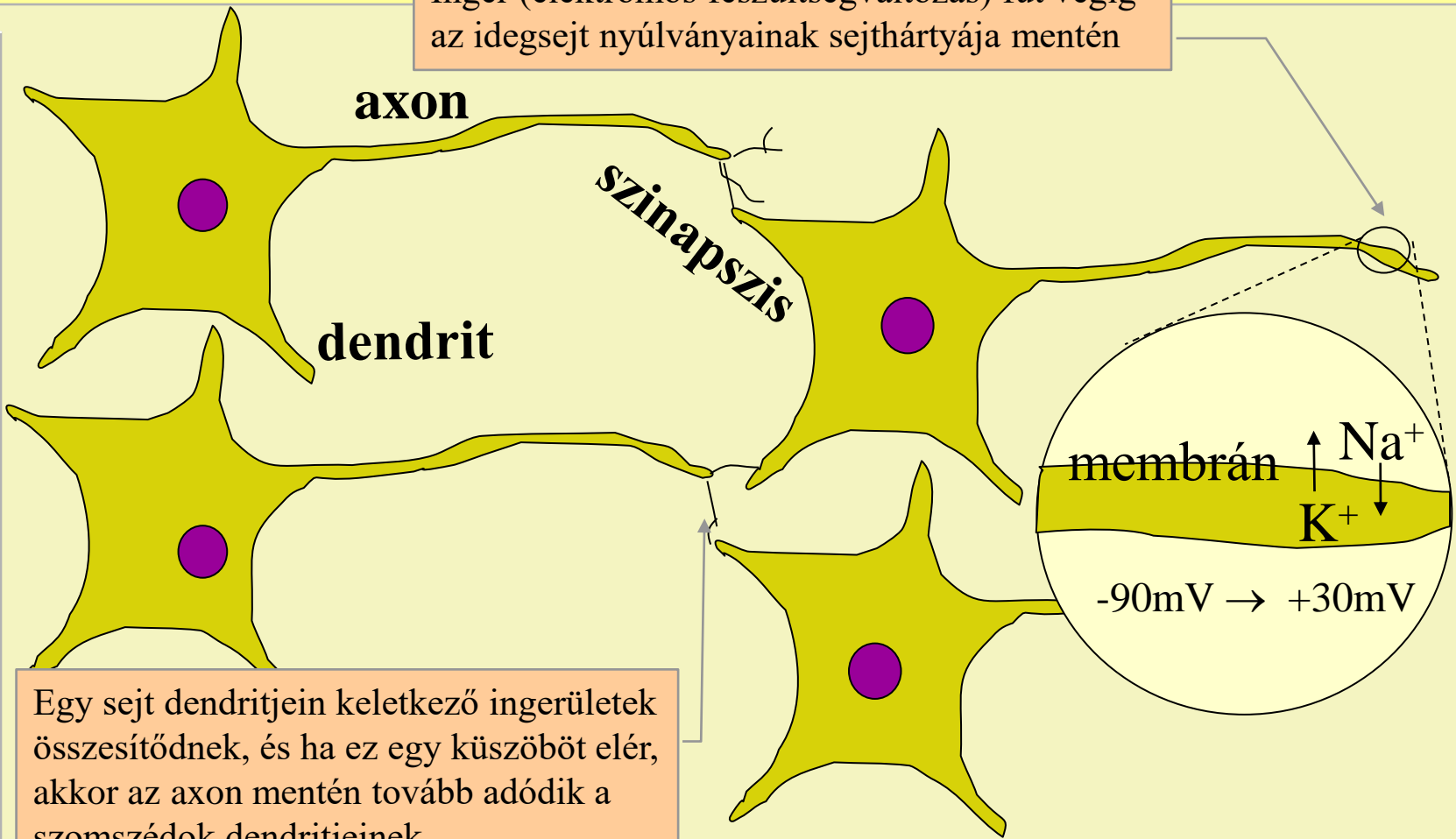


# 3. Mesterséges neuronhálók

Inger (elektromos feszültségváltozás) fut végig az idegsejt nyúlványainak sejthártyája mentén

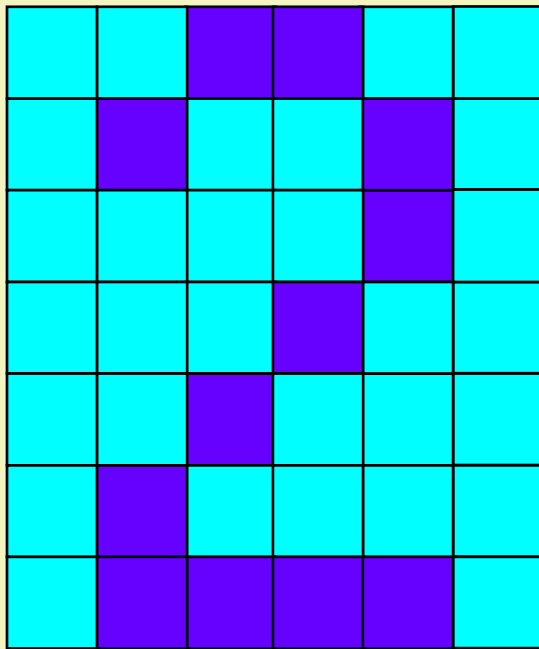


Egy sejt dendritjein keletkező ingerületek összesítődnek, és ha ez egy küszöböt elér, akkor az axon mentén tovább adódik a szomszédok dendritjeinek.

# Számjegy felismerés egy mesterséges neuronhálózata

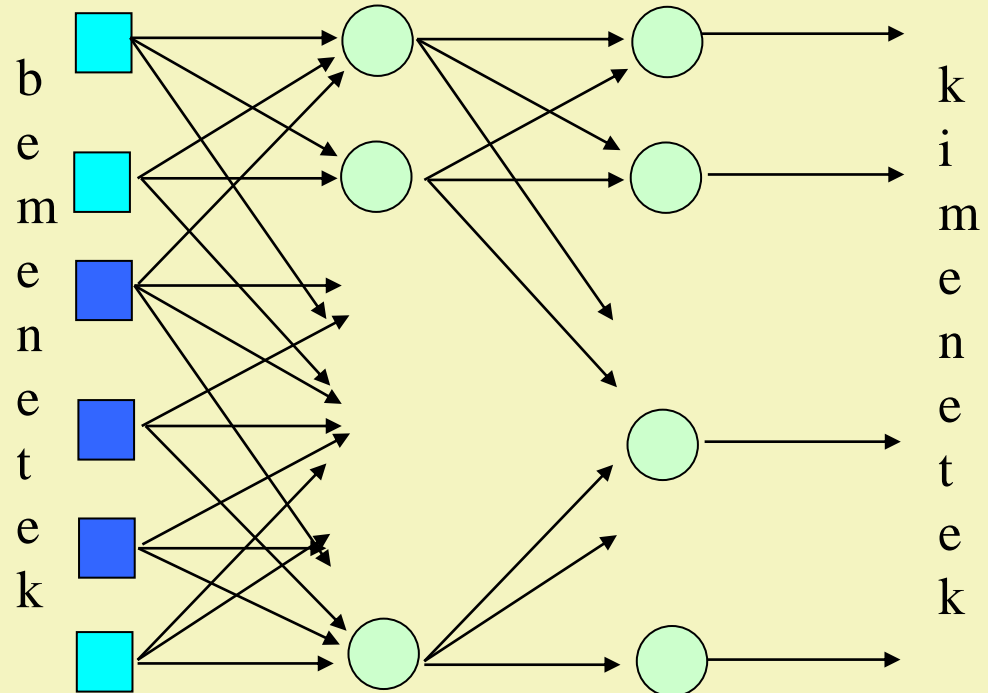
Bemeneti értékek száma: 42

Minden pixel generál  
egy bemeneti értéket.



Kimenetek száma: 10

Minden számjegyhez tartozik egy neuron a  
kimeneti rétegben, amely akkor tüzel, ha az  
adott számjegyet véli felismerni a rendszer.



# *Mesterséges neuronhálóak alkotóelemei*

## □ Mesterséges neuron

- bemenő értékekből kimenő értéket számoló egység, amelynek számítási képlete változtatható, tanítható

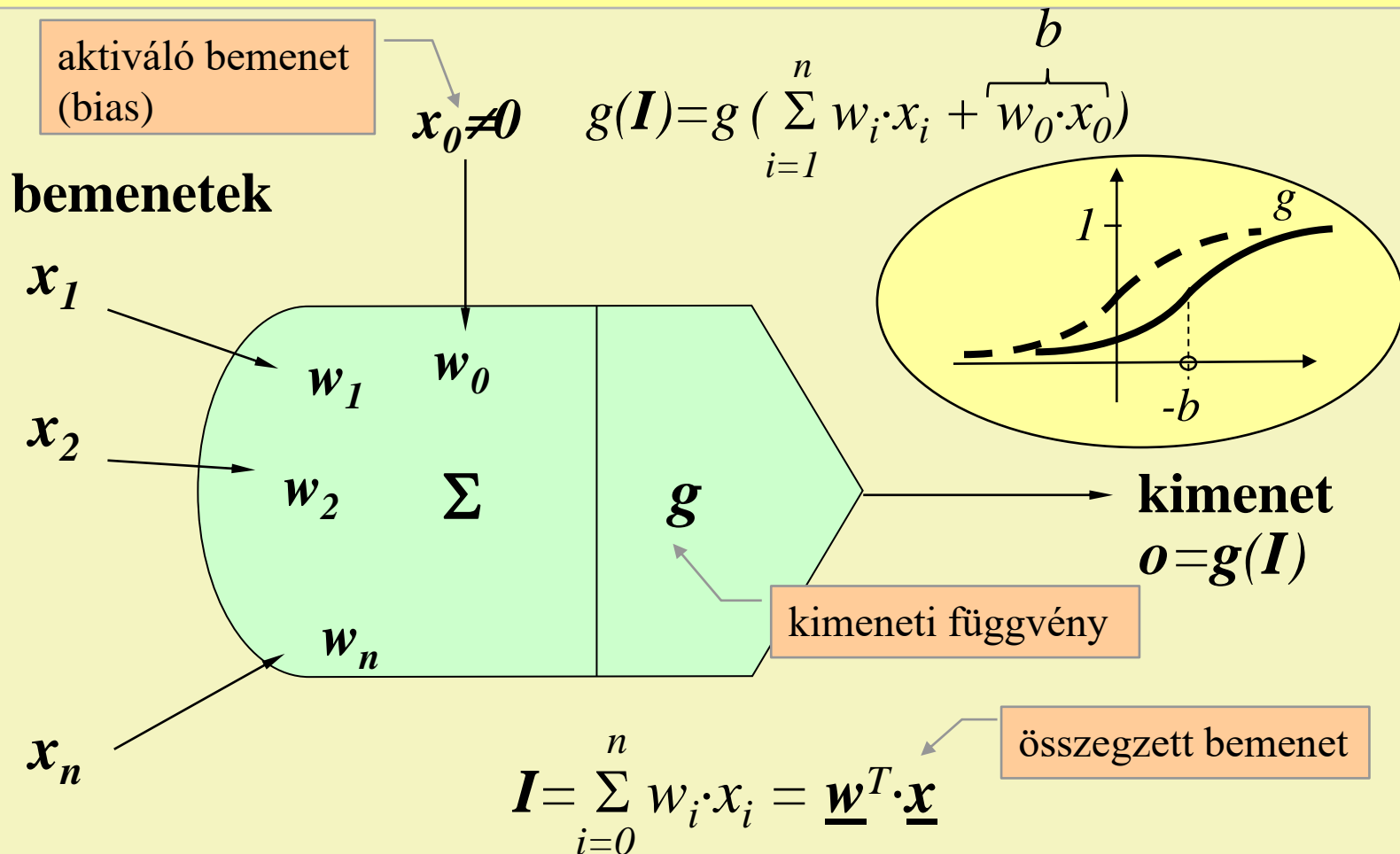
## □ Hálózati topológia

- sok mesterséges neuron egymáshoz kapcsolva, ahol egyik neuron kimenete egy másik neuron bemenete lesz
- bizonyos neuronok a bemenetüket a hálózaton kívülről kapják, mások kimeneteit pedig hálózat kimenetének tekintjük

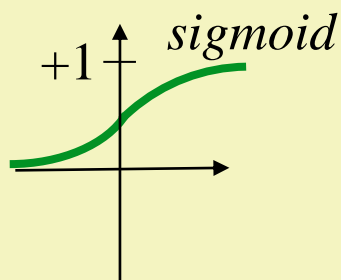
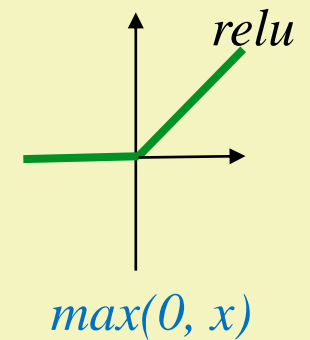
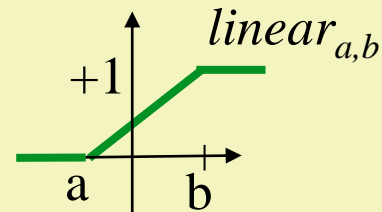
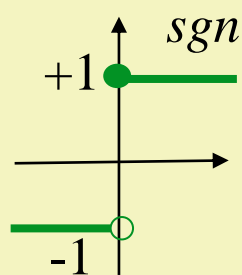
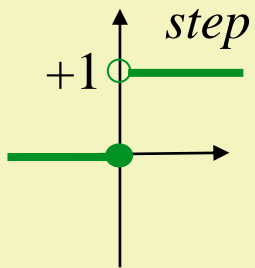
## □ Tanulási szabály

- egy neuron számítási képletét meghatározó eljárás, amely lehet egy tanító példák alapján működő algoritmus is.

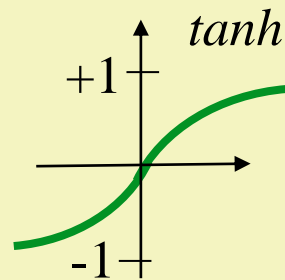
# Általánosított perceptron



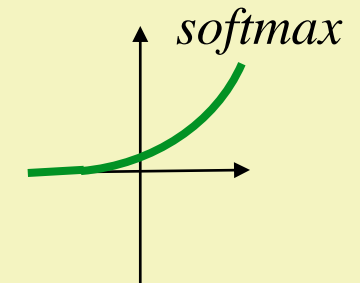
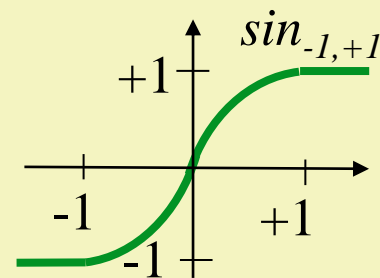
# Kimeneti függvények



$$\frac{1}{1+e^{-x}}$$



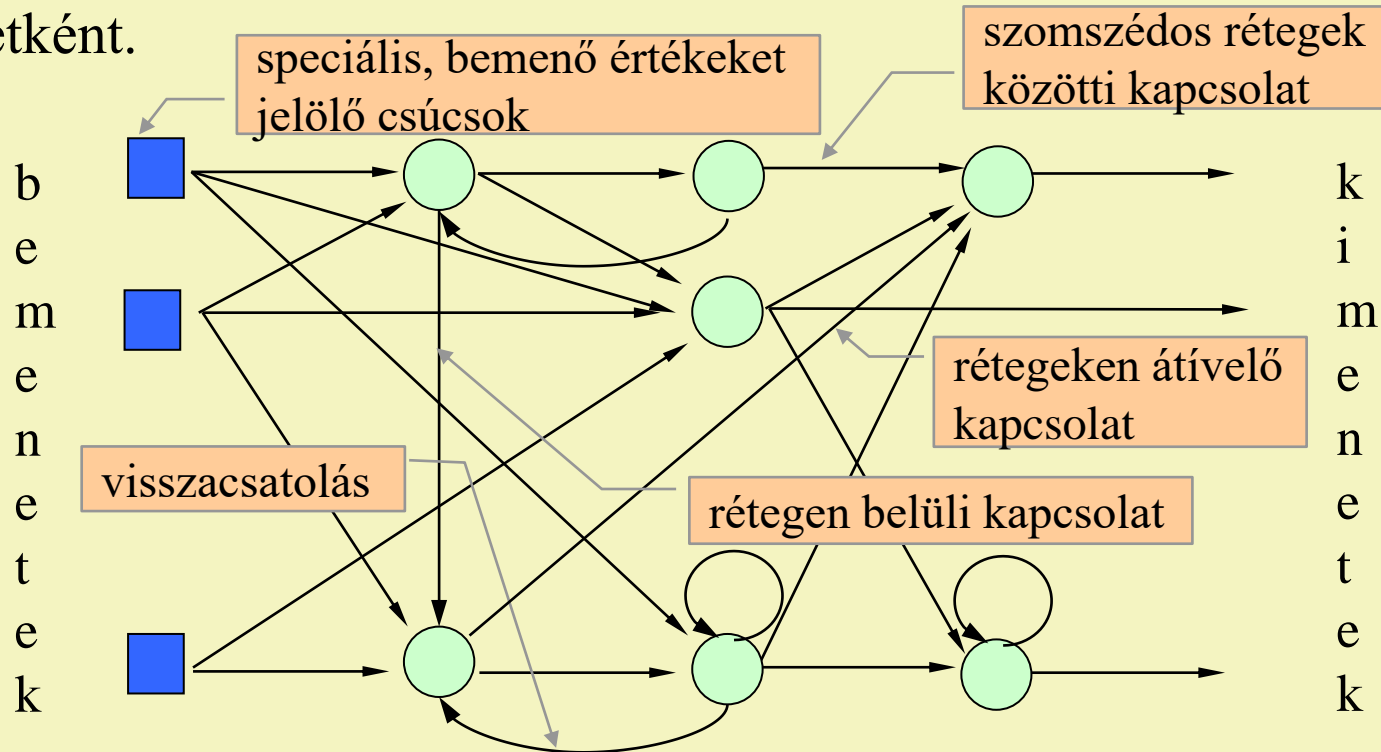
$$\frac{1-e^{-x}}{1+e^{-x}}$$



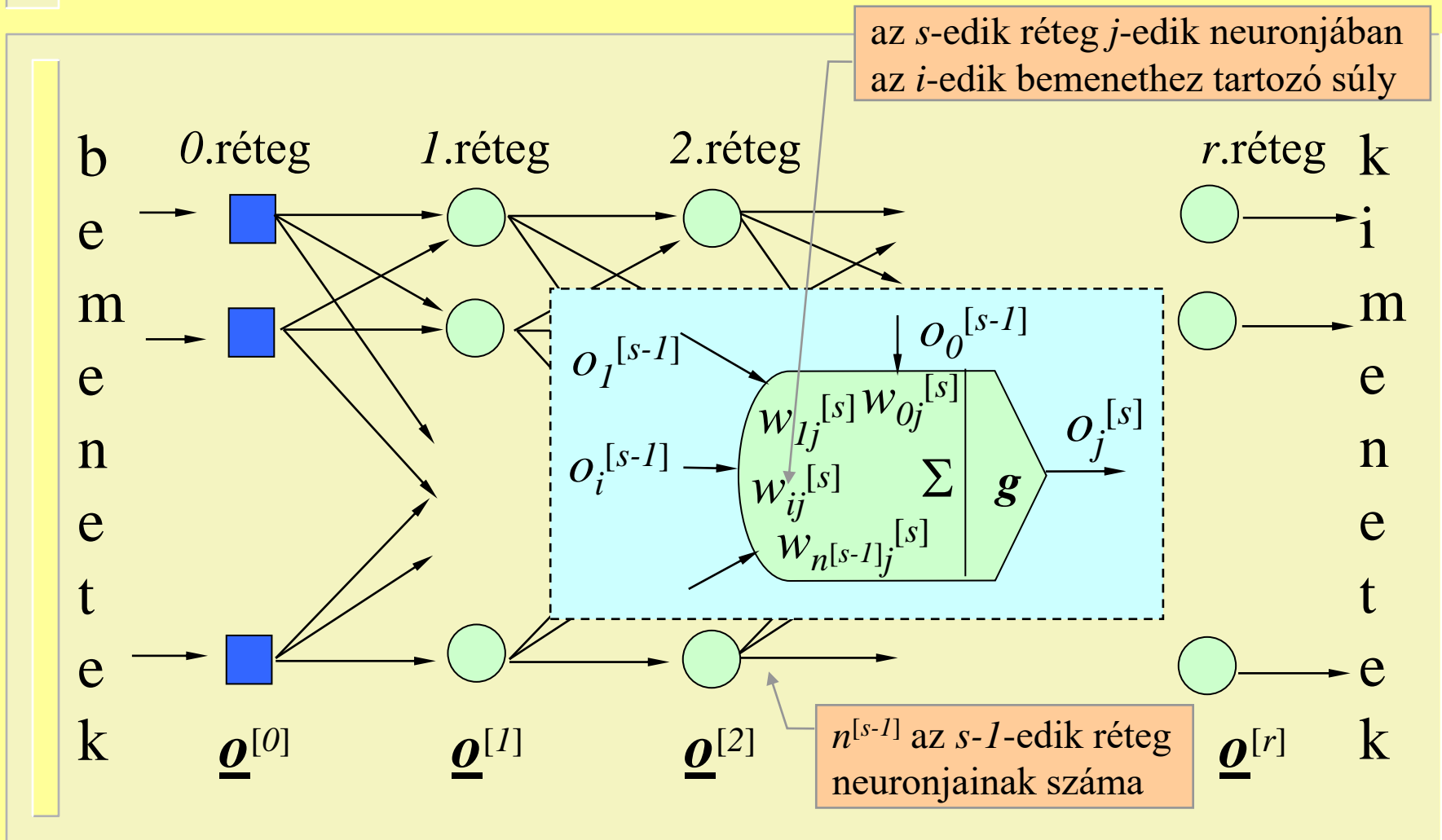
$$\frac{e^{x_j}}{\sum e^{x_k}}$$

# Hálózati topológia

Írányított gráf, amelynek csúcsai mesterséges neuronok, amelyek rétegekbe csoportosíthatók. Az irányított élek az adatáramlás irányát jelölik:  $a \rightarrow b$  : az  $a$  neuron kimeneti értékét kapja meg a  $b$  neuron bemenetként.

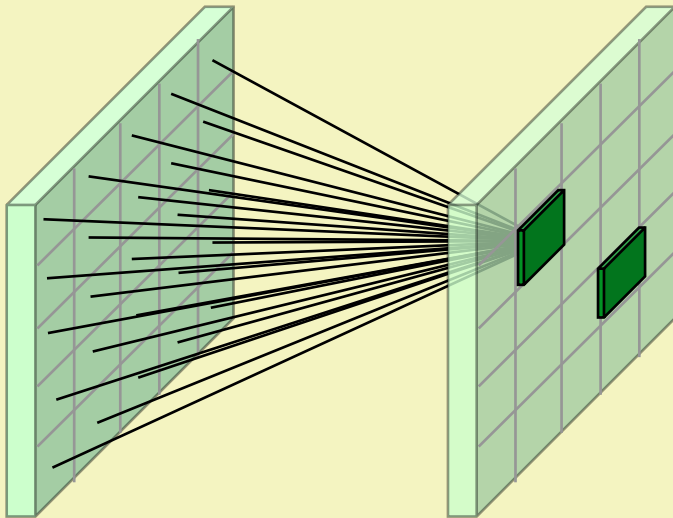


# Többrétegű előrecsatolt hálózat (feed forward MLP)



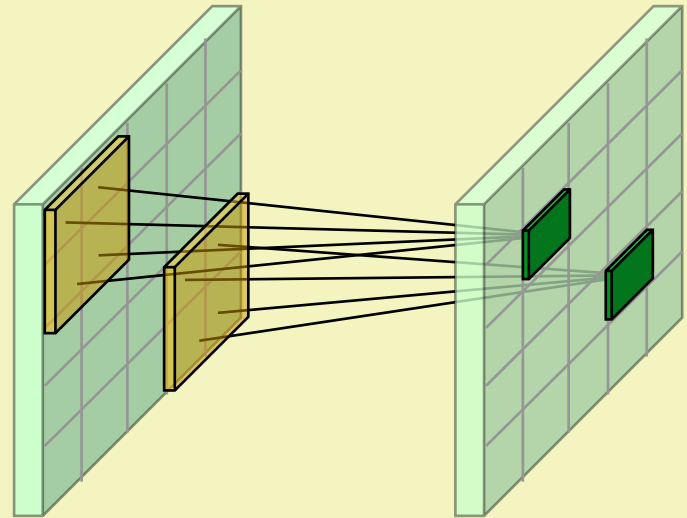
# *Konvolúciós neuron hálózat*

Teljesen összekötött (sűrű)  
fully connected neural net



Péld.:  $1000 \times 1000$ -es első réteg esetén  
a második réteg egy neuronjában  
 $10^6$  darab súlyt kell tárolni.

Lokálisan összekötött  
convolutional neural net

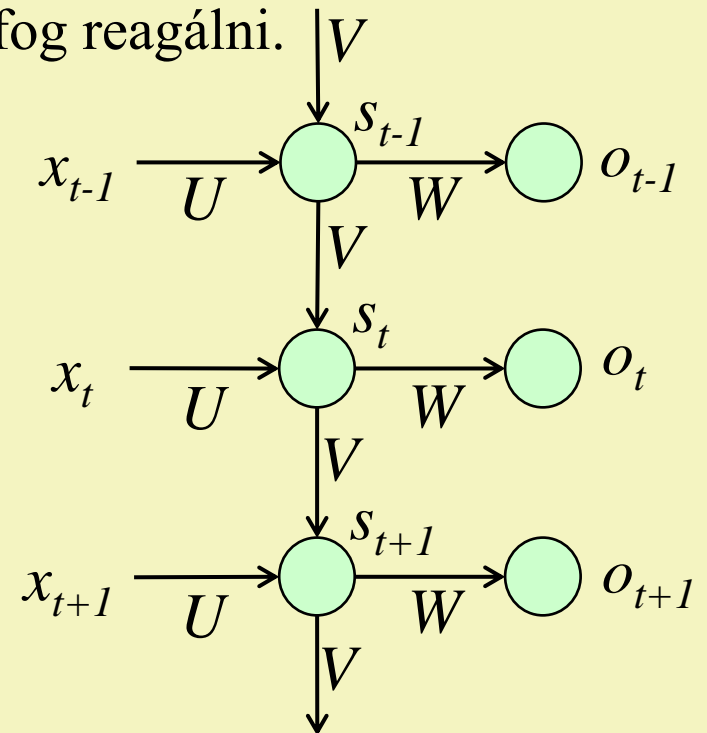
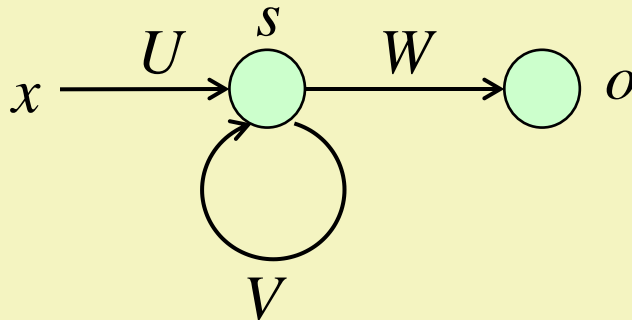


Péld.:  $1000 \times 1000$ -es első réteg esetén  
egy  $2 \times 2$ -es szűrőt használva  
a második réteg egy neuronjában  
csak 4 súlyt kell tárolni.



# Rekurrens neurális hálózat

- Az input és/vagy az output változó hosszúságú sorozat.
- A számítási képlet azt utánozza, mintha a neuron emlékezne a megelőző inputokra, mivel egy új inputra a megelőző inputokra kiszámolt outputot is felhasználva fog reagálni.

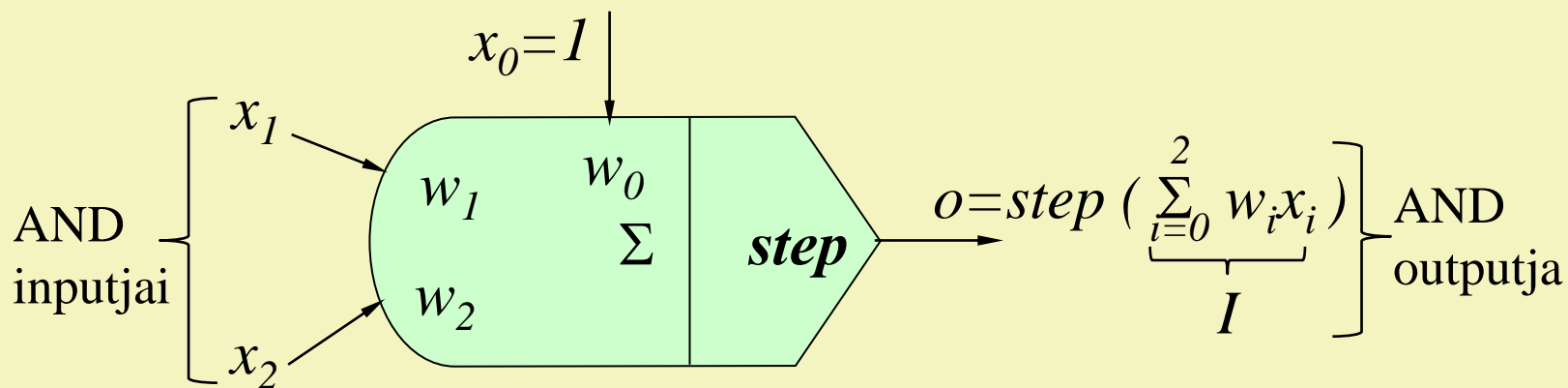


# *Általánosított perceptron tanulása*

- Egy neuron számítási képletét a neuron  $w$  súlyai határozzák meg.
- A súlyok implicit módon a hálózat topológiáját is kijelölik, hiszen neuronhoz vezető nulla értékű súllyal ellátott él lényegében az él figyelmen kívül hagyását (törlését) jelenti.
- Tanulás során a súlyokat fokozatosan módosítjuk ( $w := w + \Delta w$ ).
- A  $\Delta w$  a neuron **bemeneti értékeitől** és a neuron által **kiszámított kimeneti értéktől** függ.
  - **Felügyelt tanulás** esetén felhasználjuk az **elvárt kimenetet**.
  - **Felügyelet nélküli tanulás** esetén az elvárt kimenetre nincs szükség.

## *Példa felügyelt tanulásra*

Tanítsuk meg egy egyszerű számoló egységnek (egyetlen mesterséges neuronnak) a logikai AND művelet működését!



$x_i$  a neuron  $i$ -dik bemenete ( $x_i \in \{0, 1\}$ ,  $x_0 = 1$ )

$w_i$  a neuron  $i$ -dik bemenetének súlya ( $w_0, w_1, w_2 \in \mathbb{R}$ )

$I$  a neuron összegzett bemenete

$o$  a neuron számított kimenete ( $o \in \{0, 1\}$ )

Delta szabály

hiba ( $e$ )

$y \sim$  várt

$o \sim$  számított

$\eta = 0.1$

felügyelt tanulási szabály:  $\Delta w_i = \eta \cdot x_i \cdot (y - o)$

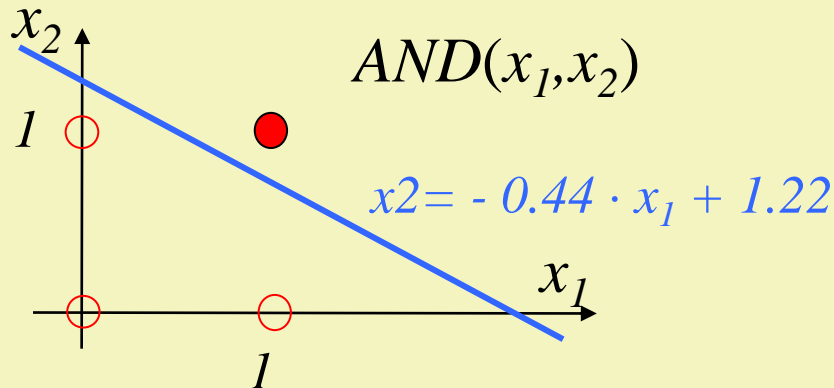
	$x_1$	$x_2$	$y$	$w_0$	$w_1$	$w_2$	$I$	$o$	$e$
				<b>0.08</b>	<b>0.08</b>	<b>0.08</b>			
1.	1	0	0	0.08 +	0.08 +	0	= 0.160	1	-1
				<b>-0.02</b>	<b>-0.02</b>	<b>0.08</b>			
2.	0	1	0	-0.02 +	0	+ 0.08	= 0.06	1	-1
				<b>-0.12</b>	<b>-0.02</b>	<b>-0.02</b>			
3.	1	1	1	-0.12 +	-0.02 +	-0.02	= -0.16	0	1
				<b>-0.02</b>	<b>0.08</b>	<b>0.08</b>			
4.	1	0	0	-0.02 +	0.08 +	0	= 0.06	1	-1
				<b>-0.12</b>	<b>-0.02</b>	<b>0.08</b>			
5.	0	1	0	-0.12 +	0	+ 0.08	= -0.04	0	0
				<b>-0.12</b>	<b>-0.02</b>	<b>0.08</b>			
6.	1	1	1	-0.12 +	-0.02 +	0.08	= -0.06	0	1
...									
13.				<b>-0.22</b>	<b>0.08</b>	<b>0.18</b>			
14.	1	0	0	-0.22 +	0.08 +	0	= -0.14	0	0
15.	0	1	0	-0.22 +	0	+ 0.18	= -0.04	0	0
16.	1	1	1	-0.22 +	0.08 +	0.18	= 0.04	1	0
17.	<b>0</b>	<b>0</b>	<b>0</b>	-0.22 +	0	+ 0	= -0.22	<b>0</b>	0

első epoch

második epoch

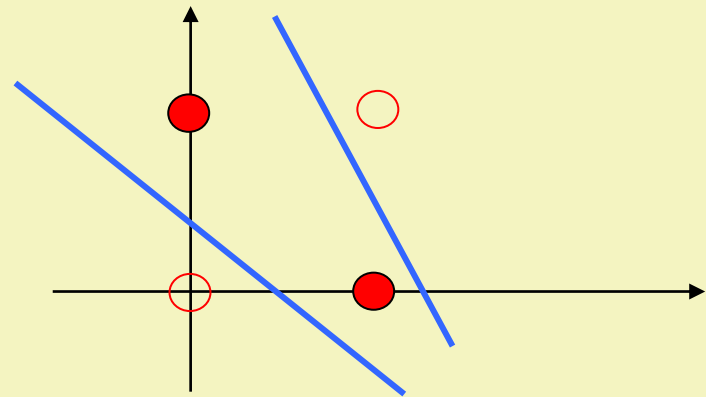
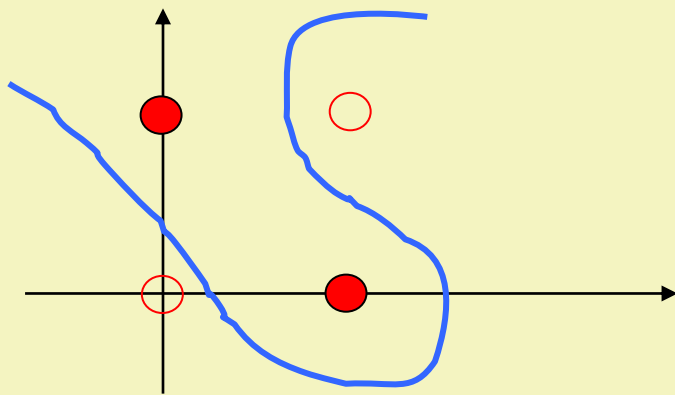
# Mit tanultunk meg?

- A neuron működése azonos az  $AND(x_1, x_2)$  működésével, mert az összegzett bemenet előjele csak az  $(1, 1)$  bementre pozitív:
  - ha  $I(x_1, x_2) \leq 0$  akkor  $step(I(x_1, x_2)) = 0$
  - ha  $I(x_1, x_2) > 0$  akkor  $step(I(x_1, x_2)) = 1$
- A  $w$  együtthatók megtanulásával azt az  $I(x_1, x_2) = w_0 + w_1 \cdot x_1 + w_2 \cdot x_2$  egyenest kaptuk meg a lehetséges bemenet-párok síkján, amely elszeparálja (osztályozza) a bemenet-párokat: egy oldalra (félsíkra) kerülnek az azonos eredményű bemenetek.

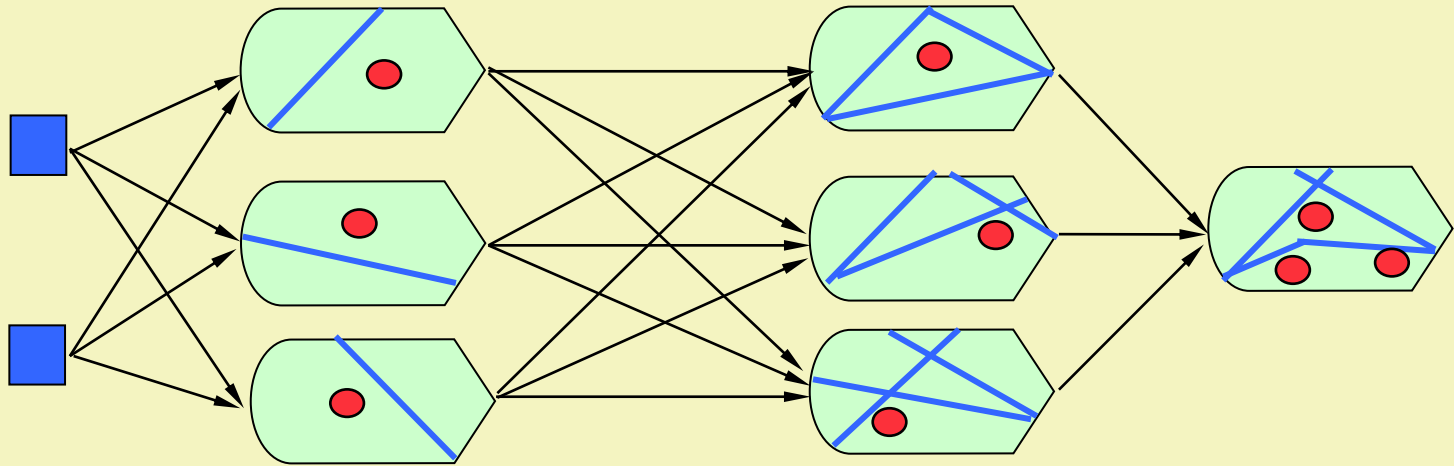


# Lineáris szeparálhatóság

- Egyetlen perceptronnal olyan bonyolultságú feladatot vagyunk képesek megoldani, ahol az eredményük alapján a bemeneteket egy hipersík választja szét, azaz **lineárisan szeparálhatók**.
- De nem lehet például a XOR műveletet egyetlen perceptronnal megvalósítani, mert ez a feladat lineárisan nem szeparálható.



## Rétegek „tudása”



- ❑ Több réteggel összetettebb problémák is megoldhatók, de mivel a *step* függvény (**nem deriválható**) egyszerű perceptronokkal eddig nem sikerült megfelelő tanuló algoritmust találni ehhez.
- ❑ De ha lecseréljük a kimeneti függvényt (pl. szigmoidra), akkor már létrehozható olyan algoritmus, amellyel a háló tanítható (pl. error backpropagation).

# Homogén MLP háló számítási modellje

- A teljes háló számítási modellje:

$$f(\Theta, \underline{x}) = g(\underline{w}^{[r]} \cdot \dots \cdot g(\underline{w}^{[s]} \cdot \dots \cdot g(\underline{w}^{[2]} \cdot g(\underline{w}^{[1]} \cdot \underline{x})) \dots) \dots)$$

deriválható  
aktivizációs  
függvény

a  $\Theta$  paraméter a  $(w_{ij}^{[s]})$  súlyok összessége

- Egy réteg számítási modellje

$$\underline{o}^{[s]} = g(\underline{w}_j^{[s]} \cdot \underline{o}^{[s-1]}) = g\left(\sum_{i=0}^{n^{[s-1]}} w_{ij}^{[s]} \cdot o_i^{[s-1]}\right)$$



# Hiba visszaterjesztés módszere (error backpropagation)

A modell hibafüggvénye:  $L(\Theta) = \frac{1}{2} \sum_{j=1}^{n^{[r]}} (y_j - o_j^{[r]})^2$   $f(\Theta, \underline{x})$

Az  $L(\Theta)$  egy olyan több változós függvény, amely a  $(w_{ij}^{[s]})$  súlyoktól függ. A tanulás során ennek a függvénynek keressük a minimum helyét **gradiens módszerrel**, azaz lépésről lépésre módosítjuk a súlyokat megfelelő irányban kis mértékben.

$$w_{ij}^{[s]} := w_{ij}^{[s]} - \Delta w_{ij}^{[s]} \quad I_j^{[s]} = \sum_{i=0}^n w_{ij}^{[s]} \cdot o_i^{[s-1]}$$

$$\Delta w_{ij}^{[s]} = \eta \cdot \frac{\partial L}{\partial w_{ij}^{[s]}} = \eta \cdot \frac{\partial L}{\partial I_j^{[s]}} \cdot \frac{\partial I_j^{[s]}}{\partial w_{ij}^{[s]}} = \eta \cdot \left( \frac{\partial L}{\partial I_j^{[s]}} \right) \cdot o_i^{[s-1]}$$

a számítási hibának az  $s$ -edik réteg  $j$ -edik neuronjára jutó hányada

$e_j^{[s]}$

# Egy neuronra visszavetített számítási hiba

Ha  $s=r$  akkor

$$e_j^{[r]} = o_j^{[r]}(1 - o_j^{[r]})(t_j - o_j^{[r]})$$

$$\Delta w_{ij}^{[r]} = \eta e_j^{[r]} o_i^{[r-1]}$$

$$e_j^{[s]} = - \frac{\partial E}{\partial I_j^{[s]}}$$

ha  $g$  a szigmoid függvény

Ha  $s < r$  akkor

$$e_j^{[s]} = o_j^{[s]}(1 - o_j^{[s]}) \sum_{k=1}^n e_k^{[s+1]} w_{jk}^{[s+1]}$$

$$\Delta w_{ij}^{[s]} = \eta e_j^{[s]} o_i^{[s-1]}$$

ha  $g$  a szigmoid függvény

rekurzív szabályt kapjuk a súlyok módosítására.

# Backpropagation tanuló algoritmus

1. Az  $\underline{x}$  bemeneti vektorból indulva rétegenként számoljuk a neuronok kimenetét:  $o_j^{[s]}$ , a kimeneti réteg kimeneteit  $o_j^{[r]}$  is.

2. A kimeneti réteg minden neuronjára kiszámoljuk a lokális hibát:

$$e_j^{[r]} := o_j^{[r]} \cdot (1 - o_j^{[r]}) \cdot (t_j - o_j^{[r]})$$

ha  $g$  a szigmoid függvény

3. Rétegenként hátulról előre haladva számoljuk a belső neuronok hibáit:

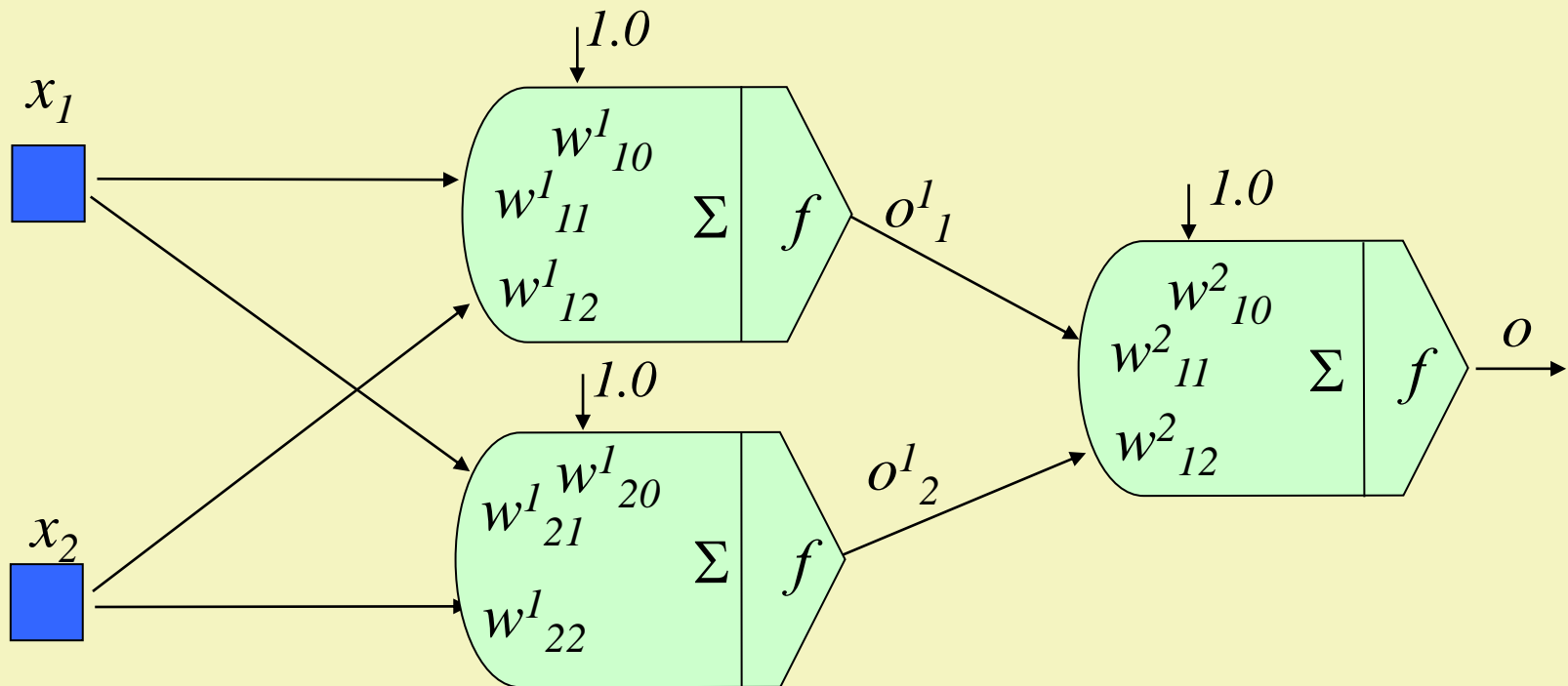
$$e_j^{[s]} := o_j^{[s]} \cdot (1 - o_j^{[s]}) \cdot \left( \sum_{k=1}^n e_k^{[s+1]} \cdot w_{jk}^{[s+1]} \right)$$

ha  $g$  a szigmoid függvény

4. Végül módosítjuk a hálózat súlyait:  $w_{ij}^{[s]} := w_{ij}^{[s]} + \Delta w_{ij}^{[s]}$  ahol a súlytényező-változás:  $\Delta w_{ij}^{[s]} := \eta \cdot e_j^{[s]} \cdot o_i^{[s-1]}$

# XOR művelet példája

Beállítások:  $x_1, x_2 \in \{0, 1\}$   $f(x) = \text{sigmoid}(x)$   $o^s_i \in (0, 1)$   
 $w^s_{ij} = \text{rand}(-0.1, 0.1)$   $o^s_0 = 1.0$   $\eta = 1.0$



# Rétegenként eltérő aktivizációs függvény

- Egy inhomogén MLP háló számítási modellje:  $f: P \times X \rightarrow Y$

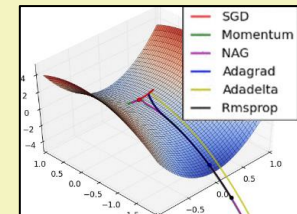
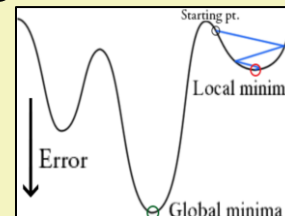
$$f(\Theta, \underline{x}) = g_r(\underline{w}^{[r]}, g_{r-1}(\dots g_2(\underline{w}^{[2]}, g_1(\underline{w}^{[1]}, \underline{x})) \dots))$$

- $\Theta = \{w^{[1]}, \dots, w^{[r]}\}$  azaz a paraméterek a súlyok
- $g_s: P \times X^{s-1} \rightarrow X^s$   $s$ -edik réteg kimeneti függvénye ( $X = X^0$ ,  $Y = X^r$ )

- A gradiens elméleti kiszámítása nehéz, ezért erre numerikus módszereket használnak. (Keras, TensorFlow)

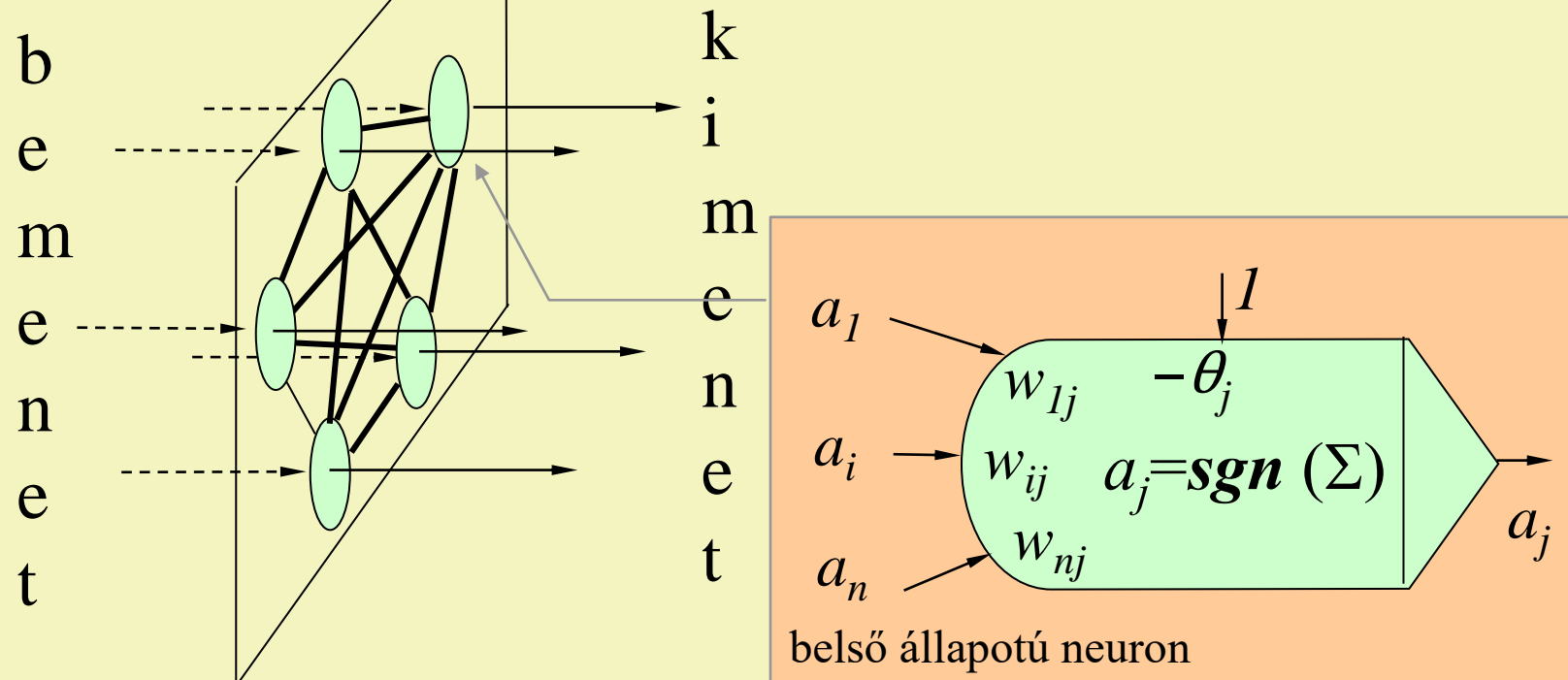
- További problémák:

- A tanító minták kiválasztása nehéz (homogén minták, overfitting).
- A hiper-paraméterek ( $N$ ,  $\eta$ ) megtanulásának kérdése.
- A lokális minimum-, illetve a nyeregpontok problémája.



# Hopfield modell

- Aszinkron működésű Hopfield topológia: egyrétegű, teljes összekötöttségű hálózat  $n$  darab  $+1$  v.  $-1$  állapotú neuronnal.



# Hopfield modell működése

- **Kezdetben** a neuronok kívülről kapják értékül az állapotaikat. A neuronok által felvett állapot-együttest hívjuk a háló egy konfigurációjának.  
(Az összes konfiguráció alkotja a **konfigurációs teret**.)

- **Ezután** elkezdik újra számolni az állapotukat, **aszinkron** módon többször is, folyamatosan változtatva a hálózat konfigurációját:

$$a_j^{új} = \text{sgn}\left(\sum_{i=1}^n w_{ij} a_i - \theta_j\right)$$

( $a_i$  az  $i$ -dik neuron állapota,  $w_{ij}$   $i$ -dik neuronból a  $j$ -dik neuronba vezető kapcsolat súlya, a  $\theta_j$  a  $j$ -dik neuron küszöbértéke)

- **Végül**, amikor a hálózat egy **stabil konfigurációba** jut (azaz az állapotok újraszámolásuk ellenére sem változnak tovább), akkor a neuronok állapotait **a háló kimeneteinek tekintjük**.

# *Hopfield hálózat felhasználása*

- ❑ A modell eredetileg **asszociatív memória** megvalósítására készült. Ilyenkor a hálózat egy-egy stabil konfigurációja ad meg egy eltárolt mintát (pl. egy kép pixelpontjait).
- ❑ Amikor a minta zajos, torzított vagy hiányos változatát adjuk meg bemenetként a háló neuronjainak (kibillentve őket egy stabil konfigurációból), akkor a hálózat addig változtatgatja az állapotát a neuronjainak, amíg újra stabil konfigurációba nem kerül, azaz fel nem idézi az eredeti mintát (több minta esetén a leghasonlóbbat).
- ❑ A működéssel kapcsolatban megvizsgálandó kérdések:
  - Hogyan lehet (felügyelet nélkül) betanítani arra a hálózatot, hogy a tárolt minták a hálózat stabil konfigurációi legyenek?
  - Konvergál-e a hálózat a stabil konfigurációk valamelyikéhez?



# Hopfield hálózat egyetlen minta tárolására

- Kell, hogy a tárolandó  $\underline{p} \in \{+1, -1\}^n$  minta egy stabil állapot-konfigurációja legyen a hálónak, azaz  $p_j = \text{sgn}(\sum_i w_{ij} \cdot p_i)$  – feltéve, hogy  $\theta_j = 0$ . Ez akkor teljesül, ha  $w_{ij} \cong p_i \cdot p_j$ .
- Az ilyen minta jelentős vonzáskörzettel bír: ekkor ugyanis minden olyan  $\underline{a}$  konfigurációból, amely komponenseinek több, mint a fele azonos a mintáéval (tehát  $\sum_i p_i \cdot a_i > 0$ ), a háló véges lépésben a mintához konvergál.

- Ez abból következik, hogy amikor egy neuron újra számolja az állapotát, akkor az azonnal a minta megfelelő komponensével válik azonossá:

$$a_j^{\text{új}} = \text{sgn}(\sum_i w_{ij} \cdot a_i) = \text{sgn}(\sum_i p_i \cdot p_j \cdot a_i) = \text{sgn}(p_j \cdot \sum_i p_i \cdot a_i) = p_j.$$

# Hopfield hálózat több minta tárolására

- Legyen  $d$  darab tárolandó mintánk:  $\underline{p}^k \in \{+1, -1\}^n$  ( $k=1 \dots d$ ).
- Válasszuk a súlyoknak a  $w_{ij} = 1/n \cdot \sum_k p_i^k \cdot p_j^k$  szuperpozíciókat. Ugyanez inkrementális tanulási szabályként is felírható:  
 $\Delta w_{ij} = 1/n \cdot p_i^k \cdot p_j^k$ . Hebb szabály
- Véletlenszerűen választott (ortogonális) minták esetén annak, hogy a minták stabil konfigurációk legyenek, akkor a legnagyobb a valószínűsége, ha  $d \leq n/\log n$ .
- Konvergencia: Vegyük a  $-1/2 \sum_{i,j} w_{ij} \cdot a_i \cdot a_j - \sum_i \theta_i \cdot a_i$  hibafüggvényt. Belátható, hogy ha a súlymátrix szimmetrikus, és a diagonális elemei nem-negatívak, akkor egy konfiguráció-váltás során a függvény értékének csökkennie kell. Mivel a konfigurációk száma véges, így a hálózat véges lépésben stabil konfigurációba fog jutni.