{1}Out_i = x_i^{(k)}, \forall i$

Stage 2 (Forward propagation)

Propagate the input signals through the network, until the network output values are received (at the output layer ${}^{Q}Out_i$

$$ {}^{q}Out_i = f\left({}^{q}Net_i\right) = f\left(\sum_j {}^{q}w_{ij}\,{}^{q-1}Out_j\right) $$

Stage 3 (output error calculation)

Calculate the output error of the network and the error signal ${}^{Q}\delta_i$ of each neuron in the output layer

$$E = E + \frac{1}{2}\sum_{i=1}^{n}(d_i^{(k)} - {}^{Q}Out_i)^2$$

$${}^{Q}\delta_i = (d_i^{(k)} - {}^{Q}Out_i) f'({}^{Q}Net_i)$$

Stage 4 (Error backpropagation)

Error backpropagation to update the weights and compute the error signals ${}^{q-1}\delta_i$ for the prior layers

$$\Delta^{q}w_{ij} = \eta.({}^{q}\delta_i).({}^{q-1}Out_j); \qquad {}^{q}w_{ij} = {}^{q}w_{ij} + \Delta^{q}w_{ij}$$

$${}^{q-1}\delta_i = f'({}^{q-1}Net_i)\sum_j {}^{q}w_{ji}\,{}^{q}\delta_j; \quad \text{for all } q = Q, Q-1, \ldots, 2$$

Stage 5 (Check the end of a learning cycle - epoch)

Check if the entire learning set has been used (one learning cycle done – one epoch)

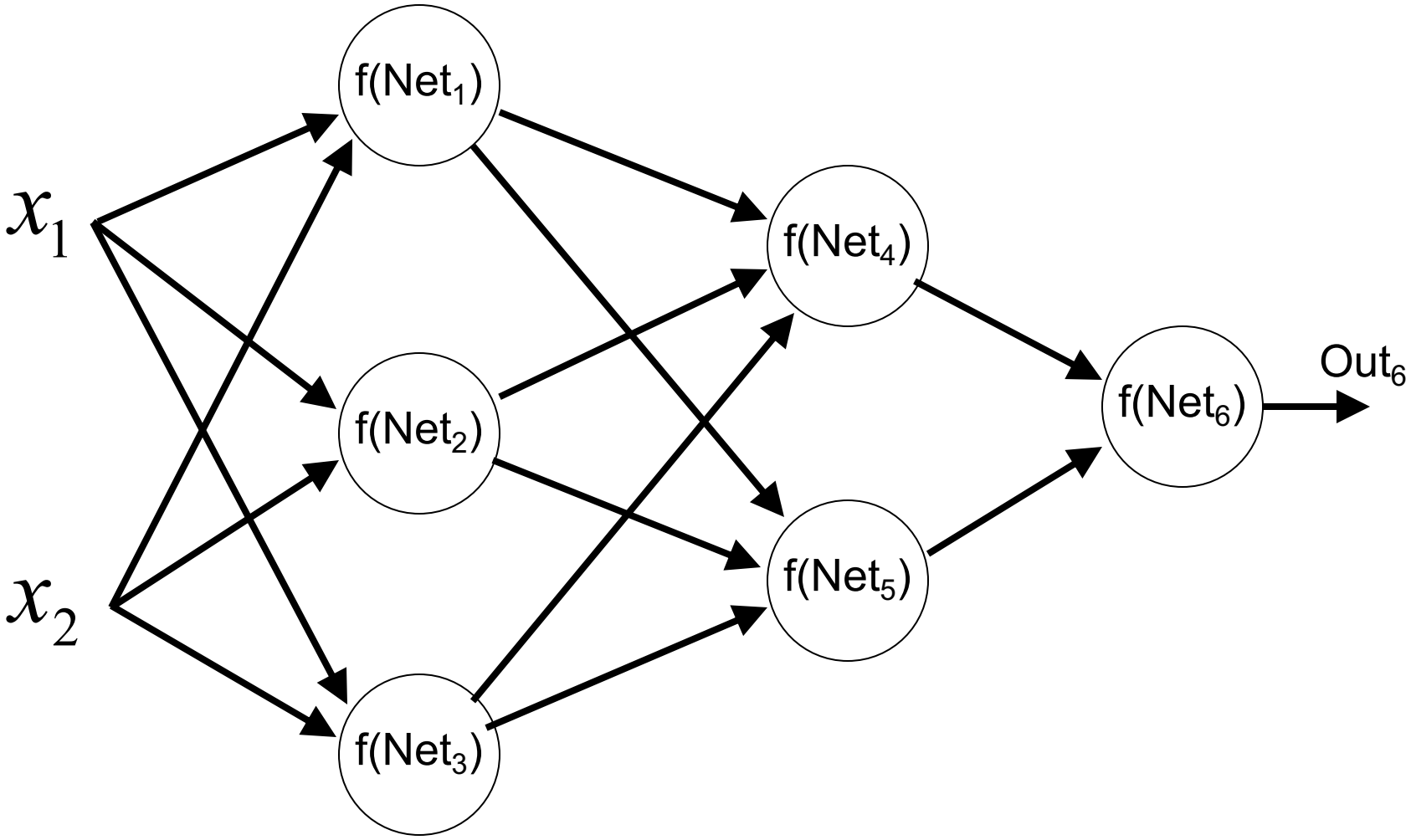
If the entire learning set has been used, go to Step 6; otherwise, go to Step 1

Stage 6 (Check the overall error)

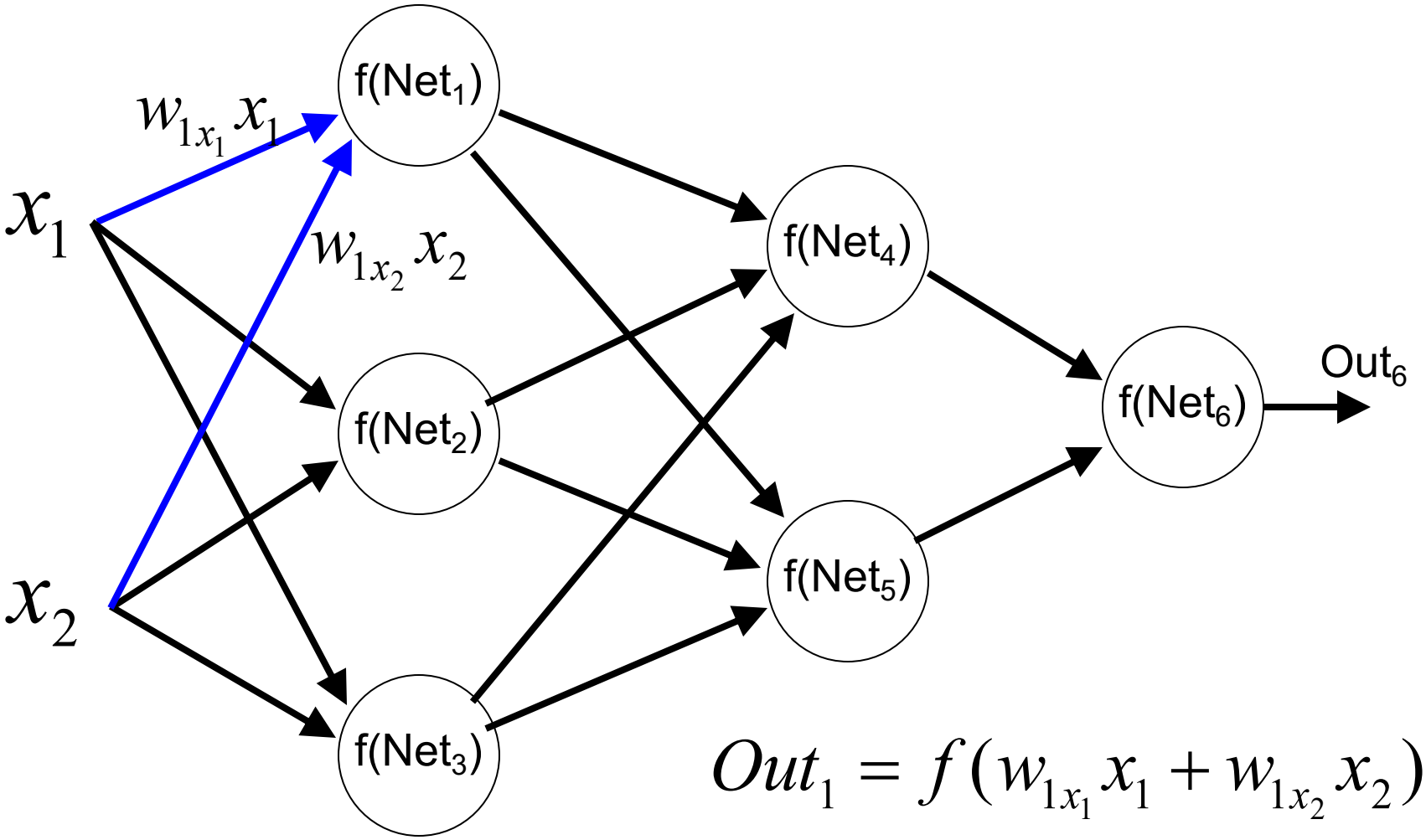
If the overall error E is less than the acceptable error threshold ($<E_{threshold}$), the learning ends and the learned weights are returned;

Otherwise, reassign E=0, and start a new learning cycle (return to Step 1)

BP Algorithm: Forward Propagation (1)

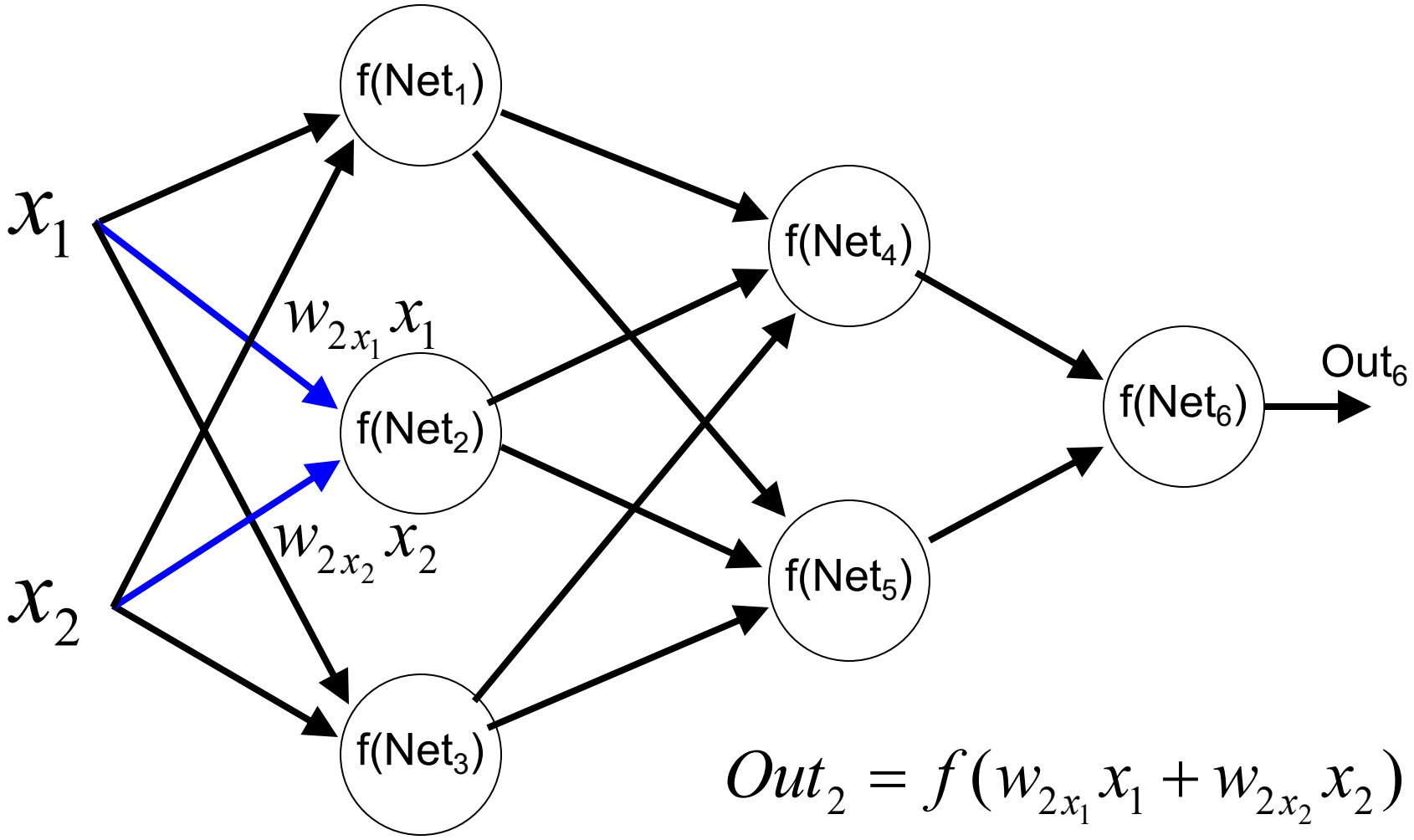


BP Algorithm: Forward Propagation (2)



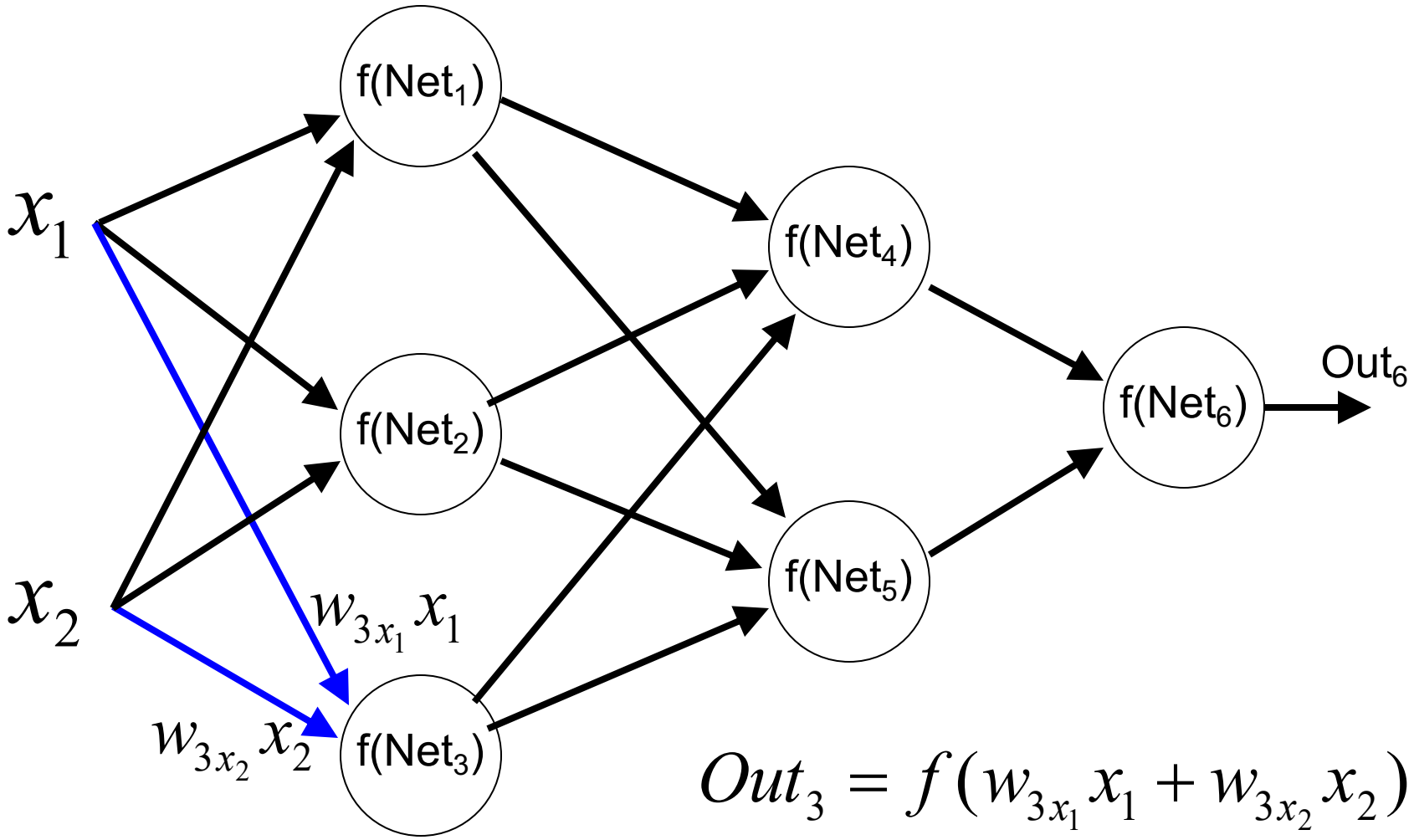
$$Out_1 = f(w_{1x_1} x_1 + w_{1x_2} x_2)$$

BP Algorithm: Forward Propagation (3)

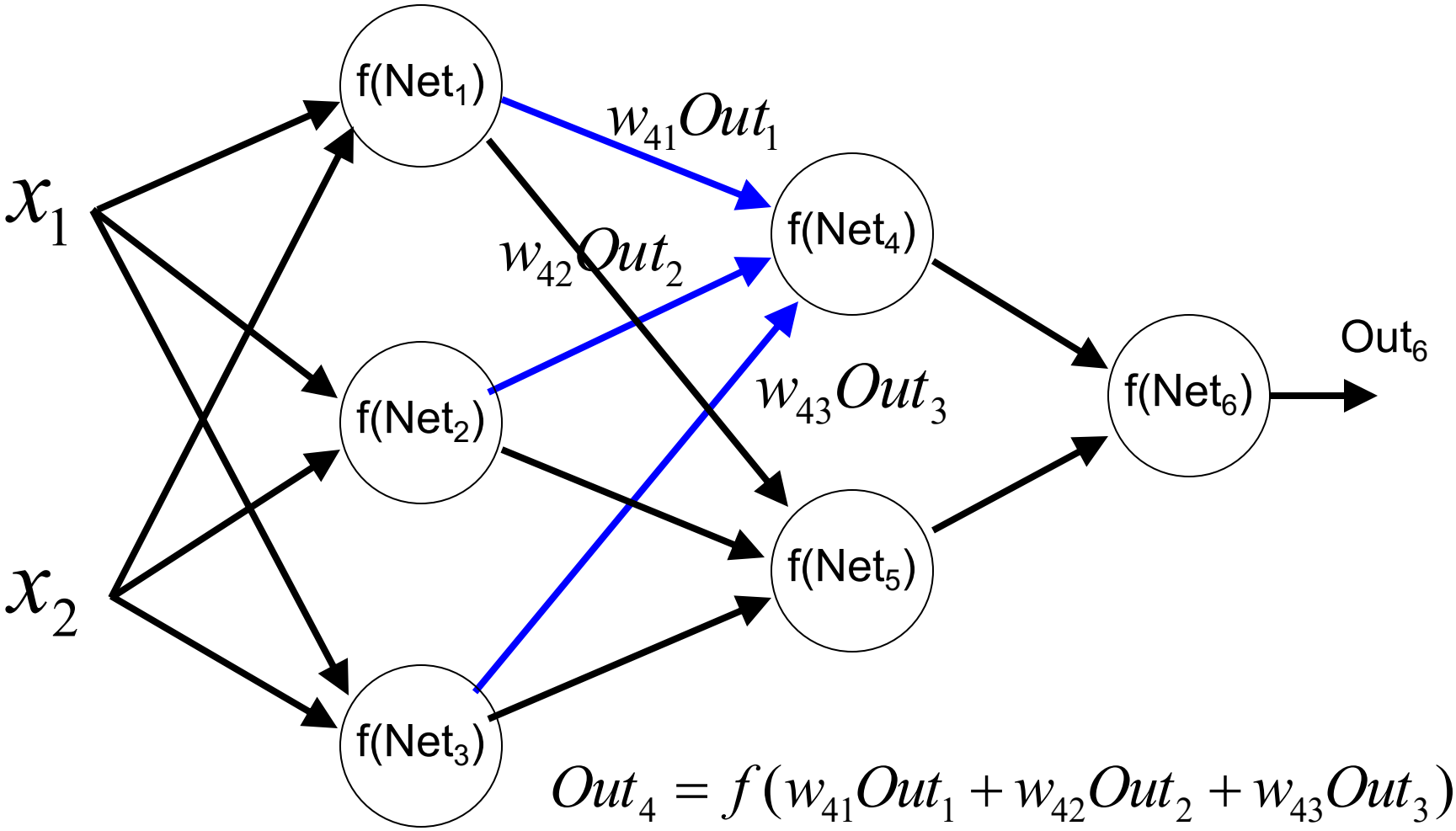


$$Out_2 = f(w_{2x_1} x_1 + w_{2x_2} x_2)$$

BP Algorithm: Forward Propagation (4)

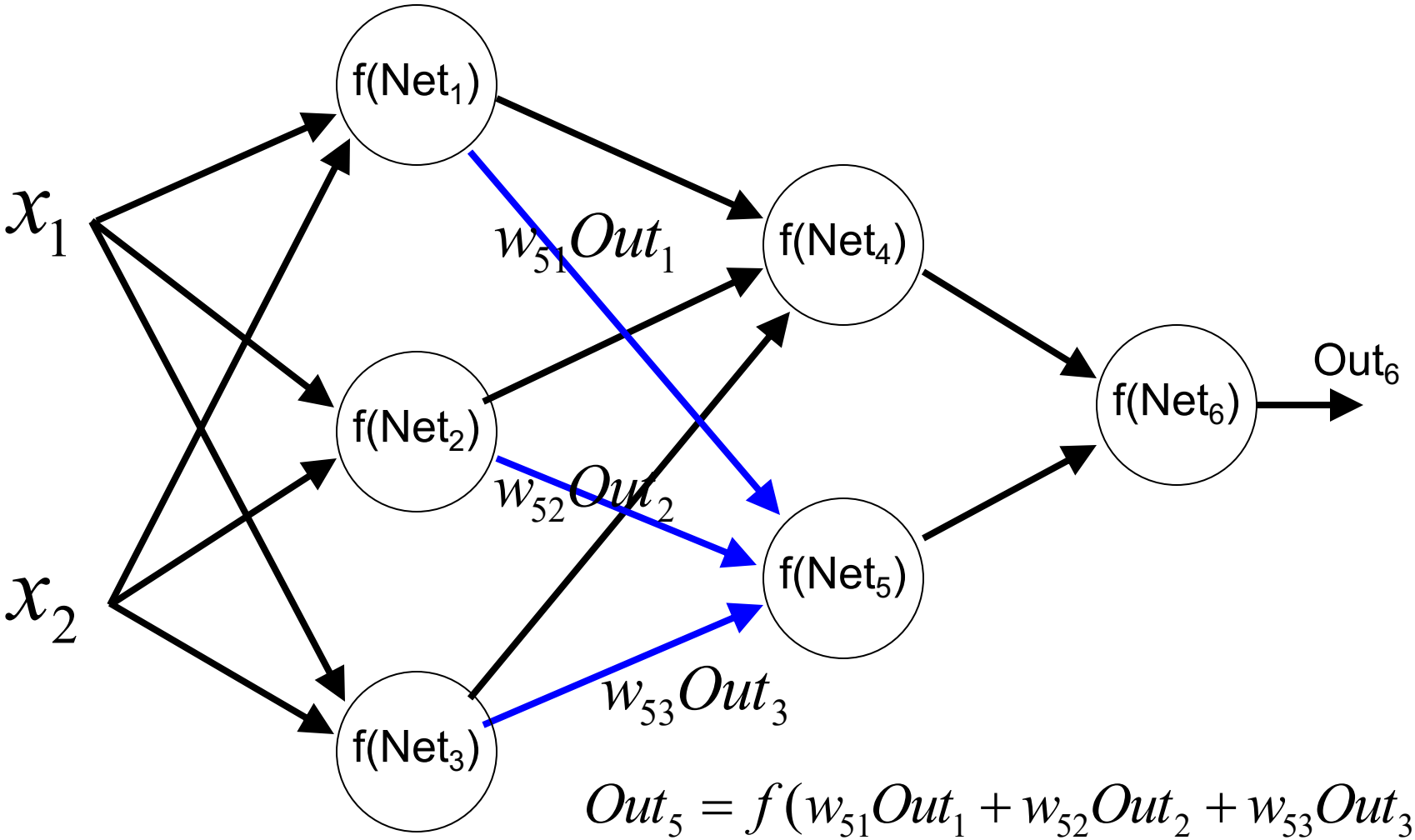


BP Algorithm: Forward Propagation (5)



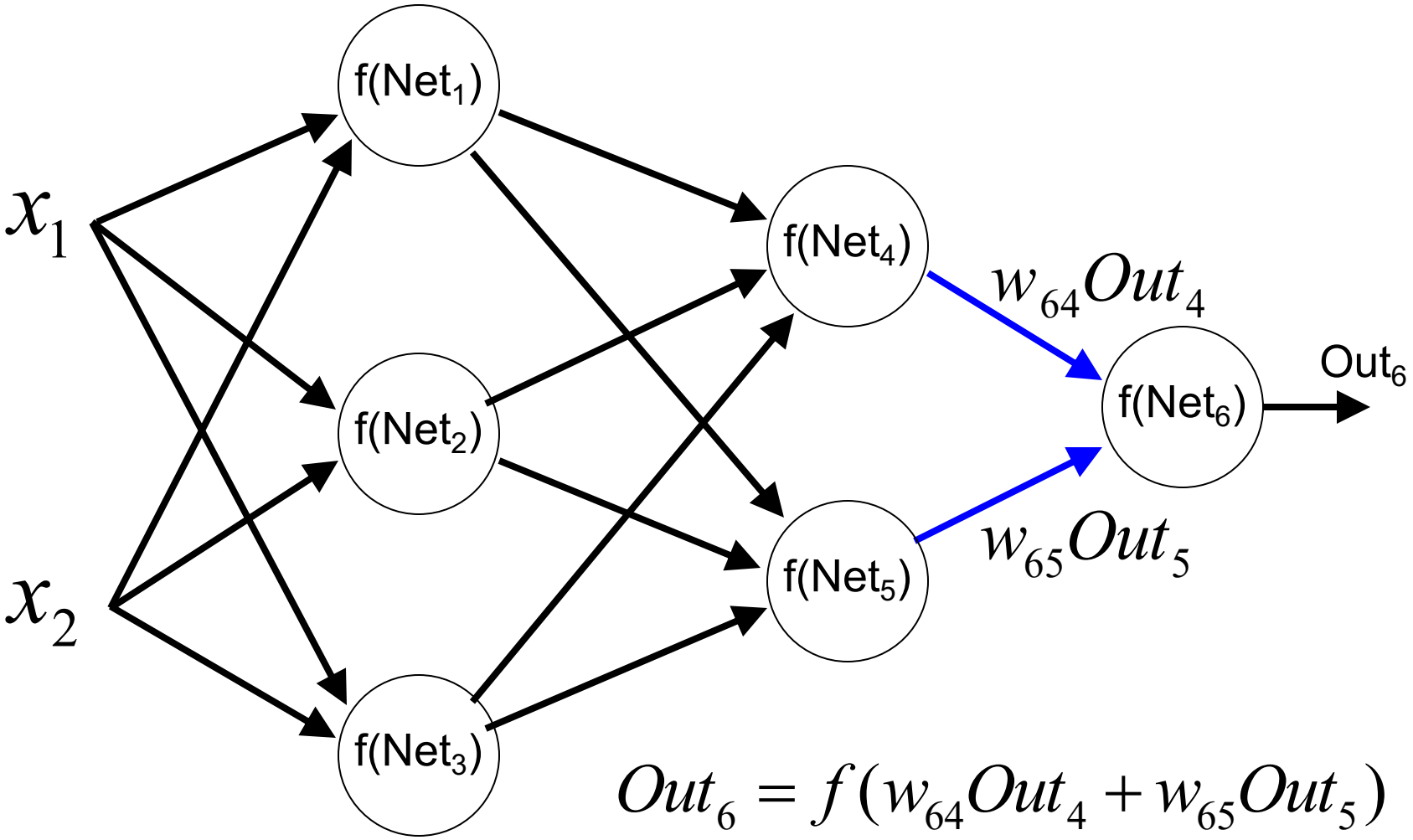
$$Out_4 = f(w_{41}Out_1 + w_{42}Out_2 + w_{43}Out_3)$$

BP Algorithm: Forward Propagation (6)



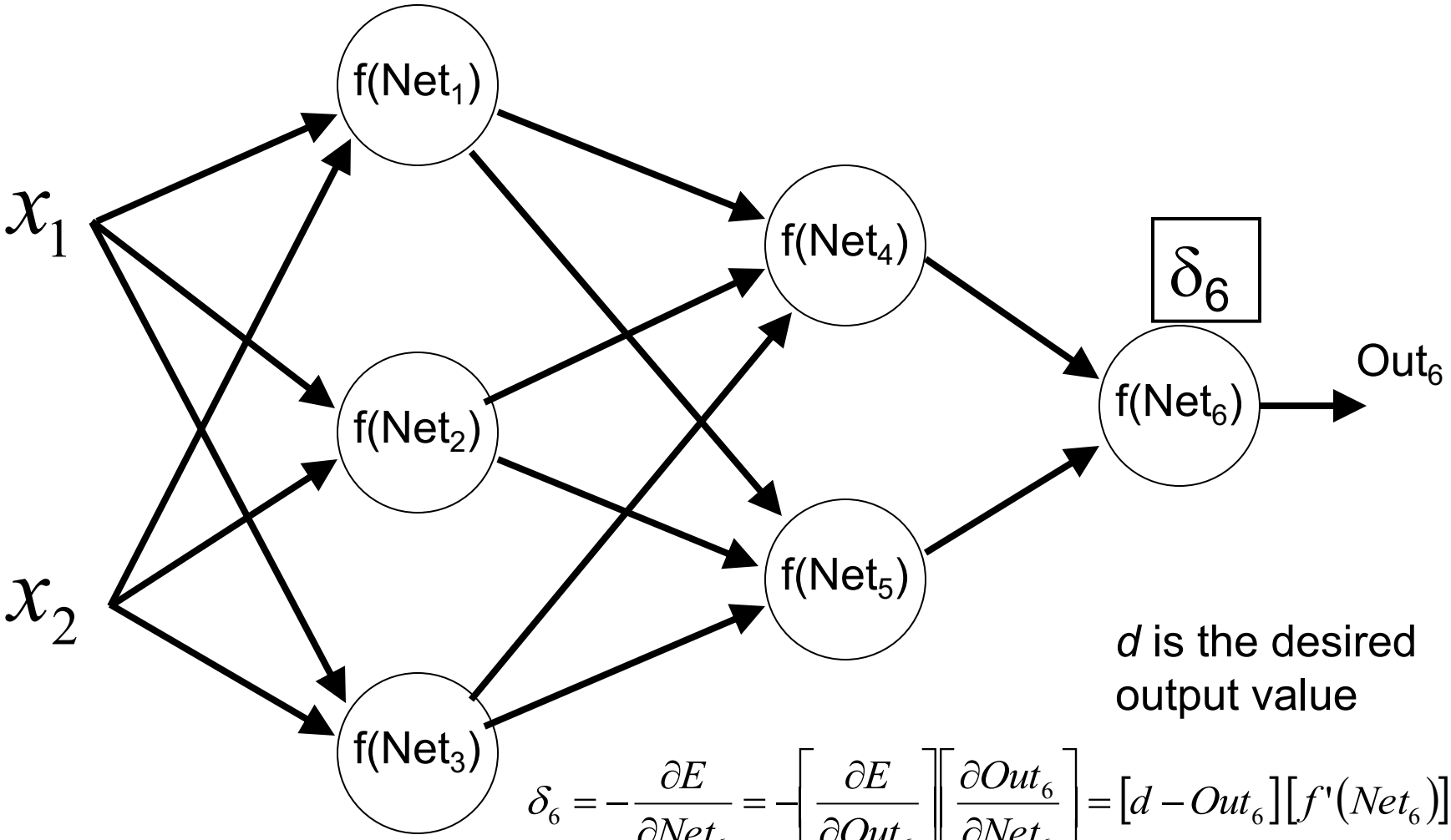
$$Out_5 = f(w_{51}Out_1 + w_{52}Out_2 + w_{53}Out_3)$$

BP Algorithm: Forward Propagation (7)



$$Out_6 = f(w_{64}Out_4 + w_{65}Out_5)$$

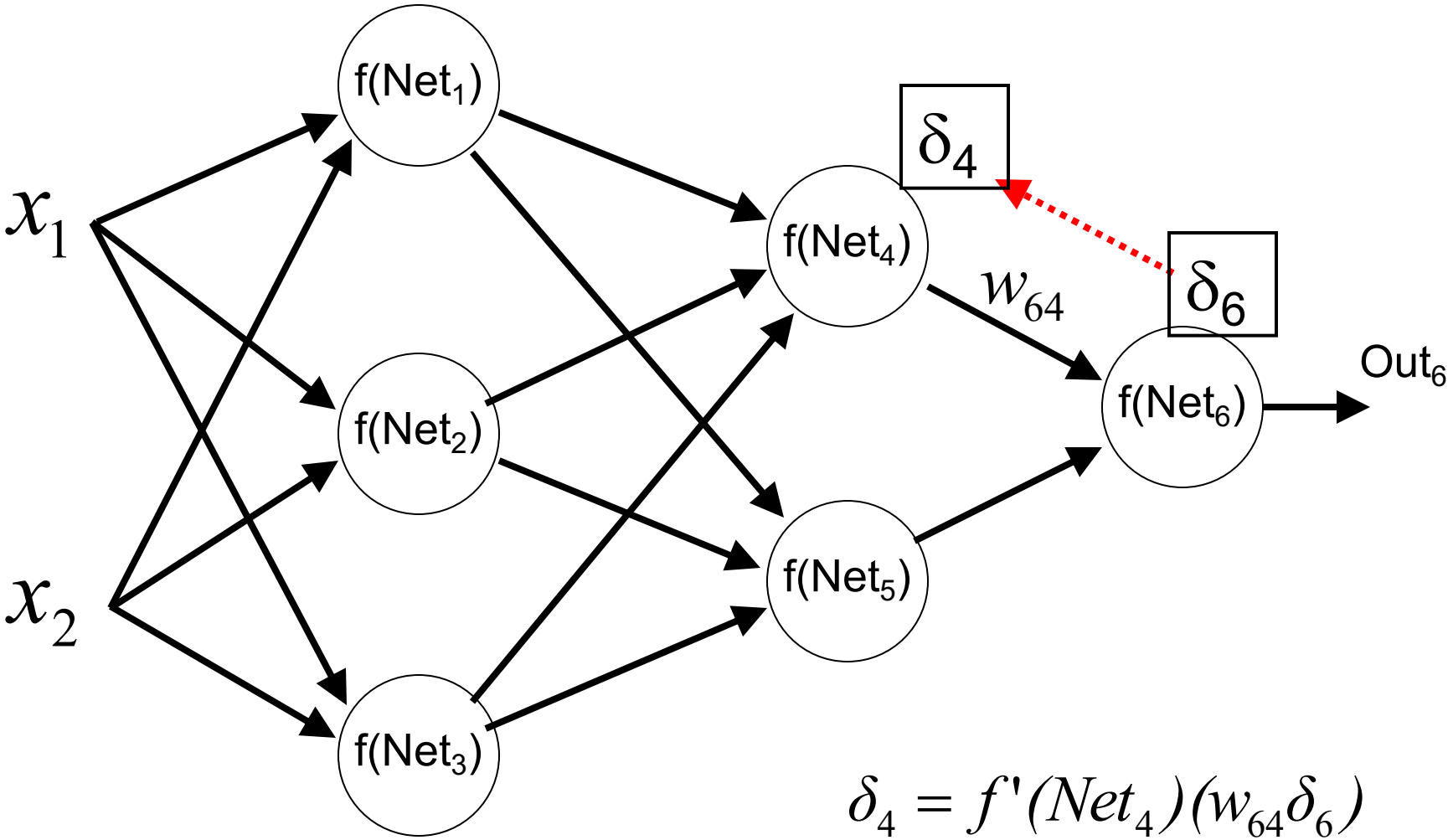
BP Algorithm: Error calculation



d is the desired output value

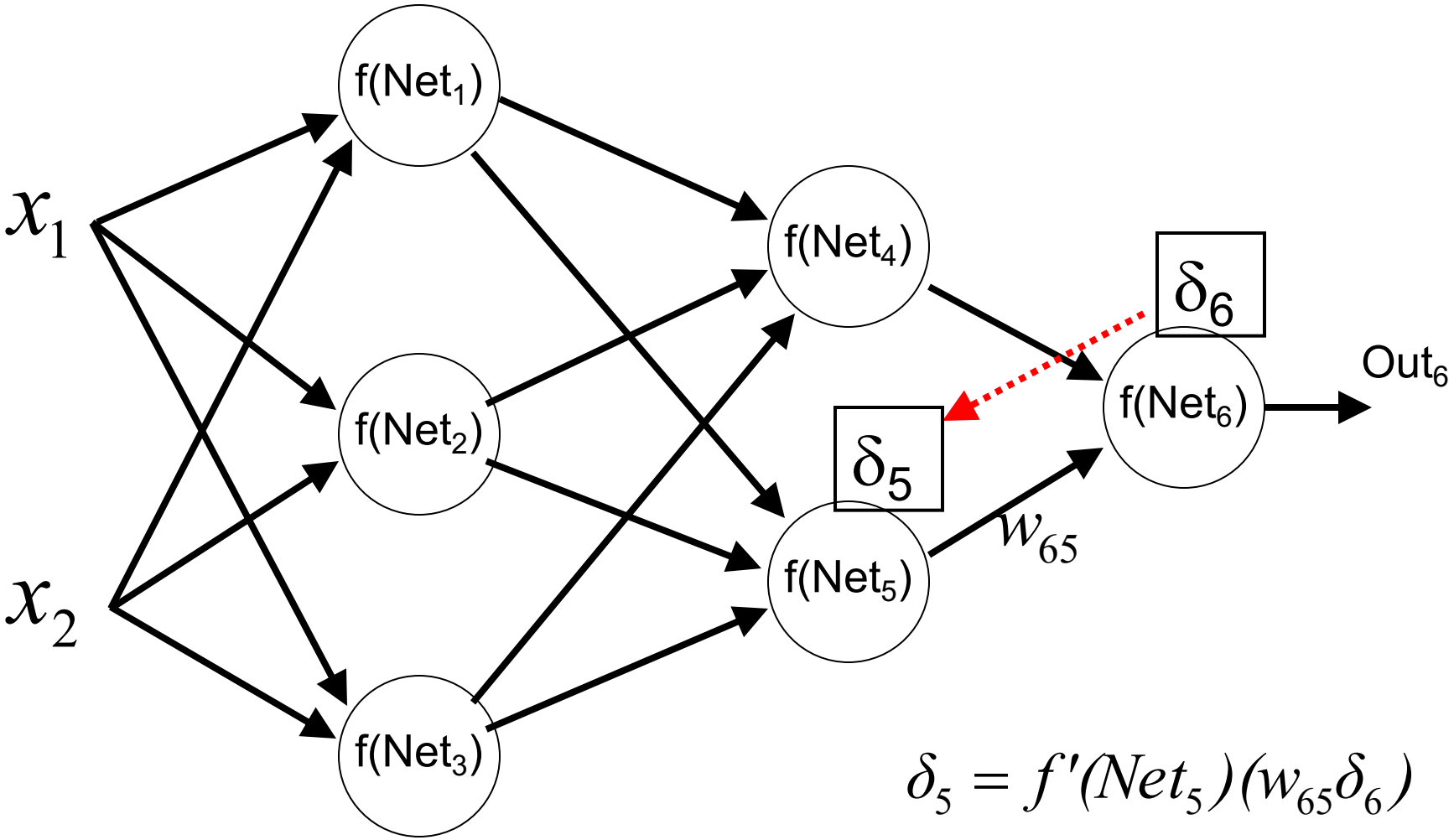
$$\delta_6 = -\frac{\partial E}{\partial Net_6} = -\left[\frac{\partial E}{\partial Out_6}\right]\left[\frac{\partial Out_6}{\partial Net_6}\right] = [d - Out_6][f'(Net_6)]$$

BP Algorithm: Back Propagation (1)



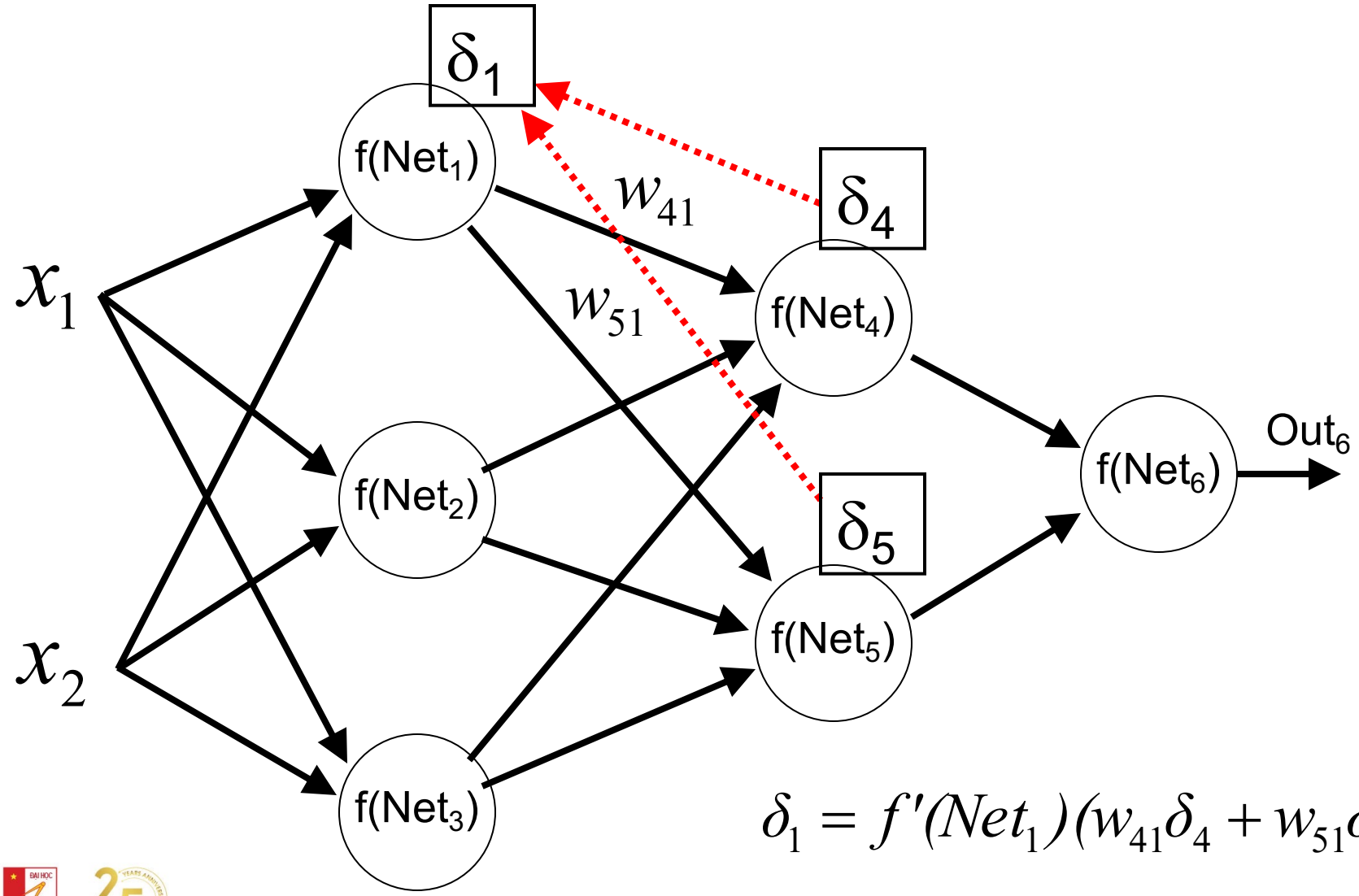
$$\delta_4 = f'(Net_4)(w_{64}\delta_6)$$

BP Algorithm: Back Propagation (2)



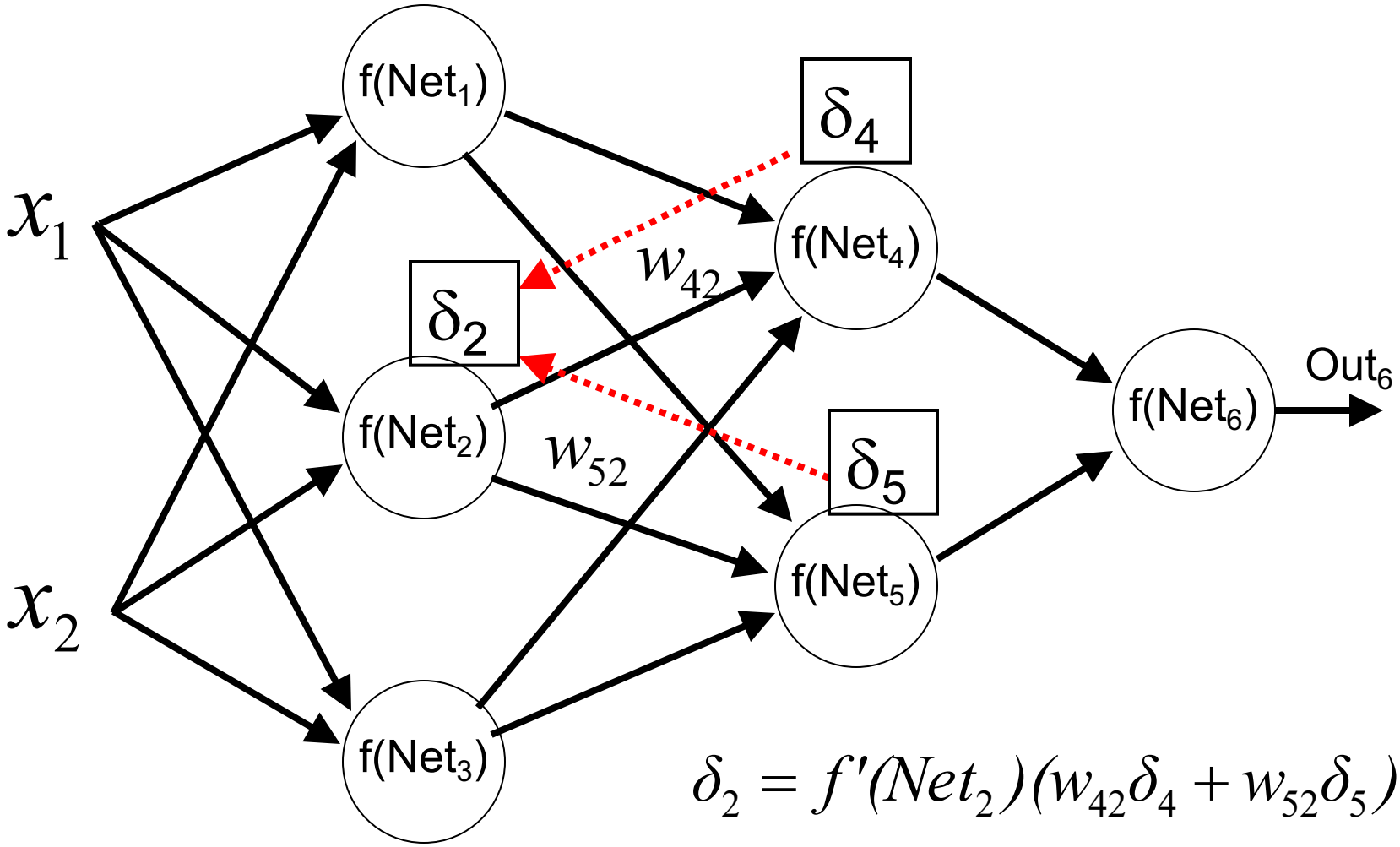
$$\delta_5 = f'(Net_5)(w_{65}\delta_6)$$

BP Algorithm: Back Propagation (3)



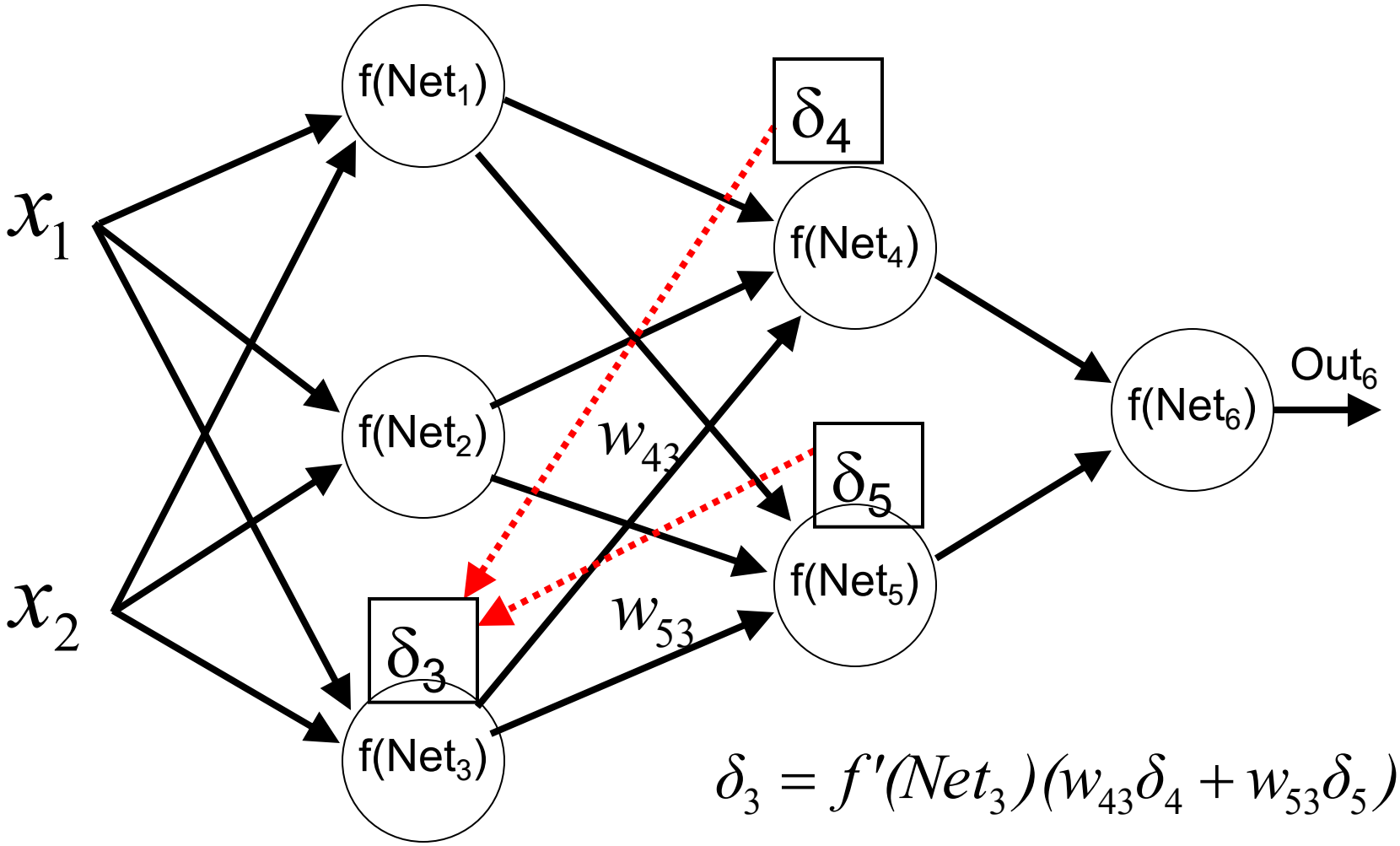
$$\delta_1 = f'(Net_1)(w_{41}\delta_4 + w_{51}\delta_5)$$

BP Algorithm: Back Propagation (4)



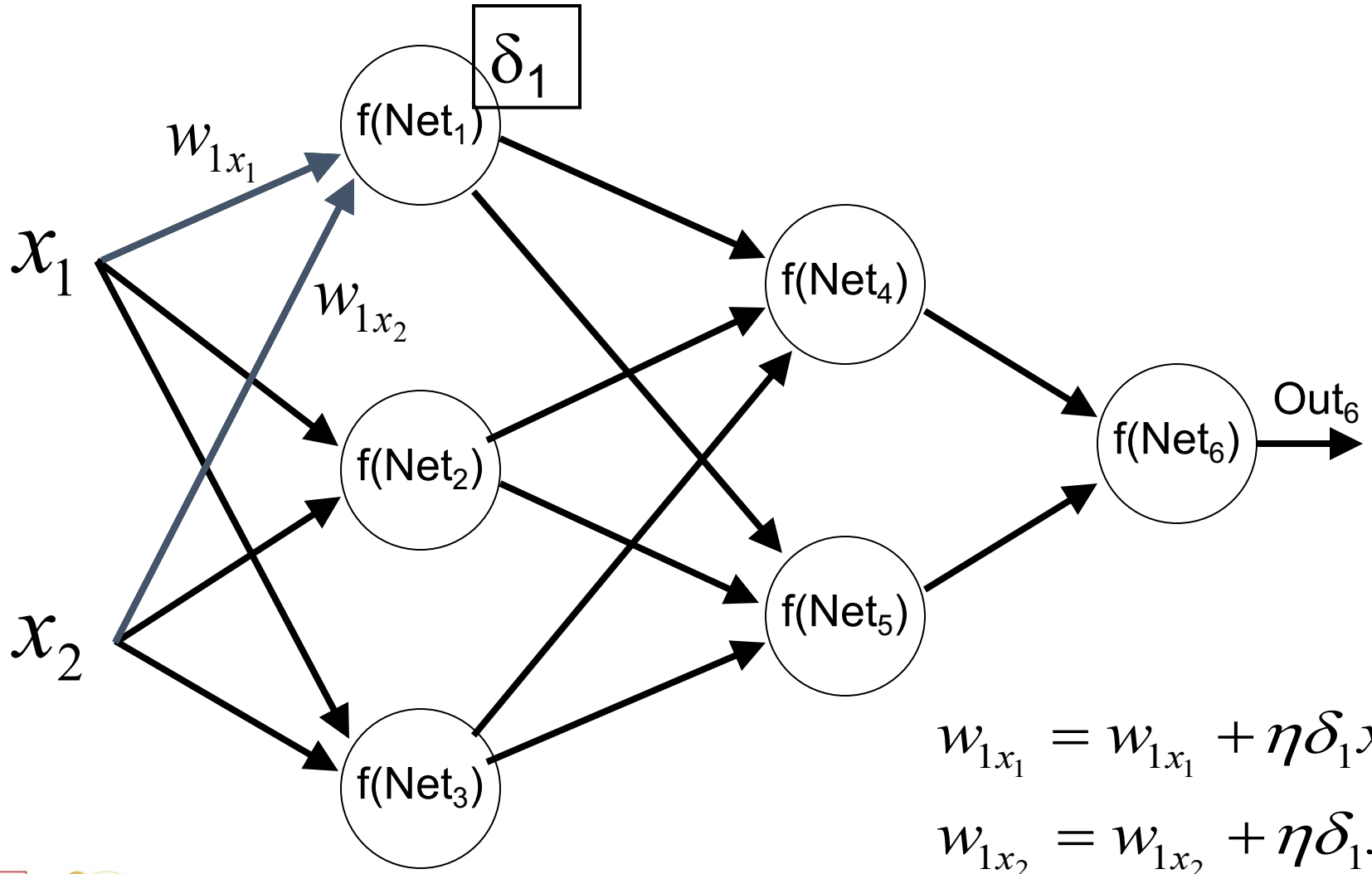
$$\delta_2 = f'(Net_2)(w_{42}\delta_4 + w_{52}\delta_5)$$

BP Algorithm: Back Propagation (5)



$$\delta_3 = f'(Net_3)(w_{43}\delta_4 + w_{53}\delta_5)$$

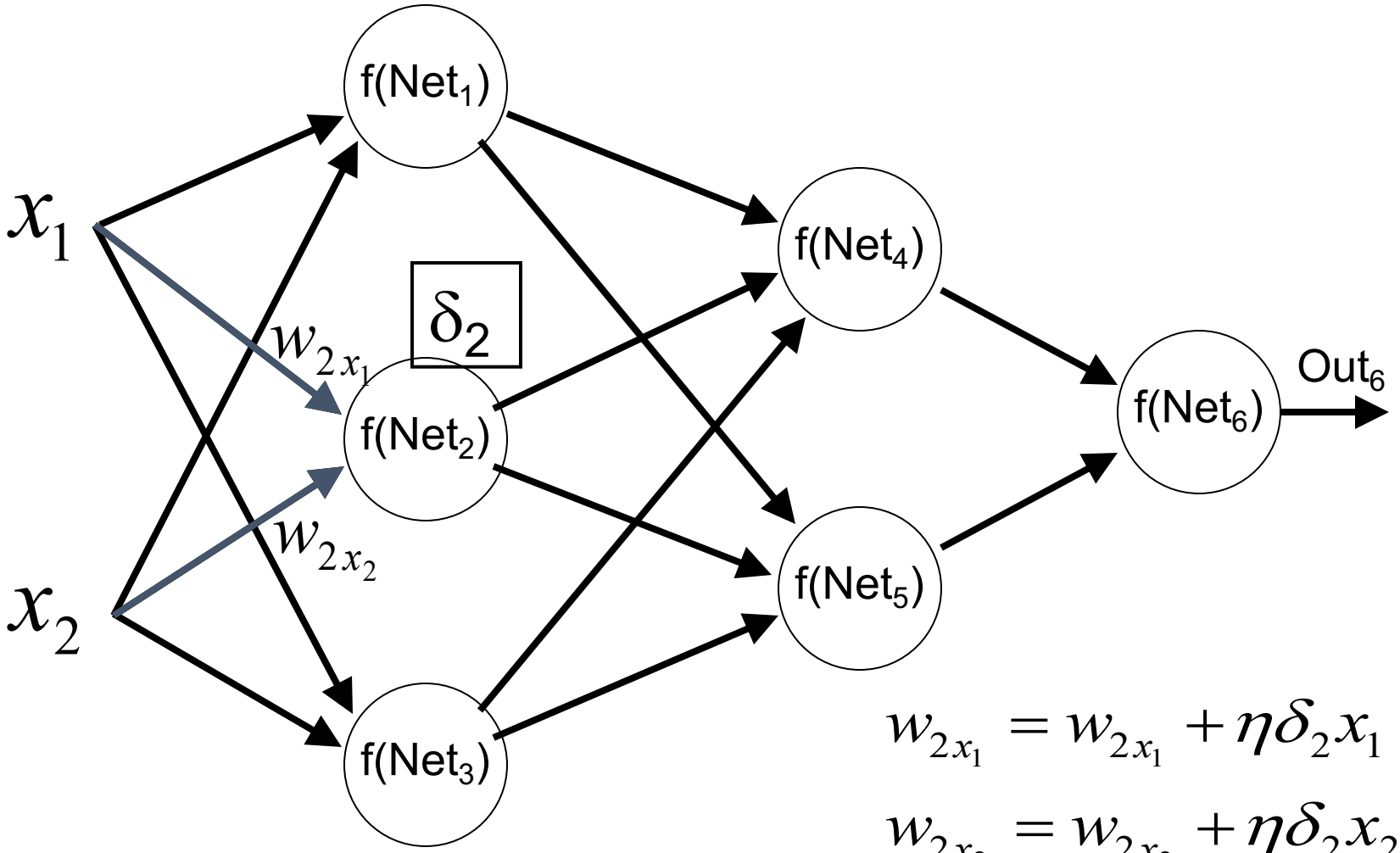
BP Algorithm: Update Weights (1)



$$w_{1x_1} = w_{1x_1} + \eta\delta_1 x_1$$

$$w_{1x_2} = w_{1x_2} + \eta\delta_1 x_2$$

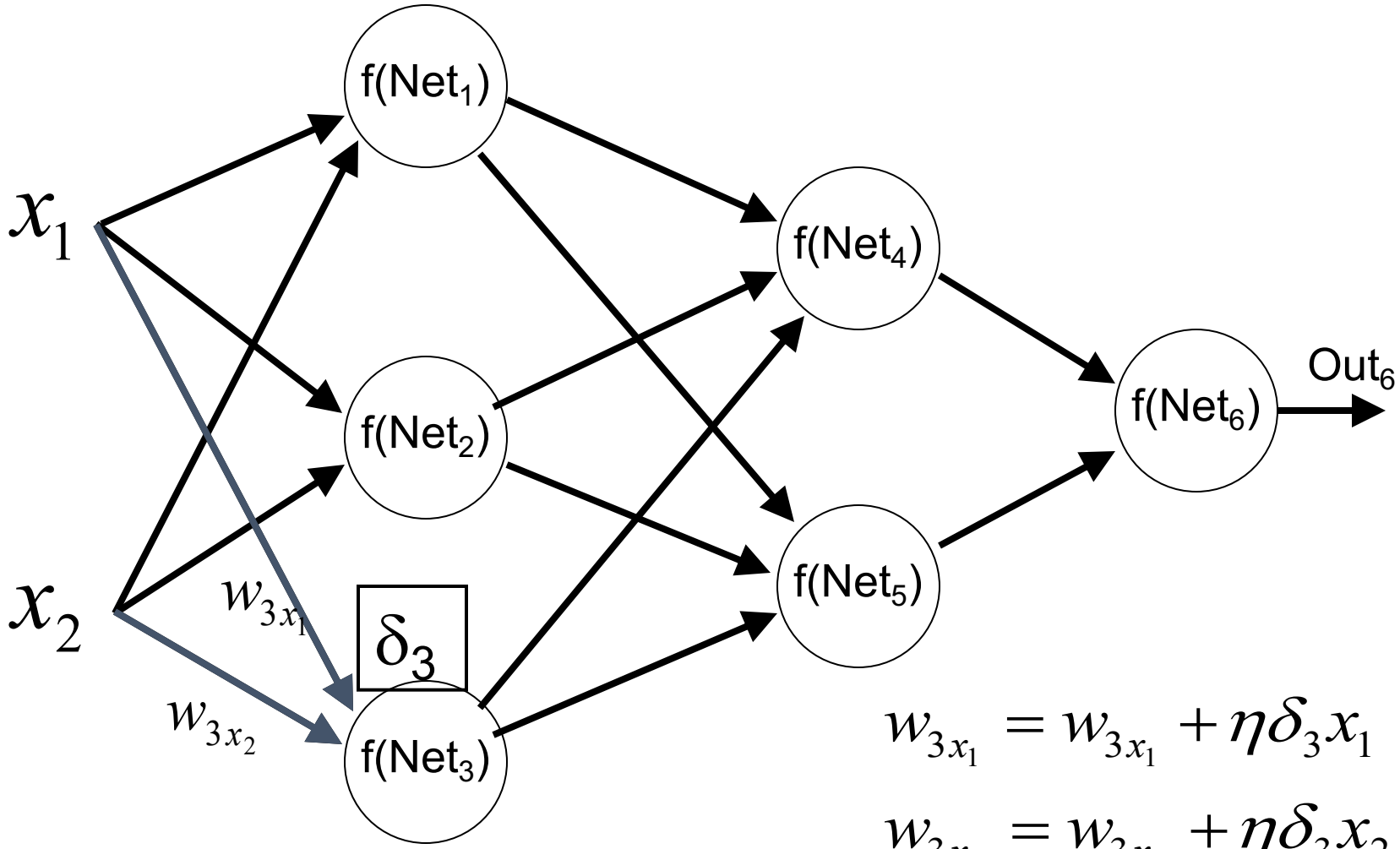
BP Algorithm: Update Weights (2)



$$w_{2x_1} = w_{2x_1} + \eta\delta_2 x_1$$

$$w_{2x_2} = w_{2x_2} + \eta\delta_2 x_2$$

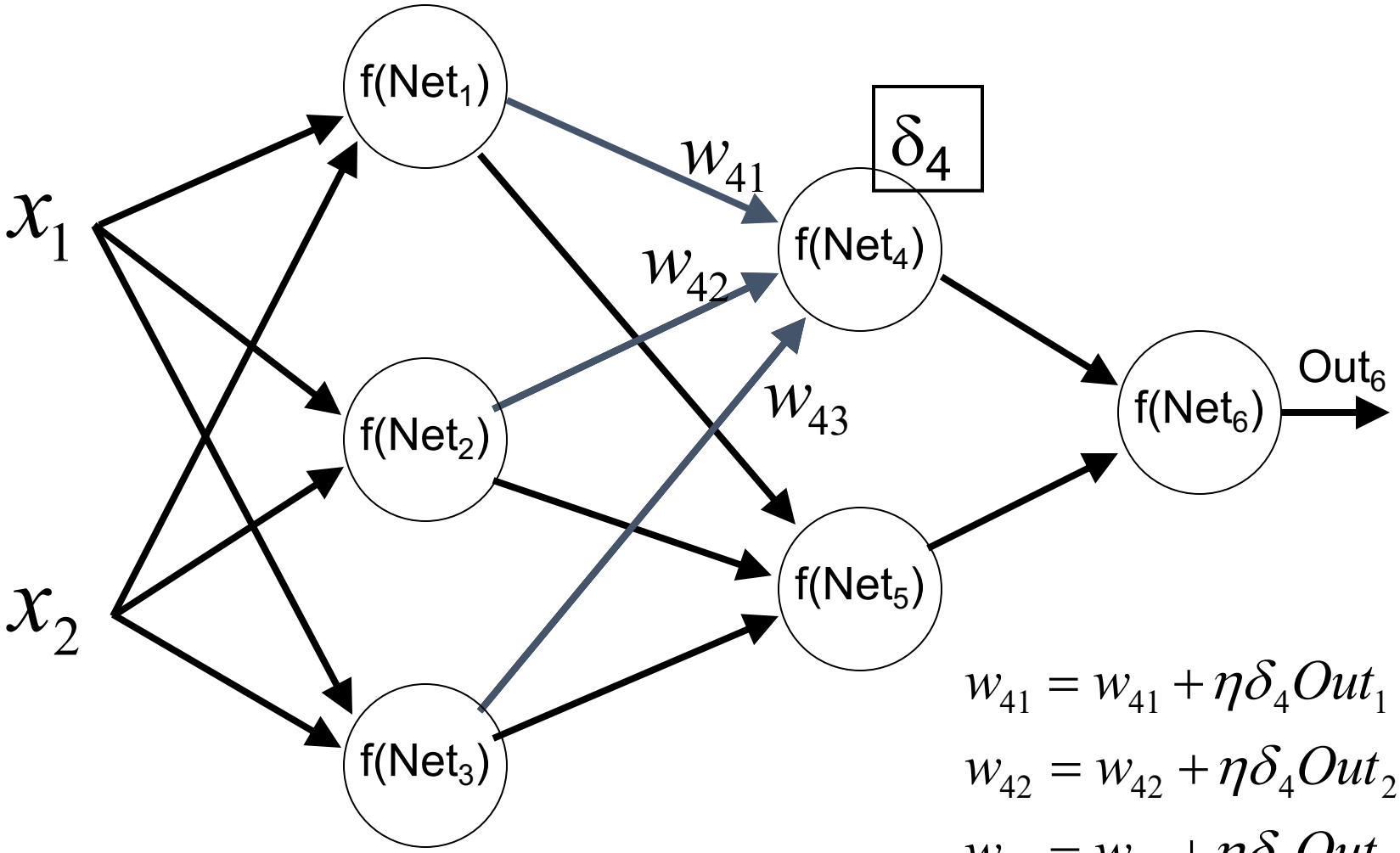
BP Algorithm: Update Weights (3)



$$w_{3x_1} = w_{3x_1} + \eta\delta_3 x_1$$

$$w_{3x_2} = w_{3x_2} + \eta\delta_3 x_2$$

BP Algorithm: Update Weights 4)

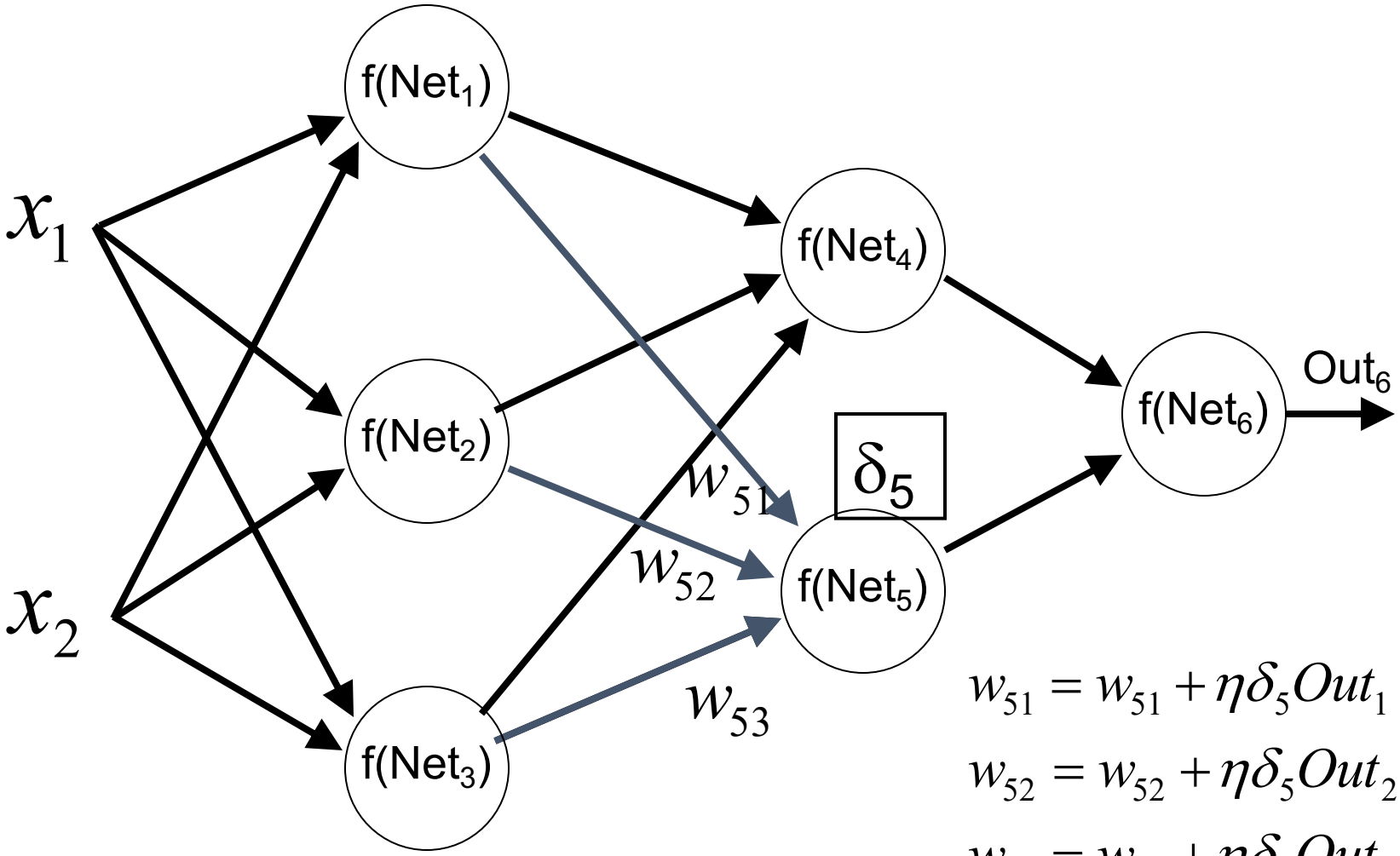


$$w_{41} = w_{41} + \eta\delta_4 Out_1$$

$$w_{42} = w_{42} + \eta\delta_4 Out_2$$

$$w_{43} = w_{43} + \eta\delta_4 Out_3$$

BP Algorithm: Update Weights (5)

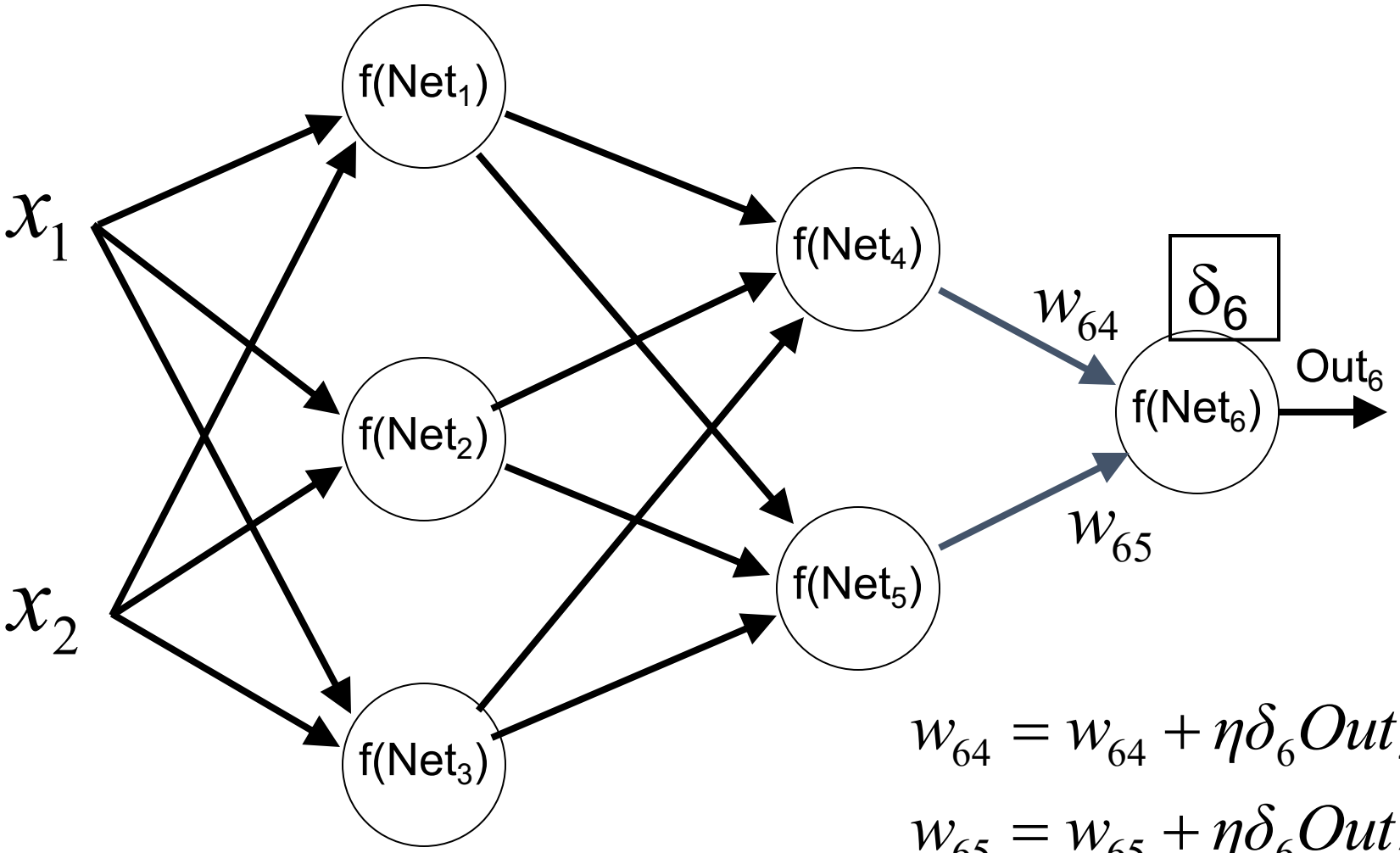


$$w_{51} = w_{51} + \eta\delta_5 Out_1$$

$$w_{52} = w_{52} + \eta\delta_5 Out_2$$

$$w_{53} = w_{53} + \eta\delta_5 Out_3$$

BP Algorithm: Update Weights (6)



$$w_{64} = w_{64} + \eta\delta_6 Out_4$$

$$w_{65} = w_{65} + \eta\delta_6 Out_5$$

BP: Initialize the weight values

- Usually, the weights are initialized with small random values
- If the weights have large initial values
  - The sigmoid functions will reach saturation early
  - The system will stall at a saddle/stationary point

BP: Tốc độ học (Learning rate)

- Ảnh hưởng quan trọng đến hiệu quả và khả năng hội tụ của giải thuật học BP
  - Một giá trị η lớn có thể đẩy nhanh sự hội tụ của quá trình học, nhưng có thể làm cho hệ thống bỏ qua điểm tối ưu toàn cục hoặc hội tụ vào điểm không tốt (saddle points)
  - Một giá trị η nhỏ có thể làm cho quá trình học kéo dài rất lâu
- Thường được chọn theo thực nghiệm (experimentally) đối với mỗi bài toán
- Các giá trị tốt của tốc độ học ở lúc bắt đầu (quá trình học) có thể không tốt ở một thời điểm sau đấy
  - Sử dụng một tốc độ học thích nghi (động)?

BP: Number of neurons in the hidden layer

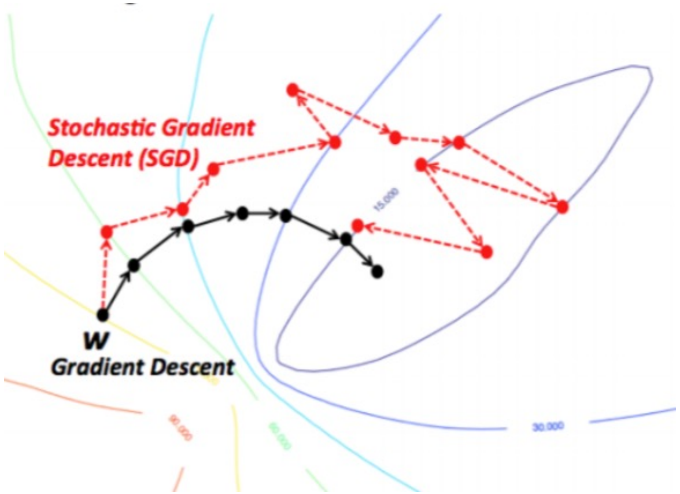
- The size (number of neurons) of the hidden layer is an important question for the application of multi-layer neural networks to solve real-world problems.
- In practice, it is difficult to determine the exact number of neurons required to achieve a desired system accuracy.
- The size of the hidden layer is usually determined by experiment (trial and test).

Stochastic gradient descent (SGD)

- Gradient descent algorithm
  1. Initialize weights w
  2. Compute $\nabla loss = \sum_m gradient$
  3. $w \leftarrow w - \alpha\,(\nabla loss)$
  4. Repeate 2 – 3 until convergence
- Stochastic gradient descent
  1. Initialize weights w
  2. Compute for each sample (x, y) $\nabla loss = gradient\ at\ (x, y)$
  3. $w \leftarrow w - \alpha\,(\nabla loss)$
  4. Repeate 2 – 3 until convergence
- Mini-batch gradient descent
  1. $\nabla loss = gradient\ for\ n\ sample\ (x, y)n$

GD vs SGD

- Gradient descent goes in steepest descent directions, but slower
- SGD is faster even though it does not follows the steepest path
- Mini-batch SGD

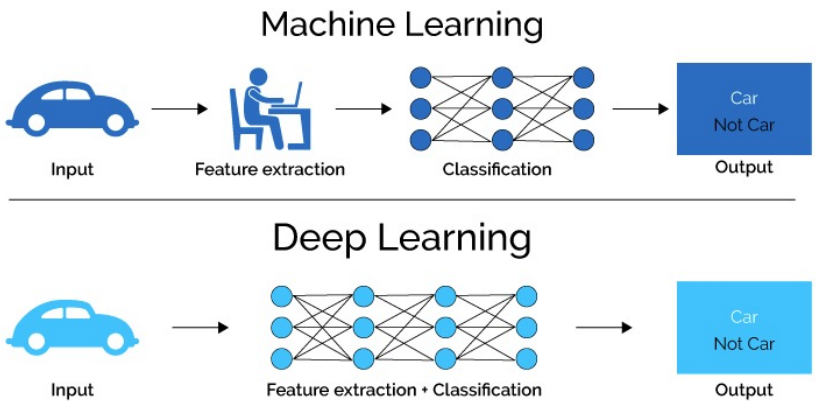


© bogotobogo.com

Summary

ANN: Pros

- The nature (structurally) of neural networks supports a high degree of parallelism
- High accuracy in many problems (photo, video, audio, text)
- Very flexible in network architecture
- Deep learning allows automatic feature extraction from data
  - Traditional machine learning methods require manual feature extraction, experience and depend on specific problems.



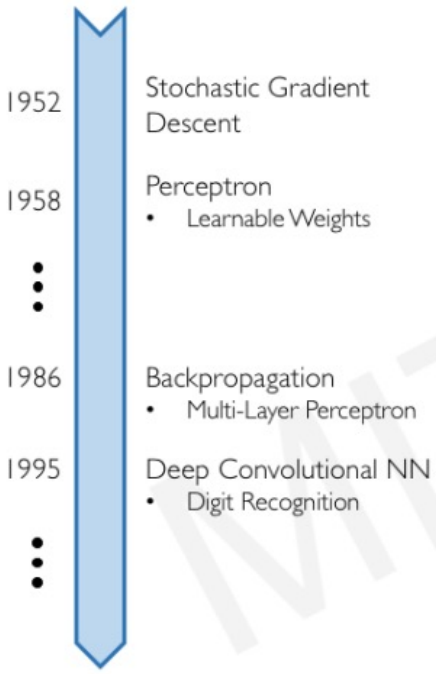
ANN: Cons

- There is no general rule to determine the optimal network structure and learning parameters for a given problem (class).
- There is no general method to evaluate the inner workings of ANNs (thus, the ANN system is seen as a "black box").
- It is very difficult (impossible) to give an explanation to the user
- There is little theoretical foundation, to help explain the successes in practice

ANN: When?

- The form of the learning function is not known in advance
- It is not necessary (or not important) to provide an explanation to the user for the results
- Accept a (quite) long time for the training process
- It is possible to collect a large number of labels for data.
- Related domains: image, video, speech, text

Why is deep learning so popular now?



Neural Networks date back decades, so why the resurgence?

1. Big Data

- Larger Datasets
- Easier Collection & Storage

IMAGENET

WIKIPEDIA
The Free Encyclopedia

2. Hardware

- Graphics Processing Units (GPUs)
- Massively Parallelizable

3. Software

- Improved Techniques
- New Models
- Toolboxes

TensorFlow

Open libraries

Keras

Caffe2

TensorFlow

PYTORCH

References

- Cybenko, G. (1989) "Approximations by superpositions of sigmoidal functions", Mathematics of Control, Signals, and Systems, 2 (4), 303-314
- Kurt Hornik (1991) "Approximation Capabilities of Multilayer Feedforward Networks", Neural Networks, 4(2), 251–257

ĐẠI HỌC BÁCH KHOA

25 YEARS ANNIVERSARY SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Thank you for your attention!!!