Minia University

Faculty of Computers & information

Artificial Neural Networks and Deep Learning

Slides By:

A.T. Sarah Osama Talaat

E-mail: SarahOsama.fci@gmail.com

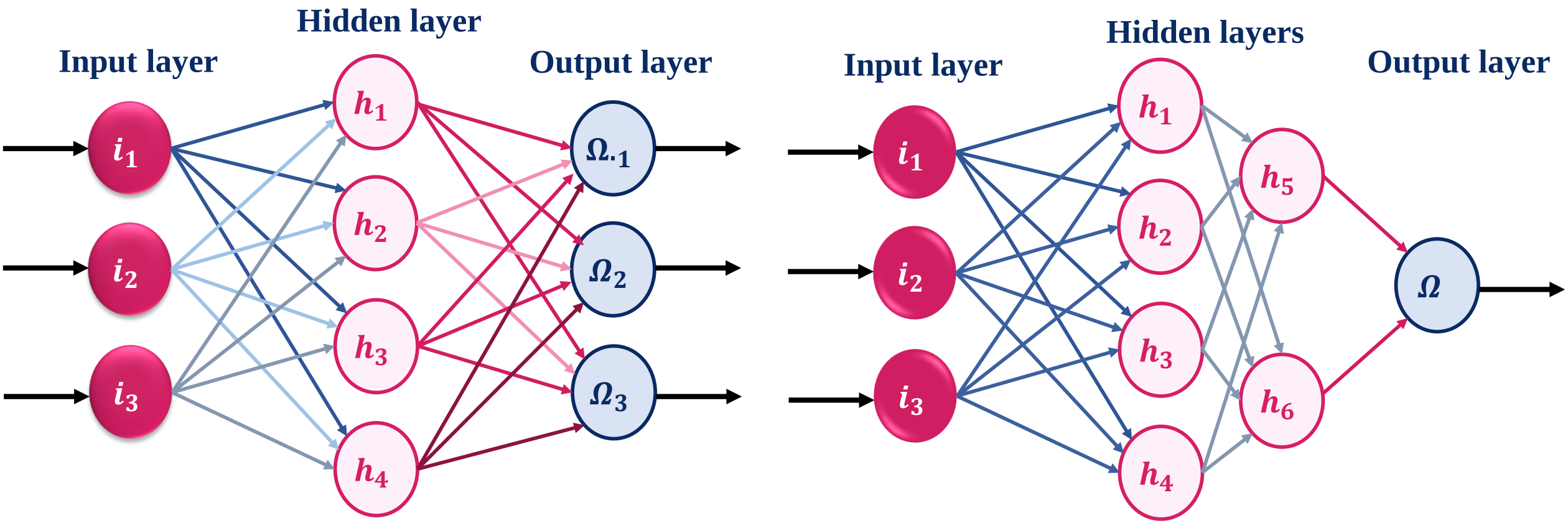
Slides were prepared based on set of references mentioned in the last slide

Agenda

- Revision
- Supervised Learning Network Paradigms
  - Backpropagation network
  - Learning rate
  - The selection of the learning rate
  - Backpropagation-specific properties
  - Resilient backpropagation network
- Initial configuration of a multilayer perceptron

Let's Start

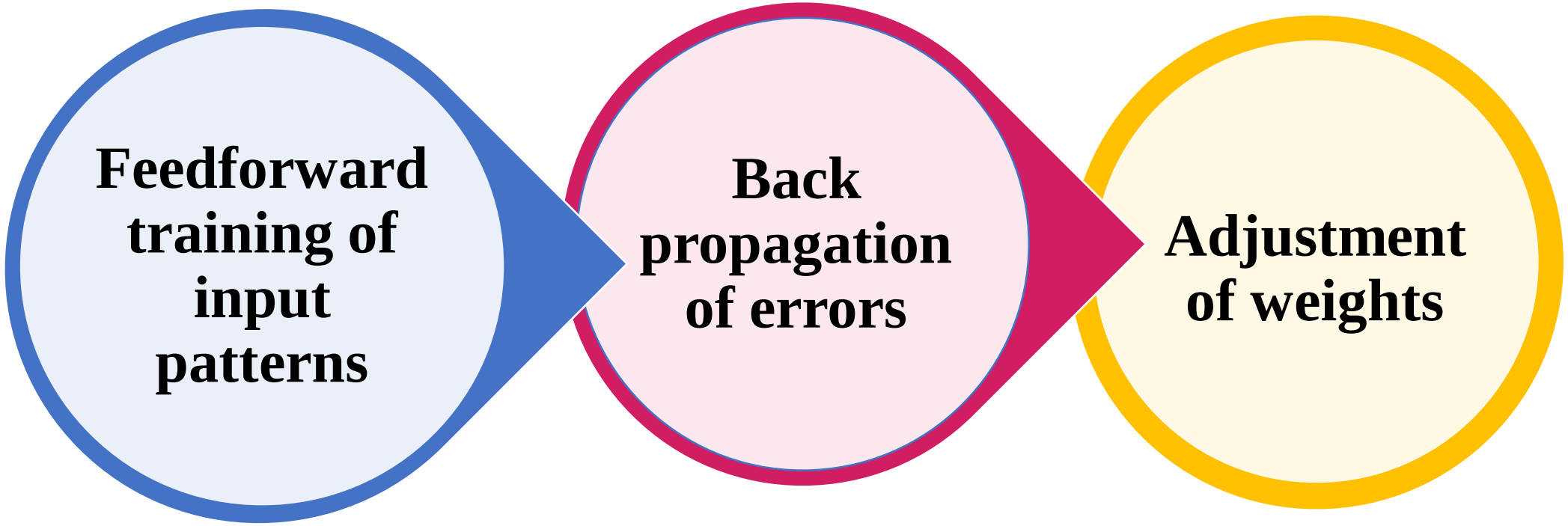
Revision
Multi-Layer perceptron



Figure(5.3): Left side: illustration of a single layer perceptron with four input neurons and one output neuron. Right side: illustration of a single layer perceptron with four input neurons and three output neuron.

Supervised Learning Network Paradigms

Backpropagation network main steps



Figure(5.4): illustration of a Backpropagation network (is a Multi-layer perceptron learning algorithm) training phase

Supervised Learning Network Paradigms

Backpropagation network algorithm

❑ **Algorithm 5.3: Backpropagation algorithm for single training sample(tr,y)**

Input x: set the corresponding activation a^1 for the input layer.

Output: Gradient of the cost function

1. **Initialization Initialize all weights by small random values (e.g. -0.5:0.5) and select a suitable learning rate $\eta(e.g.0.1)$**
2. **Feedforward:**
3. **For each** $l = 2,3,4,,..,L$ **do**
4. Feed the training sample through the network and compute $net^l = w^l a^{l-1} + b^l$,
5. Then apply the activation function $a^l = \sigma(net^l)$
6. **End foreach**
7. *Foreach output unit k,compue delta* $\delta^l = \nabla_a C \odot \sigma'(net^L) = (a^L - y) \odot \sigma'(net^L)$
8. **Backpropagate the error:**
9. **For each** $l = L-1, L-2,..,2$ **do**
10. Compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(net^L)$,
11. **End foreach**
12. **Output:** The gradient of the cost function is given by $\frac{\partial C}{\partial w_{jk}^l} = a_{in}\, \delta_{out}$ and $\frac{\partial C}{\partial b_j^l} = \delta_j^l = \delta$

Supervised Learning Network Paradigms

Backpropagation network algorithm

- **Algorithm 5.3: Backpropagation algorithm for each training samples(tr,y)**

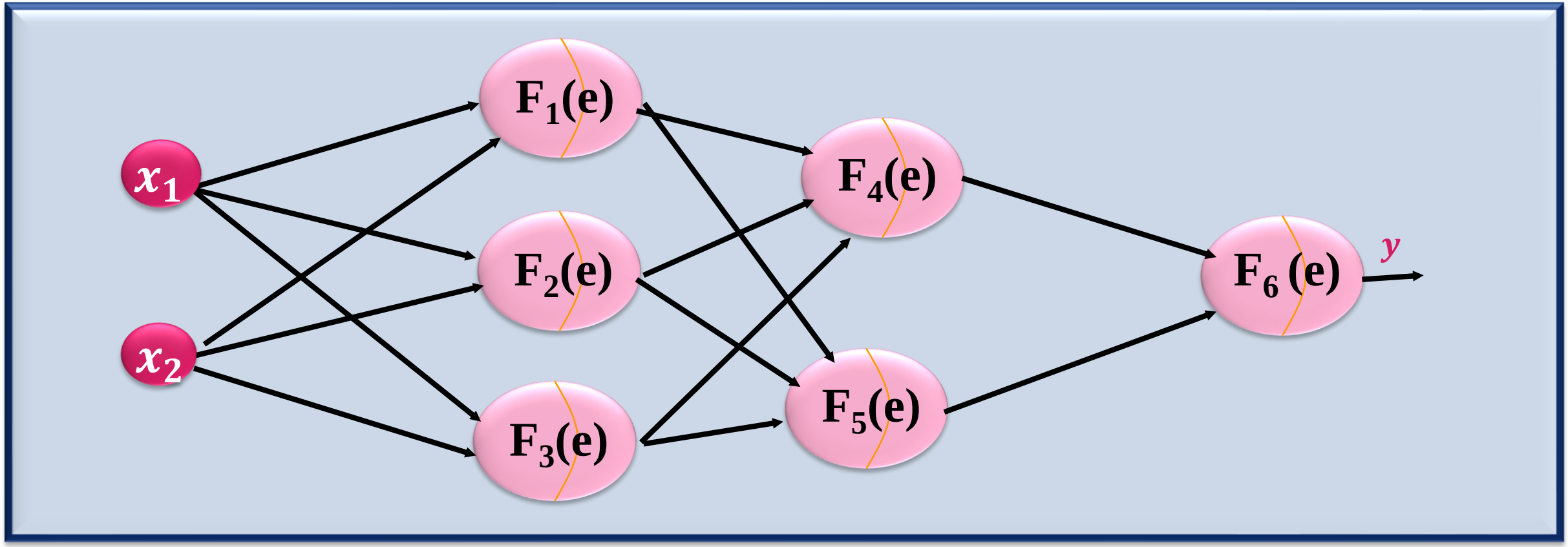
Input: set of training examples.

Output: Gradient of the cost function

1. **For each training example x: Set the corresponding input activation $a^{x,1}$, and perform the following steps:**
2. **Feedforward:**
3. **Foreach** $l = 2,3,4,,\ldots,L$ **do**
4. Feed the training samples through the network and compute $\boldsymbol{net}^{x,l} = \boldsymbol{w}^{l}\boldsymbol{a}^{x,l-1} + \boldsymbol{b}^{l}$,
5. Then apply the activation function $\boldsymbol{a}^{x,l} = \boldsymbol{\sigma}\left(\boldsymbol{net}^{x,l}\right)$
6. **End foreach**
7. $Foreach\ output\ unit\ k, compue\ delta\ \boldsymbol{\delta}^{x,L} = \boldsymbol{\nabla}_{a}\boldsymbol{C}_{x} \odot \boldsymbol{\sigma}'\left(\boldsymbol{net}^{x,L}\right)$
8. **Backpropagate the error:**
9. **For each** $l = L-1, L-2,\ldots,2$ **do**
10. Compute $\boldsymbol{\delta}^{x,l} = ((\boldsymbol{w}^{l+1})^{T}\boldsymbol{\delta}^{x,l+1}) \odot \boldsymbol{\sigma}'\left(\boldsymbol{net}^{x,L}\right)$,
11. **End foreach**
12. **Gradient descent: For each** $l = L-1, L-2,\ldots,2$ update the weights according to the rule $w_{l,new} = w_{l,old} - \frac{\boldsymbol{n}}{\boldsymbol{m}}\sum_{x}\boldsymbol{\delta}^{x,l}\left(\boldsymbol{a}^{x,l-1}\right)^{T}$, and the biases according to the rule $b_{l,new} = b_{l,old} - \frac{\boldsymbol{n}}{\boldsymbol{m}}\sum_{x}\boldsymbol{\delta}^{x,l}$

Supervised Learning Network Paradigms
Backpropagation graphical description

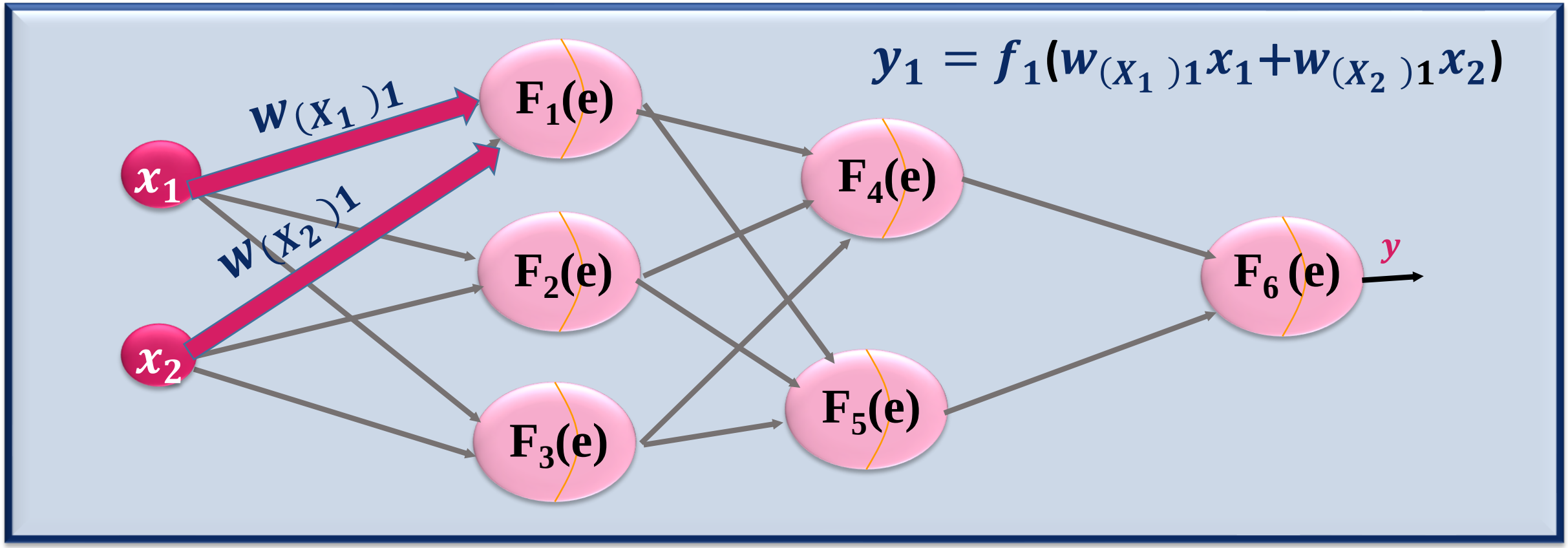
- **Graphical description of training multi-layer neural network using Backpropagation algorithm**



Supervised Learning Network Paradigms
Backpropagation graphical description

❑ Feedforward stage

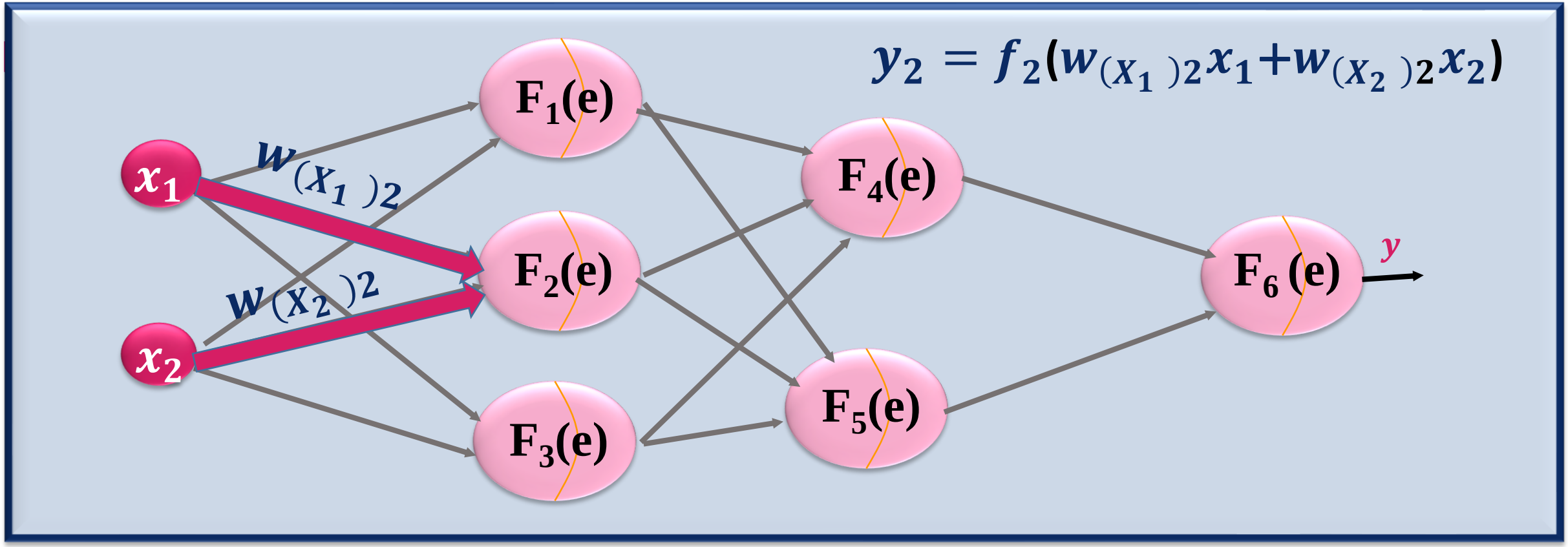
- Signal is propagating through the network, symbols $w_{(xm)n}$ represent weights of connections between network input x_m and neuron n in input layer. Symbols y_n represents output signal of neuron n.



Supervised Learning Network Paradigms
Backpropagation graphical description

❑ Feedforward stage

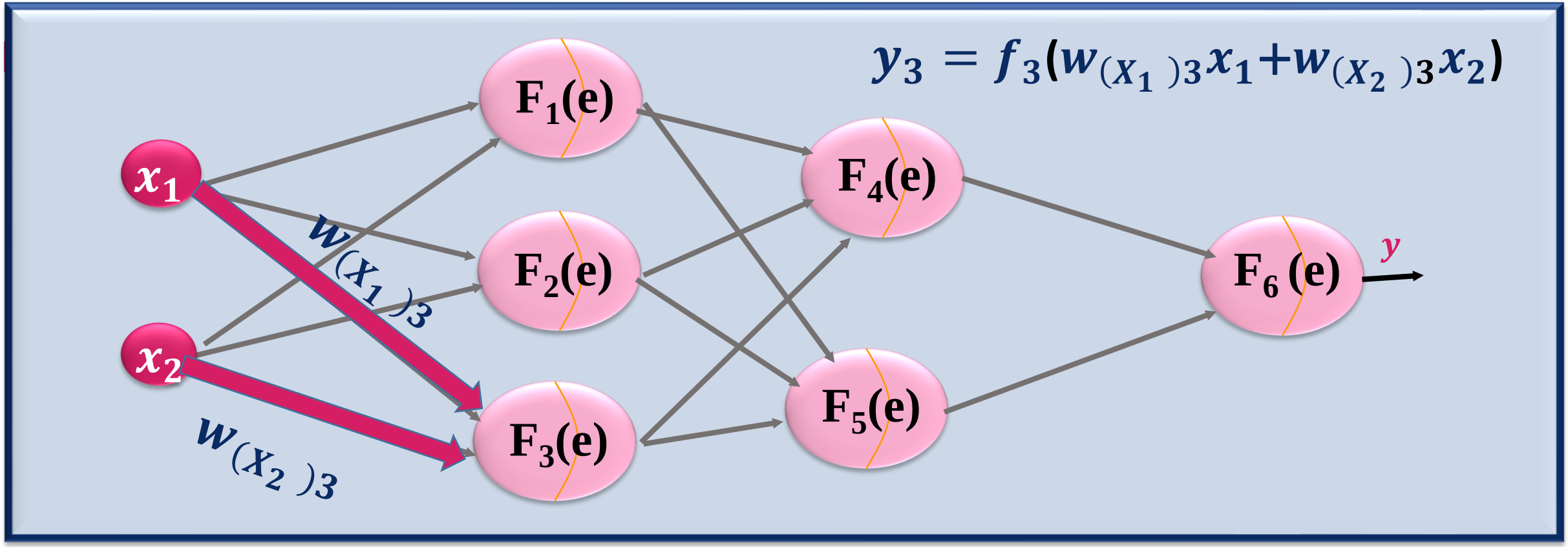
- Signal is propagating through the network, symbols $w_{(xm)n}$ represent weights of connections between network input x_m and neuron n in input layer. Symbols y_n represents output signal of neuron n.



Supervised Learning Network Paradigms
Backpropagation graphical description

❑ Feedforward stage

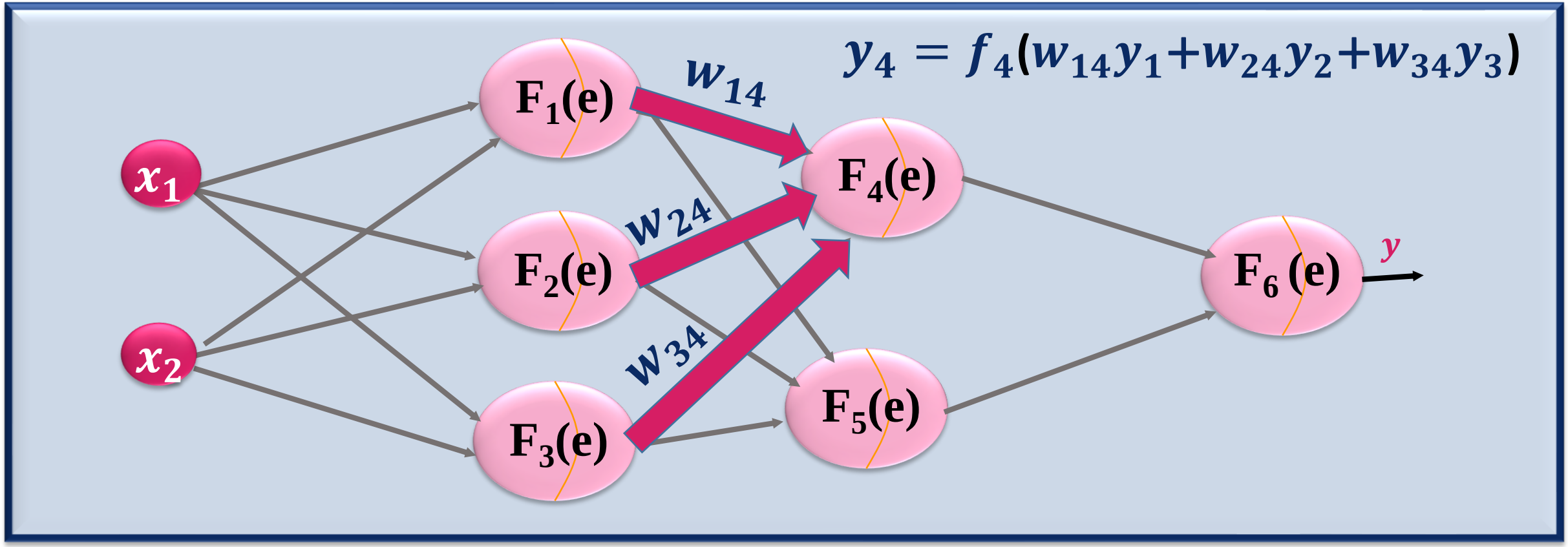
- Signal is propagating through the network, symbols $w_{(xm)n}$ represent weights of connections between network input x_m and neuron n in input layer. Symbols y_n represents output signal of neuron n.



Supervised Learning Network Paradigms
Backpropagation graphical description

❑ Feedforward stage

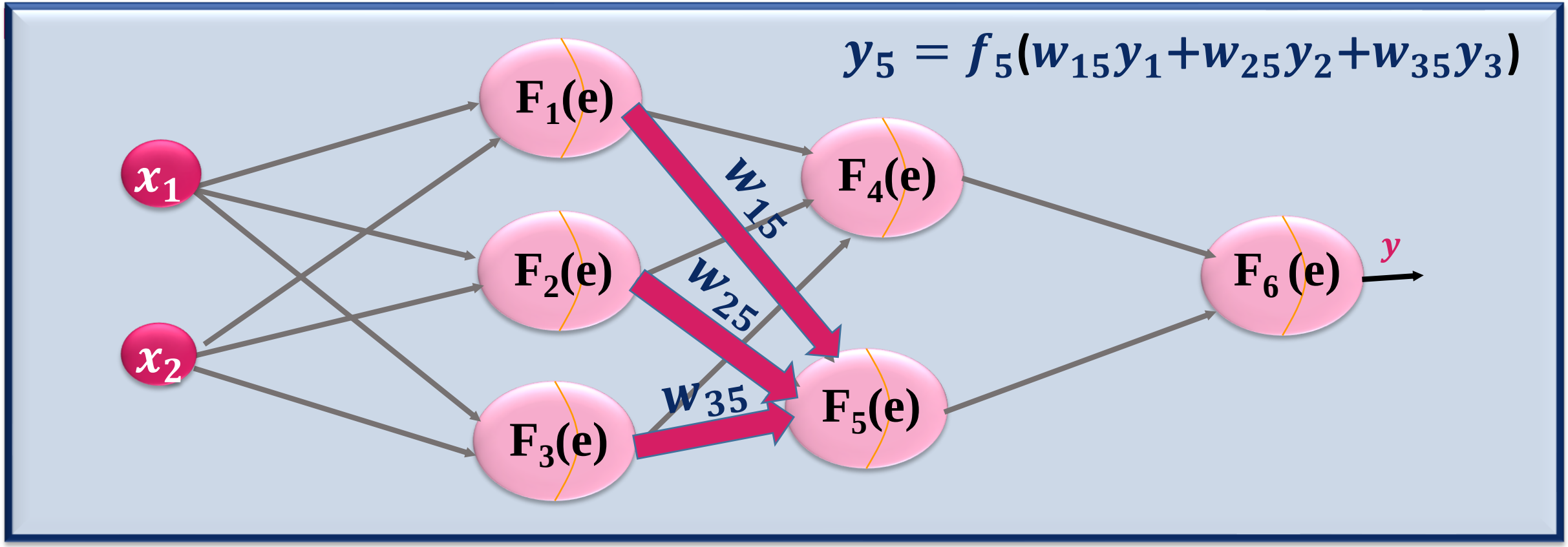
- Propagation of signals through the hidden layer. Symbols w_{mn} represent weights of connections between output of neuron ***m*** and input of neuron ***n*** in the next layer.



Supervised Learning Network Paradigms
Backpropagation graphical description

❑ Feedforward stage

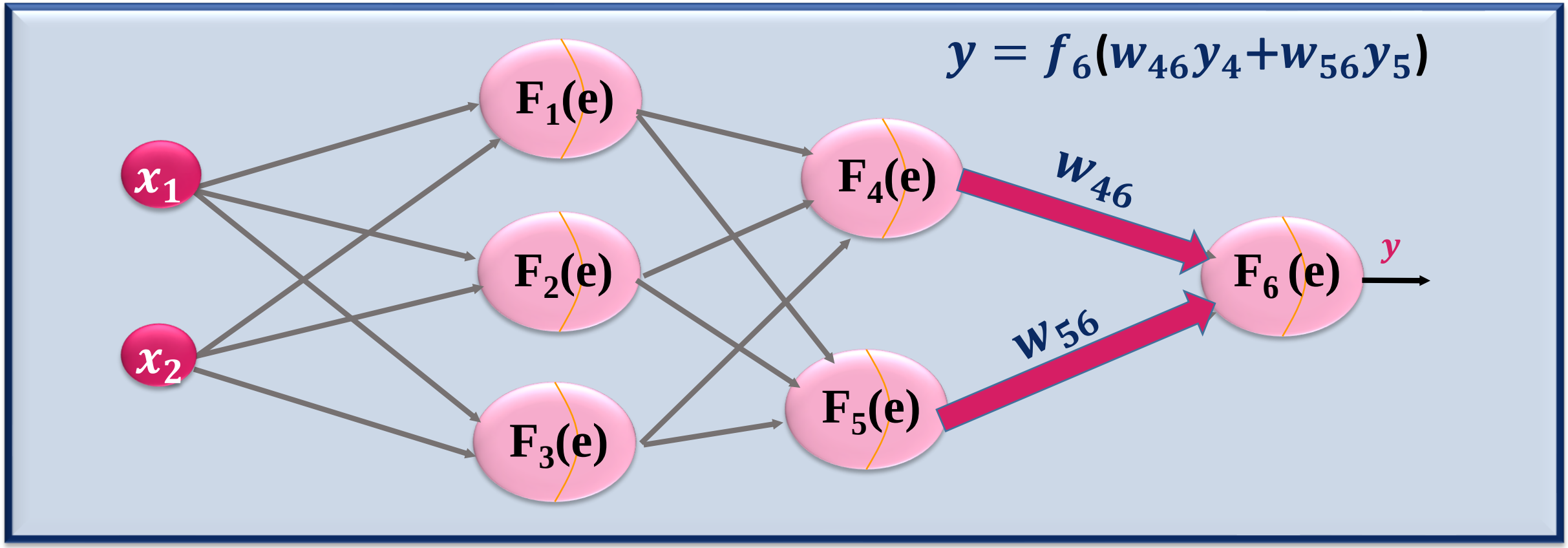
- Propagation of signals through the hidden layer. Symbols w_{mn} represent weights of connections between output of neuron ***m*** and input of neuron ***n*** in the next layer.



Supervised Learning Network Paradigms
Backpropagation graphical description

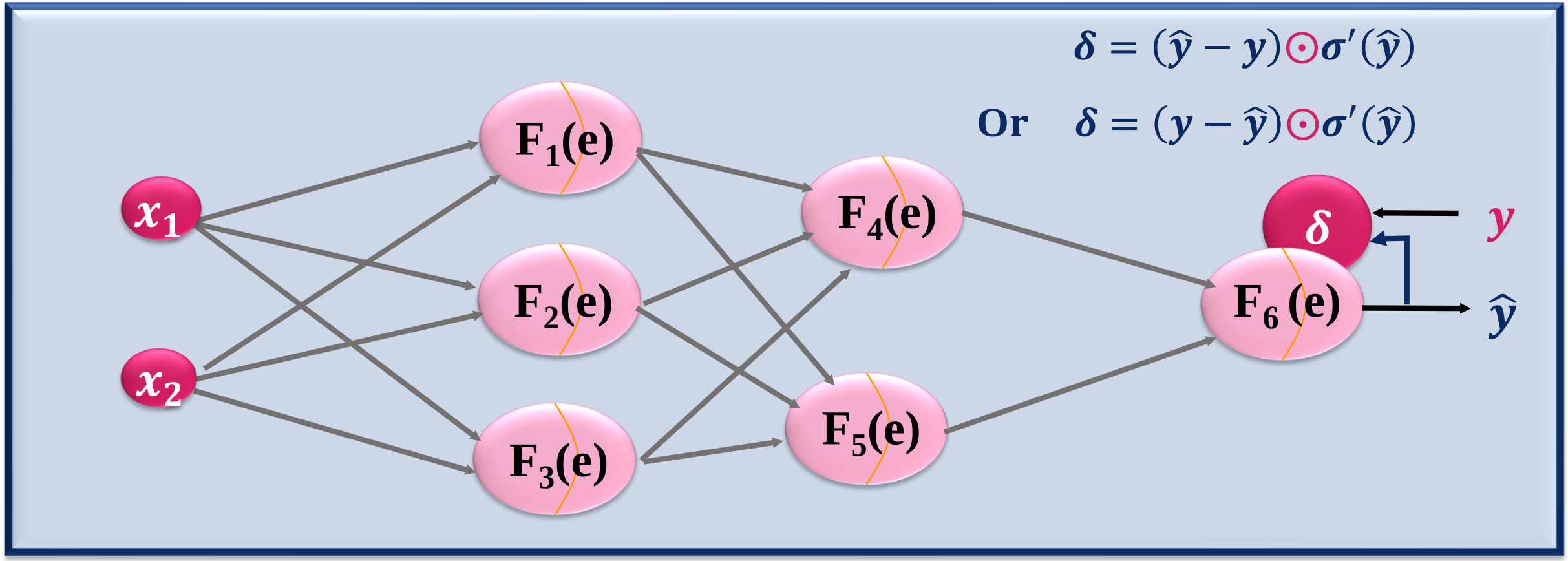
❑ Feedforward stage

- Propagation of signals through the output layer.



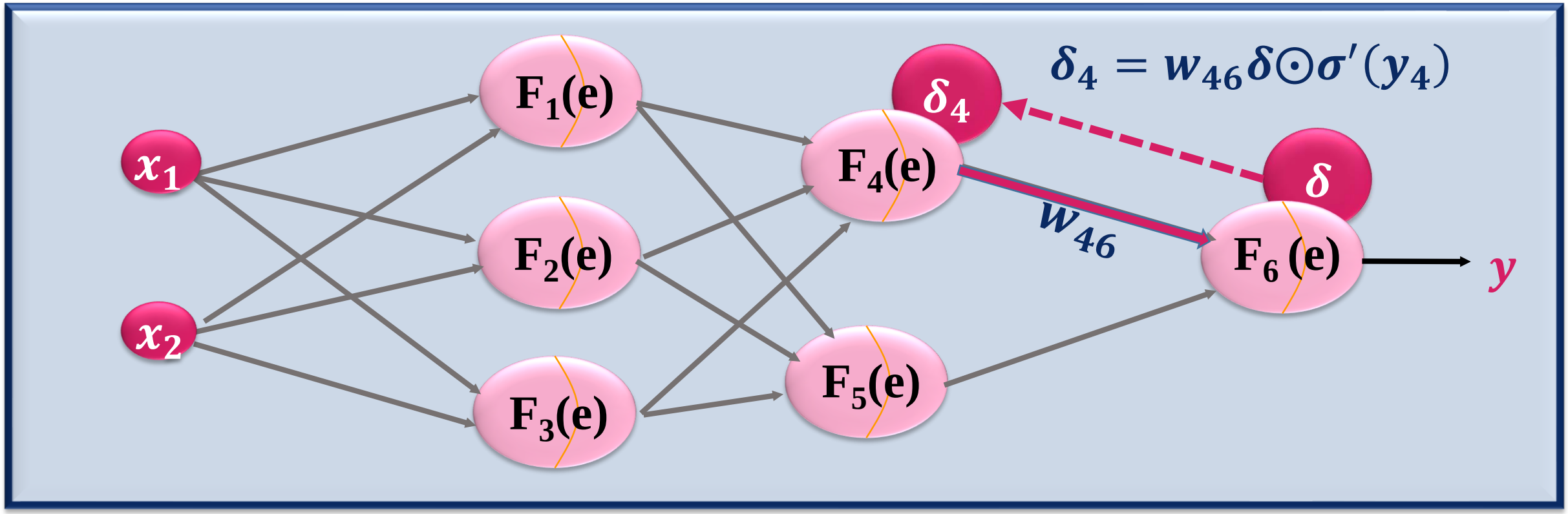
Supervised Learning Network Paradigms
Backpropagation graphical description

❑ Backpropagation stage



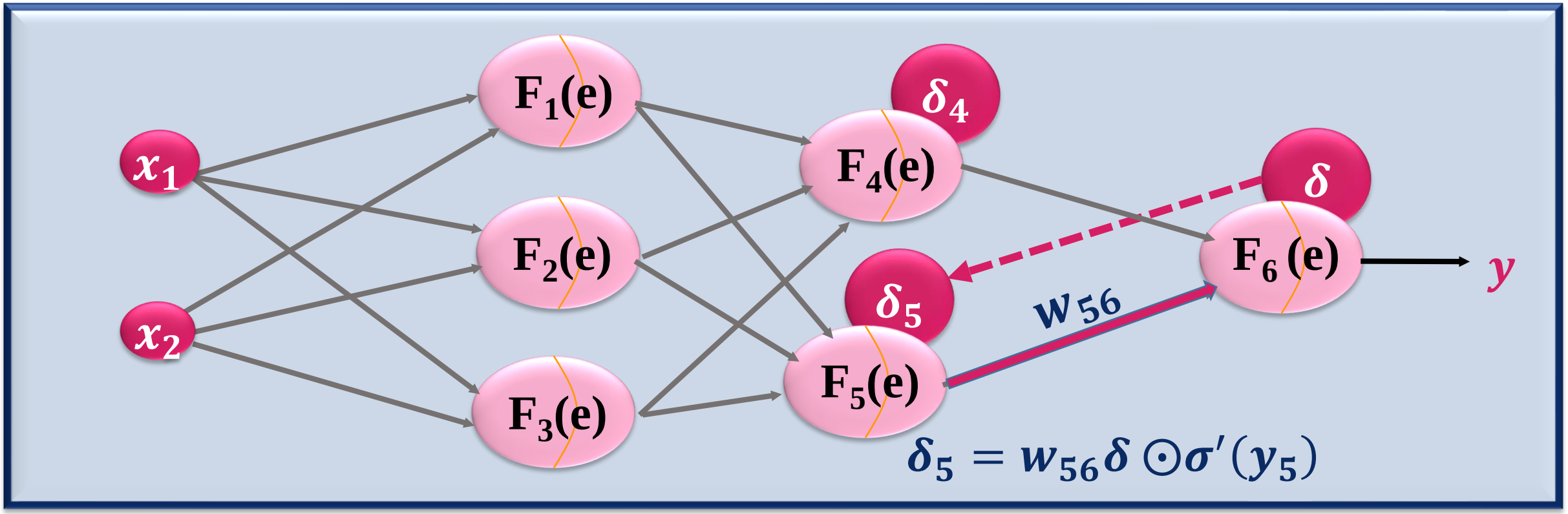
Supervised Learning Network Paradigms
Backpropagation graphical description

❑ Backpropagation stage



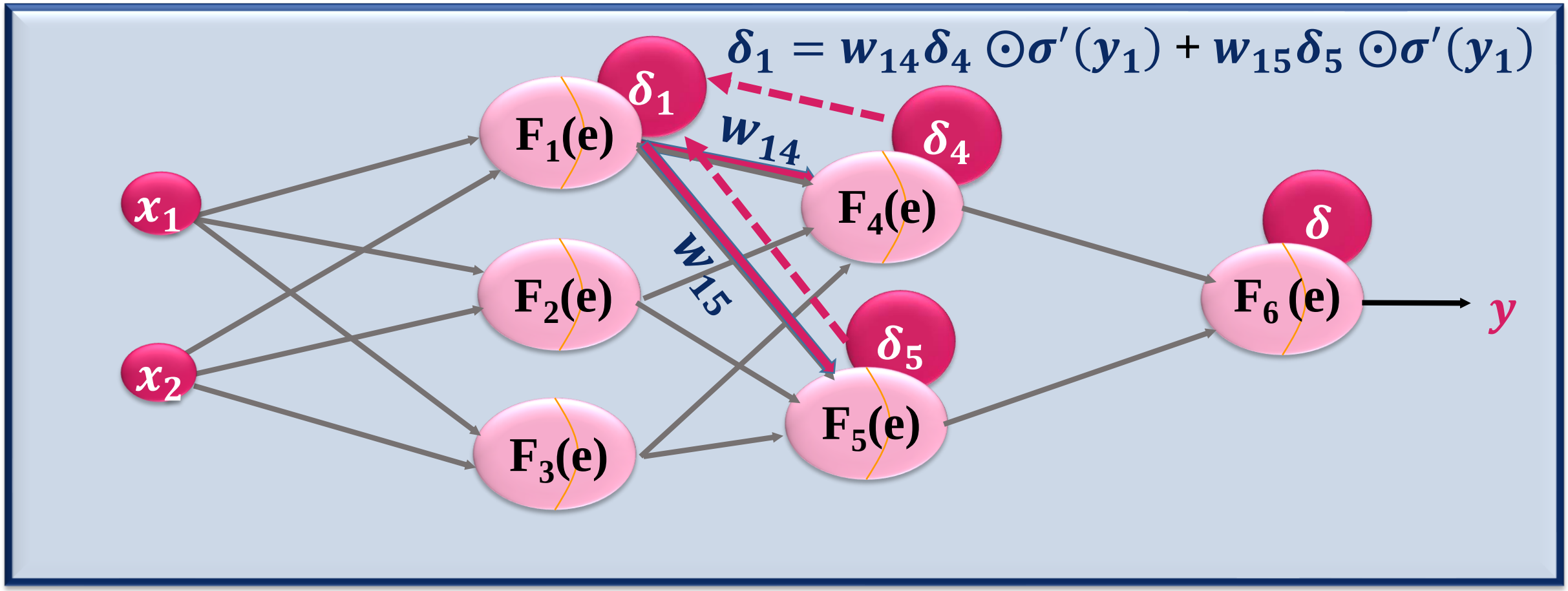
Supervised Learning Network Paradigms
Backpropagation graphical description

❑ Backpropagation stage



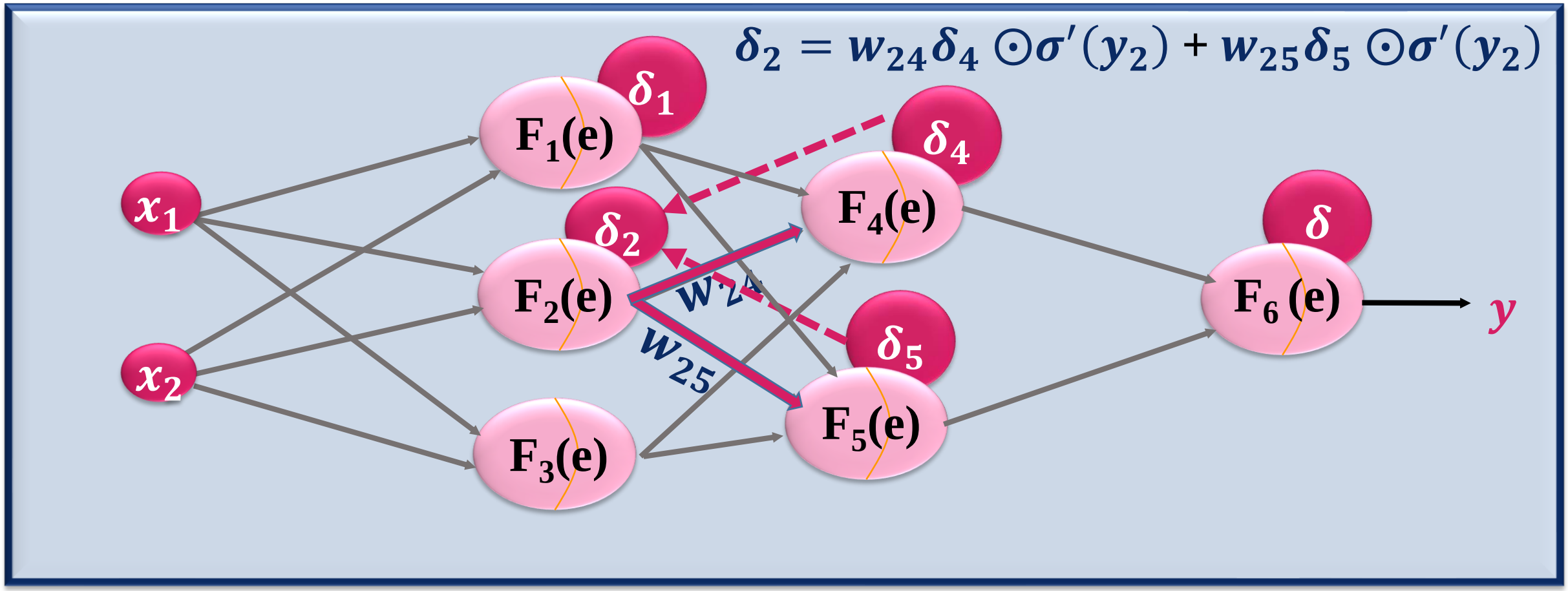
Supervised Learning Network Paradigms
Backpropagation graphical description

❑ Backpropagation stage



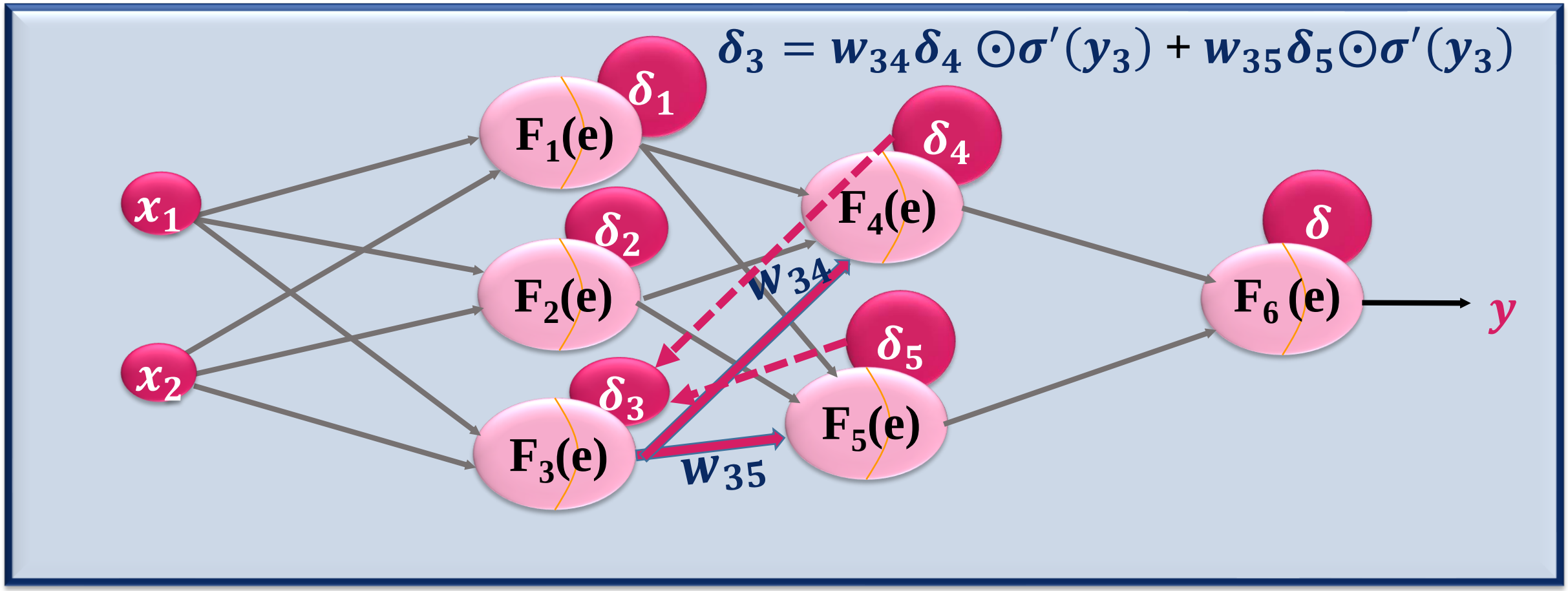
Supervised Learning Network Paradigms
Backpropagation graphical description

❑ Backpropagation stage



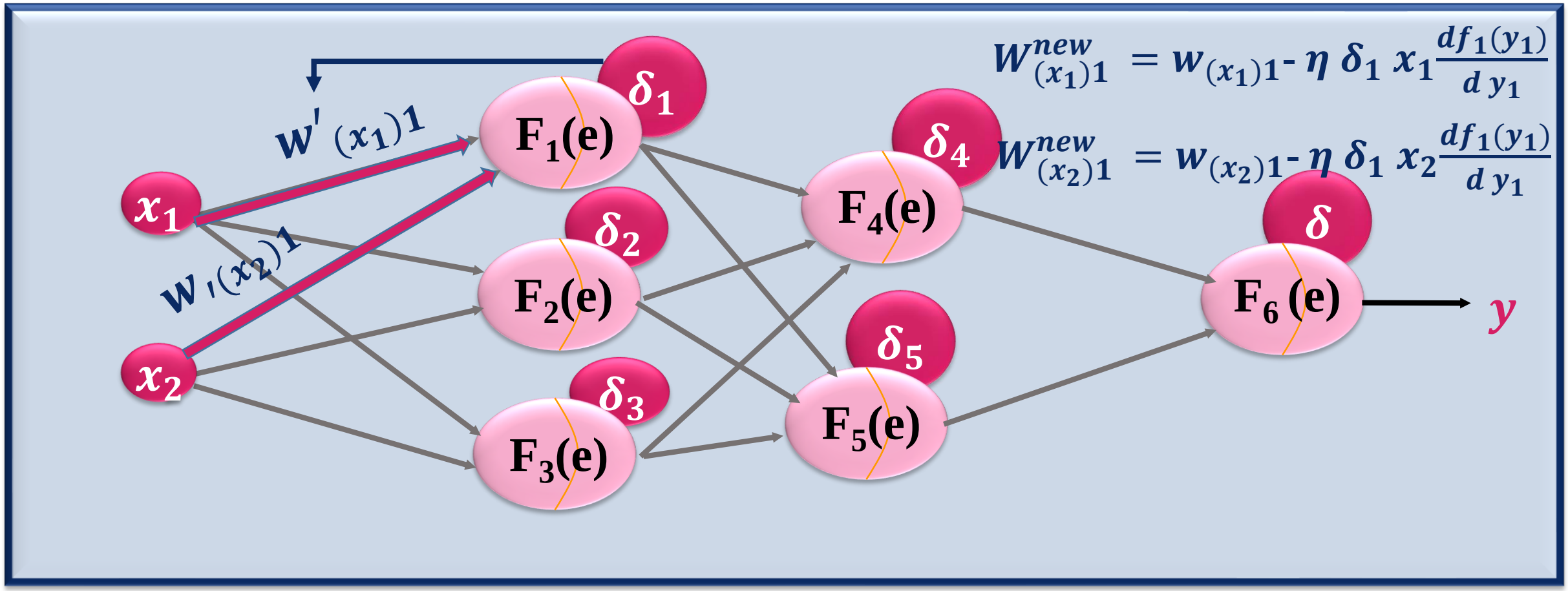
Supervised Learning Network Paradigms
Backpropagation graphical description

❑ Backpropagation stage



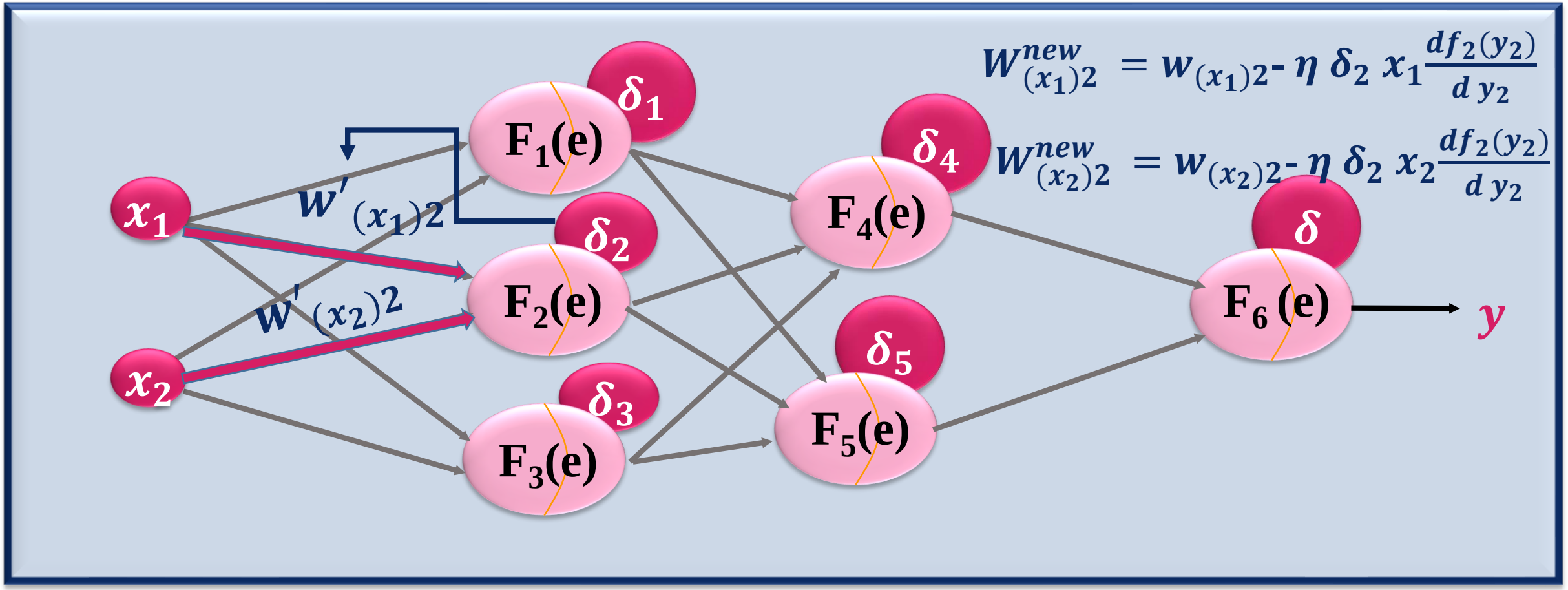
Supervised Learning Network Paradigms
Backpropagation graphical description

❑ Weights adjustment stage



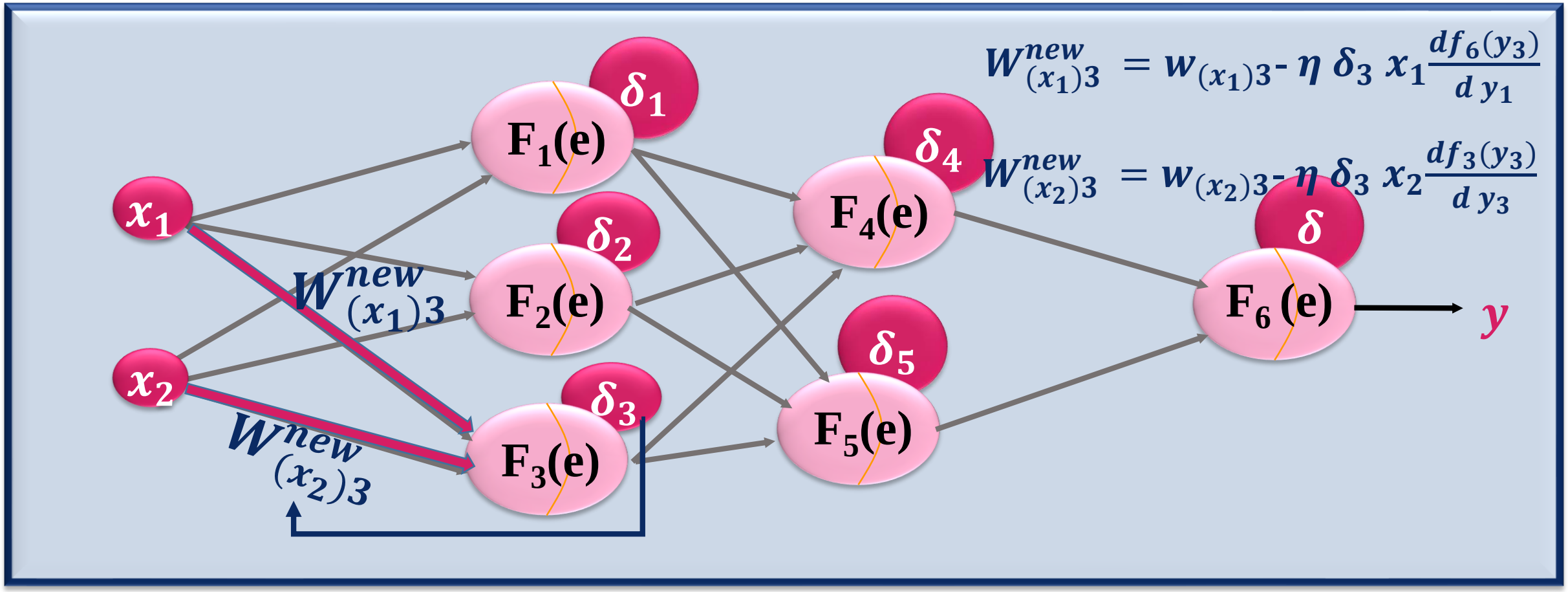
Supervised Learning Network Paradigms
Backpropagation graphical description

❑ Weights adjustment stage



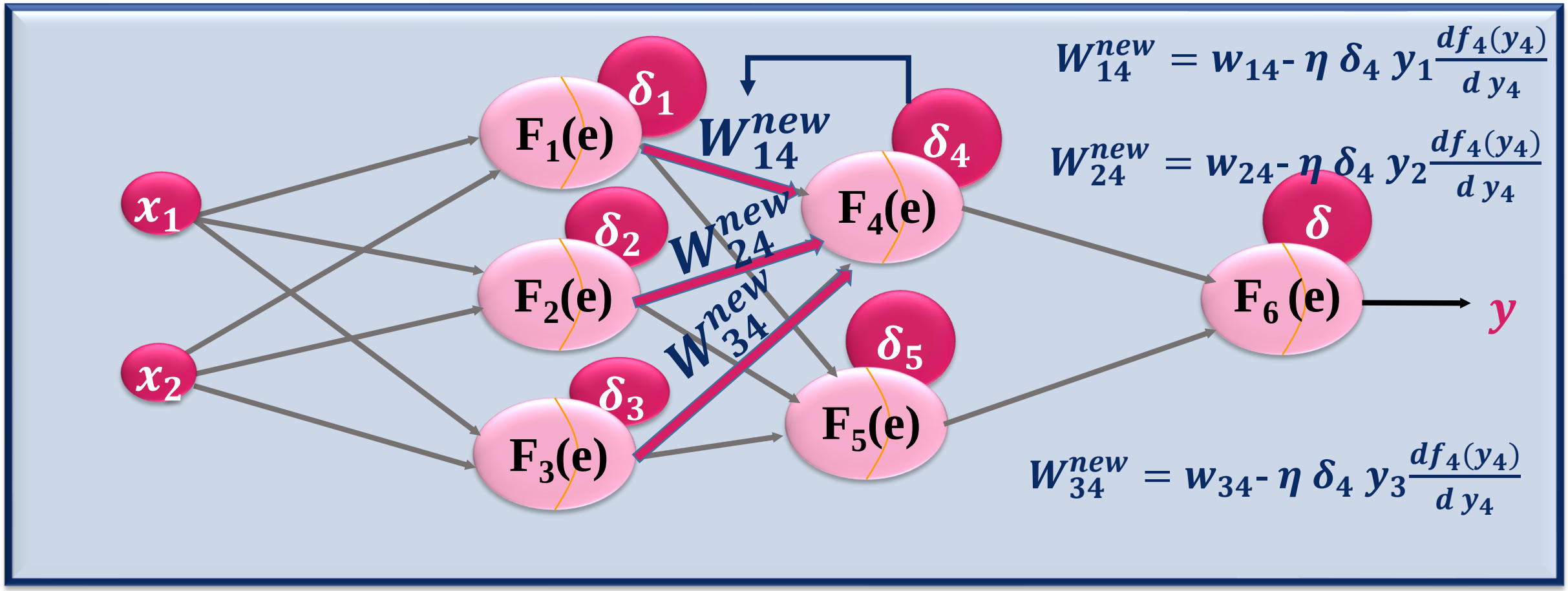
Supervised Learning Network Paradigms
Backpropagation graphical description

❑ Weights adjustment stage



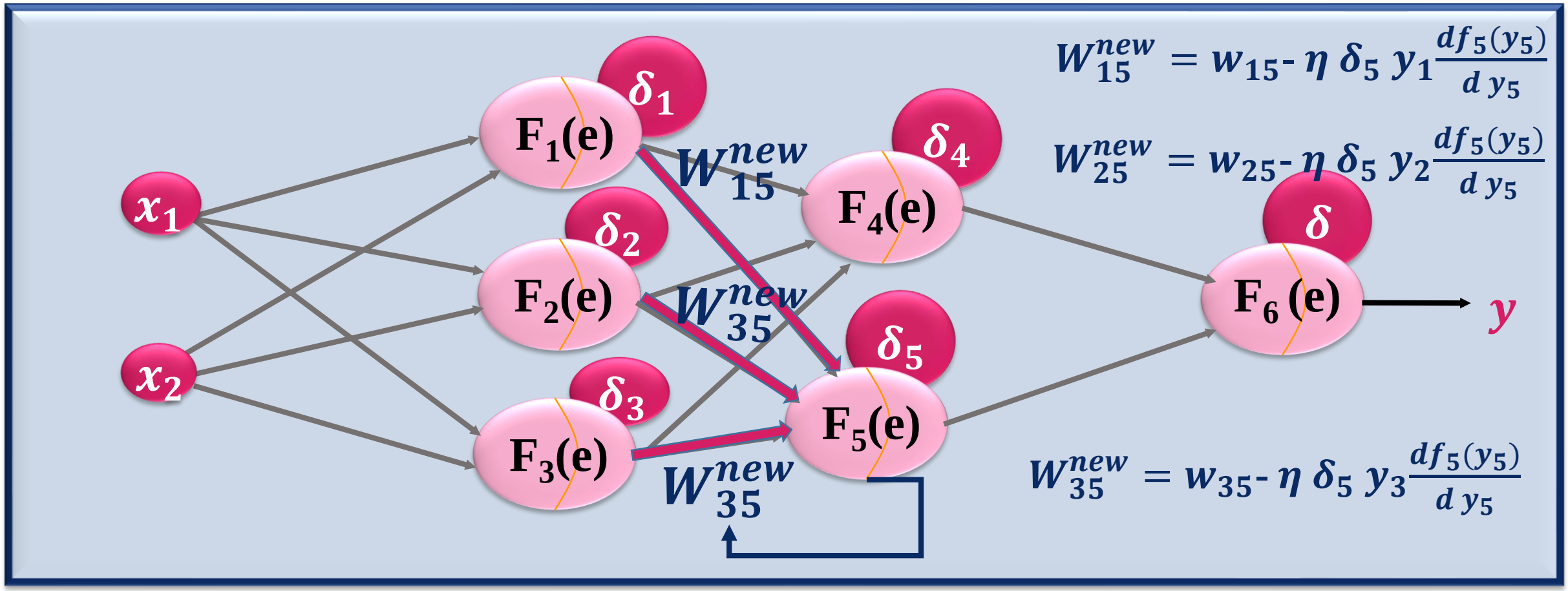
Supervised Learning Network Paradigms
Backpropagation graphical description

❑ Weights adjustment stage



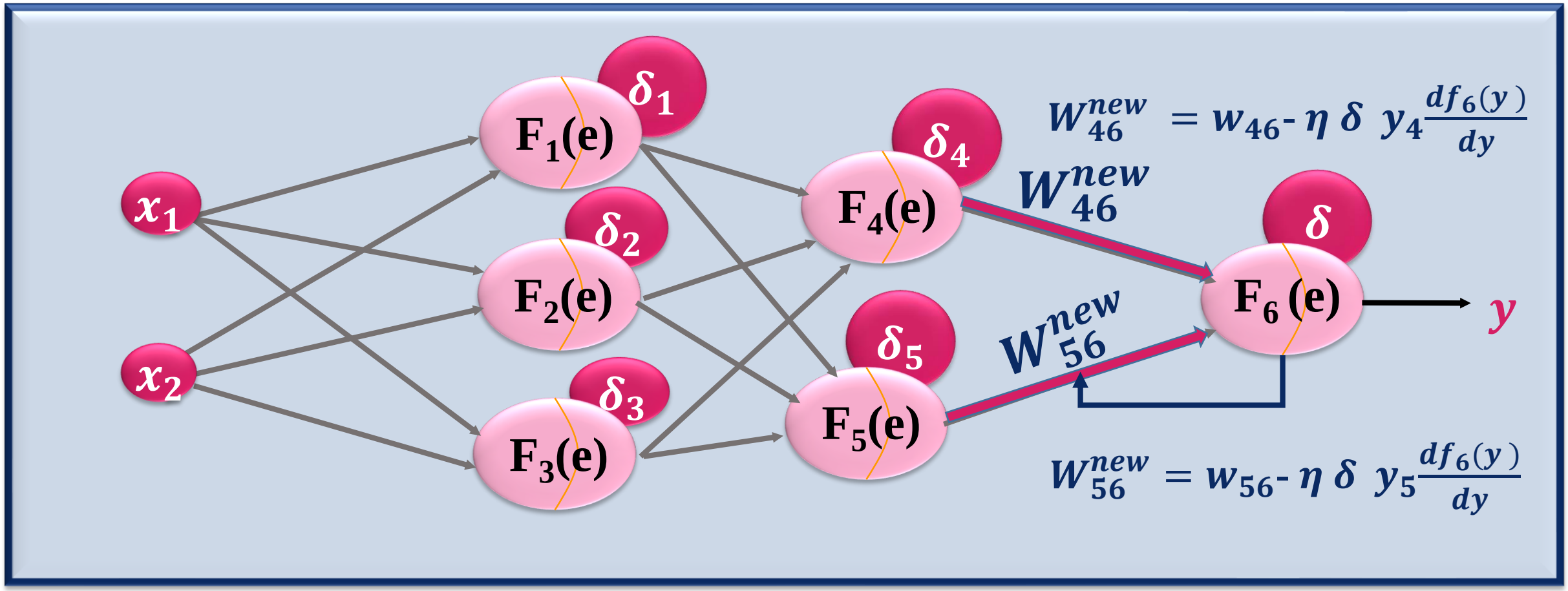
Supervised Learning Network Paradigms
Backpropagation graphical description

❑ Weights adjustment stage



Supervised Learning Network Paradigms
Backpropagation graphical description

❑ Weights adjustment stage



Supervised Learning Network Paradigms
Backpropagation Example

❑ Example:

- Construct a **XOR** function could not be solved by a single layer perceptron network, let η = 0.1 and the sigmoid activation function is

$$a(net_i) = \frac{1}{1+e^{net_i}}$$

- The truth table for **XOR** function and the net:

x_1	x_2	t = x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

Supervised Learning Network Paradigms
Backpropagation Example

❑ Solution:

- The sigmoid activation function is

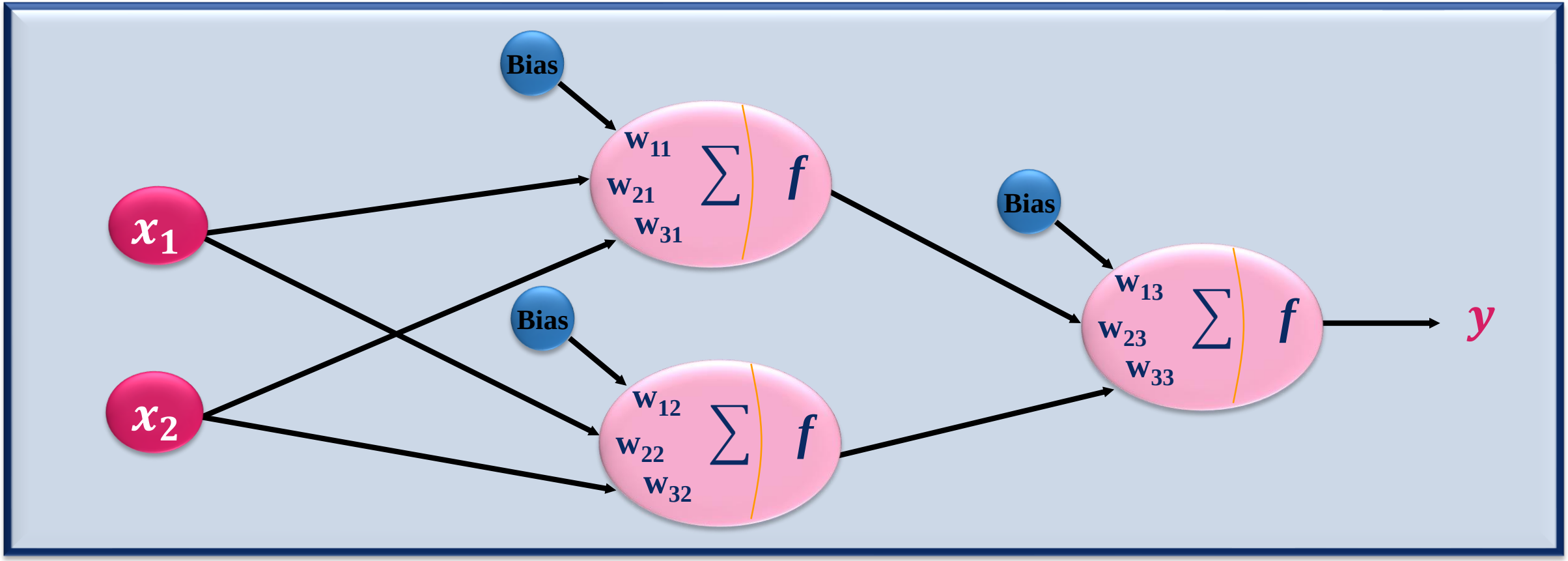
$$\mathbf{f}(net_i) = \sigma(net_i) = \frac{1}{1+e^{-net_i}}$$

- The derivative of the sigmoid function can be calculated as follows;

$$\sigma'(net_i) = \sigma(net_i)\,[1-\sigma(net_i)]$$

Supervised Learning Network Paradigms
Backpropagation Example

- **Solution:**
  - **XOR architecture**

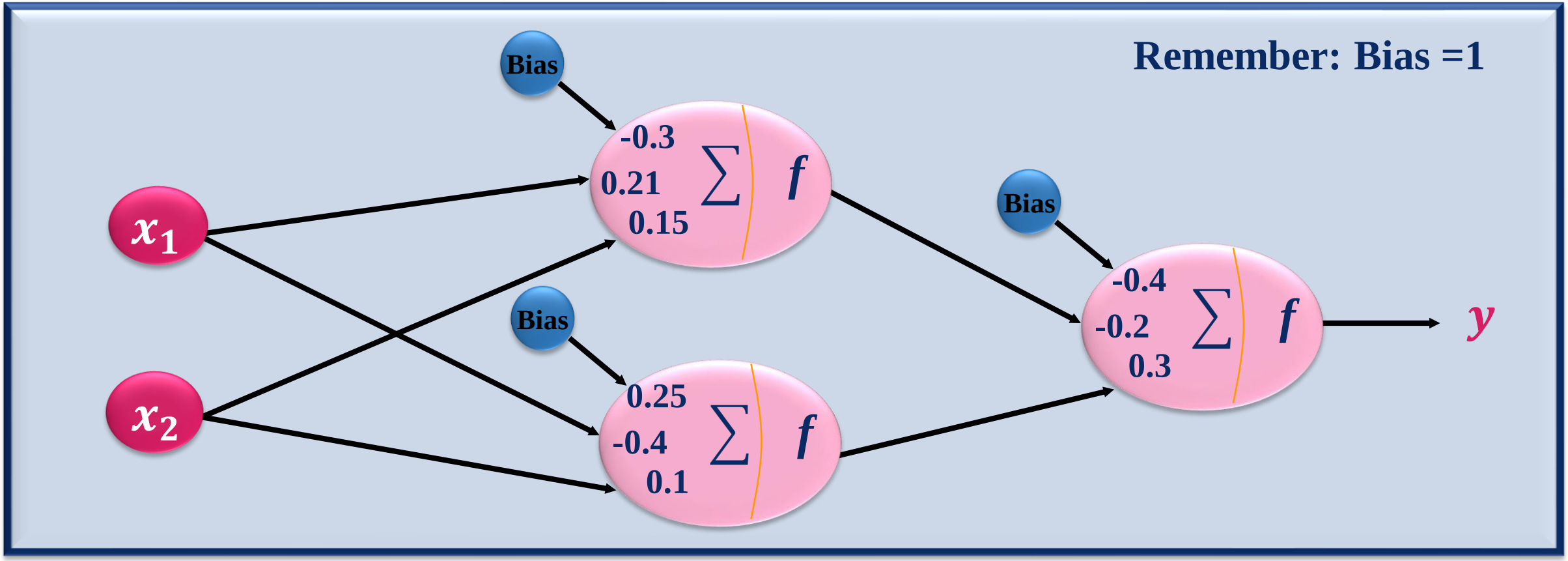


Supervised Learning Network Paradigms

Backpropagation Example

❑ Solution:

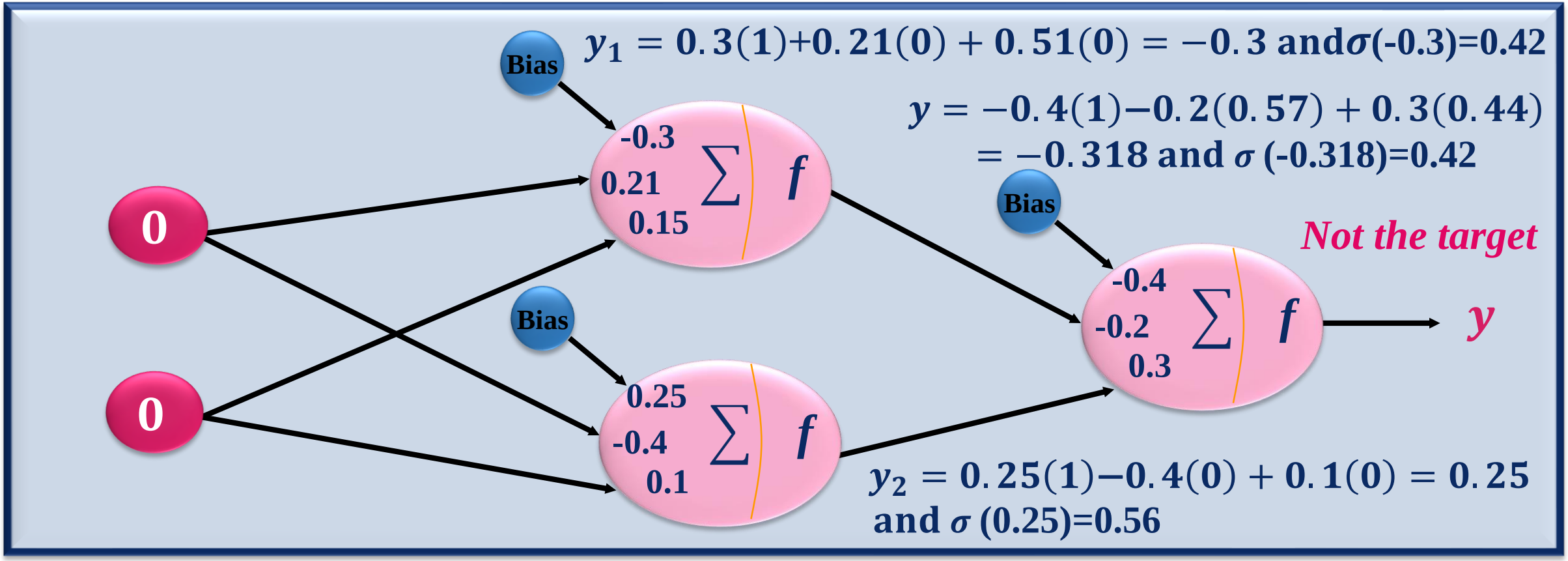
- Initialize all weights by small random values (between -1 and 1)



Supervised Learning Network Paradigms
Backpropagation Example

❑ **Solution:**

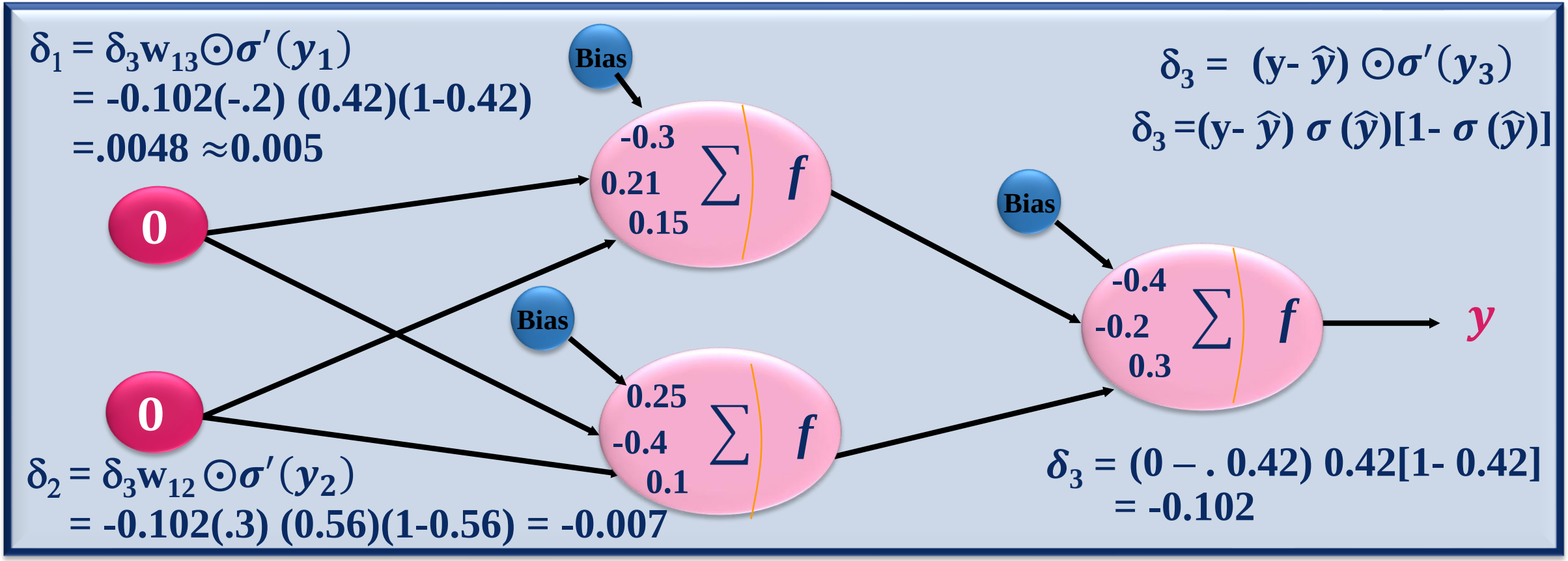
- **Feedforward stage: First iteration:** first training input is (bias,0,0) and target is 0



Supervised Learning Network Paradigms
Backpropagation Example

❑ **Solution:**

- **Backpropagate stage: First iteration:** first training input is (bias,0,0) and target is 0



Supervised Learning Network Paradigms
Backpropagation Example

- **Solution:**
  - **Weights adjustment stage: First iteration:** first training input is (bias,0,0) and target is 0

- **1st neuron : old weights (-0.3 , 0.21 , 0.15), $\delta_1 = 0.005$, inputs (1 , 0 , 0)**
  - W_{11}=-0.3 – (0.1)(0.005) (1)[(0.42)(1-0.42)] =-0.30
  - W_{21}=0.21- (0.1)(0.005) (0)[(0.42)(1-0.42)]= 0.21
  - W_{31}= 0.15- (0.1)(0.005) (0)[(0.42)(1-0.42)]= 0.15

Supervised Learning Network Paradigms
Backpropagation Example

- **Solution:**
  - **Weights adjustment stage: First iteration:** first training input is (bias,0,0) and target is 0

- **2nd neuron : old weights (0.25 , -0.4 , 0.1), $\delta_2 = -0.009$, inputs (1 , 0 , 0)**
  - W_{12}= 0.25- 0.1(0.005)[(0.56)(1-0.56)](1)= 0.249 ≈ 0.25
  - W_{22}= -0.4- 0.1(0.005)[(0.56)(1-0.56)](0)= -0.4
  - W_{32}= 0.1- 0.1(0.005) [(0.56)(1-0.56)](0)= 0.1

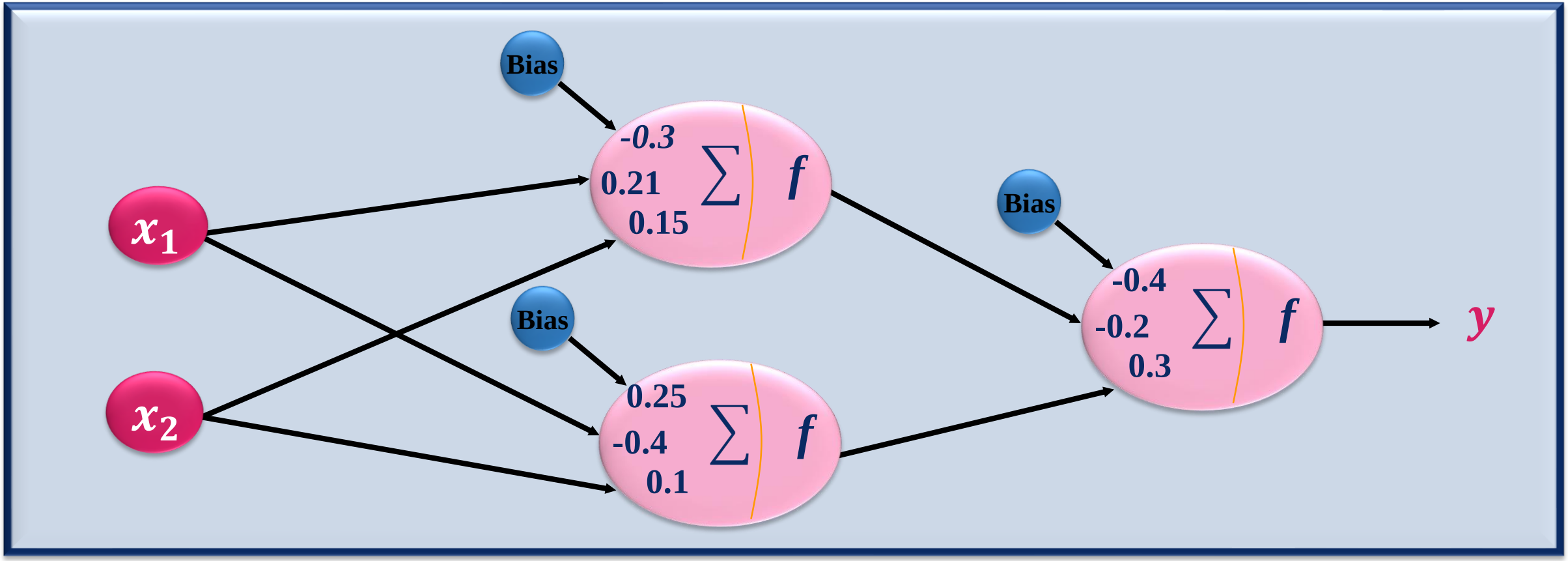
Supervised Learning Network Paradigms
Backpropagation Example

- **Solution:**
  - **Weights adjustment stage: First iteration:** first training input is (bias,0,0) and target is 0

- **3rd neuron : old weights (-0.4, -0.2 , 0.3), $\delta_3 = -0.14$, inputs (1 , 0.57 , 0.44)**
  - W_{13}=-0.4- (0.1)(-0.14)[(0.42)(1-0.42)](1)= -0.39 ≈ -0.4
  - W_{23}= -0.2 – (0.1)(-0.14)[(0.42)(1-0.42)] (0.42)= -0.19 ≈ −0.2
  - W_{33}= 0.3- (0.1)(-0.14)[(0.42)(1-0.42)] (0.56) =0.30

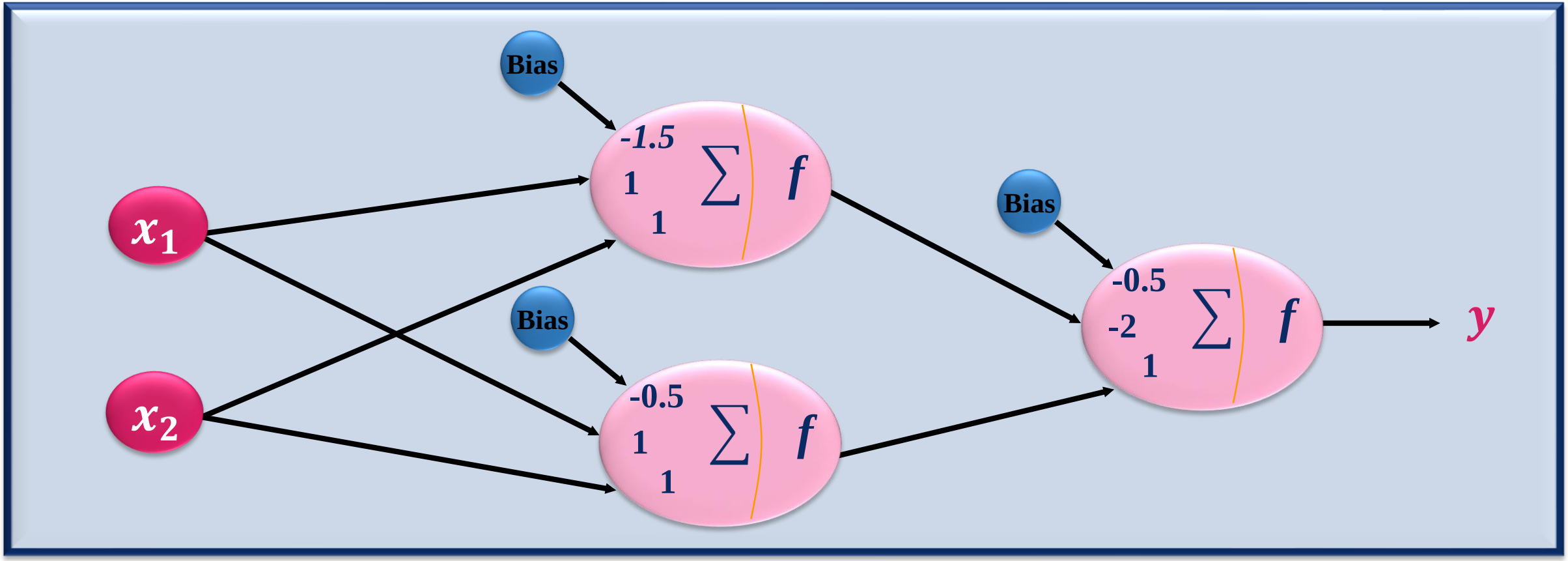
Supervised Learning Network Paradigms
Backpropagation Example

- **Solution:**
  - **New weights are:**



Supervised Learning Network Paradigms
Backpropagation Example

- **Solution:**
  - **The solution of XOR problem appears after about 500 iterations**



Supervised Learning Network Paradigms

Learning Rate (η)

❑ Definition 8.1 (Learning rate η):

- Speed and accuracy of a learning procedure can always be controlled by and are always proportional to a learning rate which is written as η.

Supervised Learning Network Paradigms

The selection of the learning rate

❑ Selection of learning rate:

- The selection of the learning rate has heavy influence on the learning process.
- If the value of the chosen **η** is too **large**, the jumps on the error surface are also too. Additionally, the movements across the **error surface** would be very uncontrolled. Thus, a **small η** is the desired input, which, however, can cost a huge, often unacceptable amount of time. Experience shows that good learning rate values are in the range of

$$0.01 \leq \eta \leq 0.9$$

Supervised Learning Network Paradigms

The selection of the learning rate

❑ Selection of learning rate:

- The selection of **η** significantly depends on the problem, the network and the training data, so that it is barely possible to give practical advise. But for instance it is popular to **start with a relatively large η, e.g. 0.9**, and to **slowly decrease it down to 0.1.** For simpler problems η can often be kept constant.

Supervised Learning Network Paradigms

The selection of the learning rate

❑ Variation of the learning rate over time

- During training, another stylistic device can be a ***variable learning rate***: In the beginning, **a large learning rate** leads to good results, but later it results in **inaccurate learning**. **A smaller learning** rate is more time-consuming, but the result is **more precise**. Thus, during the learning process the learning rate needs to be **decreased by one order of magnitude once or repeatedly.**

Supervised Learning Network Paradigms

The selection of the learning rate

❑ Variation of the learning rate over time

- A common error (which also seems to be a very neat solution at first glance) is to continually decrease the learning rate. Here it quickly happens that the descent of the **learning rate is larger than the ascent of a hill of the error function we are climbing**. The result is that we simply **get stuck at this ascent**. Solution: Rather reduce the learning rate gradually as mentioned above.

Supervised Learning Network Paradigms

Backpropagation-specific properties

- We have just raised **two backpropagation-specific properties** which are **learning rate and weight change** that can occasionally be a problem (in addition to those which are already caused by gradient descent itself):
- On the one hand, (1) users of backpropagation can choose a bad learning rate. On the other hand, (2) the further the **weights are from the output layer, the slower backpropagation learns.**

Supervised Learning Network Paradigms

Resilient backpropagation network

- **Resilient backpropagation is an extension to backpropagation of error**
- As we discussed before, the backpropagation algorithm have two properties that can caused a problems. For this problems, Martin Riedmiller et al. enhanced backpropagation and called their version **resilient backpropagation (short Rprop)**.
- Before actually dealing with formulas, let us informally show the two primary ideas behind Rprop (and their consequences) to the already familiar backpropagation.

Supervised Learning Network Paradigms

Resilient backpropagation network

❑ Learning rate

- First,, there is no **global learning rate** but each weight $w_{i,j}$ has its own learning rate η.
- These **learning rates are not chosen by the user**, but are **automatically set by Rprop itself.**
- The weight changes **are not static** but are adapted for each time step of Rprop. To account for the temporal change, we have to correctly call it $\eta_{i,j}(t)$.
- **This not only enables more focused learning, also the problem of an increasingly slowed down learning throughout the layers is solved in an elegant way.**

Supervised Learning Network Paradigms

Resilient backpropagation network

❑ Weigh change

- The amount of weight change $\Delta w_{i,j}$ simply directly corresponds to the automatically adjusted learning rate $\eta_{i,j}$. Thus the change in weight is no proportional to the gradient, **it is only influenced by the sign of the gradient.** Until now we still do not know how exactly the η are adapted at run time, but let me anticipate that the resulting process looks considerably less rugged than an error function.
- **In contrast to backprop the weight update step is replaced and an additional step for the adjustment of the learning rate is added. In the next slides, we are know how exactly are these ideas being implemented?**

Supervised Learning Network Paradigms

Resilient backpropagation network

❑ Weight changes are not proportional to the gradient

- **The gradient g hence no longer determines the strength, but only the direction of the weight change.**

- If the sign of the gradient $g = \frac{d\,Err(W)}{dw_{i,j}}$ is positive, we must decrease the weight $w_{i,j}$. So the weight is reduced by $\eta w_{i,j}$. If the sign of the gradient is negative, the weight needs to be increased. So $\eta_{i,j}$ is added to it. If the gradient is exactly 0, nothing happens at all.

Supervised Learning Network Paradigms

Resilient backpropagation network

❑ Definition 8.2 (Weigh change in Rprop)

$$\Delta w_{i,j} = \begin{cases} -\eta_{i,j}(t), & if\ g(t) > 0, \\ +\eta_{i,j}(t), & if\ g(t) < 0, \\ 0 & otherwise \end{cases},$$

Supervised Learning Network Paradigms

Resilient backpropagation network

❑ Many dynamically adjusted learning rates instead of one static

- To adjust the learning rate η , we again have to consider the **associated gradients g** of two time steps: **the gradient that has just passed** **g(t − 1)** and the current one **g(t)**.
- Again, only the sign of the gradient matters, and we now must ask ourselves: What can happen to the sign over two time steps? It can stay the same, and it can flip.
- If the sign changes from **g(t − 1)** to g(t), we have skipped a local minimum in the gradient. Hence, the last update was too large and $\boldsymbol{\eta_{i,j}(t)}$ has to be reduced as compared to the previous $\boldsymbol{\eta_{i,j}(t-1)}$. One can say, that the search needs to be more accurate.

Supervised Learning Network Paradigms

Resilient backpropagation network

❑ Many dynamically adjusted learning rates instead of one static

- In mathematical terms, we obtain a new $\eta_{i,j}(t)$ by multiplying the old $\eta_{i,j}(t-1)$. with a constant $\eta^{\downarrow}$, which is between 1 and 0. In this case we know that in the last time step **(t−1)** something went wrong – hence we additionally reset the weight update for the weight $w_{i,j}$ at time step (t) to 0, so that it not applied at all (not shown in the following formula).
- However, if the sign remains the same, one can perform a (careful!) increase of $\eta_{i,j}$ to get past shallow areas of the error function. Here we obtain our new $\eta_{i,j}(t)$ by multiplying the old $\eta_{i,j}(t)$ with a constant $\eta^{\uparrow}$ which is greater than 1.

Supervised Learning Network Paradigms

Resilient backpropagation network

❑ **Definition 8.3 (Adaptation of learning rate in Rprop)**

$$\eta_{i,j}(t) = \begin{cases} \eta^{\uparrow}\eta_{i,j}(t-1), & if\ g(t-1)g(t) > 0 \\ \eta^{\downarrow}\eta_{i,j}(t-1), & if\ g(t-1)g(t) < 0, \\ \eta_{i,j}(t-1) & otherwise \end{cases}$$

Supervised Learning Network Paradigms

Resilient backpropagation network

❑ More details to use Rprop in practice

- How to choose $\eta_{i,j}(0)$ (i.e. how are the weight-specific learning rates initialized)?
- What are the upper and lower bounds η_{min} and η_{max} for $\eta_{i,j}$?
- How large are $\boldsymbol{\eta}^{\uparrow}$ and $\boldsymbol{\eta}^{\downarrow}$ (i.e. how much are learning rates reinforced or weakened)?

Supervised Learning Network Paradigms

Resilient backpropagation network

- **How to choose $\eta_{i,j}(0)$ (i.e. how are the weight-specific learning rates initialized)?**
  - The initial value for the learning rates should be somewhere in the order of the initialization of the weights **$\eta_{i,j}(0) = 0.1$** has proven to be a good choice.
  - The authors of the Rprop paper explain in an obvious way that this value – as long as it is positive and without an exorbitantly high absolute value – does not need to be dealt with very critically, as it will be quickly overridden by the automatic adaptation anyway.

Supervised Learning Network Paradigms

Resilient backpropagation network

- **What are the upper and lower bounds η_{min} and η_{max} for $\eta_{i,j}$?**
  - One can set $\eta_{max} = 50$, which is used throughout most of the literature.
  - One can set this parameter to lower values in order to allow only very cautious updates.
  - Small update steps should be allowed in any case, so we set $\eta_{min} = 10^{-6}$

Supervised Learning Network Paradigms

Resilient backpropagation network

- **How large are $\eta^{\uparrow}$ and $\eta^{\downarrow}$ (i.e. how much are learning rates reinforced or weakened)?**
  - Let us start with $\eta^{\downarrow}$ If this value is used, we have skipped a minimum, from which we do not know where exactly it lies on the skipped track. Analogous to the procedure of binary search, where the target object is often skipped as well, we assume it was in the middle of the skipped track. So we need to halve the learning rate, which is why the canonical choice $\eta^{\downarrow} = \mathbf{0.5}$ is being selected.

Supervised Learning Network Paradigms

Resilient backpropagation network

- **How large are $\eta^{\uparrow}$ and $\eta^{\downarrow}$ (i.e. how much are learning rates reinforced or weakened)?**
  - If the value of $\eta^{\uparrow}$ is used, learning rates shall be **increased** with caution.
  - Here we cannot generalize the principle of binary search and simply use the value **2.0**, otherwise the learning rate update will end up consisting almost exclusively of changes in direction.
  - Independent of the particular problems, a value of $\eta^{\uparrow}$**=1.2** has proven to be promising. Slight changes of this value have not significantly affected the rate of convergence. This fact allowed for setting this value as a constant as well.

Supervised Learning Network Paradigms

Resilient backpropagation network

- For **deep networks (also called deep learning)** is crucial to prefer **Rprop over the original backpropagation**, because backprop, as already indicated, learns very slowly at weights witch are far from the output layer.
- For problems with a smaller number of layers, I would recommend testing the more widespread backpropagation (with both offline and online learning) and the less common Rprop equivalently.

Initial configuration of a multilayer perceptron

- After having discussed the backpropagation of error learning procedure and knowing how to train an existing network, it would be useful to consider how to implement such a network.

Initial configuration of a multilayer perceptron

Number of layers

❑ Number of layers:

- Two or three may often do the job, but more are also used.
- Let us begin with the trivial circumstance that a network should have one layer of input neurons and one layer of output neurons, which results in at least two layers.
- Additionally, we need – as we have already learned during the examination of linear separability – at least one hidden layer of neurons, if our problem is not linearly separable (which is, as we have seen, very likely).

Initial configuration of a multilayer perceptron

Number of layers

❑ Number of layers:

- In this respect, experience shows that **two hidden neuron layers** (or three trainable weight layers) can be very useful to solve a problem, since many problems can be represented by a hidden layer but are very difficult to learn.
- One should keep in mind that any additional layer generates additional sub-minima of the error function in which we can get stuck. All these things considered, a promising way is to try it with one hidden layer at first and if that fails, retry with two layers. Only if that fails, one should consider more layers. However, given the increasing calculation power of current computers, ***deep networks*** with a lot of layers are also used with success.

Initial configuration of a multilayer perceptron

Number of neurons

❑ The number of neurons has to be tested:

- The number of neurons (apart from input and output layer, where the number of input and output neurons is already defined by the problem statement) principally corresponds to the number of free parameters of the problem to be represented.
- Since we have already discussed the network capacity with respect to memorizing or a too imprecise problem representation**, it is clear that our goal is to have as few free parameters as possible but as many as necessary**.

Initial configuration of a multilayer perceptron

Number of neurons

❑ The number of neurons has to be tested:

- But we also know that there is no standard solution for the question of how many neurons should be used. Thus, the most useful approach is **to initially train with only a few neurons and to repeatedly train new networks with more neurons** until the result significantly improves and, particularly, the generalization performance is not affected (bottom-up approach).

Initial configuration of a multilayer perceptron

Selection of activation function

❑ Selecting an activation function:

- The first question to be asked is whether we actually want to use the same activation function in the hidden layer and in the output layer – no one prevents us from choosing different functions. Generally, the activation function is the same for all hidden neurons as well as for the output neurons respectively.

Initial configuration of a multilayer perceptron

Selection of activation function

❑ Selecting an activation function:

- For tasks of ***function approximation*** it has been found reasonable to use the hyperbolic tangent as activation function of the hidden neurons, while a **linear activation function** is used in the output. The latter is absolutely necessary so that we do not generate a limited output interval. Contrary to the input layer which uses linear activation functions as well, the output layer still processes information, because it has threshold values. However, linear activation functions in the output can also cause huge learning steps and jumping over good minima in the error surface. This can be avoided by setting the learning rate to very small values in the output layer.

Initial configuration of a multilayer perceptron

Selection of activation function

❑ Selecting an activation function:

- An unlimited output interval is not essential for ***pattern recognition*** tasks. If the hyperbolic tangent is used in any case, the output interval will be a bit larger. Unlike with the hyperbolic tangent, with the Fermi function it is difficult to learn something far from the threshold value (where its result is close to 0). However, here a lot of freedom is given for selecting an activation function. But generally, the disadvantage of sigmoid functions is the fact that they hardly learn something for values far from they threshold value, unless the network is modified.

Initial configuration of a multilayer perceptron

Number of layers

- **Weights should be initialized with small, randomly chosen values:**
  - The initialization of weights is not as trivial as one might think. If they are simply initialized with 0, there will be no change in weights at all. If they are all initialized by the same value, they will all change equally during training. The simple solution of this problem is called **symmetry breaking**.

Initial configuration of a multilayer perceptron

Number of layers

- **Weights should be initialized with small, randomly chosen values:**
  - **Symmetry breaking** is the initialization of weights with small random values. The range of random values could be the interval [−0.5; 0.5] not including 0 or values very close to 0. This random initialization has a nice side effect: Chances are that the average of network inputs is close to 0, a value that hits (in most activation functions) the region of the greatest derivative, allowing for strong learning impulses right from the start of learning.

References

- Kriesel, David. "A Brief Introduction to Neural Networks. 2007." URL http://www. dkriesel. com (2007).
- http://neuralnetworksanddeeplearning.com/chap2.html

Any Questions!?

Thank you