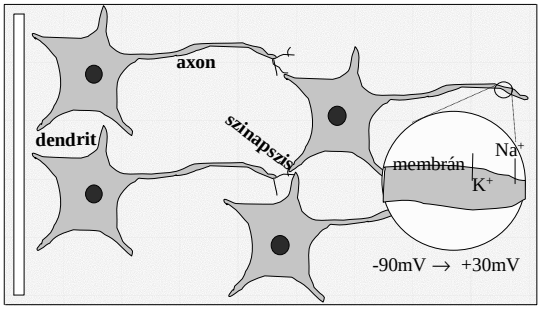
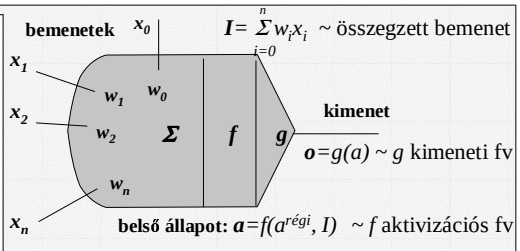
Mesterséges neuron hálók



1. Mesterséges neuronháló részei

- Mesterséges neuron
  - bemenő értékekből kimenő értéket számoló egység, amelynek számítási képlete változtatható
- Hálózati topológia
  - több mesterséges neuron egymáshoz kapcsolásának módja
- Tanulási szabály
  - egy neuron számítási képletének módosítása

1.1. Mesterséges neuron



$I = \sum_{i=0}^{n} w_i x_i$ ~ összegzett bemenet

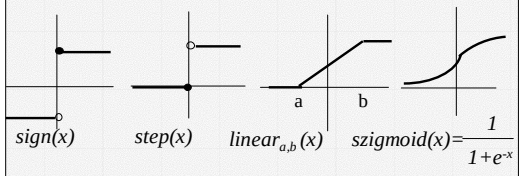
$o = g(a)$ ~ g kimeneti fv

belső állapot: $a = f(a^{régi}, I)$ ~ f aktivációs fv

- A g gyakran identitás, az f pedig csak az I-től függ, azaz a belső állapotnak nincs jelentősége.

Megjegyzés

Az $f: \Re \rightarrow \Re$ aktivizációs függvény

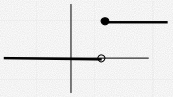


$sign(x)$ $step(x)$ $linear_{a,b}(x)$ $szigmoid(x) = \frac{1}{1+e^{-x}}$

Megjegyzés

- Az x_0 (stimuláló) bemenetnek speciális szerepe van: meghatározza a sejt ingerküszöb értékét.
- Például a Θ küszöbértékű *step* aktivizációs függvény esetén

$step_\Theta(x) = step(x - \Theta)$



Ha a küszöbérték: $\Theta = - w_0 x_0$ akkor

$$step_\Theta(\sum_{i=1}^{n} w_i x_i) = step(\sum_{i=1}^{n} w_i x_i - \Theta) = step(\sum_{i=0}^{n} w_i x_i)$$

1.2. Hálózati topológia

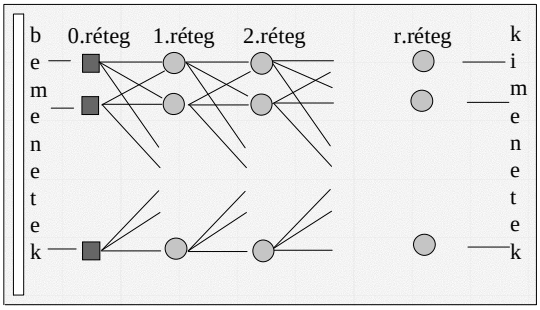
- A mesterséges neuronháló egy olyan irányított gráf, amelynek csúcsai vagy a háló bemeneti értékeit jelenítik meg, vagy egy-egy (általában azonos konfigurációjú) mesterséges neuront szimbolizálnak.

Kapcsolatok osztályozási szempontjai

- rétegelés szerint: Ha a neuronokat rétegekre osztjuk (egy rétegbe tartozó neuronok számításai párhozamosan végezhetők), akkor beszélhetünk rétegen belüli ill. rétegek közötti kapcsolatokról
- fajta szerint: erősítő illetve gátló
- kitöltöttség szerint: teljes, egy-egy, vagy véletlen
- irányítás szerint: előre csatolt vagy visszacsatolt

7

Előrecsatolt, rétegzett topológia: perceptron



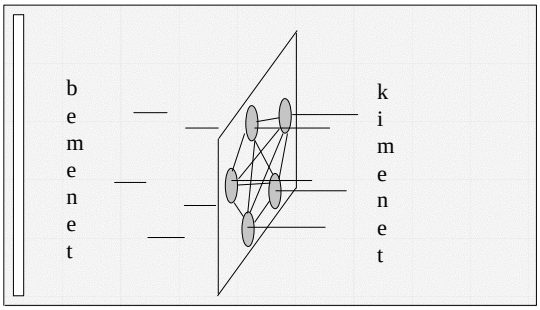
8

Perceptron topológia működése

- Számítási modell, amely bemenő értékekre kimenő értékeket számol:
  - A *0*-adik réteg az x_i bemeneti értékeket, mint o_i^0 - t, továbbítja
  - Az *s*-edik réteg *j*-edik neuronjának *i*-edik bemenete = az *s-1*-edik réteg *i*-edik neuronjának kimenete: o_i^{s-1}
  - Az *s*-edik réteg *j*-edik neuronjának *i*-edik bemenetéhez tartozó súly: w_{ij}^s
  - Az *s*-edik réteg mindegyik neuronja kiszámolja a saját kimeneti értékét: o_j^s

9

Hopfield topológia



10

Hopfield topológia működése

- Célja egy nyugalmi helyzet előállítása
  - Minden neuron kezdő állapota egy-egy bemeneti érték
  - Egy neuron mindaddig újra számolja a többi neuron kimeneti értéke alapján a belső állapotát, ameddig az eltér a korábbi állapotától.
  - A stabil helyzetben kialakult állapotok lesznek a kimeneti értékek.

11

1.3. Tanulás

- A tanulás a hálózat paramétereinek tanító példák alapján történő beállítása.
  - Paraméterek: súlyok (amely kihat egy-egy neuron bemeneteinek számára, közvetve a topológiára), topológia, aktivizációs függvény
- Leggyakrabban a súlyokat tanuljuk meg:
  - Mintákat, azaz lehetséges bemeneteket mutatunk a mesterséges neuronhálónak, amely minden mintára kiszámítja a kimenetet, majd ez alapján módosítja a súlyokat: $w_i = w_i + \Delta w_i$

12

Felügyelt tanulás | **Felügyelet nélküli tanulás**

Delta szabály:

$\underline{x}$ a neuron bemenetei
t a várt kimenet
o a számított kimenet

akkor
$\Delta w_i = \eta * x_i * (t-o)$

η a tanulási együttható

Hebb szabály:

ha a neuron kimenete (o) egyidejűleg aktív az i-edik (x_i) bemenettel (megelőző szomszéd kimenete), akkor erősíteni kell köztük a kapcsolatot

$\Delta w_i = \eta * x_i * o$

13

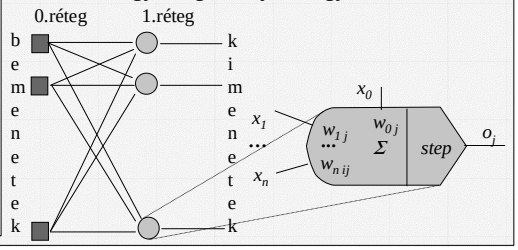
2. Nevezetes hálók

- Perceptron modell
- Backpropagation modell
- …

14

2.1. Perceptron modell

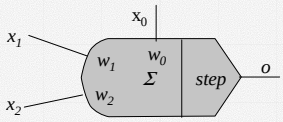
- Step aktivizációs függvényű, belső állapot nélküli neuronok egy rétegű hálója, felügyelt tanulással.



15

Példa

AND művelet



Beállítások:

$x_1, x_2, o \in \{0,1\}$
$x_0 = 1$
$w_0, w_1, w_2 \in \Re$
$f(x) = step(x)$

16

Tanulás nélkül: közvetlen beállítás

$AND(x_1, x_2) = step(w_0 + w_1 x_1 + w_2 x_2)$

$AND(0,0)=0$	ha	$w_0 < 0$
$AND(1,0)=0$	ha	$w_0 + w_1 < 0$
$AND(0,1)=0$	ha	$w_0 + w_2 < 0$
$AND(1,1)=1$	ha	$w_0 + w_1 + w_2 \geq 0$

Például:

$w_0 = -3$
$w_1 = 2$
$w_2 = 2$

17

Tanulás

- o - a számított kimenet t - a várt kimenet
- ha $t-o=1$ (azaz $t=1$ és $o=0$) akkor az o-t növelni kell a kiszámításában aktív bemenetek súlyainak növelésével
- ha $t-o=-1$ (azaz $t=0$ és $o=1$) akkor az o-t csökkenteni kell a kiszámításában aktív bemenetek súlyainak csökkentésével
- ha $t-o=0$ (t és o azonos) akkor semmit nem kell tenni
- Ez egy delta szabály: $\Delta w_i = \eta * x_i * (t-o)$

18

$\eta = 0.1$

Tanítás és alkalmazás

	x_1	x_2	w_0	w_1	w_2	I	o	t	e
0.			0.08	0.08	0.08				
1.	1	0	0.08	0.08	0.08	0.160	1	0	-1
2.	0	1	-0.02	-0.02	0.08	0.06	1	0	-1
3.	1	1	-0.12	-0.02	-0.02	-0.16	0	1	1
4.	1	0	-0.02	0.08	0.08	0.06	1	0	-1
5.	0	1	-0.12	-0.02	0.08	-0.04	0	0	0
6.	1	1	-0.12	-0.02	0.08	-0.06	0	1	1
7.	1	0	-0.02	-0.08	0.18	0.06	1	0	-1
...									
14.			**-0.22**	**0.08**	**0.18**				
14.	1	0	-0.22	0.08	0.18	-0.14	0	0	0
15.	0	1	-0.22	0.08	0.18	-0.04	0	0	0
16.	1	1	-0.22	0.08	0.18	0.04	1	1	0
17.	**0**	**0**	-0.22	0.08	0.18	-0.22	**0**	0	0

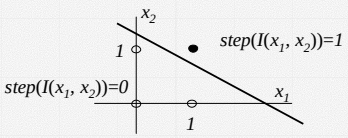
Megjegyzés

- Egy neuron súlyai annak a hipersíknak egyenletét határozzák meg, amelyik a bemeneti érék n-esek terét két részre osztja. Az egyik térfélbe tartozó bemenetekhez *1*-et, a másikhoz *0*-t rendel.
- Ezért a perceptron csak lineárisan szeparálható osztályozási problémákat képes megoldani, de azokat biztosan megoldja.

20

Példa

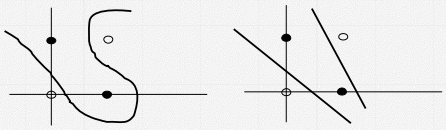
- Az AND műveletre betanított perceptron bemenetei egy síkon ábrázolhatóak.
- Az $I(x_1,x_2) = w_0 + w_1x_1 + w_2x_2 = 0$ egyenlet egy egyenest határoz meg:
  - $x_2 = -(w_1/w_2)x_1 - (w_0/w_2)$ azaz $x_2 = -0.44\, x_1 + 1.22$



21

Ellenpélda

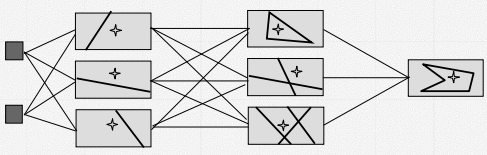
- A XOR művelethez nem rajzolható be szeparáló egyenes, tehát nem tanítható meg egy perceptronnak.



22

Megjegyzés

- A két bemenetű egyrétegű perceptron modell csak félsíkokat képes felismerni, a kétrétegű perceptron modell már konvex poliédereket is, a három rétegű háló pedig tetszőleges poliédereket.



23

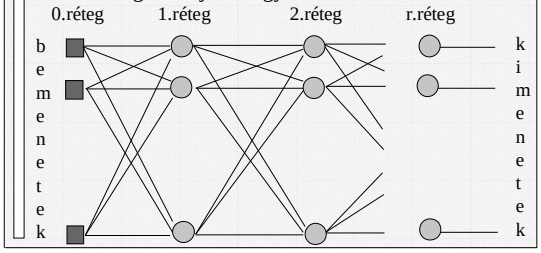
Megjegyzés

- A perceptron modellnek természetes általánosítása a többrétegű perceptron háló. (Ilyenkor elképzelhető, hogy a közbülső rétegek neuronjainak aktivizációs függvénye lineáris.)
- A többrétegű perceptron modellekhez azonban nem találtak tanuló eljárást, ezért a súlyai csak közvetlen módon állíthatók be.

24

2.1. Backpropagation modell

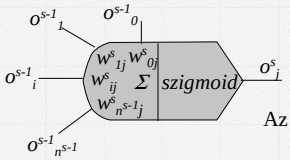
- Szigmoid függvényű, belső állapot nélküli neuronok több rétegű hálója, felügyelt tanulással.



25

Neuron

Az *s*-edik réteg *j*-edik neuronja:



Az első (*s=1*) rétegnél:
$o^{s-1}{}_i = x_i$

26

Energia függvény

A háló energiafüggvénye egyetlen tanító minta esetén:

$$E = \frac{1}{2}\sum_{j=1}^{n^r}(t_j - o^r{}_j)^2$$

Az *E* tekinthető úgy is, mint háló $\{w^s{}_{ij}\}$ súlyainak $E(w^1{}_{11}, \ldots, w^r{}_{n^{r-1}n^r})$ függvénye, ezért értékének minimalizálásához a háló súlyait a gradiens módszer alapján kell megváltoztatni:

$$w^s{}_{ij} \leftarrow w^s{}_{ij} - \eta \frac{\partial E}{\partial w^s{}_{ij}}$$

27

Tanulási szabály

Az *E* többek között függ az *s*-edik réteg *j*-edik neuronjának összegzett bemenetétől ($I^s{}_j$) is, ami viszont tekinthető az *i*-edik súly ($w^s{}_{ij}$) függvényének:

$$-\eta \frac{\partial E}{\partial w^s{}_{ij}} = -\eta \frac{\partial E}{\partial I^s{}_j} \frac{\partial I^s{}_j}{\partial w^s{}_{ij}}$$

Jelöljük $e^s{}_j$ -vel a $-\frac{\partial E}{\partial I^s{}_j}$ -t, továbbá

$I^s{}_j = \sum_{i=0}^{n^{s-1}} w^s{}_{ij}\, o^{s-1}{}_i$ -ből kiszámoljuk, hogy $\frac{\partial I^s{}_j}{\partial w^s{}_{ij}} = o^{s-1}{}_i$

Így azt kapjuk, hogy $-\eta \frac{\partial E}{\partial w^s{}_{ij}} = \eta\, e^s{}_j\, o^{s-1}{}_i$

28

s=r eset

Az $o^r{}_j = f(I^r{}_j)$ miatt az energiafüggvény

$E = \frac{1}{2}\sum_{j=1}^{n^r}(t_j - f(I^r{}_j))^2$ alakban írható fel, és ezért

$$e^r{}_j = -\frac{\partial E}{\partial I^r{}_j} = \frac{1}{2} 2\,(t_j - f(I^r{}_j))\, f'(I^r{}_j) = (t_j - o^r{}_j)\, o^r{}_j(1 - o^r{}_j)$$

Kihasználjuk, hogy $f(x) = \frac{1}{1+e^{-x}}$, azaz $f'(x) = f(x)(1-f(x))$

29

s<r eset

E függ *s+1*-edik réteg ($I^{s+1}{}_k$) összegzett bemeneteitől is, amelyek azonban mindannyian függnek az *s*-edik réteg *j*-edik neuronjának ($I^s{}_j$) összegzett bemenetétől.

$$e^s{}_j = -\frac{\partial E}{\partial I^s{}_j} = -\sum_{k=1}^{n^{s+1}} \frac{\partial E}{\partial I^{s+1}{}_k} \frac{\partial I^{s+1}{}_k}{\partial I^s{}_j} = \sum_{k=1}^{n^{s+1}} -\frac{\partial E}{\partial I^{s+1}{}_k} \frac{\partial I^{s+1}{}_k}{\partial I^s{}_j} =$$

Az $I^{s+1}{}_k$ függ az *s*-edik réteg *j*-edik neuronjának ($o^s{}_j$) kimenetétől, ezért

$$= \sum_{k=1}^{n^{s+1}} -\frac{\partial E}{\partial I^{s+1}{}_k} \frac{\partial I^{s+1}{}_k}{\partial o^s{}_j} \frac{\partial o^s{}_j}{\partial I^s{}_j} =$$

30

s<r eset

$$= \sum_{k=1}^{n^{s+1}} -\frac{\partial E}{\partial I^{s+1}_k} \frac{\partial \sum_{j=0}^{n^s} w^{s+1}_{jk} o^s_j}{\partial o^s_j} \frac{\partial f(I^s_j)}{\partial I^s_j} =$$

Felismerve a e^{s+1}_k jelölést a $-\frac{\partial E}{\partial I^{s+1}_k}$ -ban, és kiszámolva a deriváltakat:

$$= \sum_{k=1}^{n^{s+1}} e^{s+1}_k w^{s+1}_{jk} f'(I^s_j) = o^s_j(1- o^s_j) \sum_{k=1}^{n^{s+1}} e^{s+1}_k w^{s+1}_{jk}$$

Összesítve

Ha $s=r$ akkor

$e^r_j = o^r_j(1- o^r_j)(t_j- o^r_j)$

$\Delta w^r_{ij} = \eta e^r_j o^{r-1}_i$

Ha $s<r$ akkor

$e^s_j = o^s_j(1- o^s_j) \sum_{k=1}^{n^{s+1}} e^{s+1}_k w^{s+1}_{jk}$

$\Delta w^s_{ij} = \eta e^s_j o^{s-1}_i$

rekurzív szabályt kaptuk a súlyok módosítására.

Algoritmus

{*inicializálás*}
$o^s_0 \leftarrow ?$; $w^s_{ij} \leftarrow ?$; $\eta \leftarrow ?$

{*tanító példák feldolgozása:*}
for $\forall(x,t) \in P$ loop

{*kimenet számítás*}
{*bemeneti réteg*}
for $j=1 .. n^0$ loop $o^0_j \leftarrow x_j$ endloop

{*előre haladva a neuron rétegeken*}
for $s=1 .. r$ loop
for $j=1 .. n^s$ loop $o^s_j \leftarrow f(\sum_{i=0}^{n^{s-1}} w^s_{ij} o^{s-1}_i)$ endloop
endloop

Algoritmus

{*hiba visszaterjesztés és súlymódosítás*}

{*kimeneti réteg*}
for $j=1 .. n^r$ loop $e^r_j \leftarrow (t_j- o^r_j) o^r_j(1- o^r_j)$ endloop

for $i=0 .. n^{r-1}$ loop
for $j=1 .. n^r$ loop $w^r_{ij} \leftarrow w^r_{ij} + \eta e^r_j o^{r-1}_i$ endloop
endloop

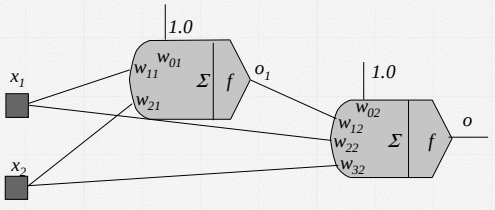
Algoritmus

{*rejtett rétegekre visszafelé haladva*}
for $s=r-1 .. 1$ loop
for $j=1 .. n^s$ loop $e^s_j \leftarrow o^s_j(1- o^s_j)(e^{s+1}_k w^{s+1}_{jk})$ endloop $\sum_{k=1}^{n^{s+1}}$
for $i=0 .. n^{s-1}$ loop
for $j=1 .. n^s$ loop $w^s_{ij} \leftarrow w^s_{ij} + \eta e^s_j o^{s-1}_i$ endloop
endloop
endloop

endloop

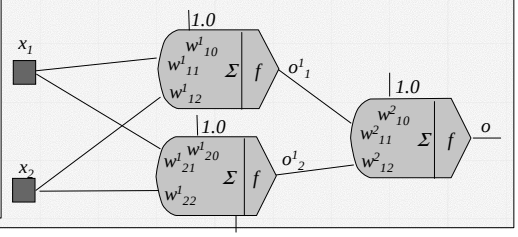
XOR művelet 1. példája

Beállítások: $x_1, x_2 \in \{0,1\}$ $f(x)=szigmoid(x)$ $o_i \in (0,1)$
$w_{ij} = rand(-0.1 , 0.1)$ $o^s_0=1.0$ $\eta=1.0$



XOR művelet 2. példája

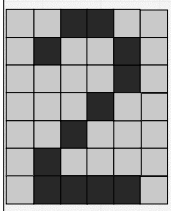
Beállítások: $x_1, x_2 \in \{0,1\}$ $f(x)=szigmoid(x)$ $o^s_i \in (0,1)$
$w^s_{ij} = rand(-0.1 , 0.1)$ $o^s_0=1.0$ $\eta=1.0$



37

Számjegy felismerés

Bemeneti értékek száma: 42 Kimenetek száma: 10



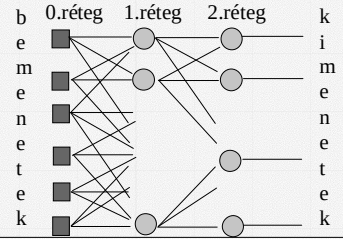
Minden számjegyhez tartozik egy kimenet

Közbülső réteg sejtjeinek száma: 11

38

Számjegy felismerés

Beállítások: $x_i \in \{0,1\}$ $f(x)=szigmoid(x)$ $o^s_i \in (0,1)$
$w^s_{ij} = rand(-0.1 , 0.1)$ $o^s_0=1.0$ $\eta=0.35$



39