



# Introduction to **Machine Learning and Data Mining** (Học máy và Khai phá dữ liệu)

**Khoat Than**

School of Information and Communication Technology  
Hanoi University of Science and Technology

# Outline

---

- Introduction to Machine Learning & Data Mining
- Unsupervised learning
- **Supervised learning**
  - **Artificial neural network**
- Practical advice

# Artificial neural network: introduction (1)

- Artificial neural network (ANN) (mạng nơron nhân tạo)
  - Simulates the biological neural systems (human brain)
  - ANN is a structure/network made of interconnection of artificial neurons
- Neuron
  - Has input/output
  - Executes a local calculation (local function)
- Output of a neuron is characterized by
  - In/out characteristics
  - Connections between it and other neurons
  - (Possible) other inputs



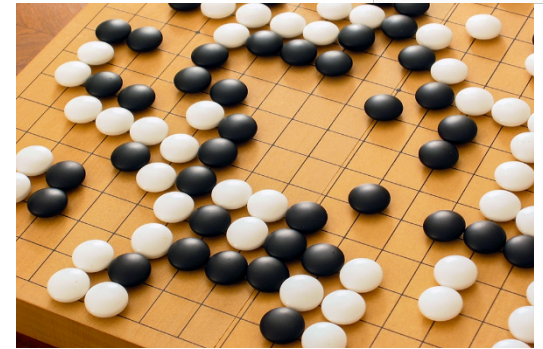
## Artificial neural network: introduction (2)

---

- ANN can be thought of as a highly decentralized and parallel information processing structure
- ANN has the ability to learn, recall and generalize from the training data
- The ability of an ANN depends on
  - Topology of the neural network
  - Input/output characteristics
  - Learning algorithm
  - Training data

# ANN: a huge breakthrough

- AlphaGo of Google the world champion at Go, 3/2016
  - Go is a 2500 year-old game.
  - Go is one of the most complex games
- AlphaGo learns from 30 millions human moves, and plays itself to find new moves



- It beat Lee Sedol (World champion)

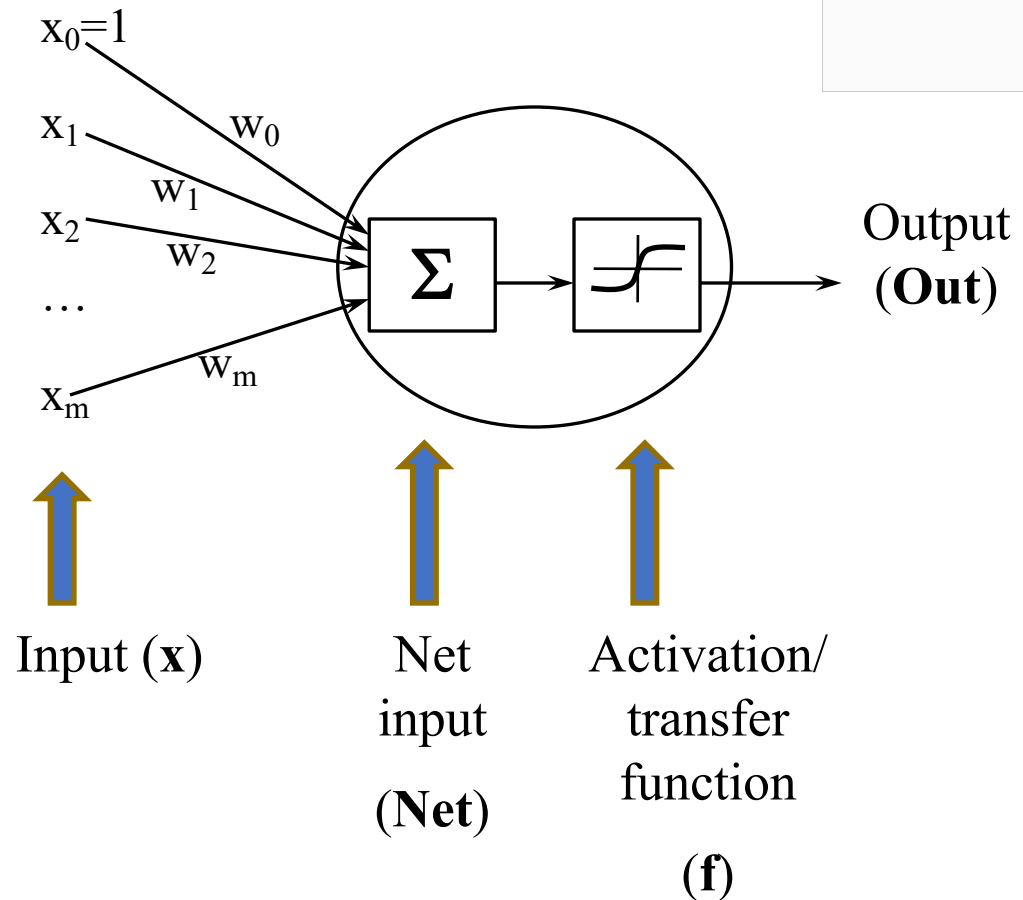
<http://www.wired.com/2016/03/two-move-future/>

<http://www.nature.com/news/google-ai-alpha-go-1.19234>



# Structure of a neuron

- Input signals of a neuron  
 $\{x_i, i = 1 \dots m\}$ 
  - Each input signal  $x_i$  is associated with a weight  $w_i$
- Bias  $w_0$  (with  $x_0 = 1$ )
- Net input is a combination of the input signals  
 $Net(\mathbf{w}, \mathbf{x})$
- Activation/transfer function  $f(\cdot)$  computes the output of a neuron
- Output  
 $Out = f(Net(\mathbf{w}, \mathbf{x}))$



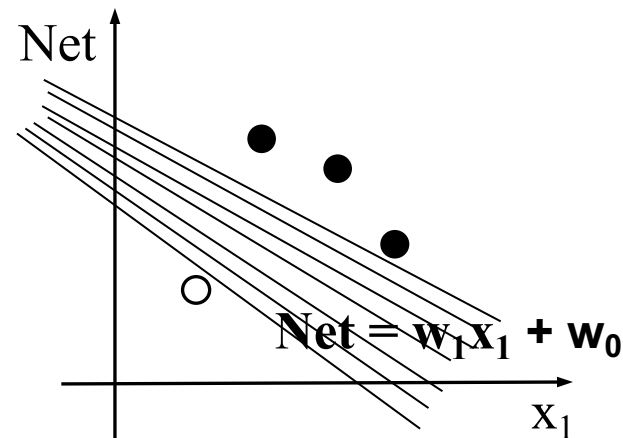
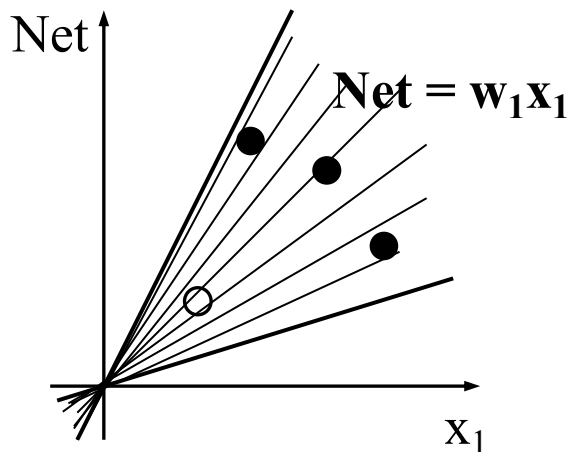
# Net Input

- Net input is usually calculated by a linear function

$$Net = w_0 + w_1x_1 + w_2x_2 + \dots + w_mx_m = w_0 \cdot 1 + \sum_{i=1}^m w_ix_i = \sum_{i=0}^m w_ix_i$$

- Role of bias:

- Net= $w_1x_1$  may not separate well the classes
- Net= $w_1x_1 + w_0$  is able to do better



# Activation function: hard-limited

- Also known as a threshold function

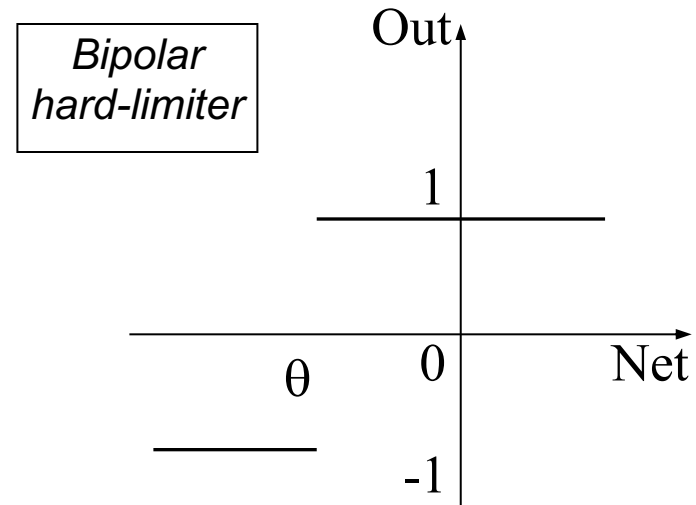
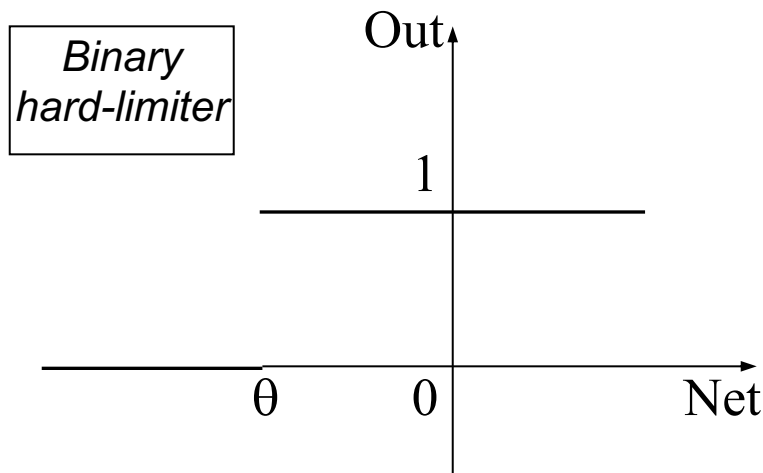
$$Out(Net) = HL(Net, \theta) = \begin{cases} 1, & \text{if } Net \geq \theta \\ 0, & \text{otherwise} \end{cases}$$

- The output takes one of the two values

$$Out(Net) = HL2(Net, \theta) = \text{sign}(Net, \theta)$$

- $\theta$  is the threshold value

- **Disadvantages:** discontinuous, non-smoothed (không trơn)



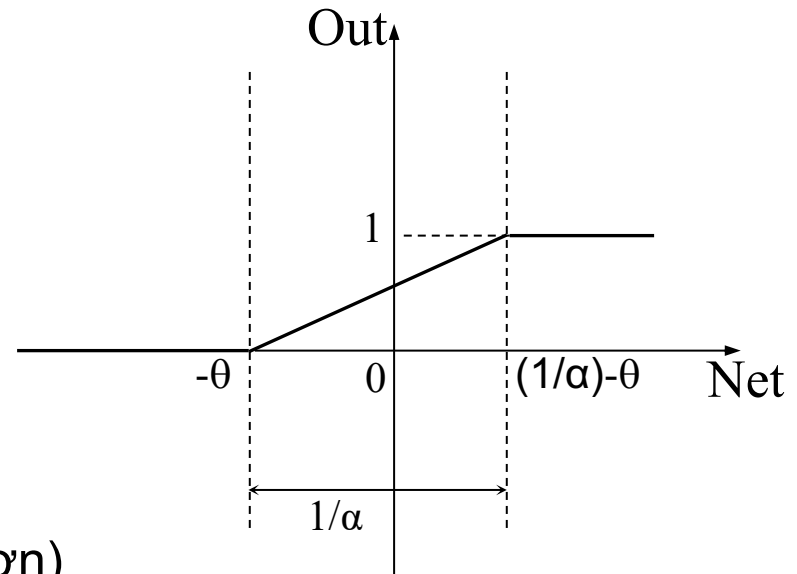


# Activation function: threshold logic

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

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

- Also known as a saturating linear function
- Combination of 2 activation functions: linear and tight limit
- $\alpha$  determines the slope of the linear range
- **Disadvantages:** continuous, non-smoothed (liên tục, nhưng không trơn)



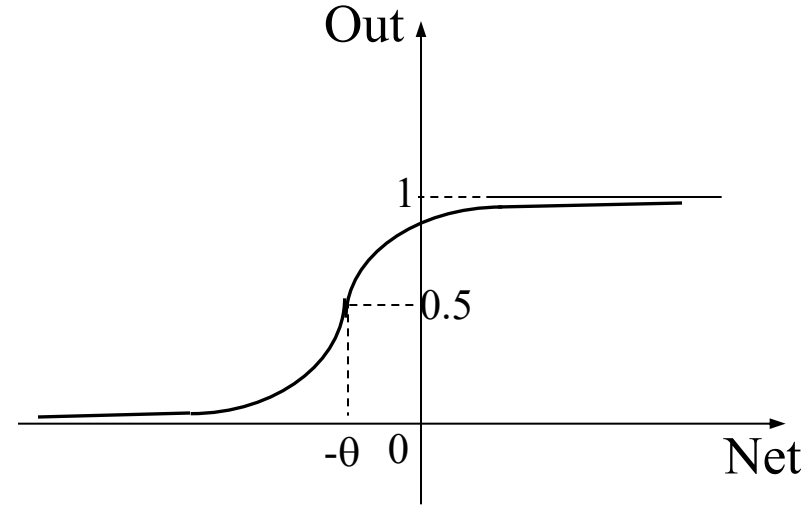
# Activation function: Sigmoid

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

- Popular
- The parameter  $\alpha$  determine the slope
- Output in the range of 0 and 1

- **Advantages**

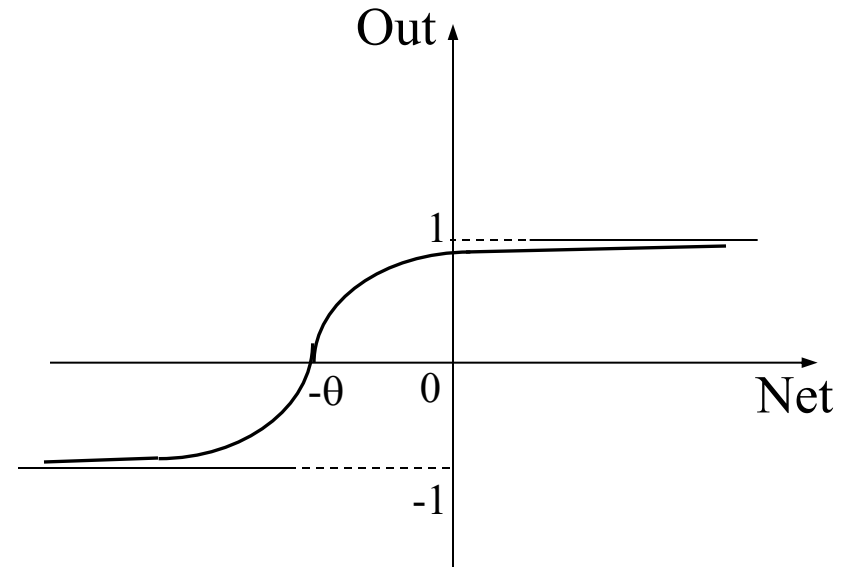
- Continuous, smoothed
- Gradient of a sigmoid function is represented by a function of itself



# Activation function: Hyperbolic tangent

$$Out(Net) = \tanh(Net, \alpha, \theta) = \frac{1 - e^{-\alpha(Net + \theta)}}{1 + e^{-\alpha(Net + \theta)}} = \frac{2}{1 + e^{-\alpha(Net + \theta)}} - 1$$

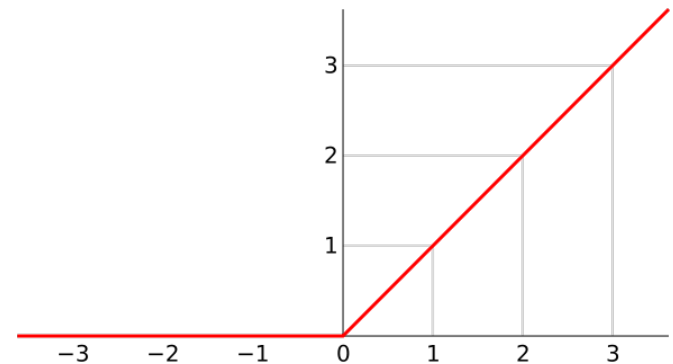
- Popular
- The parameter  $\alpha$  determine the slope
- Output in the range of -1 and 1
- Advantages
  - Continuous, continuous derivative
  - Gradient of a tanh function is represented by a function of itself



# Act. function: Rectified linear unit (ReLU)

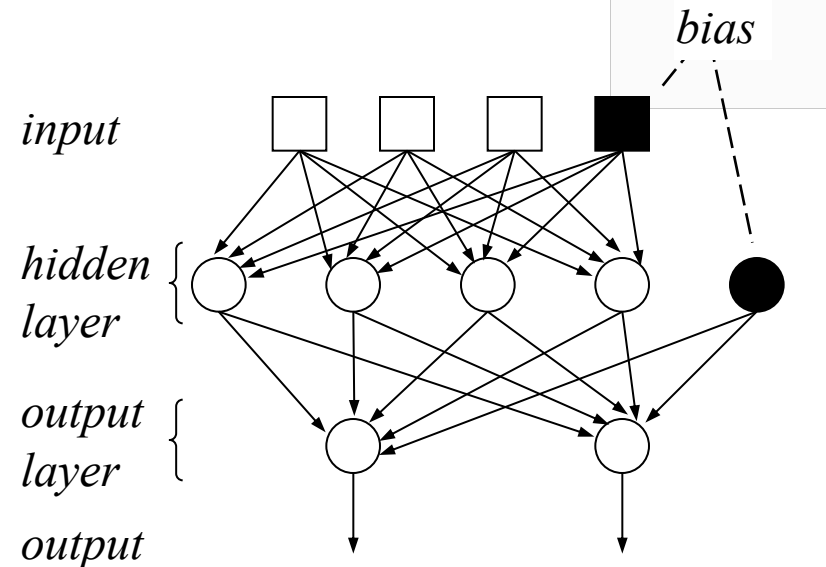
$$Out(net) = \max(0, net)$$

- Most popular
- Output is non-negative
- Advantages
  - Continuous
  - No derivative at point 0
  - Easy to calculate



# ANN: Architecture (1)

- ANN's architecture is determined by
  - Number of input and output signals
  - Number of layers
  - Number of neurons in each layer
  - Number of connection for each neuron
  - How neurons (with in a layer, or between layers) are connected
- An ANN must have
  - An input layer
  - An output layer
  - No, single, or multiple hidden layers

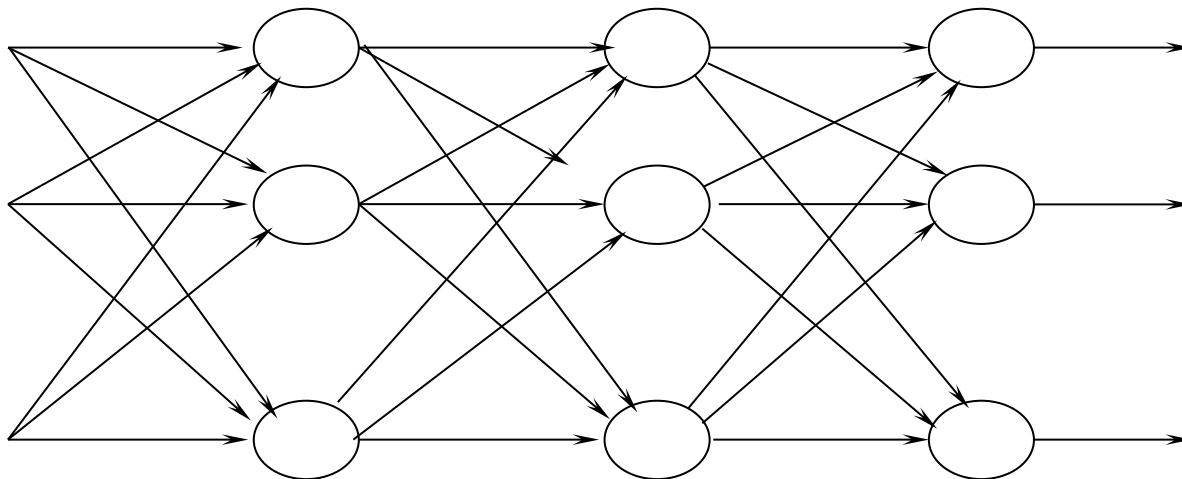


E.g: An ANN with single hidden layer

- Input: 3 signals
- Output: 2 signals
- Total, have 6 neurons
  - 4 neurons at hidden layer
  - 2 neurons at output layer

## ANN: Architecture (2)

- A layer (tầng) contains a set of neurons
- Hidden layer (tầng ẩn) is a layer between input layer and output layer
- Hidden nodes do not interact directly with external environment of the neural network
- An ANN is called a **fully connected** if outputs of a layer are connected to all neurons of the next layer



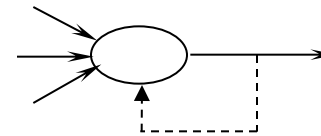
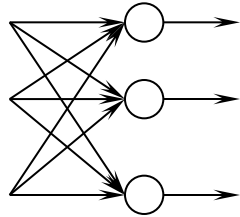
## ANN: Architecture (3)

---

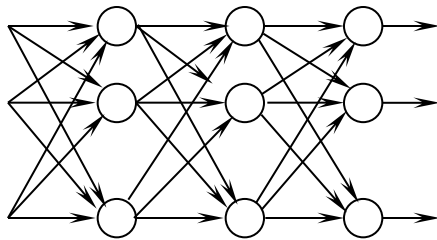
- An ANN is called a **feed-forward network** (mạng lan truyền tiến) if there is not any output of a node being input of another node of the same layer or a previous layer
- When the output of a node is the input of the node the same layer or a previous layer. It is called a **feedback network** (mạng phản hồi)
  - If feedback connects to the input of nodes of the same layer, then it is called a **lateral feedback**.
- Feedback networks with closed loops are called **recurrent networks** (mạng hồi quy)

# ANN: Architecture (4)

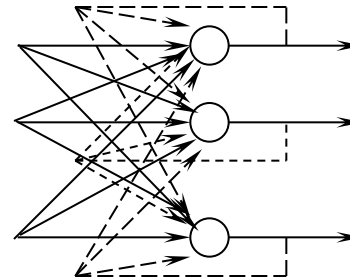
Feed-forward network



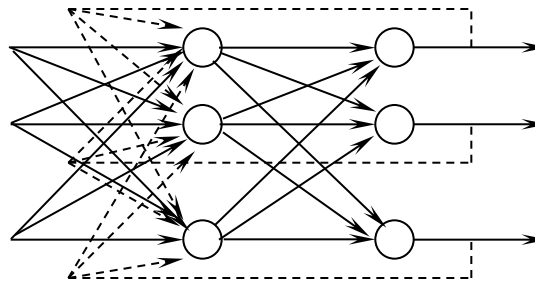
A neuron with feedback to itself



Feed-forward network with multiple layers



Recurrent network with single layer

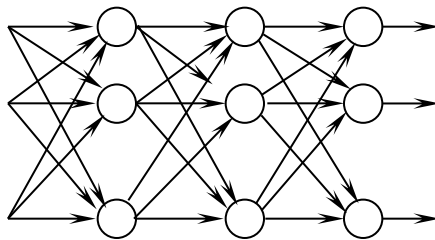


Recurrent network with multiple layers

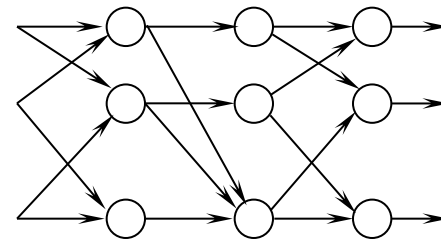


# ANN: Training

- 2 types of learning in ANNs
  - **Parameter learning:** The goal is to adapt the weights of the connections in the ANN, given a fixed network structure
  - **Structure learning:** The goal is to learn the network structure, including the number of neurons and the types of connections between them, and the weights



Or



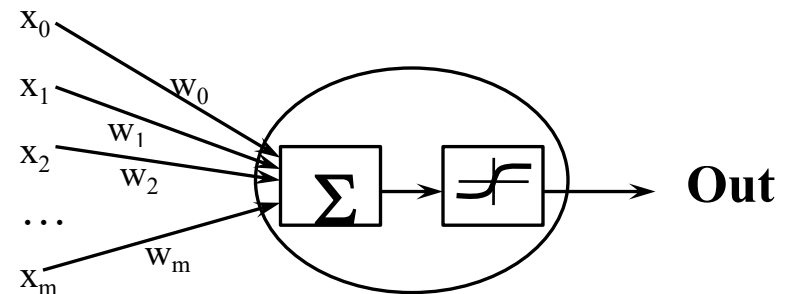
- Those two types can be done simultaneously or separately
- In this lecture, we will only consider parameter learning

# ANN: Idea for training

- Training a neural network (when fixing the architecture) is learning the weights  $\mathbf{w}$  of the network from training data  $\mathbf{D}$
- Learning can be done by minimizing an empirical error function

$$L(\mathbf{w}) = \frac{1}{|\mathbf{D}|} \sum_{\mathbf{x} \in \mathbf{D}} \text{loss}(d_{\mathbf{x}}, \text{out}(\mathbf{x}))$$

- Where  $\text{out}(\mathbf{x})$  is the output of the network, with the input  $\mathbf{x}$  labeled accordingly as  $d_{\mathbf{x}}$ ; loss is a function for measuring prediction error
- Many gradient-based methods:
  - Backpropagation
  - Stochastic gradient decent (SGD)
  - Adam
  - AdaGrad

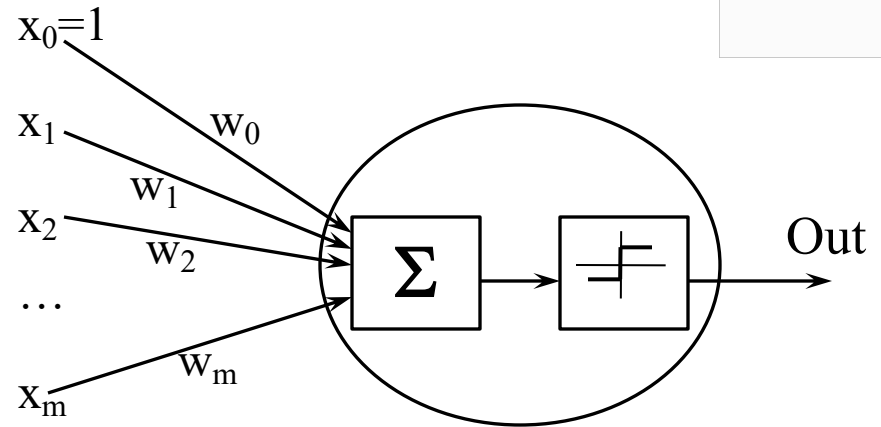


# Perceptron

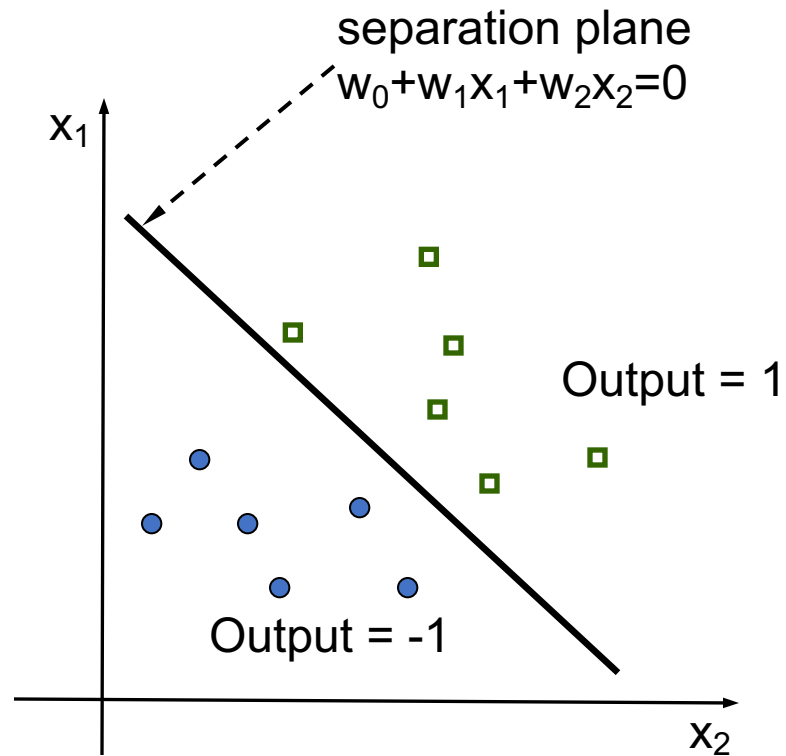
- A perceptron is the simplest type of ANNs (consists of only one neuron).
- Use the hard-limited activation function

$$Out = sign(Net(w, x)) = sign\left(\sum_{j=0}^m w_j x_j\right)$$

- For input  $\mathbf{x}$ , the output value of perceptron
  - 1 if  $Net(\mathbf{w}, \mathbf{x}) > 0$
  - -1 otherwise



# Perceptron: Illustration



# Perceptron: Algorithm

---

- Training data  $\mathbf{D} = \{(\mathbf{x}, d)\}$ 
  - $\mathbf{x}$  is input vector
  - $d$  is output (1 or -1)
- The goal of perceptron learning (training) process determines a weight vector that allows the perceptron to produce the correct output value (-1 or 1) for each data point
- For data point  $\mathbf{x}$  correctly classified by perceptron, the weight vector  $\mathbf{w}$  unchanged
- If  $d = 1$  but the perceptron produces -1 (Out = -1), then  $\mathbf{w}$  needs to be changed so that the value of Net ( $\mathbf{w}, \mathbf{x}$ ) increases
- If  $d = -1$  but the perceptron produces 1 (Out = 1), then  $\mathbf{w}$  needs to be changed so that the value of Net ( $\mathbf{w}, \mathbf{x}$ ) decreases

# Perceptron: Algorithm

---

**Perceptron\_batch( $\mathbf{D}$ ,  $\eta$ )**

Initialize  $\mathbf{w}$  ( $w_i \leftarrow$  an initial (small) random value)

do

$\Delta\mathbf{w} \leftarrow 0$

    for each training instance  $(\mathbf{x}, d) \in \mathbf{D}$

        Compute the real output value Out

        if (Out  $\neq$  d)

$\Delta\mathbf{w} \leftarrow \Delta\mathbf{w} + \eta(d - \text{Out})\mathbf{x}$

    end for

$\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w}$

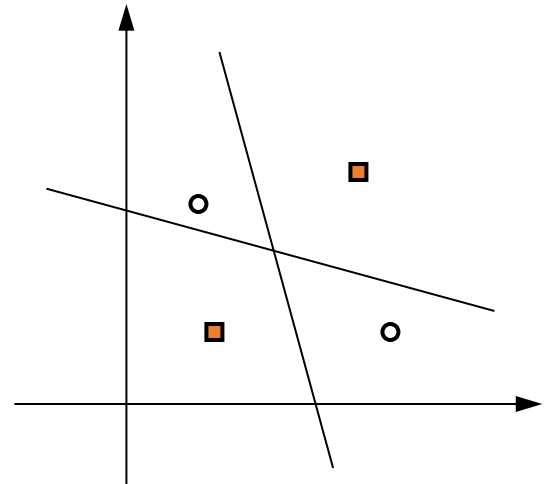
until all the training instances in  $\mathbf{D}$  are correctly classified

return  $\mathbf{w}$

# Perceptron: Limitation

- The training algorithm for perceptron is proved to converge if:
  - Data points are linearly separable
  - Use a learning rate  $\eta$  small enough
- The training algorithm for perceptron may not converge if data points are not linearly separable

A perceptron cannot classify correctly for this case!



# Loss function

---

- Consider an ANN that has  $n$  output neurons
- For data point  $(\mathbf{x}, d)$ , the **training error** value caused by the (current) weight vector  $\mathbf{w}$ :

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

- **Training error** for the training data  $\mathbf{D}$  is

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



# Minimize errors with gradients

- Gradient of E (denoted by  $\nabla E$ ) is a vector

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

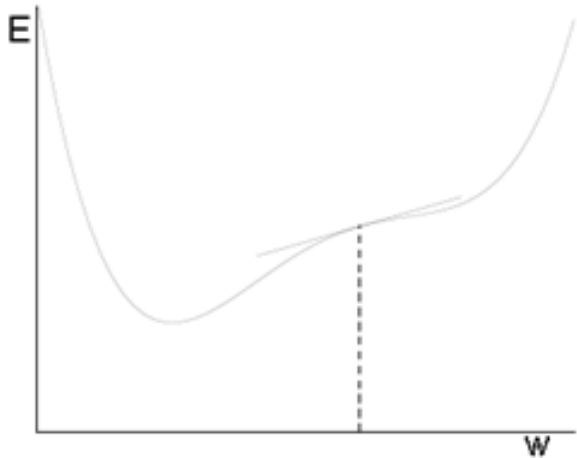
- where N is the total number of weights (connections) in the ANN
- The gradient  $\nabla E$  determines the direction that causes the **steepest increase** for the error value E
- Therefore, the direction that causes the **steepest decrease** is the opposite direction to the gradient of E

$$\Delta \mathbf{w} = -\eta \cdot \nabla E(\mathbf{w}); \quad \Delta w_i = -\eta \frac{\partial E}{\partial w_i} \text{ for } i = 1 \dots N$$

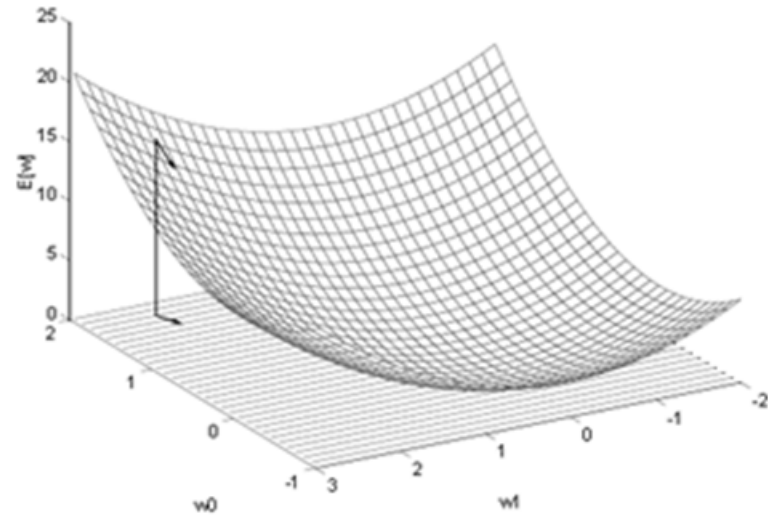
- Requirement: all the activation functions must be smoothed

# Gradient descent: Illustration

One-dimensional space  
 $E(w)$



2-dimensional space  
 $E(w_1, w_2)$



# Algorithm

## Gradient\_descent\_incremental ( $\mathbf{D}, \eta$ )

Initialize  $\mathbf{w}$  ( $w_i \leftarrow$  an initial (small) random value)

do

for each training instance  $(\mathbf{x}, d) \in \mathbf{D}$

Compute the network output

for each weight component  $w_i$

$$w_i \leftarrow w_i - \eta (\partial E_{\mathbf{x}} / \partial w_i)$$

end for

end for

until (stopping criterion satisfied)

return  $\mathbf{w}$

Stopping criterion: epochs, threshold error, ...

If we take a small *subset* (mini-batch) *randomly* from  $\mathbf{D}$  to update the weights, we will have mini-batch training.

# Backpropagation algorithm

---

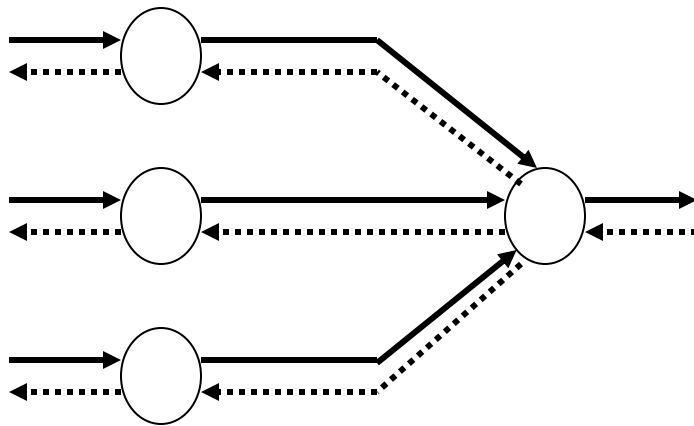
- A perceptron can only represent a linear function
- A multi-layer NN learned by the **Backpropagation** (BP) algorithm can represent a highly non-linear function
- The BP algorithm is used to learn the weights of an ANN
  - Fixed network structure (neurons and the connections between them are fixed)
  - For each neuron, the activation function must be smoothed
- The BP algorithm applies a *gradient descent* strategy to the rules for updating weights
  - To minimize errors between actual output values and desired output values, for training data

# Backpropagation algorithm (1)

---

- Back propagation algorithm seeks a vector of weights that minimizes the net errors of ANN with training data
- The BP algorithm consists of 2 phases:
  - **Forward pass:** The input signals (input vector) are forwarded from the input layer to the output layer (passing through hidden layers).
  - **Error backward:**
    - Based on the desired output value of the input vector, the system calculates the error value
    - From the output layer, the error value is backward-propagated across the network, from layer to another layer (front), to the input layer.
    - Error back-propagation is executed by calculating (regressively) the local gradient values of each neuron

# Backpropagation algorithm (2)



Signal forward phase:

- Forward signals via the network

Error backward phase:

- Calculate the error at the output
- Error back-propagation

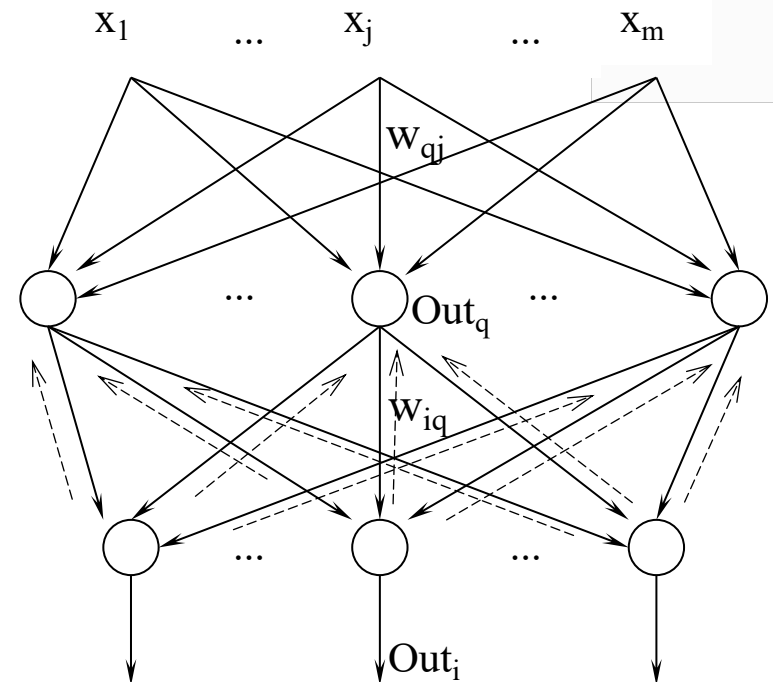
# Network structure

- Consider the 3-layer neural network (in the figure) to illustrate the BP algorithm
- $m$  input signals  $x_j$  ( $j=1..m$ )
- $l$  hidden neurons  $z_q$  ( $q=1..l$ )
- $n$  output neurons  $y_i$  ( $i=1..n$ )
- $w_{qj}$  is the weight of the connection from the input signal  $x_j$  to the hidden neuron  $z_q$

Input  $x_j$   
( $j=1..m$ )

Hidden  
neuron  $z_q$   
( $q=1..l$ )

Output  
neuron  $y_i$   
( $i=1..n$ )



- $w_{iq}$  is the weight of the connection from the hidden neuron  $z_q$  to the output  $y_i$
- $Out_q$  is the (local) output value of the hidden neuron  $z_q$
- $Out_i$  is the output value of the network corresponding to the output neuron  $y_i$

## BP algorithm: Forward (1)

- For each data point  $\mathbf{x}$ 
  - Input vector  $\mathbf{x}$  is forwarded from the input layer to the output layer
  - The network will generate an actual output value **Out** (a vector with value  $Out_i$ ,  $i = 1..n$ )
- For an input vector  $\mathbf{x}$ , a neuron  $z_q$  at the hidden layer receives the value of net input equal to:

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

then produce an (local) output value equal to

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

where  $f(.)$  is a activation function of neuron  $z_q$



## BP algorithm: Forward (2)

- Net input value of the neuron  $y_i$  at the output layer

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

- Neuron  $y_i$  produces output value (is an output value of network)

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

- Vector of the output values  $Out_i$  ( $i=1..n$ ) is the actual output value of the network, for the input vector  $\mathbf{x}$

## BP algorithm: Backward (1)

- For each data point  $\mathbf{x}$ 
  - Error signals due to the difference between the desired output value  $d$  and the actual output value **Out** are calculated
  - These error signals are **back-propagated** from the output layer to the front layers, to update weights
- To consider the error signals and their back-propagated, an error function needs to be defined

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

## BP algorithm: Backward (2)

- According to the gradient-descent method, the weights of the connections 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  $\partial E / \partial w_{iq}$ , we have

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

- $\delta_i$  is **error signals** of neuron  $y_i$  at output layer

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

where  $Net_i$  is the net input of the neuron  $y_i$  at the output layer,  
and  $f'(Net_i) = \partial f(Net_i) / \partial Net_i$

## BP algorithm: Backward (3)

- To update the weights of the connections 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: Backward (4)

- Apply the derivation chain rule, we have

$$\begin{aligned}\Delta w_{qj} &= \eta \sum_{i=1}^n [(d_i - Out_i) f'(Net_i) w_{iq}] f'(Net_q) x_j \\ &= \eta \sum_{i=1}^n [\delta_i w_{iq}] f'(Net_q) x_j = \eta \delta_q x_j\end{aligned}$$

- $\delta_q$  is **error signals** of neuron  $z_q$  at hidden layer

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

where  $Net_q$  is the net input of the neuron  $z_q$  at the hidden layer,  
and  $f'(Net_q) = \partial f(Net_q) / \partial Net_q$

## BP algorithm: Backward (5)

---

- According to the formulas for calculating the error signals  $\delta_i$  and  $\delta_q$ , 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 BP algorithm is also known as general delta learning rule
- Error signals  $\delta_q$  of neuron  $z_q$  at hidden layer determined by:
  - Error signals  $\delta_i$  of neuron  $y_i$  at output layer (to which neuron  $z_q$  are connected)
  - The weights  $w_{iq}$

## BP algorithm: Backward (6)

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

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

- $b$  and  $a$  are 2 indices corresponding to the two ends of the connect ( $b \rightarrow a$ ) (from a neuron (or input signal)  $b$  to neuron  $a$ )
- $x_b$  is the output value of the neuron at the hidden layer (or input signal)  $b$
- $\delta_a$  is error signal of neuron  $a$

# BP algorithm

## Back\_propagation\_incremental( $\mathbb{D}, \eta$ )

Neural network consists of  $Q$  layer,  $q = 1, 2, \dots, Q$

${}^qNet_i$  and  ${}^qOut_i$  are net input and output value of neuron  $i$  at the layer  $q$

Network has  $m$  input signals and  $n$  output neuron

${}^qw_{ij}$  is the weight of the connection from neuron  $j$  at the layer  $(q-1)$  to the neuron  $i$  at the layer  $q$

### Step 0 (Initialization)

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

Initialize the initial value of the weights with random small values

Assign  $E=0$

### Step 1 (Start a training cycle)

Apply the input vector of the data point  $k$  to the input layer ( $q=1$ )

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

### Step 2 (Forward)

Forward the input signals over the network, until the network output values (at the output layer) are received  ${}^QOut_i$

$${}^qOut_i = f({}^qNet_i) = f\left(\sum_j {}^qw_{ij} {}^{q-1}Out_j\right)$$



# BP algorithm

## Step 3 (Calculate the output error)

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

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

$${}^Q\delta_i = (d_i^{(k)} - {}^QOut_i) f'({}^QNet_i)$$

## Step 4 (Error backward)

Backpropagation the error to update the weights and calculate the error signals  ${}^{q-1}\delta_i$  for the front layers

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

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

## Step 5 (Check stopping criterion satisfied)

Check if the entire training data has been used yet

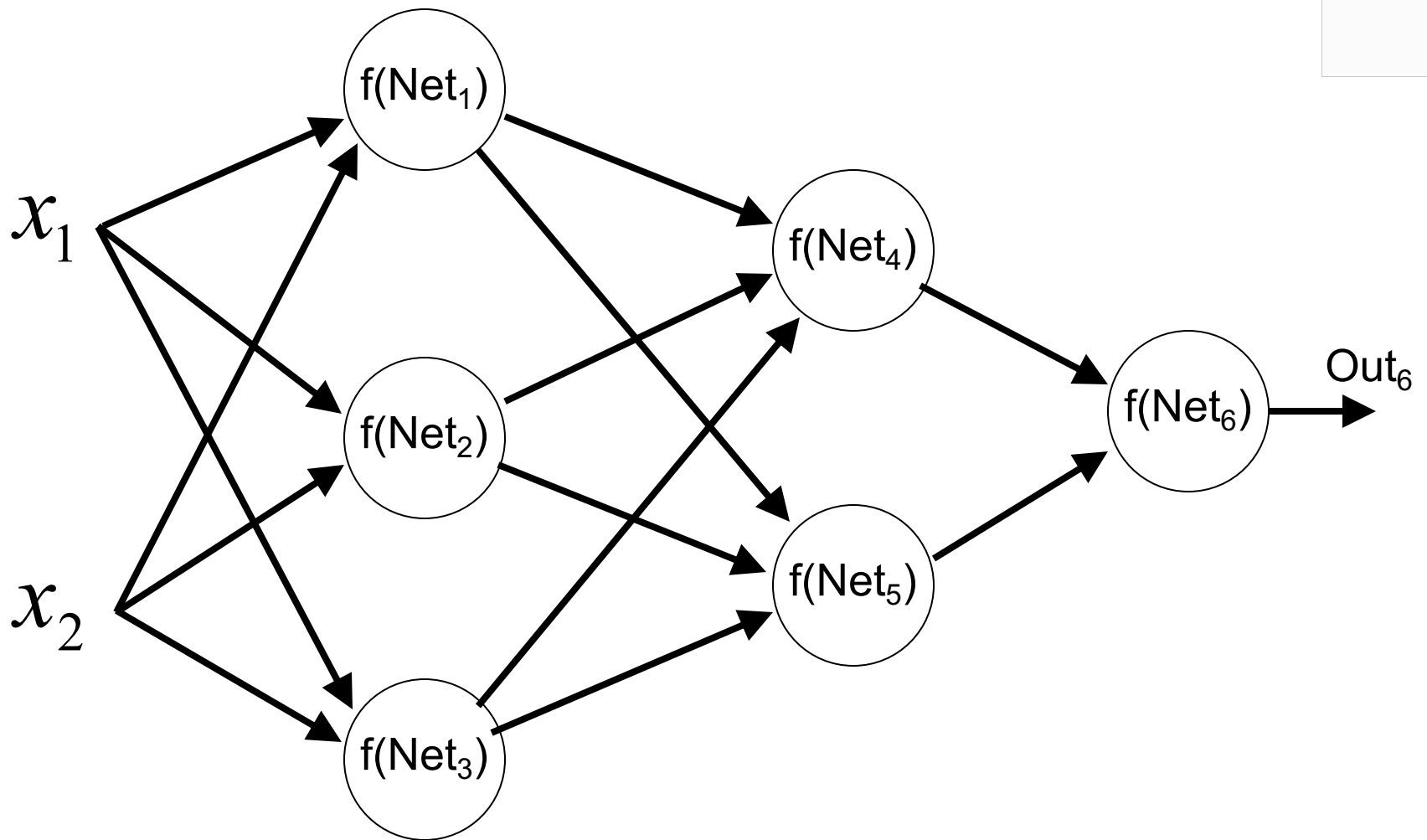
If the entire training data has used, go to Step 6, otherwise go to Step 1

## Step 6 (Check net error)

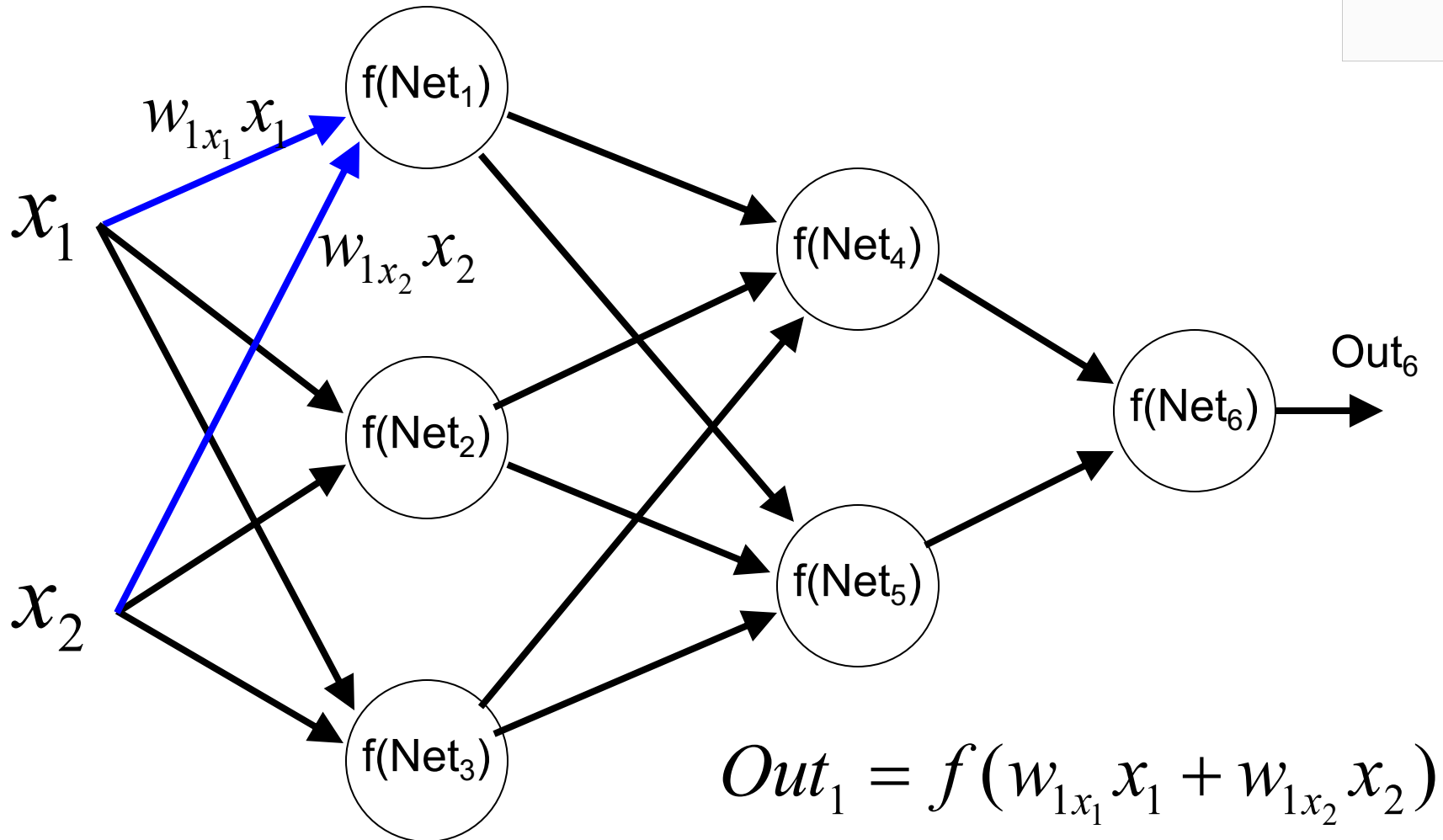
If net error E is less than the acceptable threshold ( $< E_{\text{threshold}}$ ), then training is completed and returns the learned weights;

otherwise, assign  $E=0$ , and start new training cycle (go back to Step 1)

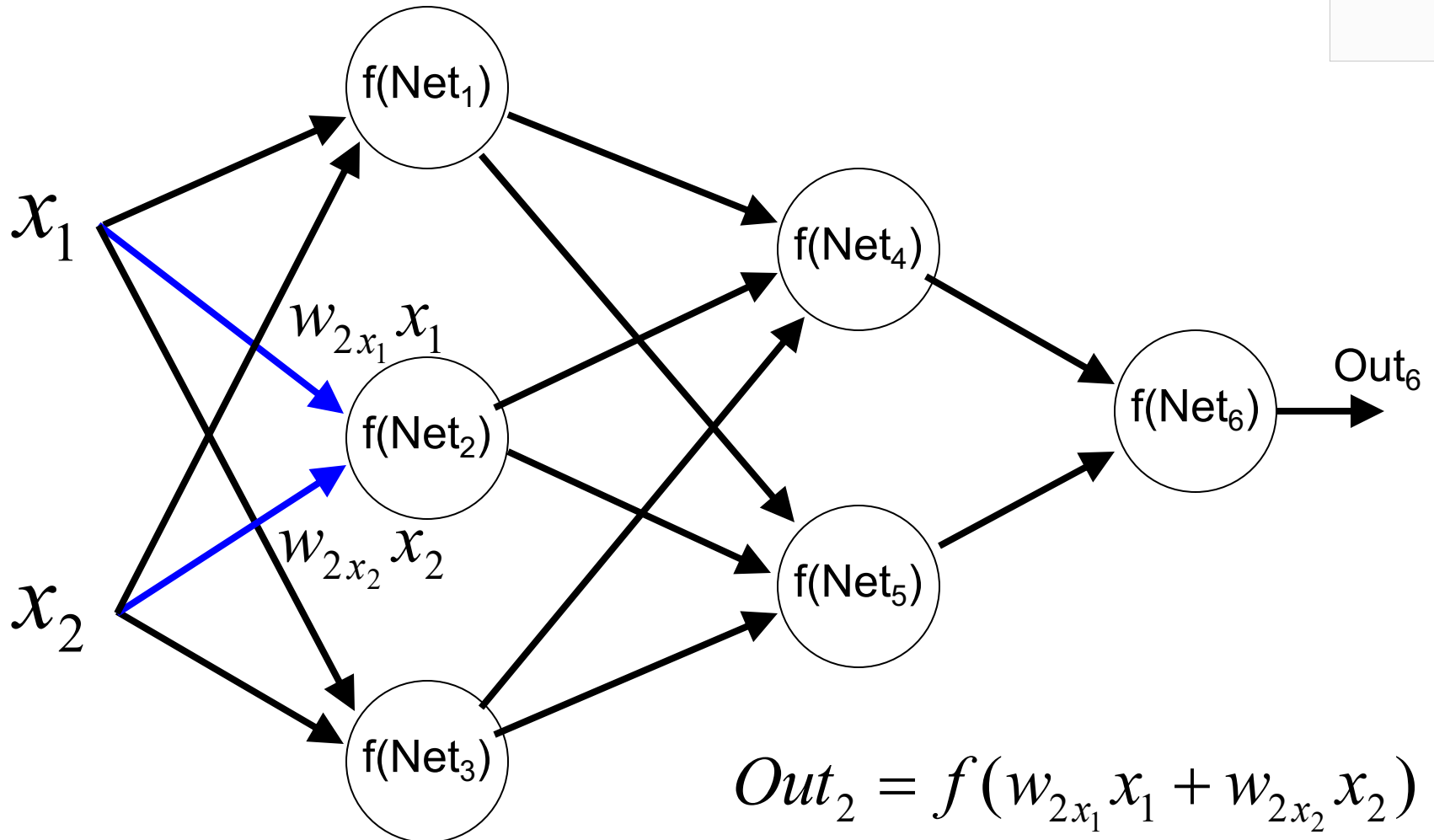
## BP algorithm: Forward (1)



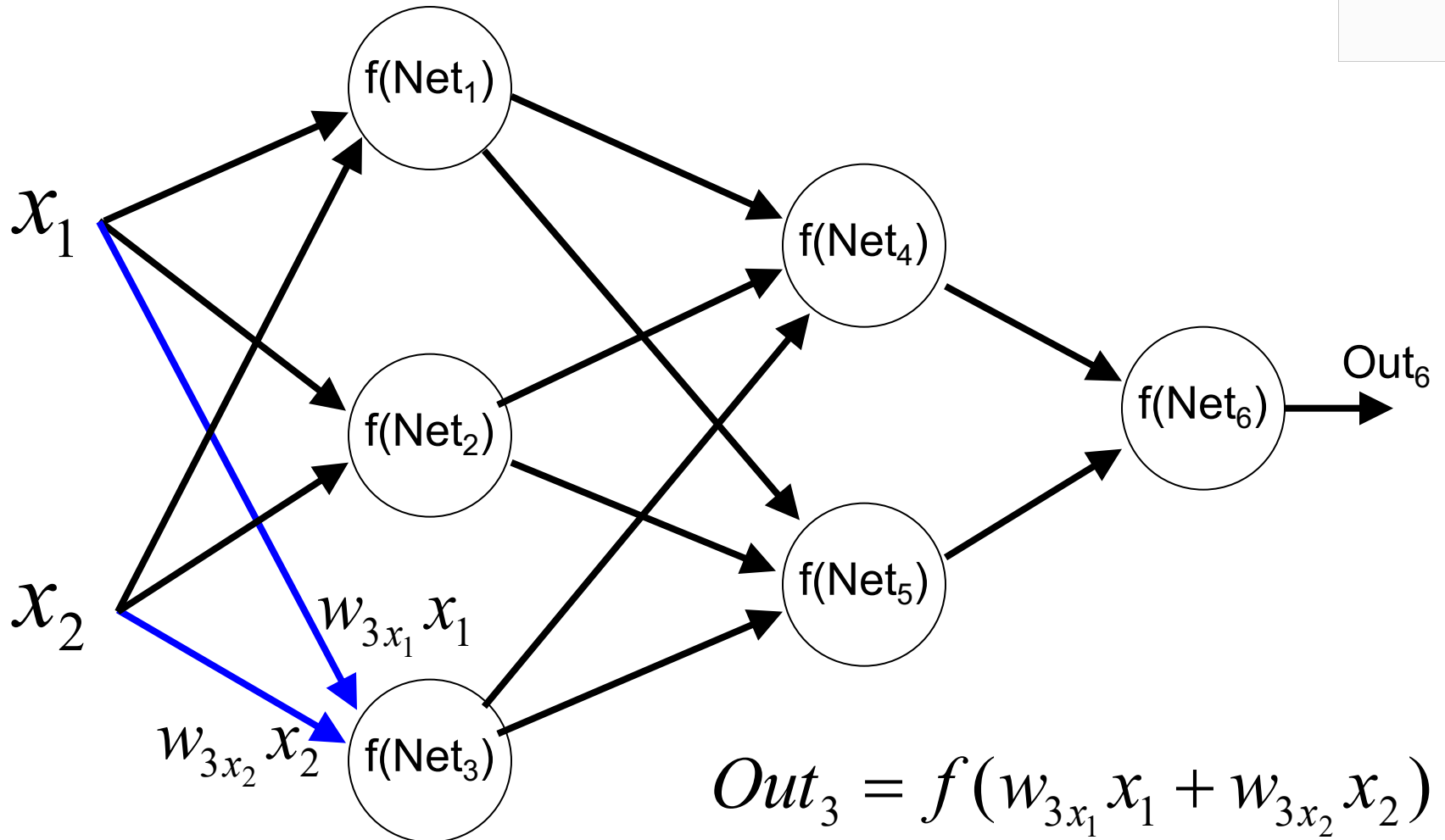
## BP algorithm: Forward (2)



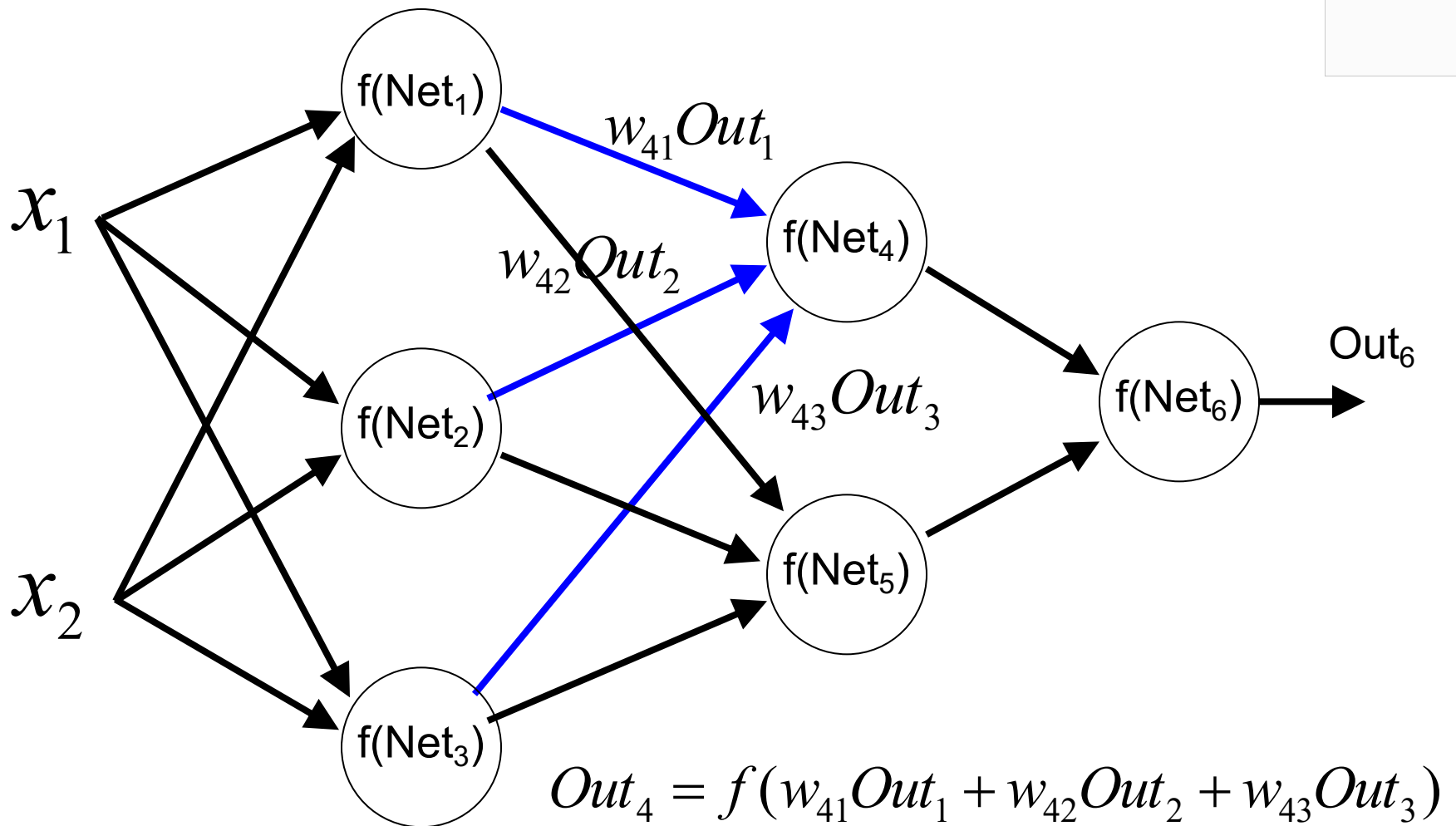
## BP algorithm: Forward (3)



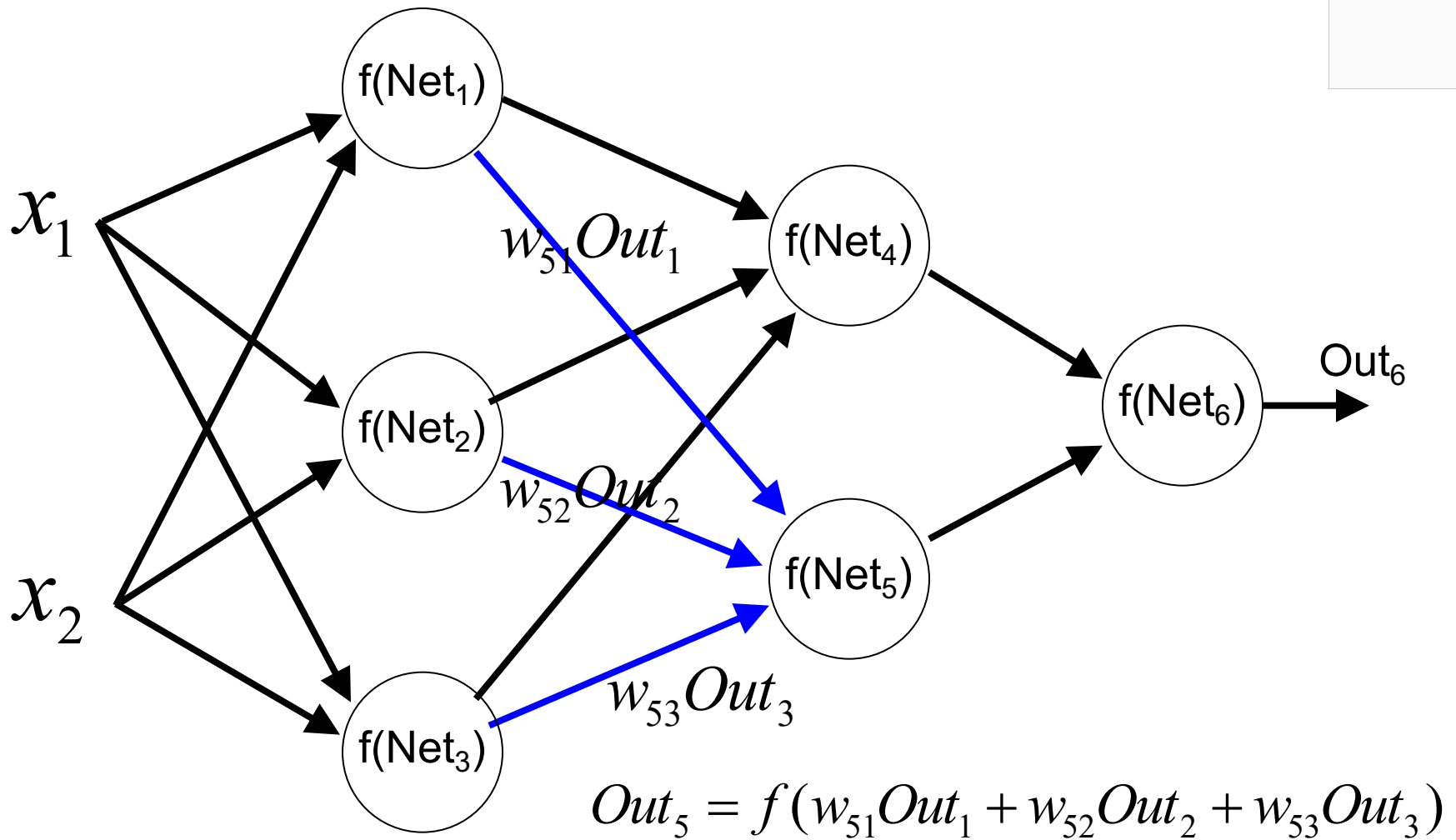
## BP algorithm: Forward (4)



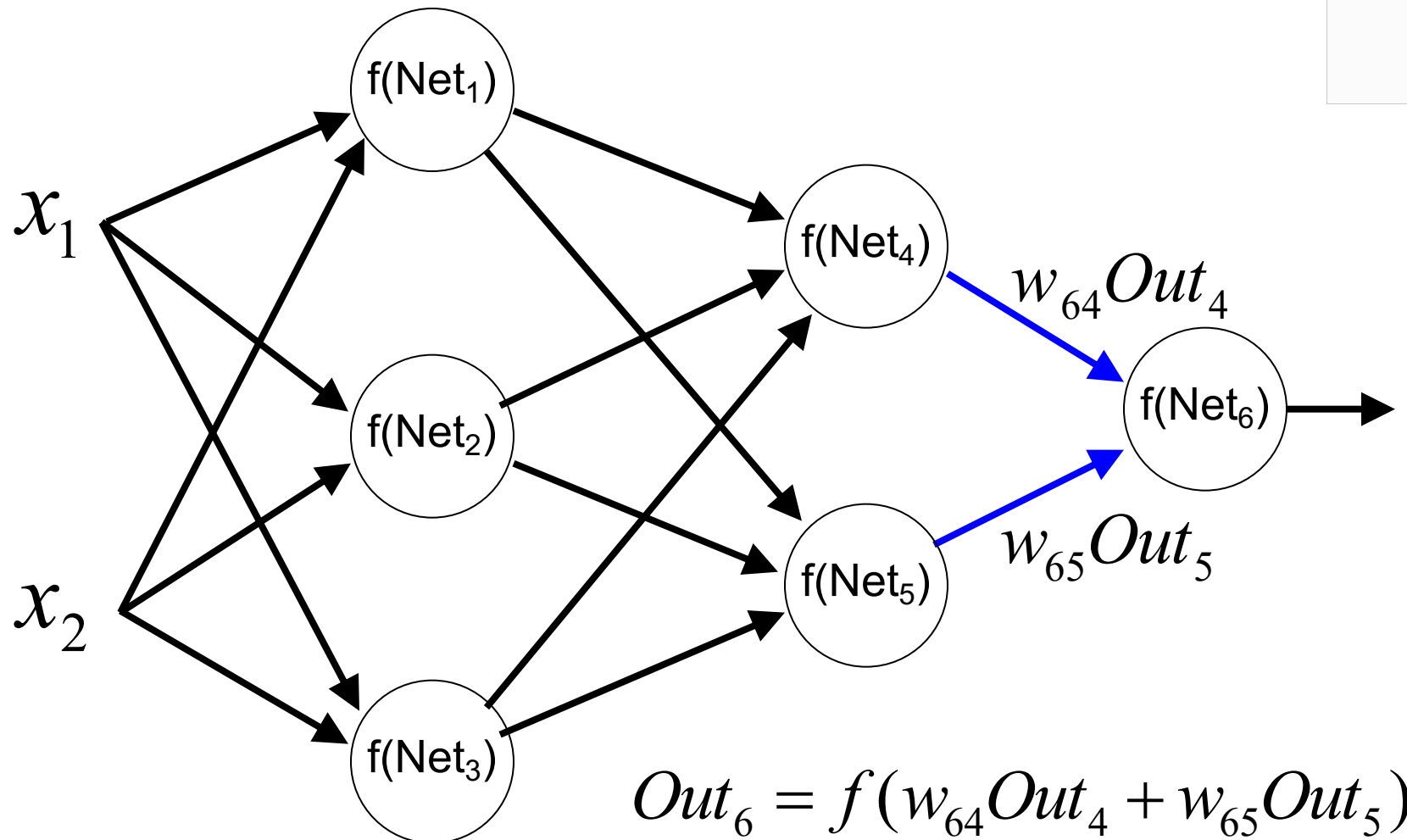
## BP algorithm: Forward (5)



## BP algorithm: Forward (6)

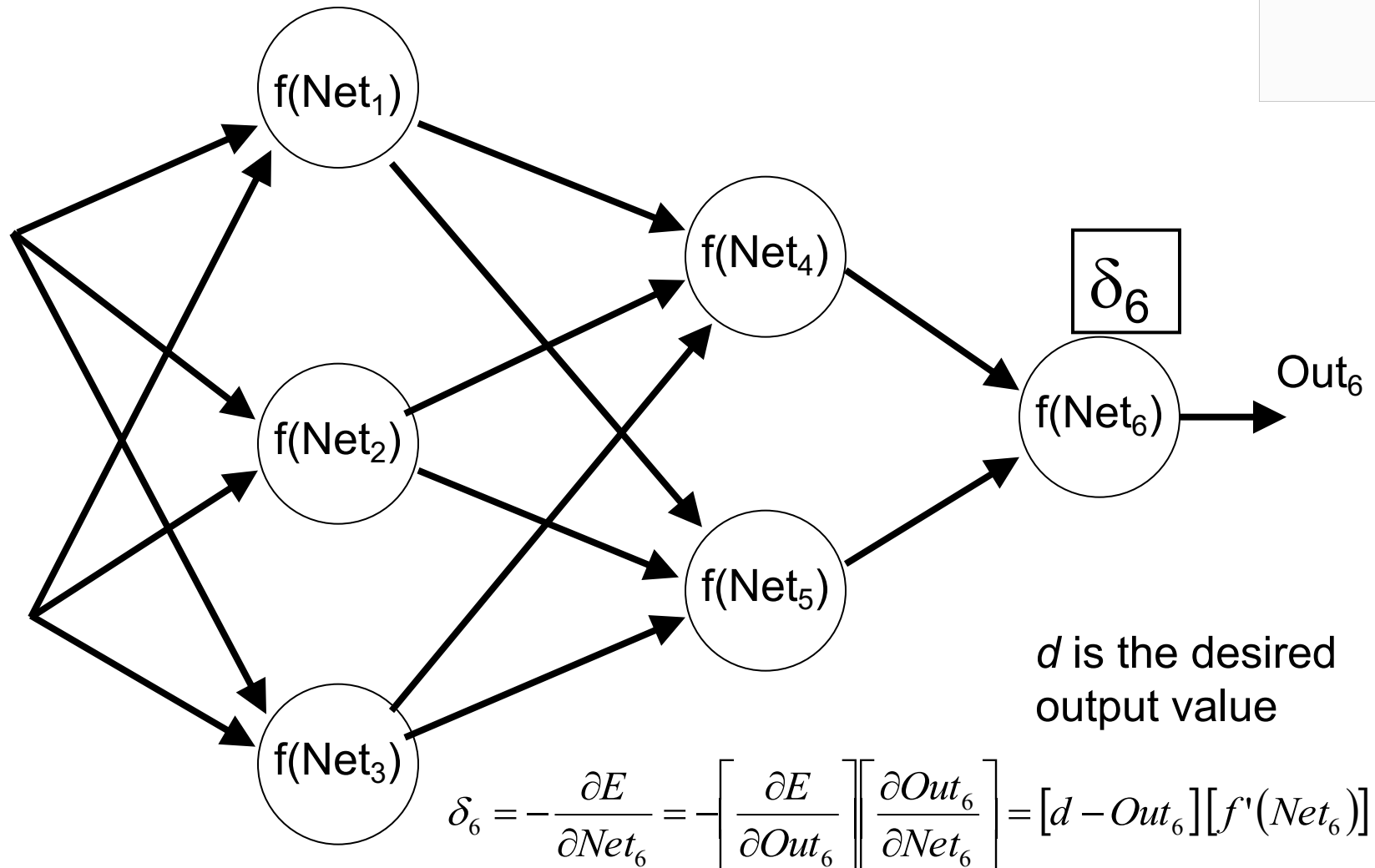


## BP algorithm: Forward (7)

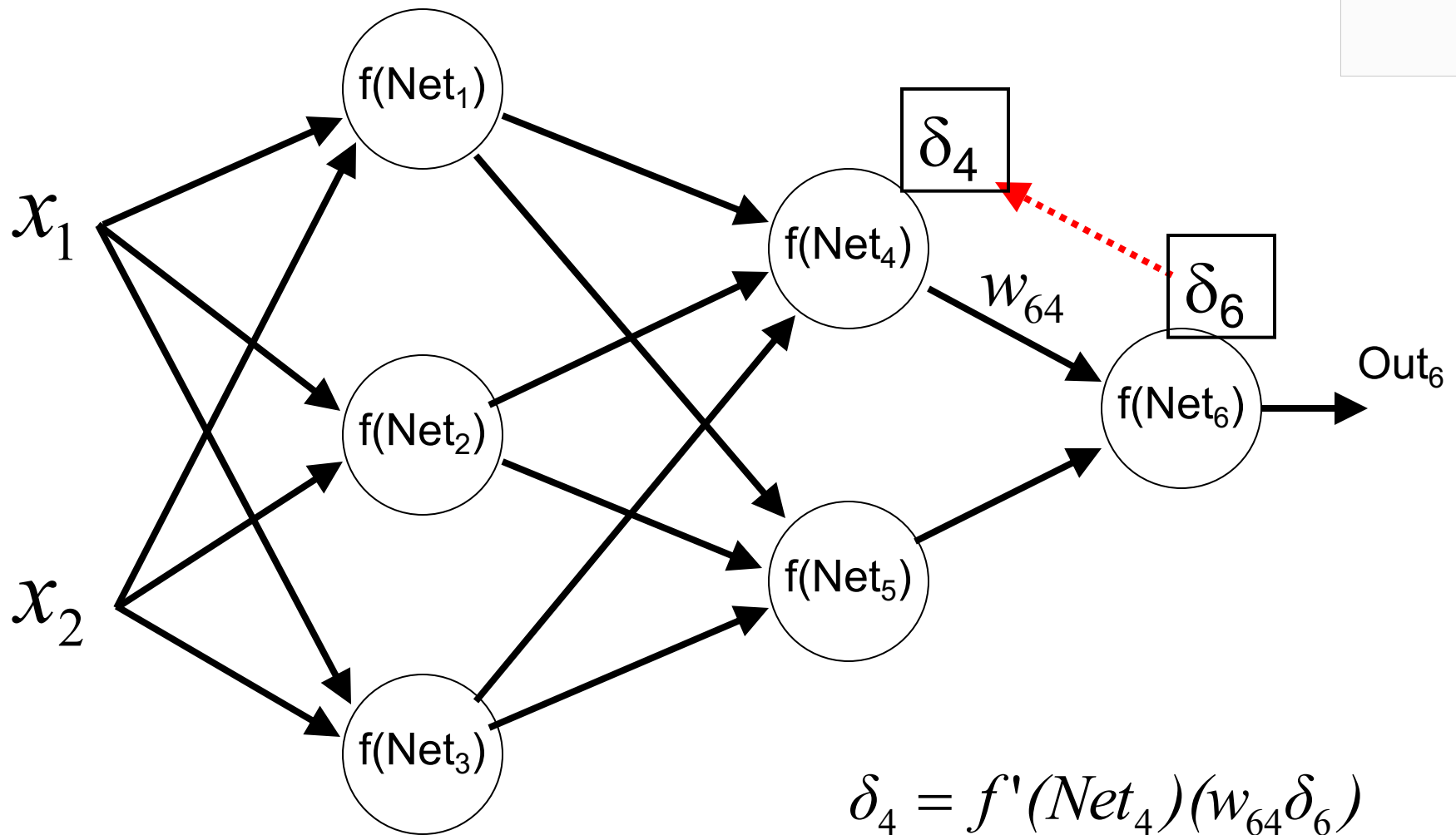




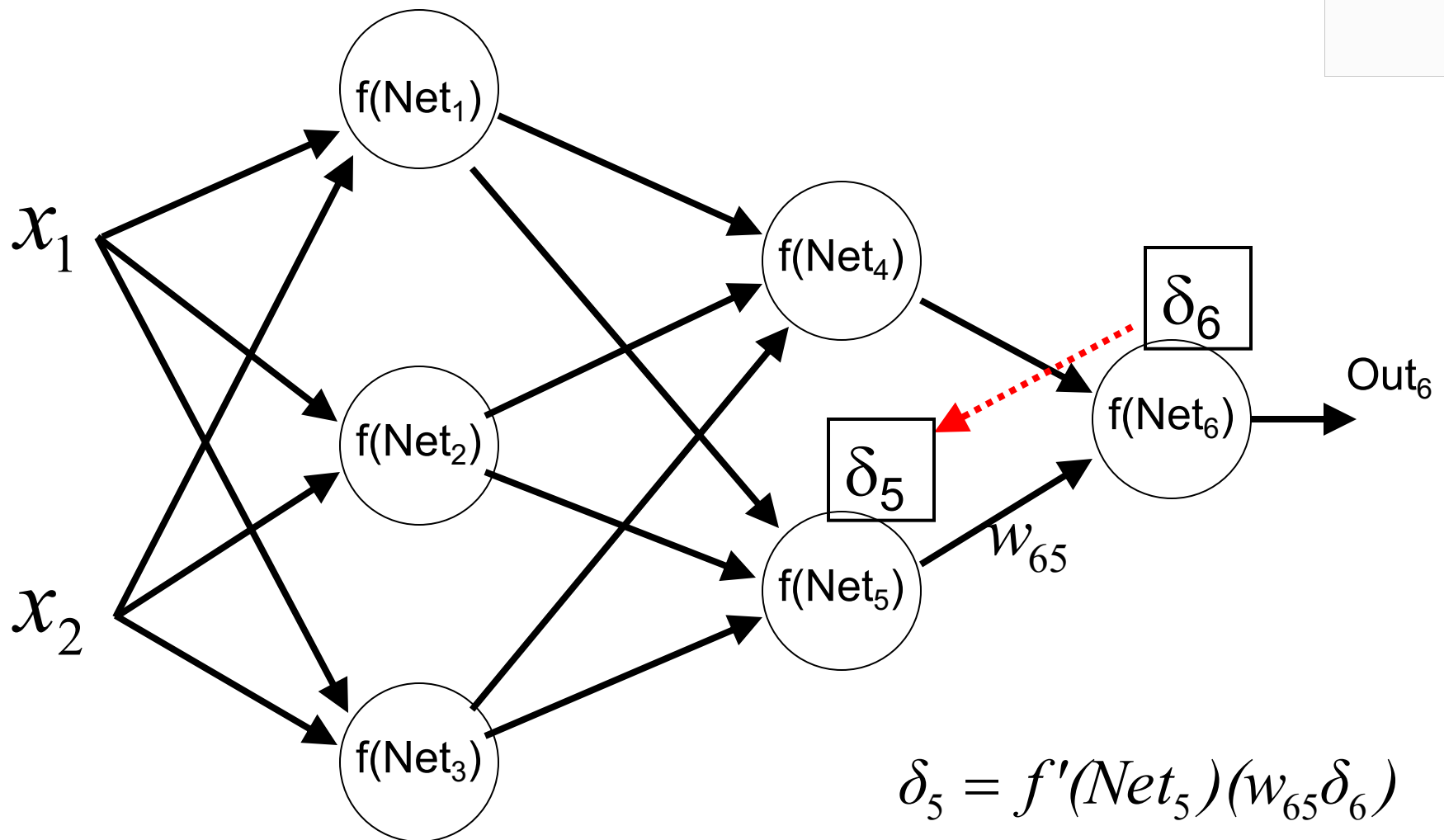
# BP algorithm: Calculate error



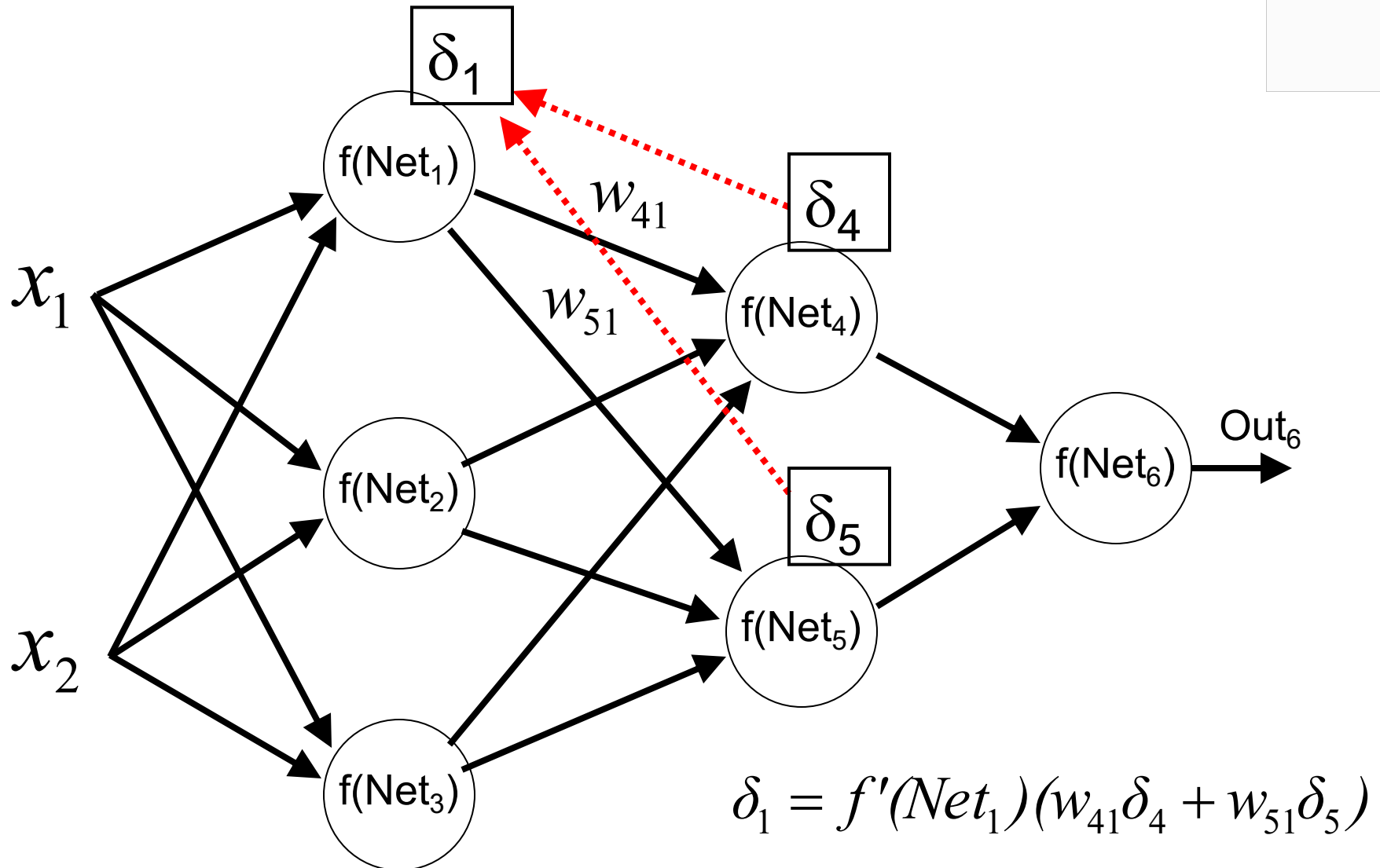
# BP algorithm: Backward(1)



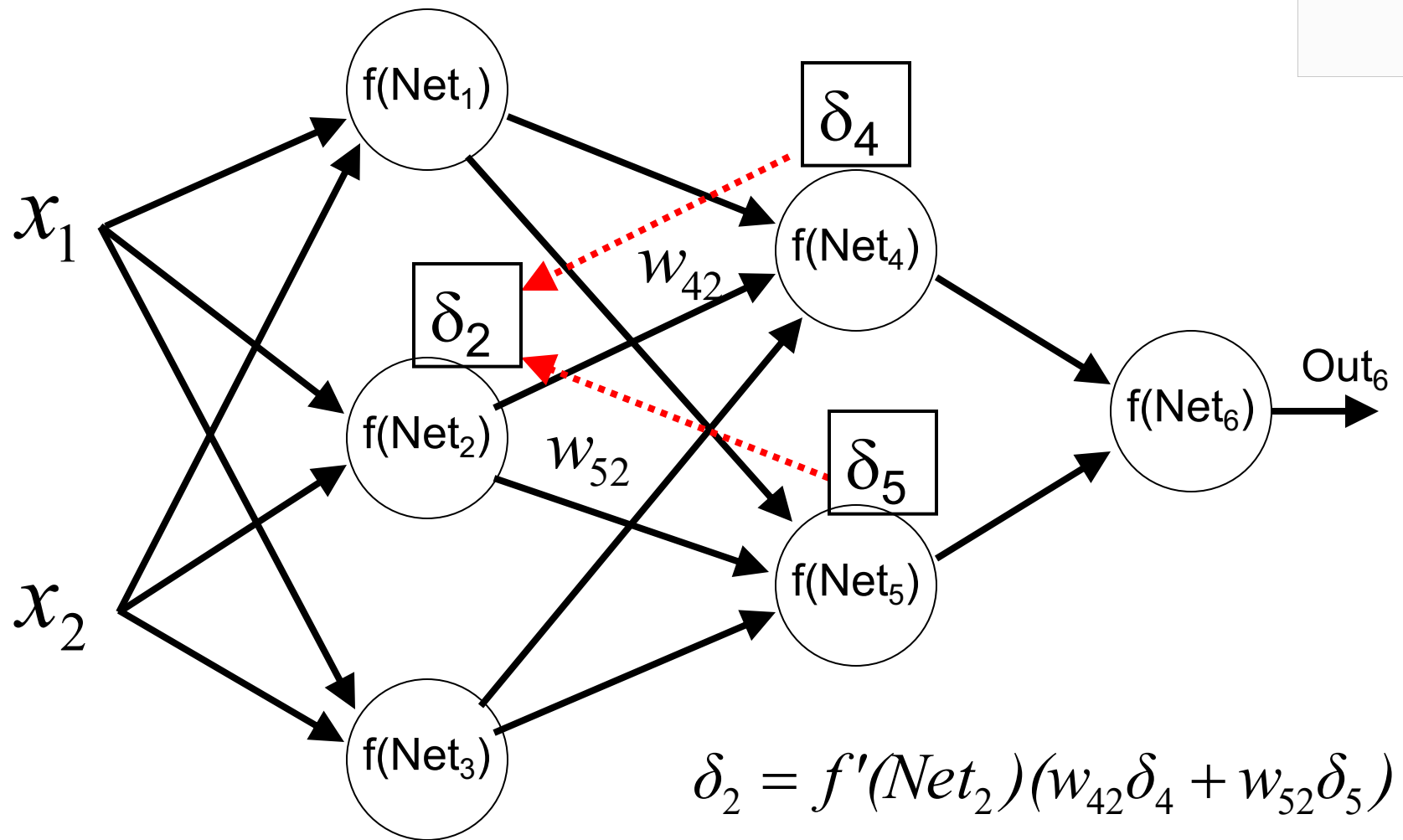
## BP algorithm: Backward(2)



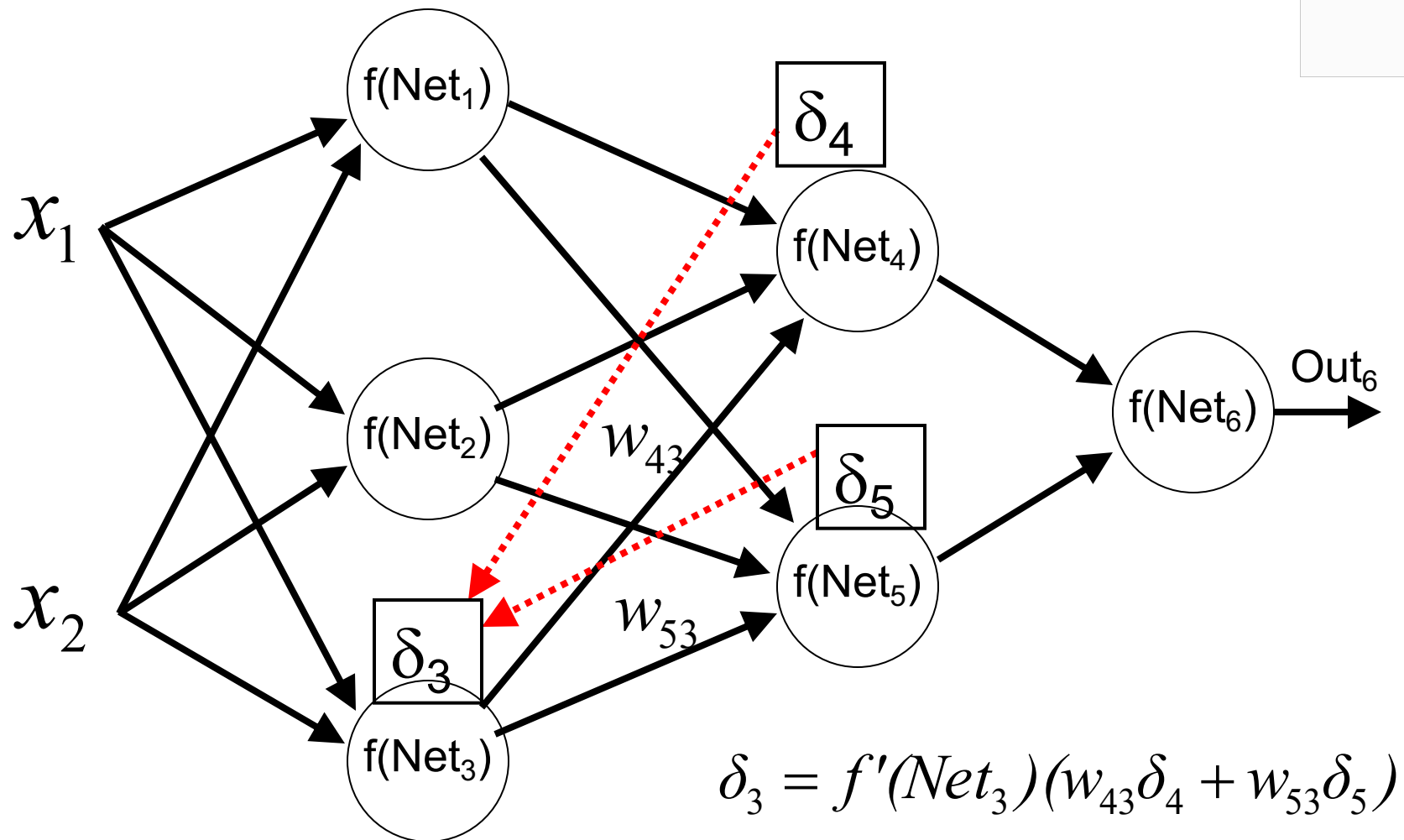
## BP algorithm: Backward(3)



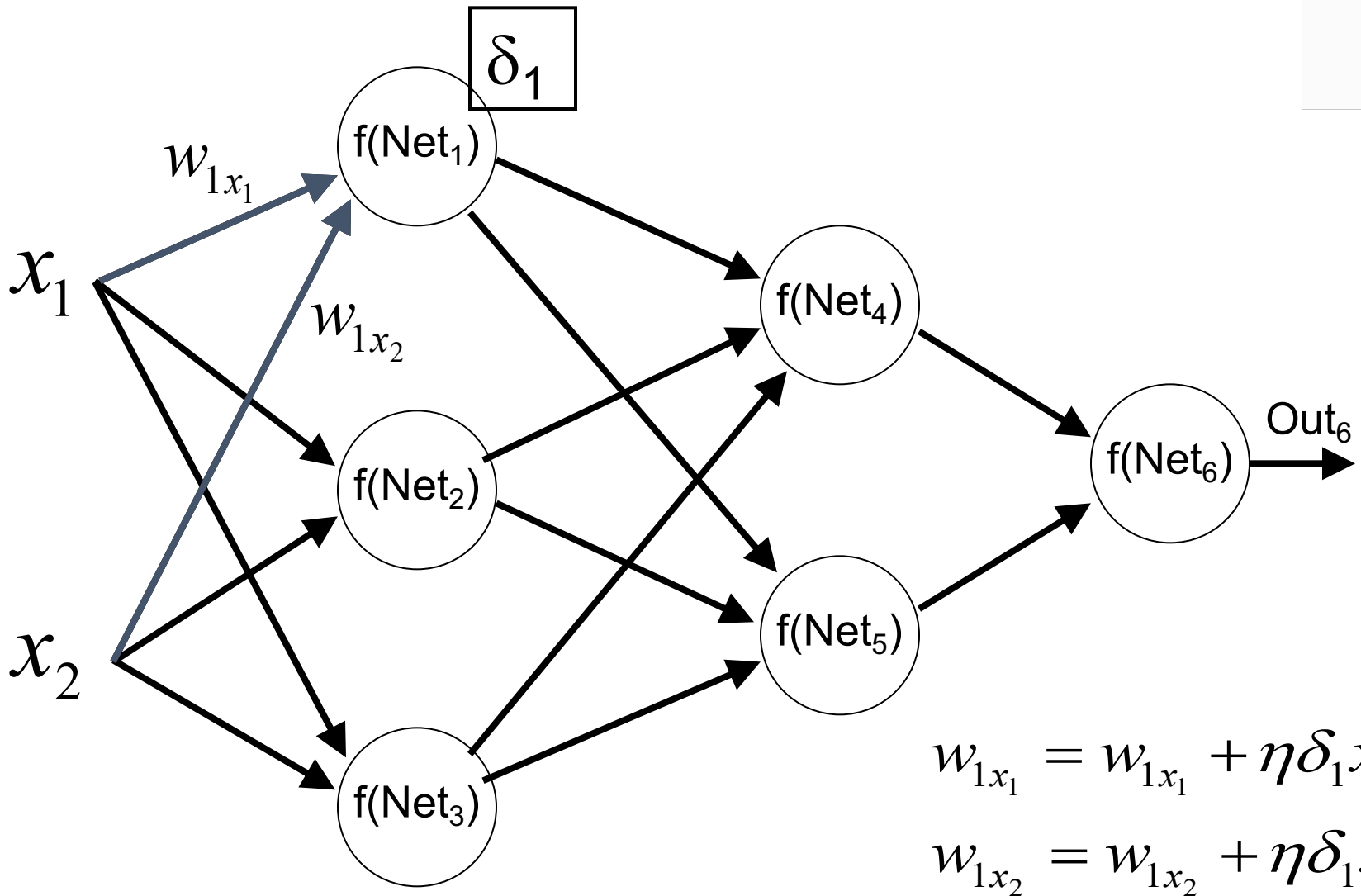
## BP algorithm: Backward(4)



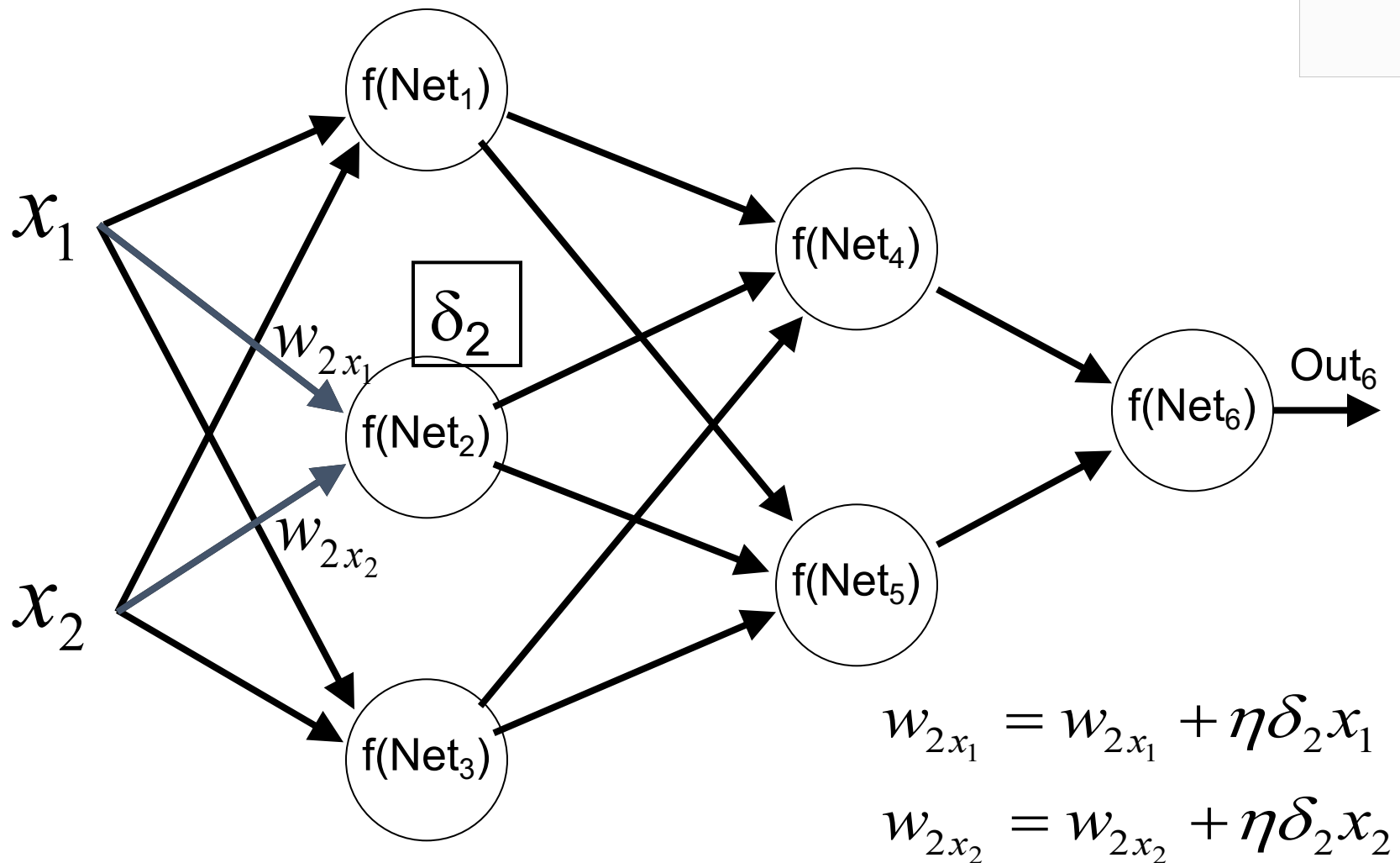
## BP algorithm: Backward(5)



## BP algorithm: Update weight(1)

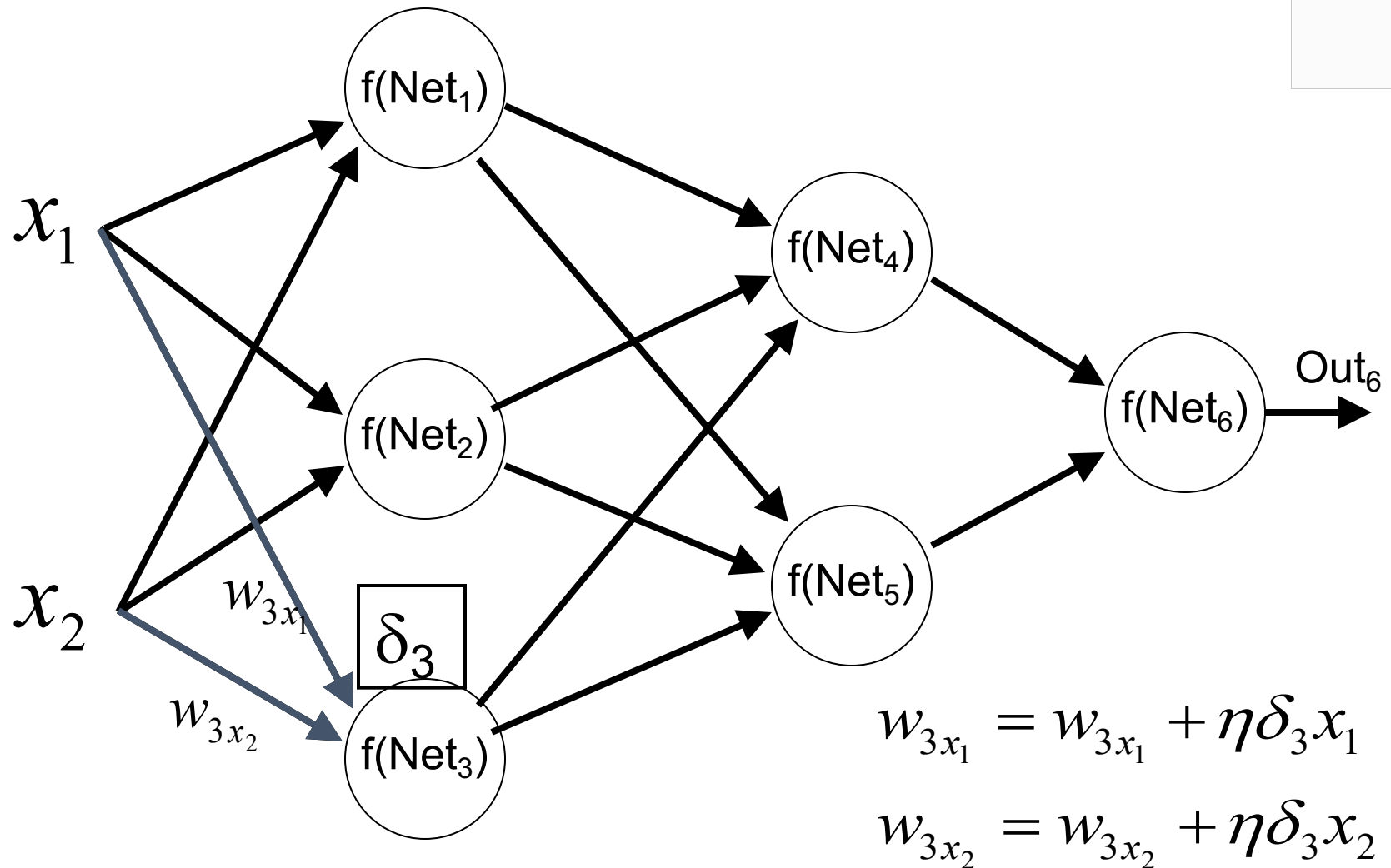


## BP algorithm: Update weight(2)

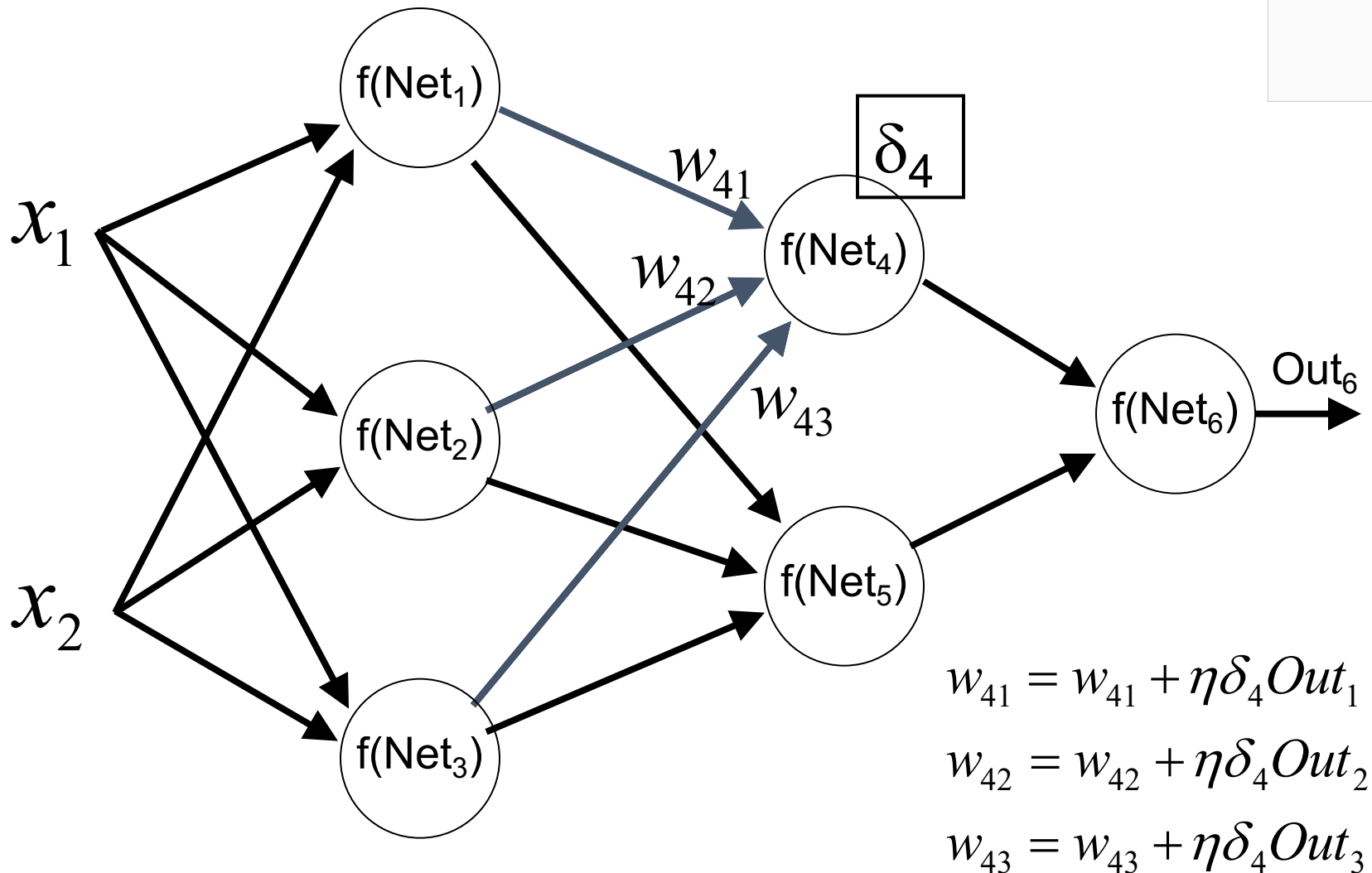




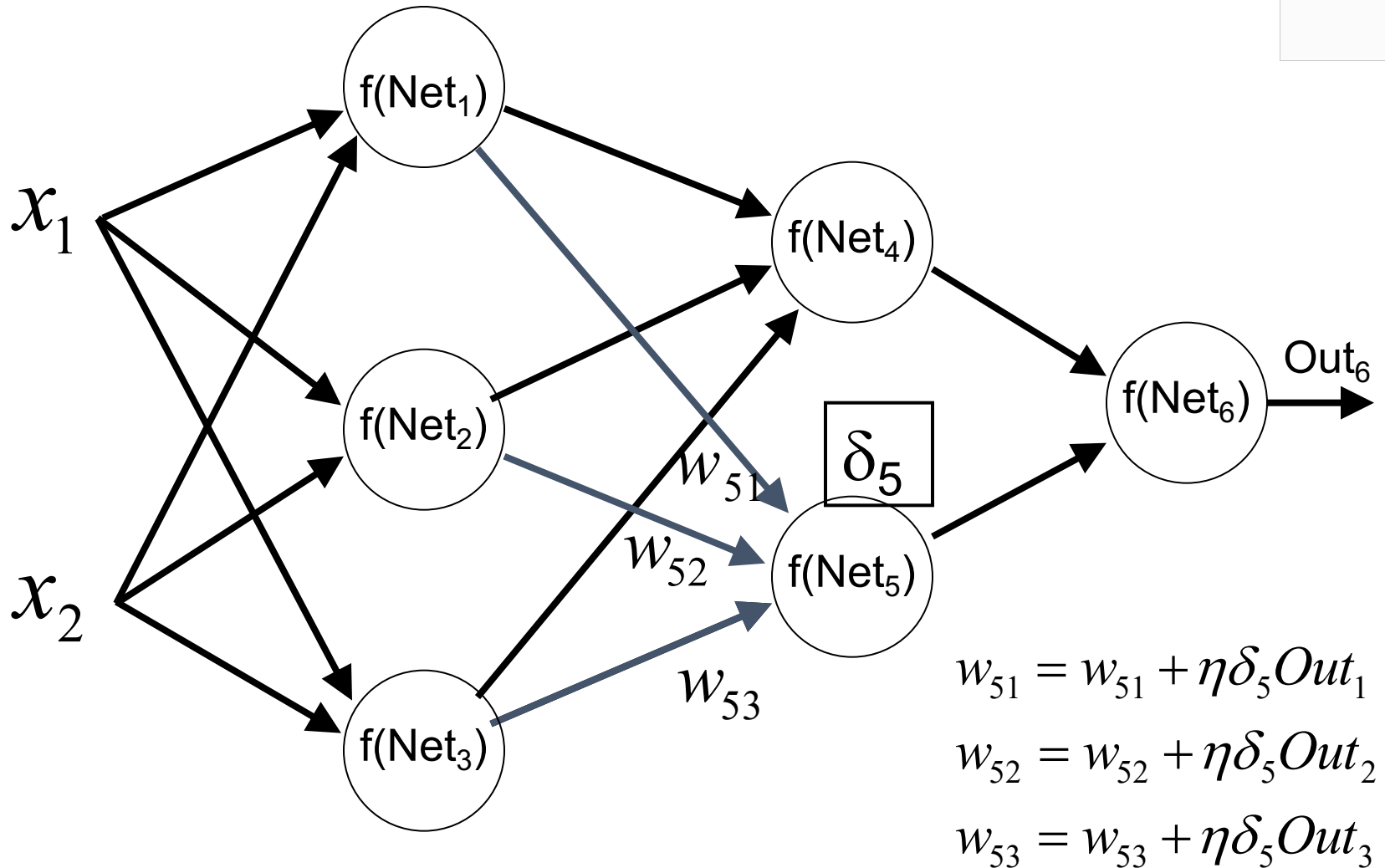
## BP algorithm: Update weight(3)



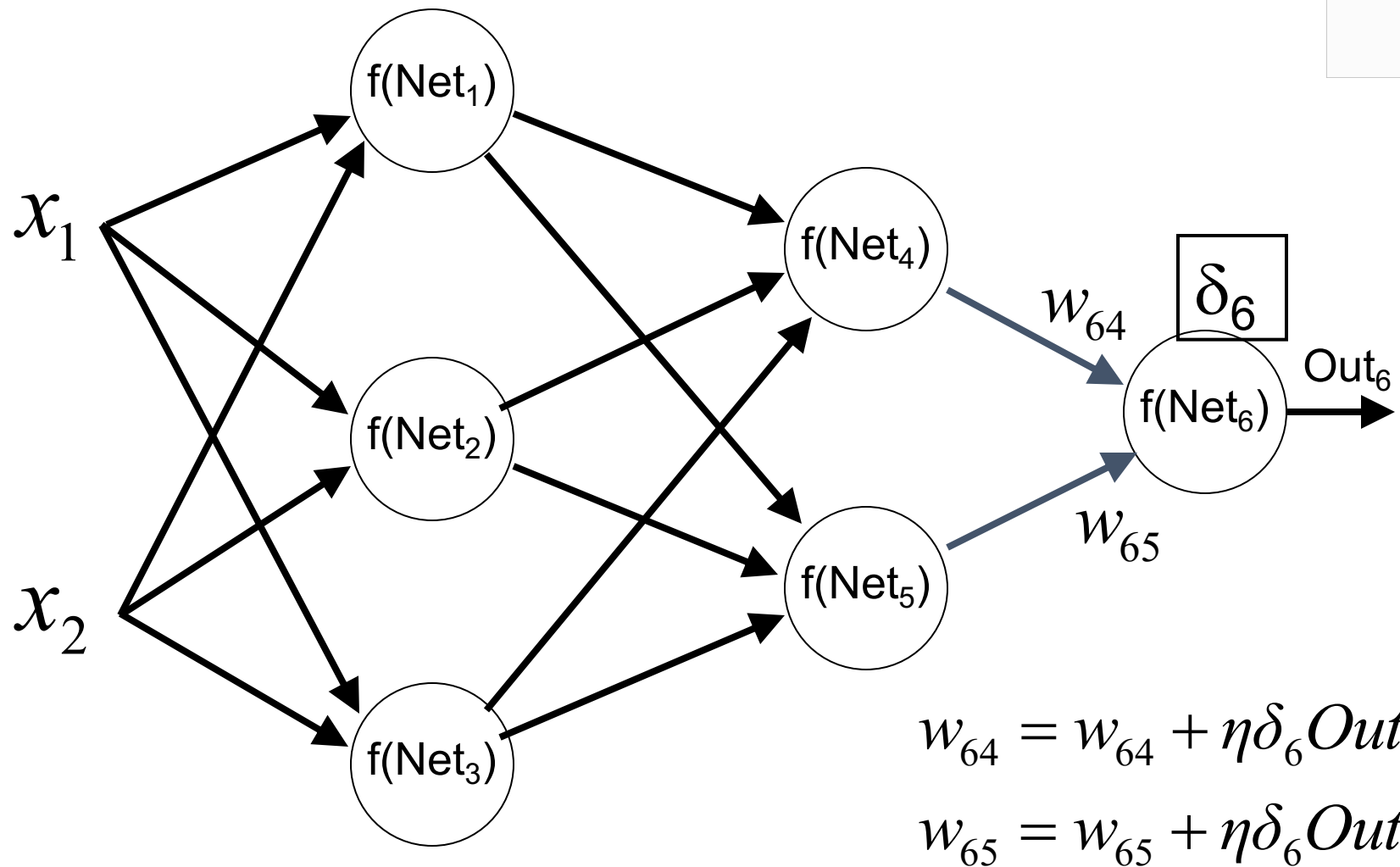
## BP algorithm: Update weight(4)



# BP algorithm: Update weight(5)



## BP algorithm: Update weight(6)



## BP algorithm: Initialize weights

---

- Normally, weights are initialized with random small values
- If the weights have large initial values
  - Sigmoid functions will reach saturation soon
  - The system will deadlock at a saddle / stationary points

## BP algorithm: Learning rate

---

- Important effect on the efficiency and convergence of BP algorithm
  - A large value of  $\eta$  can accelerate the convergence of the learning process, but can cause the system to ignore the global optimal point or focus on bad points (saddle points).
  - A small  $\eta$  value can make the learning process take a long time
- Often select according to experimentally for each problem
- Good values of learning rate at the beginning (learning process) may not be good at a later time
  - Using an adaptive (dynamic) learning rate?

# BP algorithm: Momentum

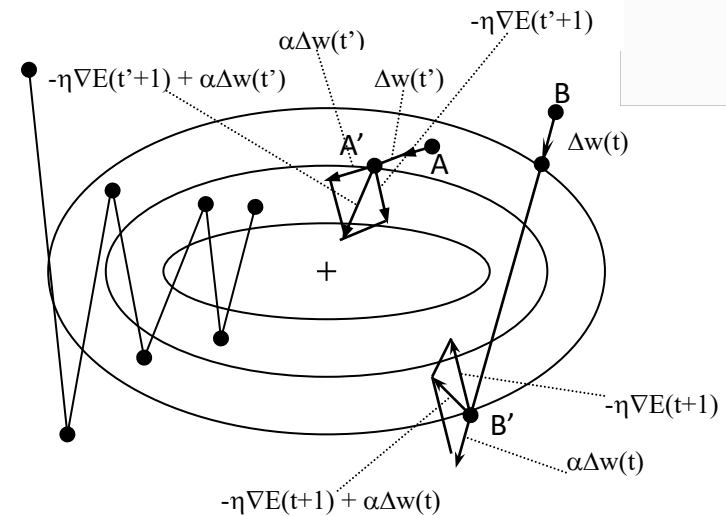
- The gradient descent method can be very slow if  $\eta$  is small and can fluctuate greatly if  $\eta$  is too large
- To reduce the level of fluctuations, it is necessary to add a momentum component

$$\Delta w^{(t)} = -\eta \nabla E^{(t)} + \alpha \Delta w^{(t-1)}$$

- where  $\alpha (\in [0,1])$  is a momentum parameter (usually assign = 0.9)
- Based on the experiences, we should choose reasonable values for learning rate and satisfying momentum

$$(\eta + \alpha) > \approx 1$$

where  $\alpha > \eta$  to avoid fluctuations



Gradient descent for a simple square error function.

The left trajectory does not use momentum.

The right trajectory uses momentum.

## BP algorithm: Number of neurons

---

- The size (number of neurons) of the hidden layer is an important question for the application of multi-layer neural network to solve practical problems
- In fact, it is difficult to identify the exact number of neurons needed to achieve the desired system accuracy
- The size of the hidden layer is usually determined through experiments (experiment/trial and test)



# ANN: Learning limit

---

- Boolean functions
  - Any binary function can be learned (approximately good) by an ANN using a hidden layer
- Continuous functions
  - Any bounded continuous function can be learned (approximately) by an ANN using a hidden layer [Cybenko, 1989; Hornik et al., 1991]

# ANN: advantages, disadvantages

---

## ■ Advantages

- Nature (structure) supports high-level parallel computation
- Obtain high accuracy in many problems (photos, video, audio, text)
- Very flexible in network architecture

## ■ Disadvantages

- There are no general rules for determining the network architecture and optimal parameters for a given problem
- There is no general method for assessing ANN's inner workings (thus, the ANN system is viewed as a "black box").
- It is difficult (impossible) to give explanations to the user
- Fundamental theories are few, to help explain the real successes

## ANN: When?

---

- The form of the function does not predetermined
- It is not necessary (or unimportant) to provide an explanation to the user about the results
- Accept long time for the training process
- Can collect a large number of labels for data
- Domains related to image, video, speech, text

Open library

---



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