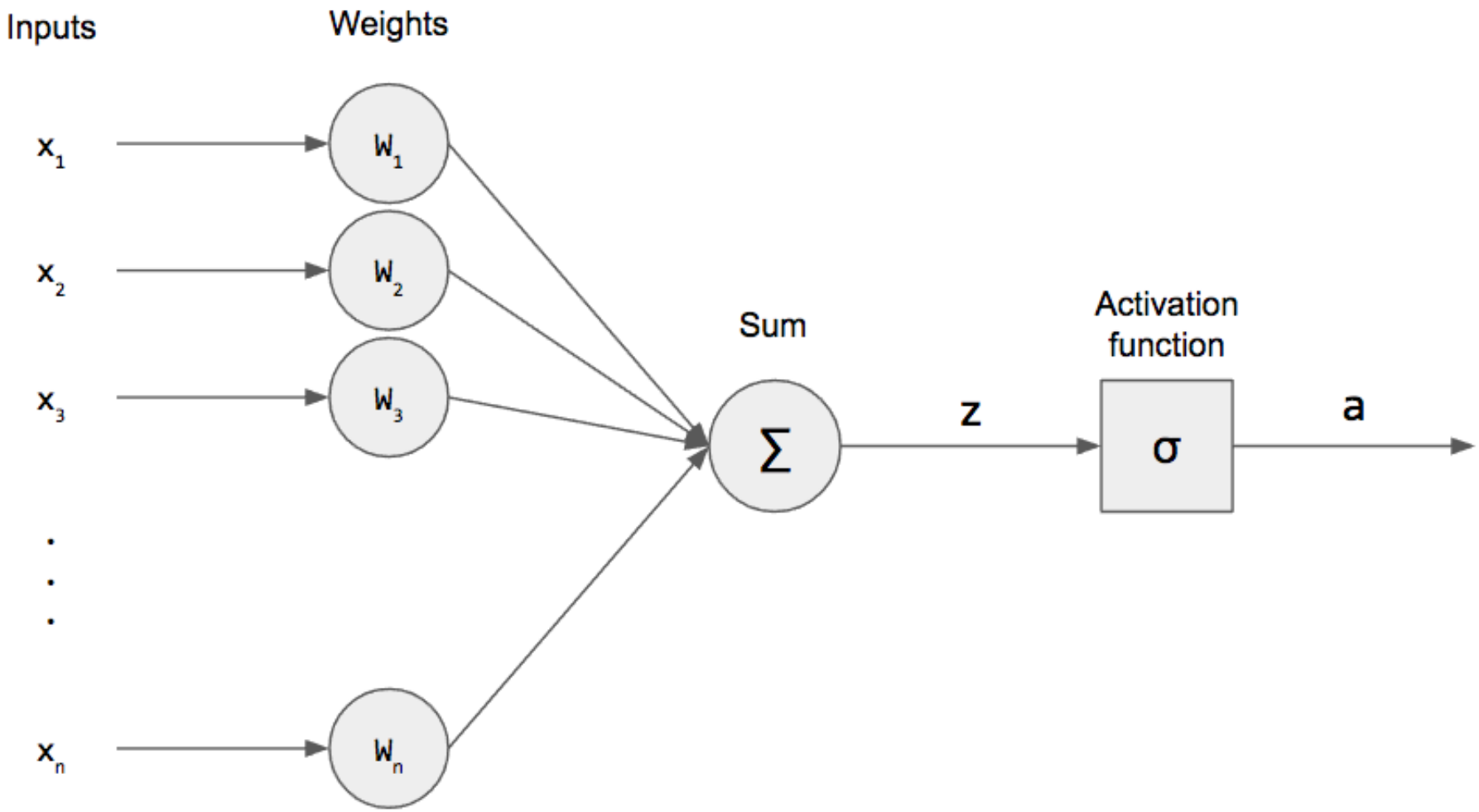
LES RESEAUX DE NEURONES

Perceptron simple

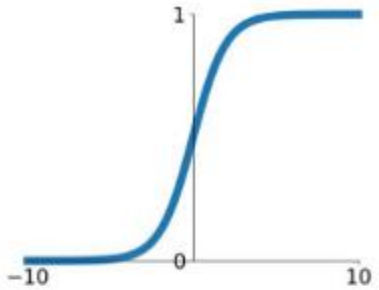


LES RESEAUX DE NEURONES

Fonctions d'activation

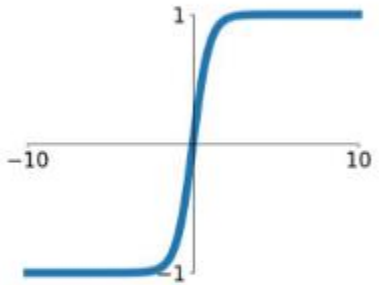
Sigmoid

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



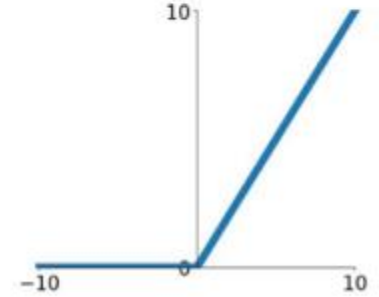
tanh

$$\tanh(x)$$



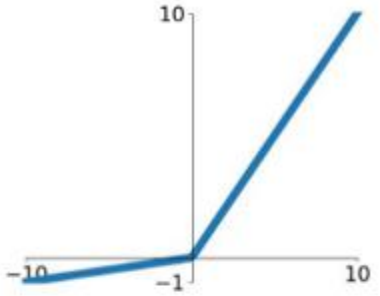
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

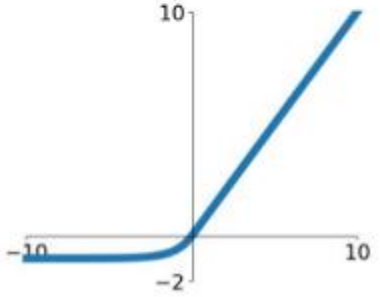


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

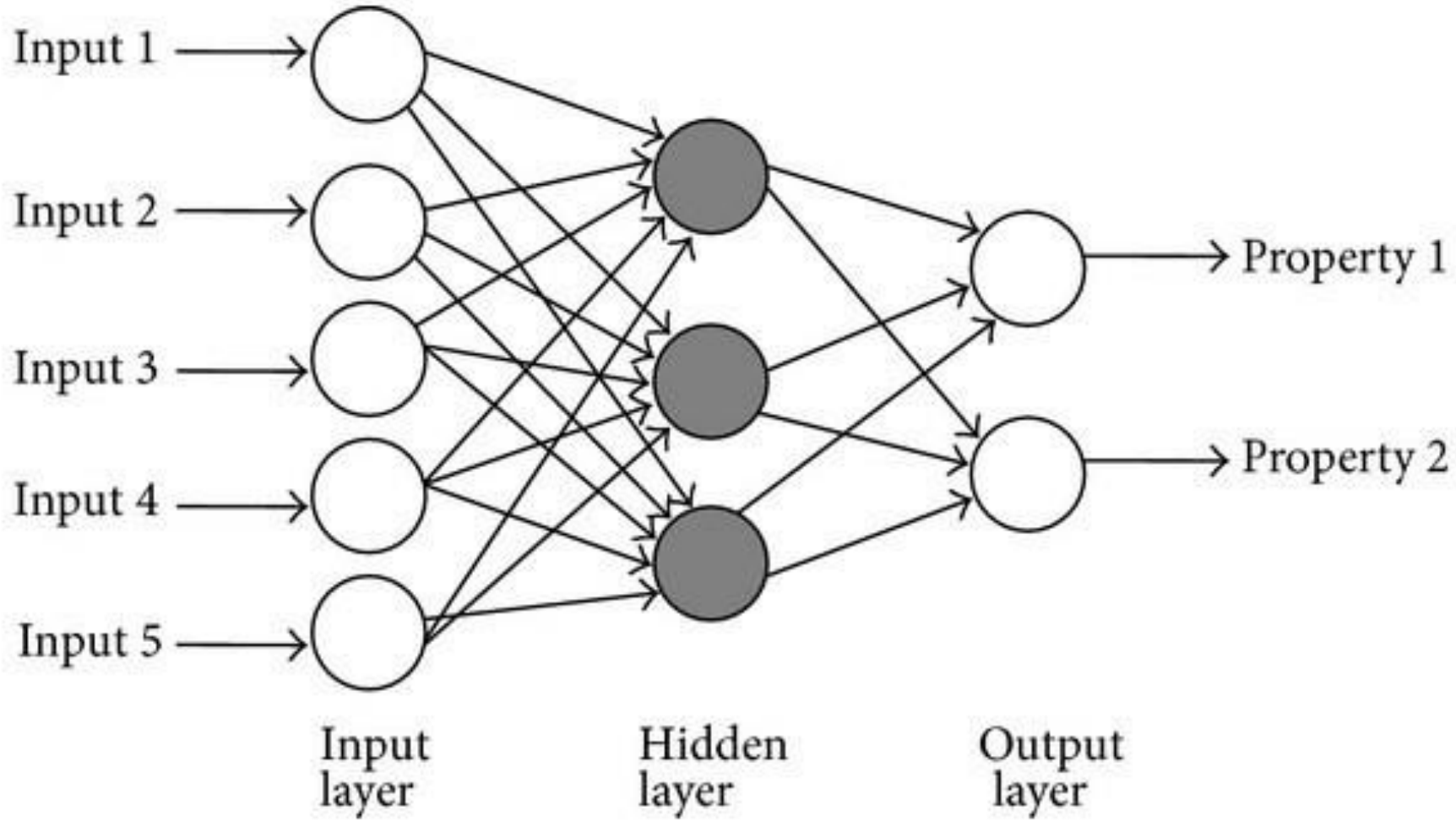
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



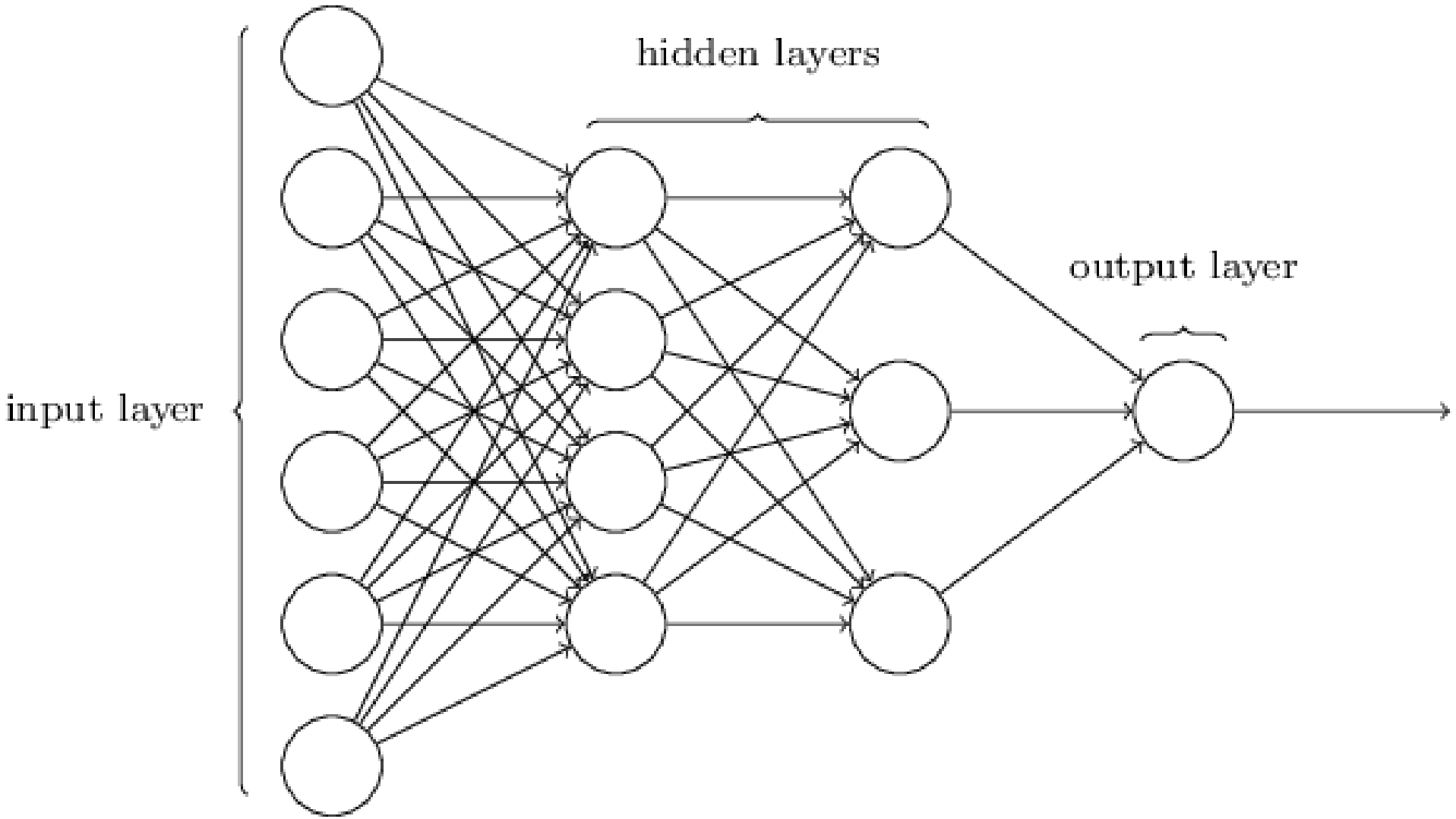
LES RESEAUX DE NEURONES

Perceptron (à couche cachée)



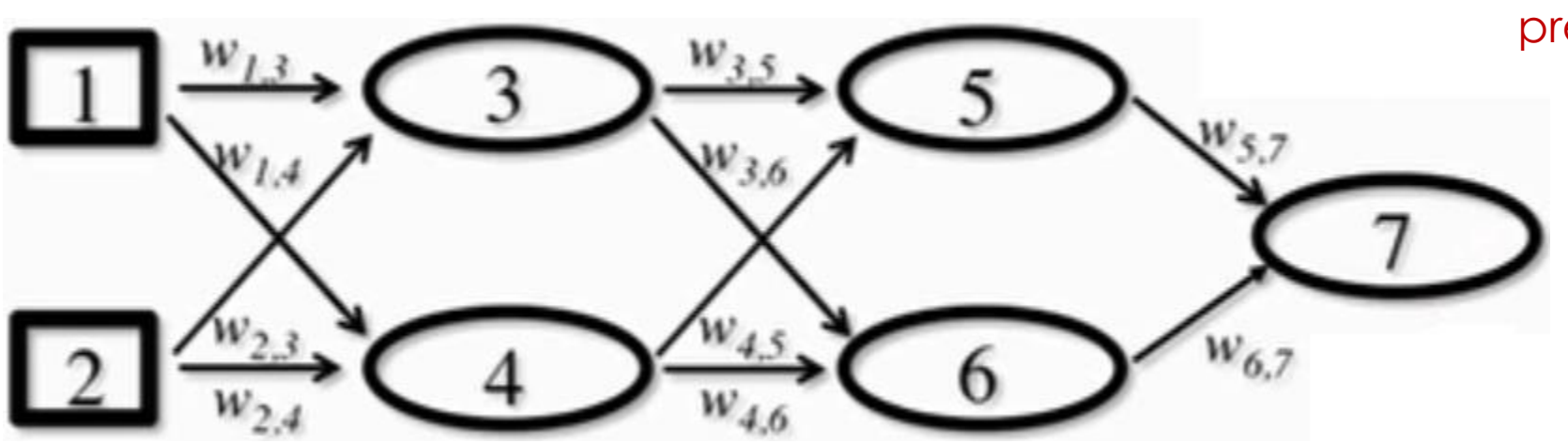
LES RESEAUX DE NEURONES

Perceptron multicouches



LES RESEAUX DE NEURONES

Algorithme d'apprentissage par rétropropagation



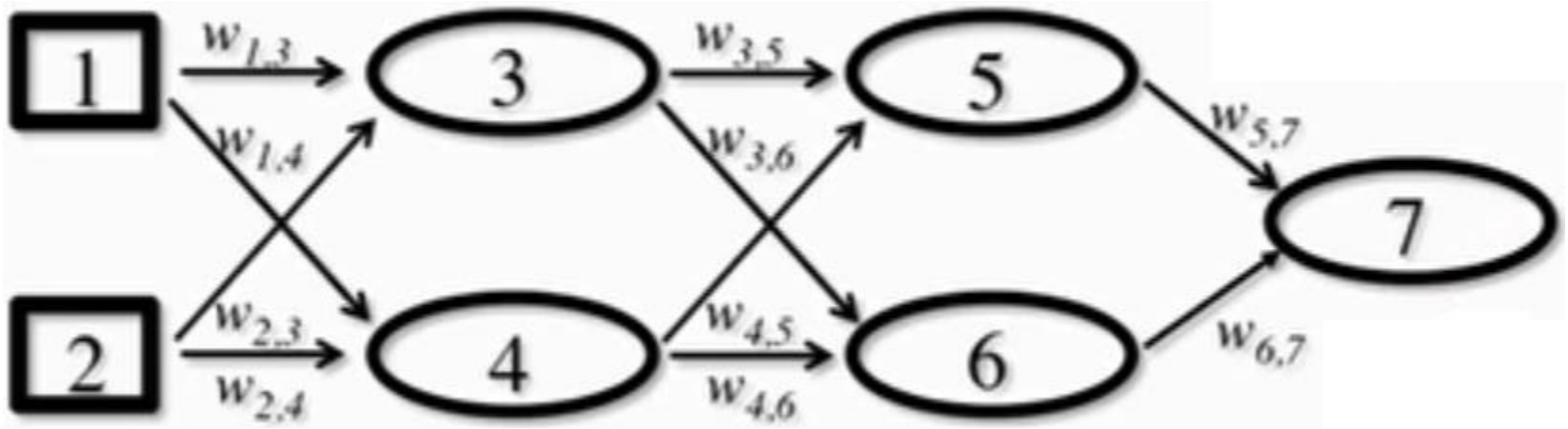
On note :

- a_j l'activité du neurone j
- in_j l'activité du neurone avant l'application de la fonction d'activation
  - $a_j = \sigma(in_j) = \sigma(\sum_i w_{i,j} a_i)$
- $h(x)$ la prédiction

LES RESEAUX DE NEURONES

Algorithme d'apprentissage par rétropropagation

Propagation avant

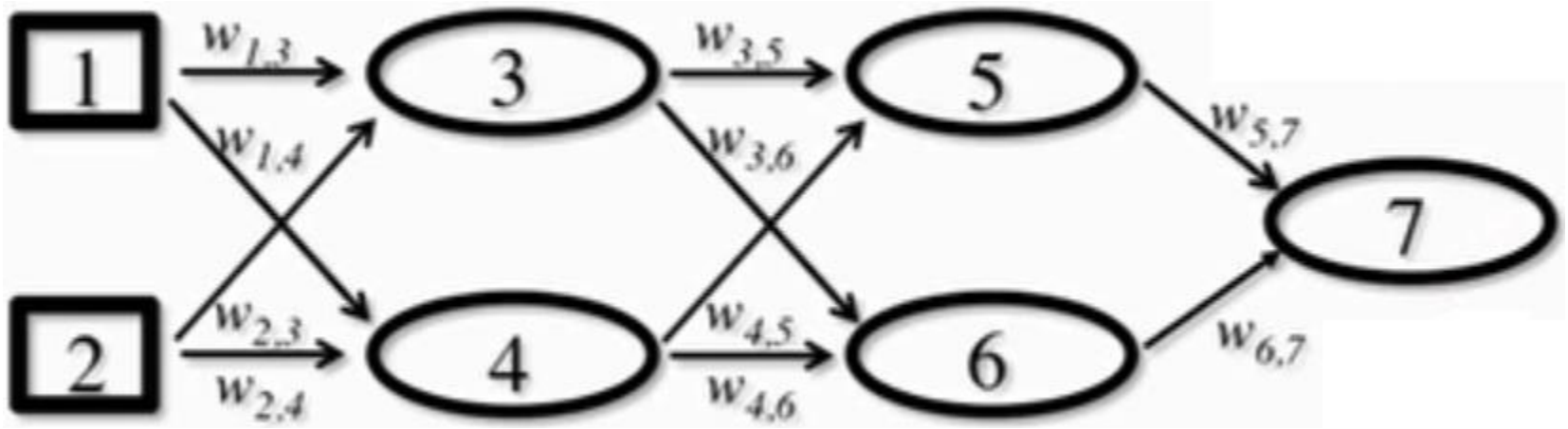


Propagation arrière

LES RESEAUX DE NEURONES

Algorithme d'apprentissage par rétropropagation

Propagation avant

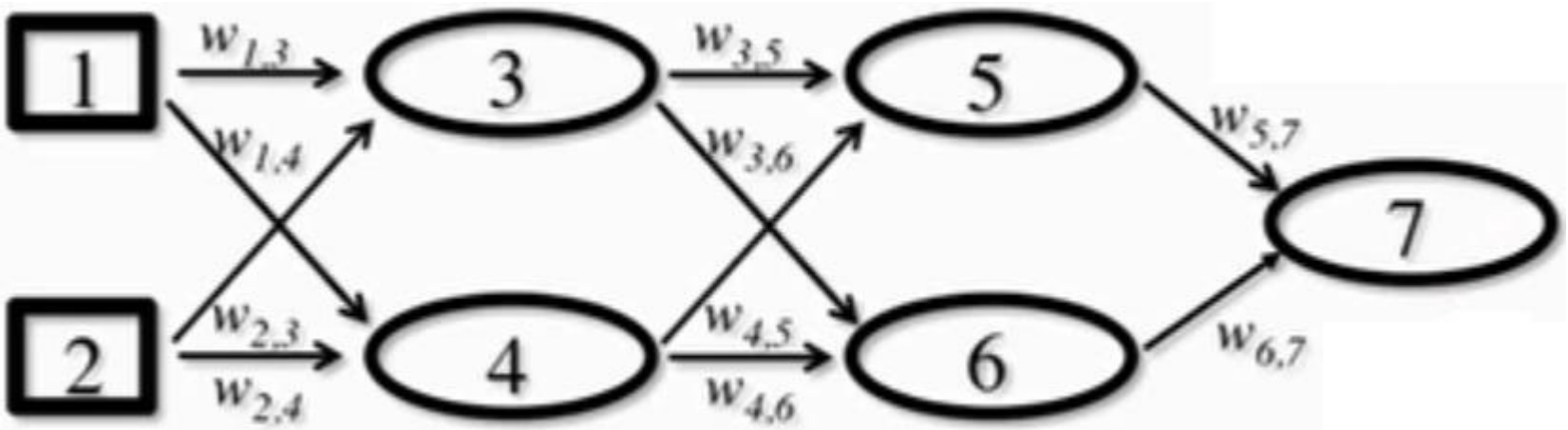


$$a_k = \sigma(\sum_j w_{j,k} a_j)$$

LES RESEAUX DE NEURONES

Algorithme d'apprentissage par rétropropagation

Mise à jour des poids



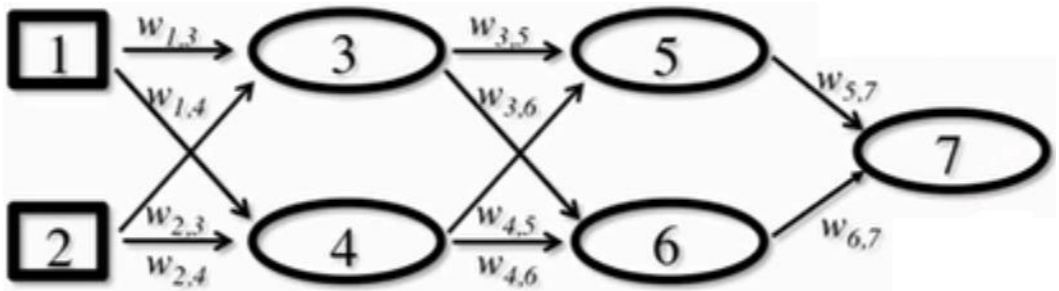
$$w_{i,j} = w_{i,j} - \alpha \frac{\partial}{\partial w_{i,j}} Loss(y_t, h_w(x_t))$$

- α : taux d'apprentissage
- $Loss$: taux de perte associée à la prédiction courante

LES RESEAUX DE NEURONES

Algorithme d'apprentissage par rétropropagation

Mise à jour des poids (simplification)



$$w_{i,j} = w_{i,j} - \alpha \underbrace{\frac{\partial}{\partial in_j} Loss(y_t, h_w(x_t))}_{-\Delta[j]} \underbrace{\frac{\partial}{\partial w_{i,j}} in_j}_{a_i}$$

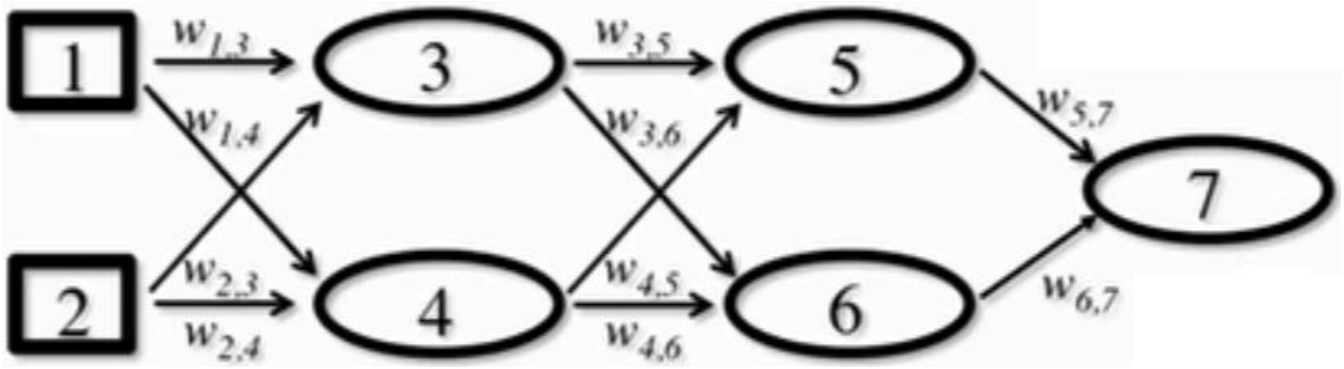
- **Réécriture de la règle de mise à jour :**

$$w_{i,j} = w_{i,j} + \alpha\, a_i \Delta[j]$$

LES RESEAUX DE NEURONES

Algorithme d'apprentissage par rétropropagation
Mise à jour des poids (simplification)

Propagation arrière



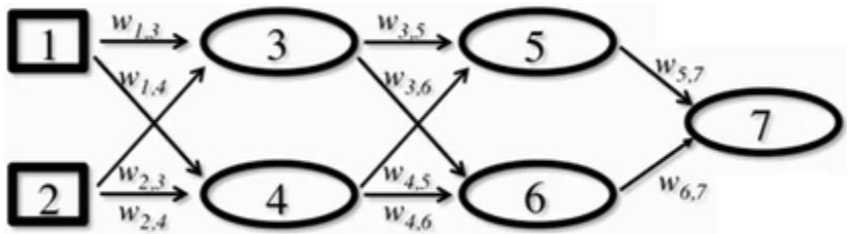
- **Simplification de** $\Delta[j]$**:**

$$\Delta[j] = \sigma(in_j)(1 - \sigma(in_j)) \sum_k w_{j,k} \Delta[k]$$

Algorithme d'apprentissage par rétropropagation

function BACK-PROP-LEARNING(*examples*, *network*) **returns** a neural network
inputs: *examples*, a set of examples, each with input vector **x** and output vector **y**
network, a multilayer network with L layers, weights $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

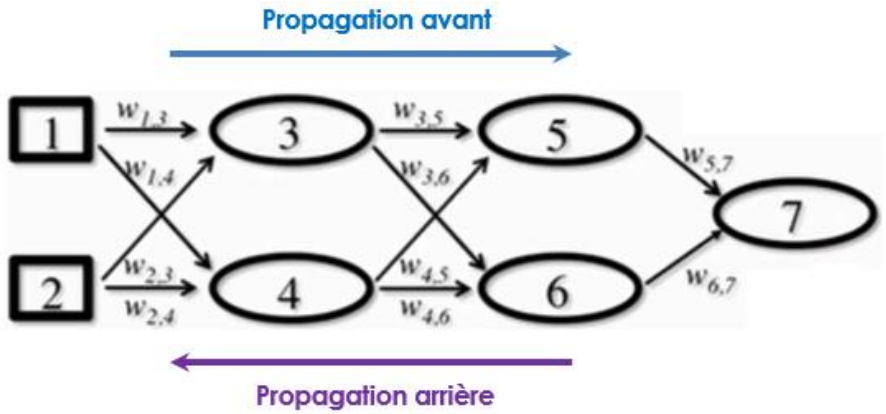
for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow$ a small random number
repeat
for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
/* *Propagate the inputs forward to compute the outputs* */
for each node i in the input layer **do**
$a_i \leftarrow x_i$
for $\ell = 2$ **to** L **do**
for each node j in layer ℓ **do**
$in_j \leftarrow \sum_i w_{i,j}\, a_i$
$a_j \leftarrow g(in_j)$
/* *Propagate deltas backward from output layer to input layer* */
for each node j in the output layer **do**
$\Delta[j] \leftarrow y_j - a_j \quad (= -\partial Loss/\partial in_j)$
for $\ell = L - 1$ **to** 1 **do**
for each node i in layer ℓ **do**
$\Delta[i] \leftarrow g(in_i)(1 - g(in_i)) \sum_j w_{i,j}\Delta[j]$
/* *Update every weight in network using deltas* */
for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
until some stopping criterion is satisfied
return *network*



$g = sigmoid$

LES RESEAUX DE NEURONES

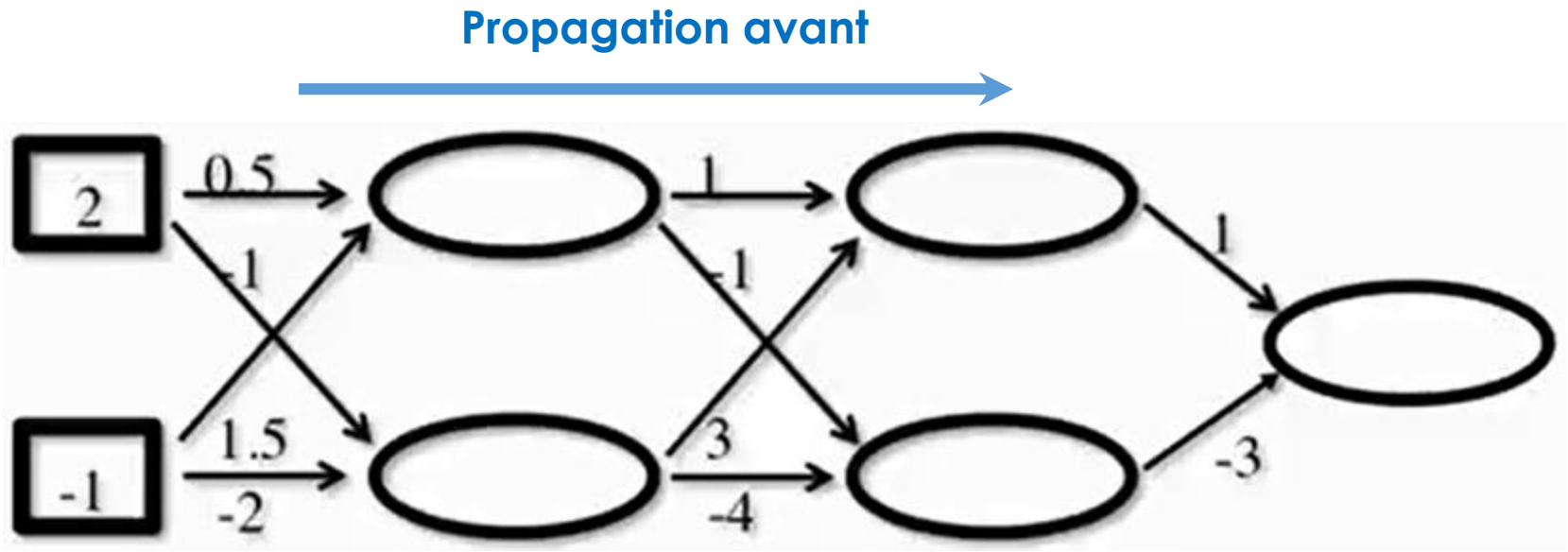
Algorithme d'apprentissage par rétropropagation



- **Propagation avant :** $a_k = \sigma(\sum_j w_{j,k} a_j)$
- **Mise à jour des poids :** $w_{i,j} = w_{i,j} + \alpha\, a_i \Delta[j]$
- **Ecart de la perte :** $\Delta[j] = \sigma(in_j)(1 - \sigma(in_j)) \sum_k w_{j,k} \Delta[k]$

Algorithme d'apprentissage par rétropropagation

Exemple : $x = [2, -1], \quad y = 1, \quad \alpha = 0.1$

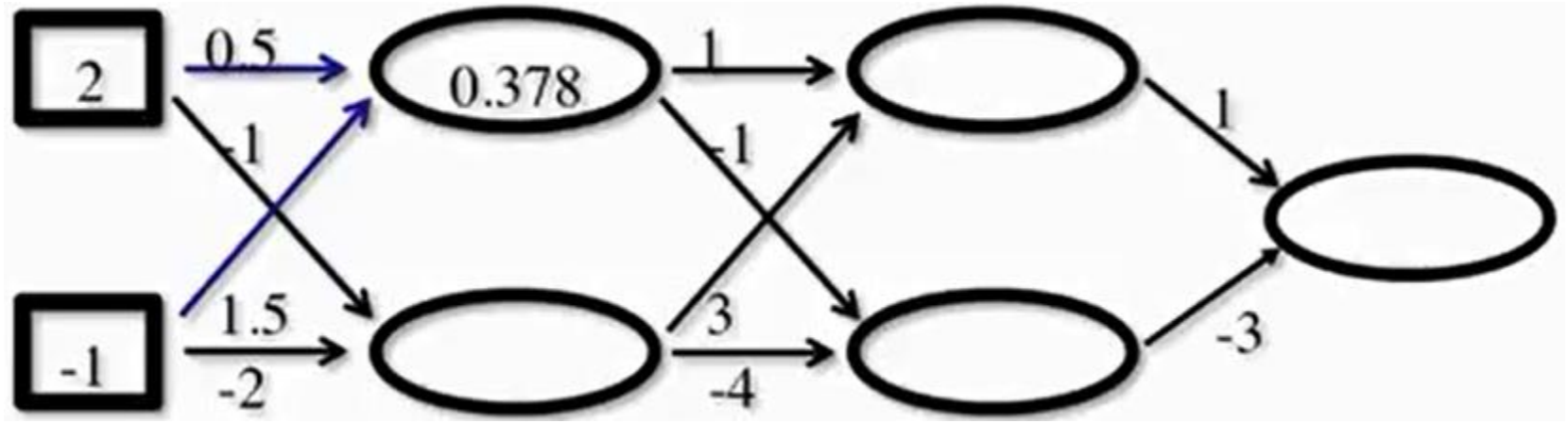


$$a_k = \sigma(\sum_j w_{j,k} a_j)$$

Algorithme d'apprentissage par rétropropagation

Exemple : $x = [2, -1], \quad y = 1, \quad \alpha = 0.1$

Propagation avant



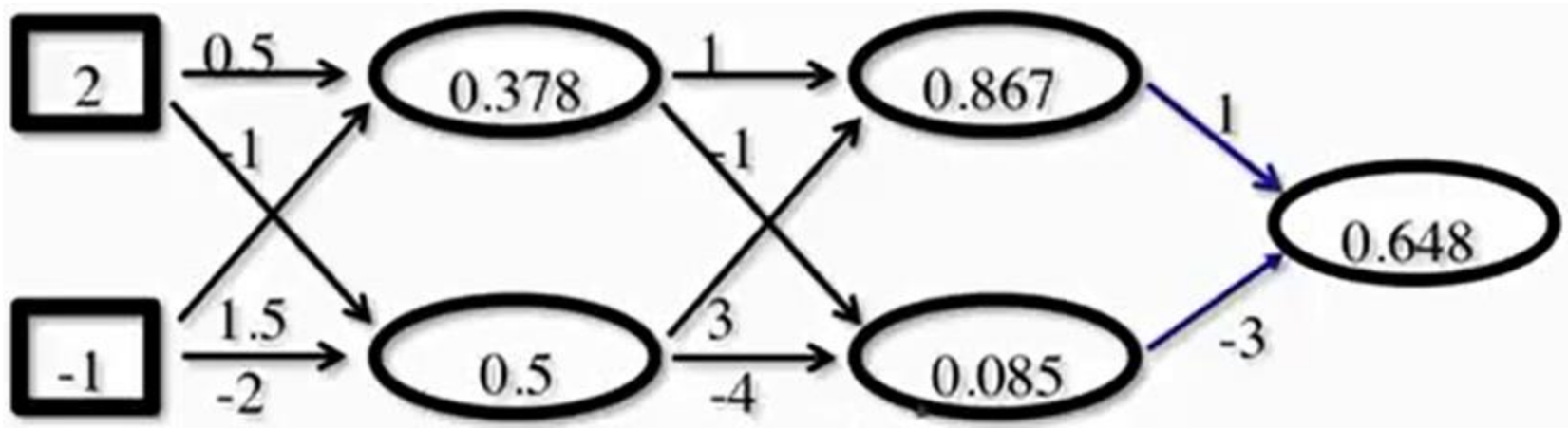
$$a_k = \sigma\left(\sum_j w_{j,k} a_j\right)$$

$$a_3 = \sigma(0.5 \times 2 + 1.5 \times -1) = \sigma(-0.5) = 0.378$$

Algorithme d'apprentissage par rétropropagation

Exemple : $x = [2, -1], \quad y = 1, \quad \alpha = 0.1$

Propagation avant



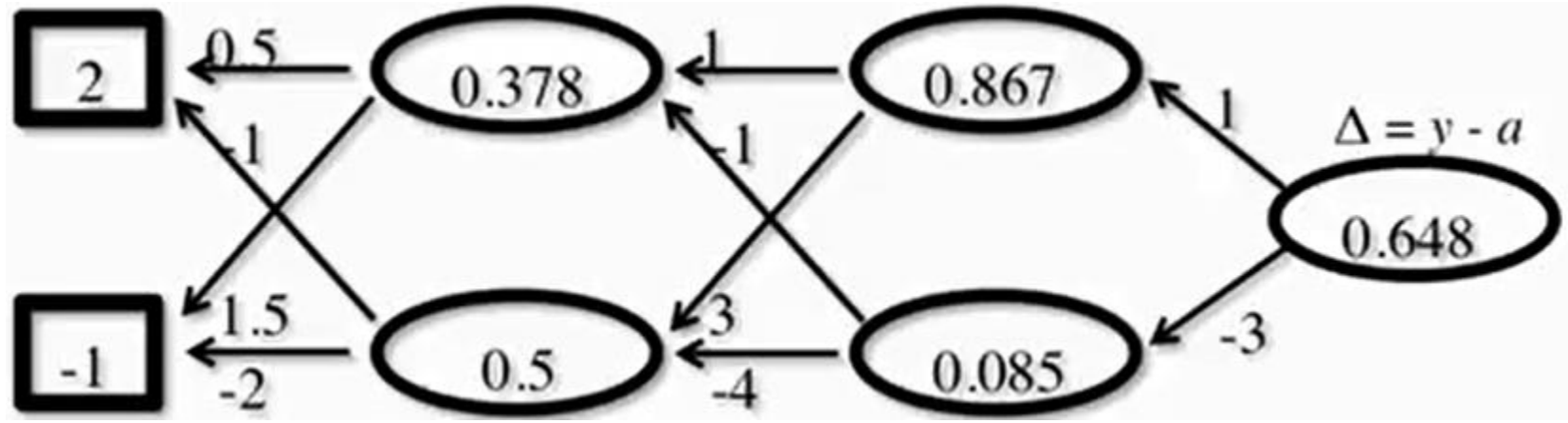
$$a_k = \sigma\left(\sum_j w_{j,k} a_j\right)$$

$$a_7 = \sigma(1 \times 0.867 + (-3) \times 0.085) = \sigma(0.612) = 0.648$$

Algorithme d'apprentissage par rétropropagation

Exemple : $x = [2, -1], \quad y = 1, \quad \alpha = 0.1$

Propagation arrière

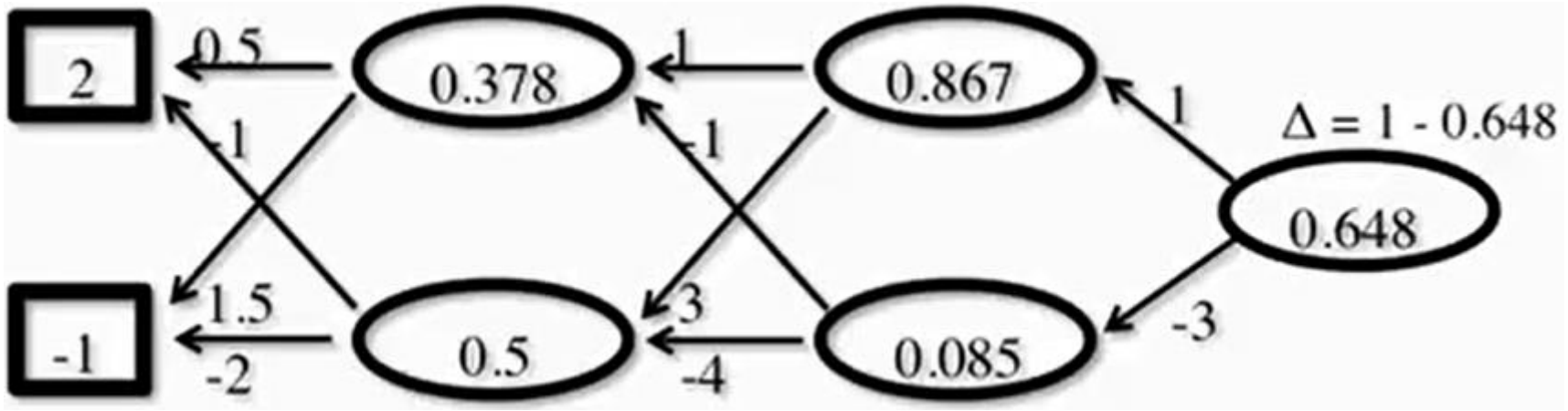


$$\Delta[j] = \sigma(in_j)(1 - \sigma(in_j)) \sum_k w_{j,k} \Delta[k]$$

Algorithme d'apprentissage par rétropropagation

Exemple : $x = [2, -1], \quad y = 1, \quad \alpha = 0.1$

Propagation arrière

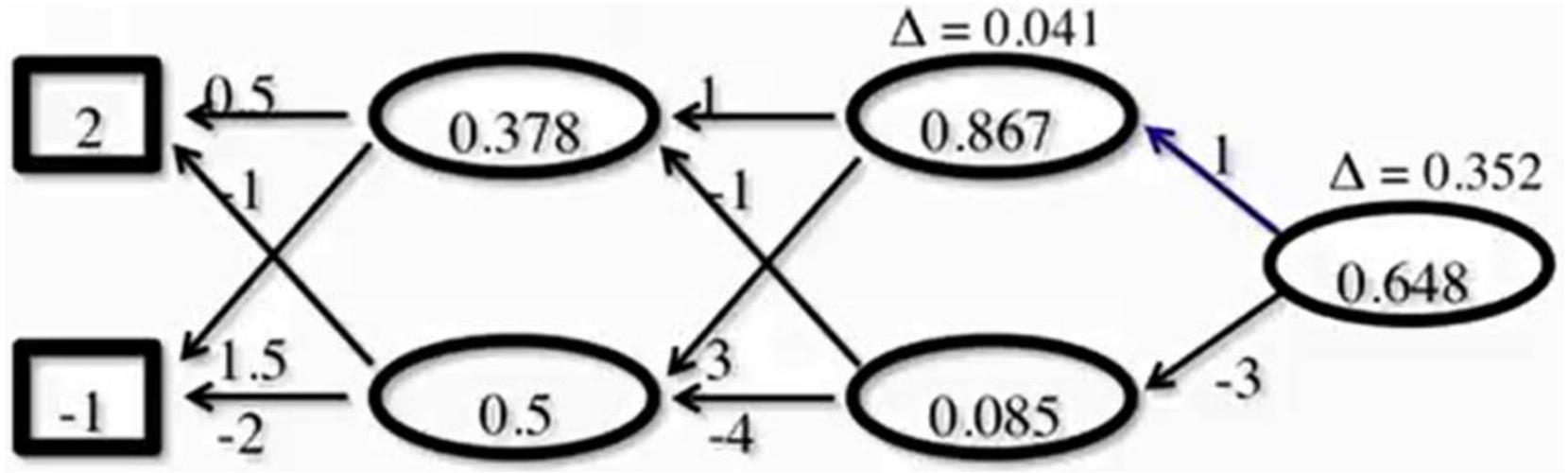


$$\Delta[j] = \sigma(in_j)(1 - \sigma(in_j)) \sum_k w_{j,k} \Delta[k]$$

Algorithme d'apprentissage par rétropropagation

Exemple : $x = [2, -1], \quad y = 1, \quad \alpha = 0.1$

Propagation arrière



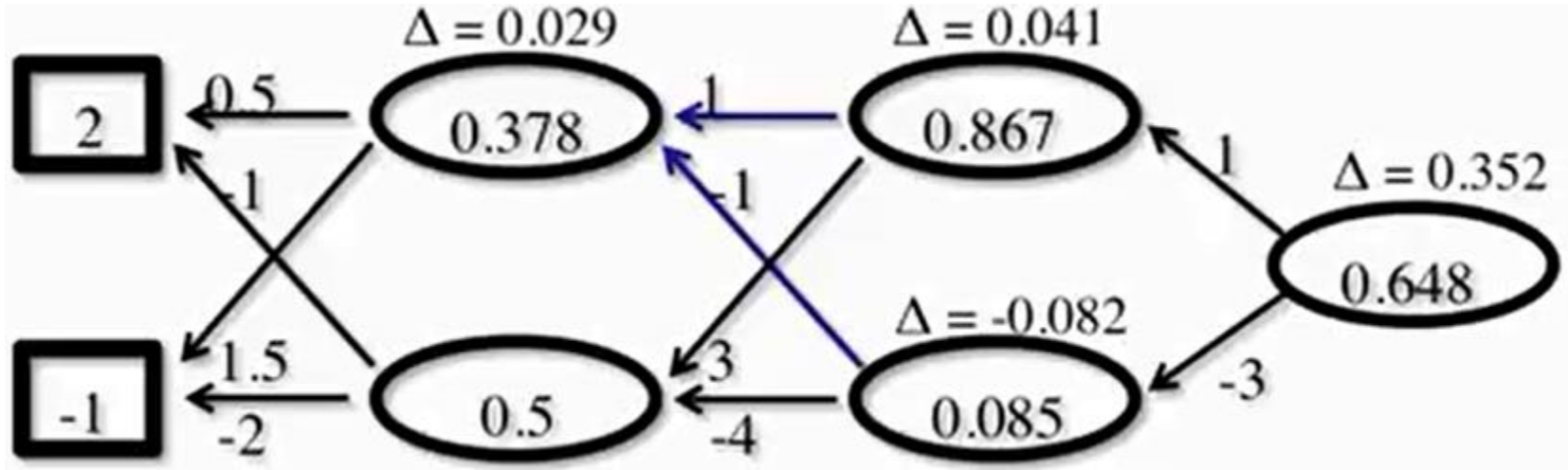
$$\Delta[j] = \sigma(in_j)(1 - \sigma(in_j)) \sum_k w_{j,k} \Delta[k]$$

$$\Delta[5] = 0.867 \times (1 - 0.867) \times 1 \times 0.352 = 0.041$$

Algorithme d'apprentissage par rétropropagation

Exemple : $x = [2, -1], \quad y = 1, \quad \alpha = 0.1$

Propagation arrière



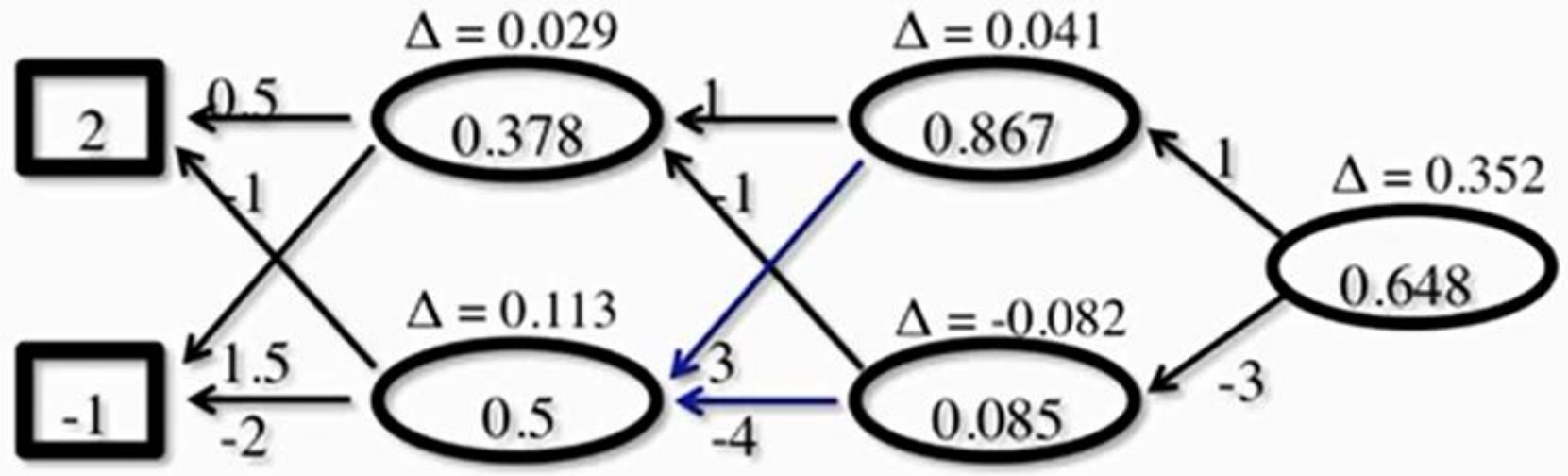
$$\Delta[j] = \sigma(in_j)(1 - \sigma(in_j)) \sum_k w_{j,k} \Delta[k]$$

$$\Delta[3] = 0.378 \times (1 - 0.378) \times (1 \times 0.041 + (-1) \times (-0.082) = 0.029$$

Algorithme d'apprentissage par rétropropagation

Exemple : $x = [2, -1], \quad y = 1, \quad \alpha = 0.1$

Propagation arrière



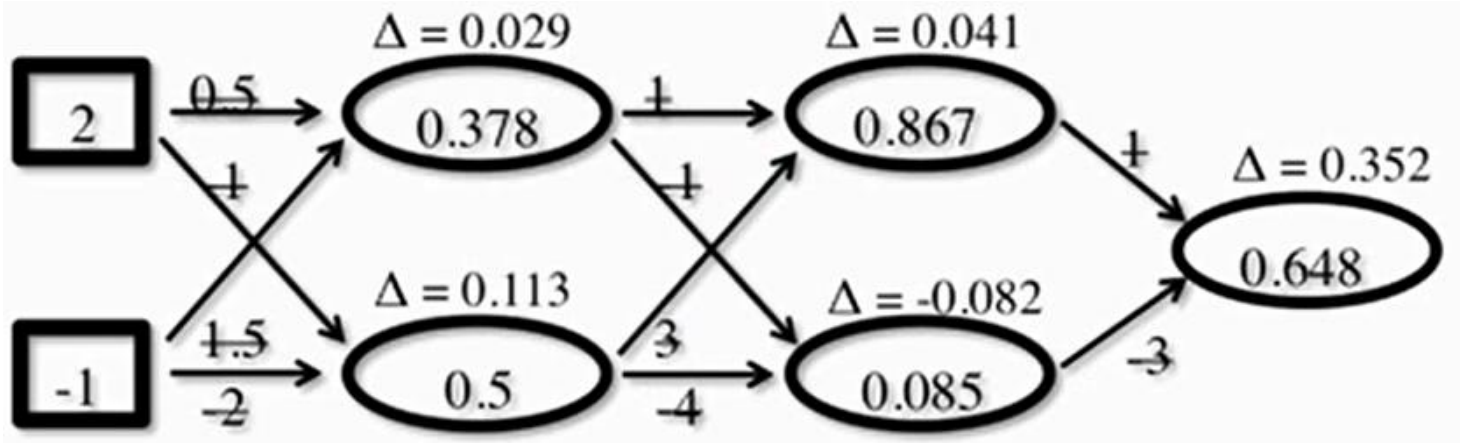
$$\Delta[j] = \sigma(in_j)(1 - \sigma(in_j)) \sum_k w_{j,k} \Delta[k]$$

$$\Delta[4] = 0.5 \times (1 - 0.5) \times (3 \times 0.041 + (-4) \times (-0.082) = 0.113$$

Algorithme d'apprentissage par rétropropagation

Exemple : $x = [2, -1], \quad y = 1, \quad \alpha = 0.1$

Mise à jour des poids



$$w_{i,j} = w_{i,j} + \alpha\, a_i \Delta[j]$$

$w_{1,3} \leftarrow 0.5 + 0.1 * 2 * 0.029 = 0.506$
$w_{1,4} \leftarrow -1 + 0.1 * 2 * 0.113 = -0.977$
$w_{2,3} \leftarrow 1.5 + 0.1 * -1 * 0.029 = 1.497$
$w_{2,4} \leftarrow -2 + 0.1 * -1 * 0.113 = -2.011$

$w_{3,5} \leftarrow 1 + 0.1 * 0.378 * 0.041 = 1.002$
$w_{3,6} \leftarrow -1 + 0.1 * 0.378 * -0.082 = -1.003$
$w_{4,5} \leftarrow 3 + 0.1 * 0.5 * 0.041 = 3.002$
$w_{4,6} \leftarrow -4 + 0.1 * 0.5 * -0.082 = -4.004$

$w_{5,7} \leftarrow 1 + 0.1 * 0.867 * 0.352 = 1.031$
$w_{6,7} \leftarrow -3 + 0.1 * 0.085 * 0.352 = -2.997$

A mostly complete chart of

Neural Networks

- Backfed Input Cell
- Input Cell
- Noisy Input Cell
- Hidden Cell
- Probablistic Hidden Cell
- Spiking Hidden Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- Different Memory Cell
- Kernel
- Convolution or Pool

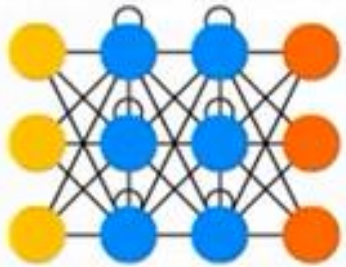
Perceptron (P)

Feed Forward (FF)

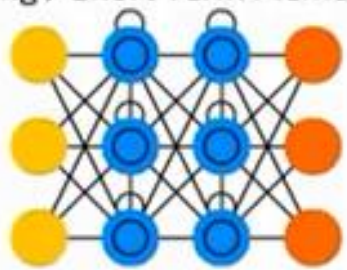
Radial Basis Network (RBF)

Deep Feed Forward (DFF)

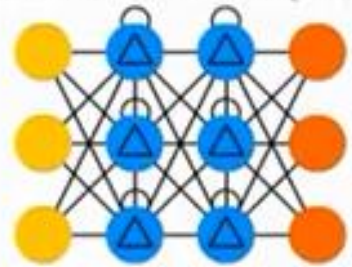
Recurrent Neural Network (RNN)



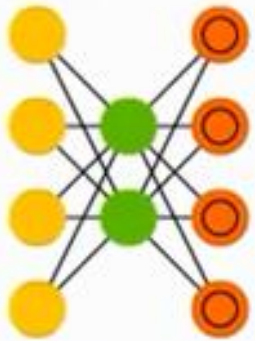
Long / Short Term Memory (LSTM)



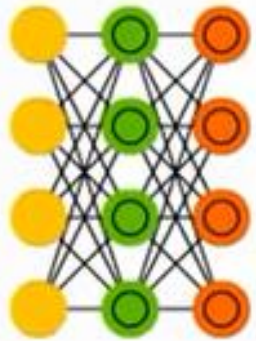
Gated Recurrent Unit (GRU)



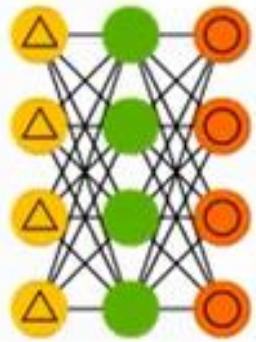
Auto Encoder (AE)



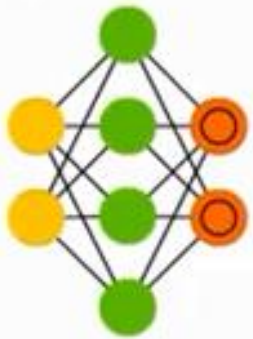
Variational AE (VAE)



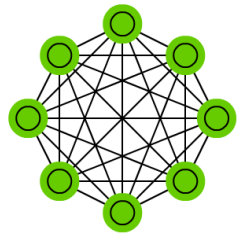
Denoising AE (DAE)



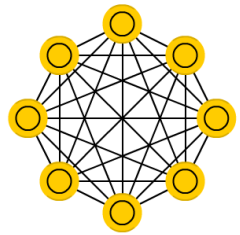
Sparse AE (SAE)



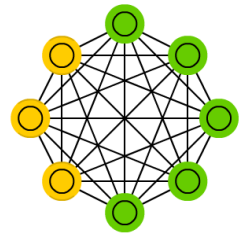
Markov Chain (MC)



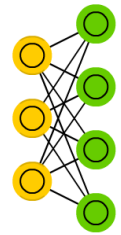
Hopfield Network (HN)



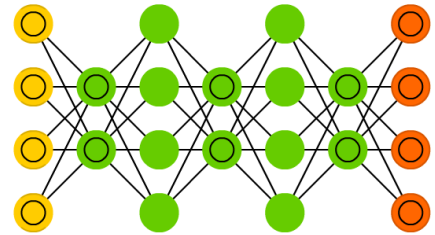
Boltzmann Machine (BM)



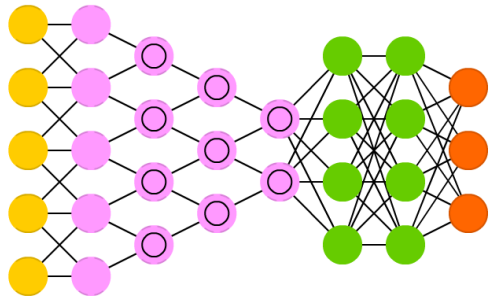
Restricted BM (RBM)



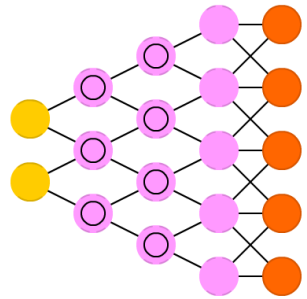
Deep Belief Network (DBN)



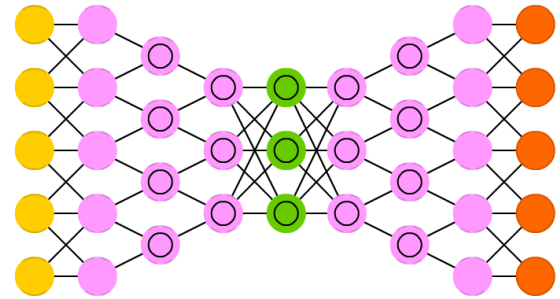
Deep Convolutional Network (DCN)



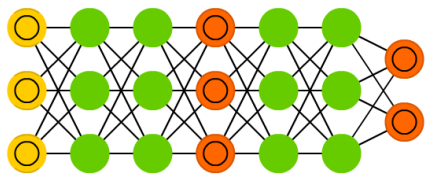
Deconvolutional Network (DN)



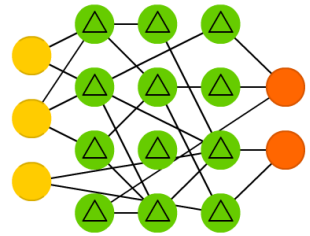
Deep Convolutional Inverse Graphics Network (DCIGN)



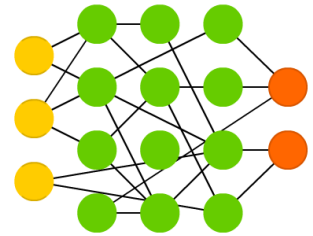
Generative Adversarial Network (GAN)



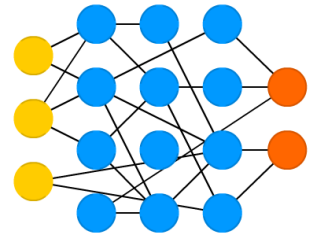
Liquid State Machine (LSM)



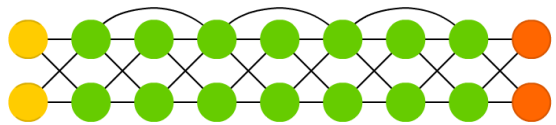
Extreme Learning Machine (ELM)



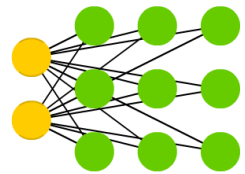
Echo State Network (ESN)



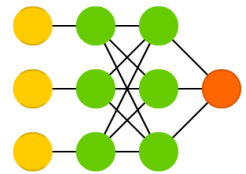
Deep Residual Network (DRN)



Kohonen Network (KN)



Support Vector Machine (SVM)



Neural Turing Machine (NTM)