

# Üzleti Intelligencia

## 7. Előadás: Mesterséges mélytanulás

Kuknyó Dániel  
Budapesti Gazdasági Egyetem

2023/24  
1.félév

1 Bevezetés

2 Gradiens ereszkedés

3 Neurális hálózatok

4 Tanítás

5 Konvolúciós hálózatok

6 Önkódoló architektúrák

## 1 Bevezetés

## 2 Gradiens ereszkedés

## 3 Neurális hálózatok

## 4 Tanítás

## 5 Konvolúciós hálózatok

## 6 Önkódoló architektúrák

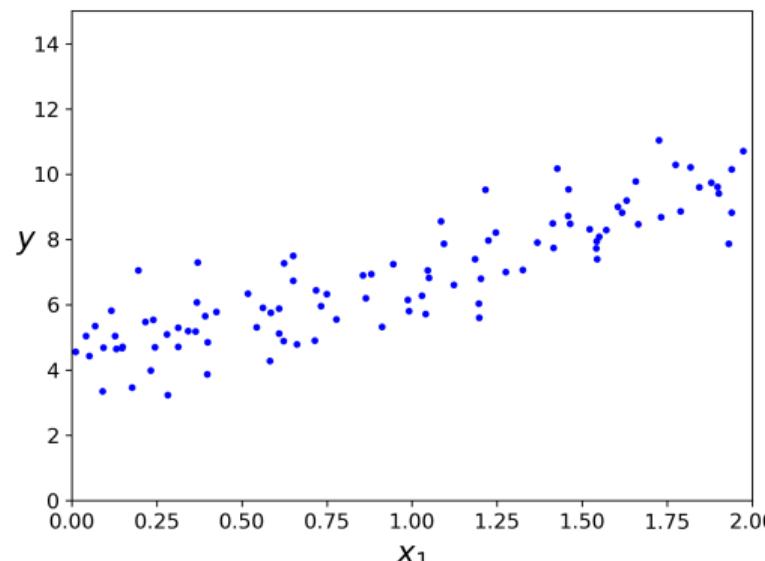
## Függvény illesztés alapjai

A függvény illesztés eljárása szerint valamely  $x$  független változóból vett minta alapján szeretnénk előre jelezni egy  $y$  függő változó értékét azért, hogy feltárja az adatpontok közötti mintázatokat.

Két eljárása ismert:

- **Regresszió:** tárgya egy folytonos változó
  - **Osztályozás:** tárgya egy diszkrét változó

A függvény illesztés eredménye a **modell**.



# Függvény illesztés alapjai

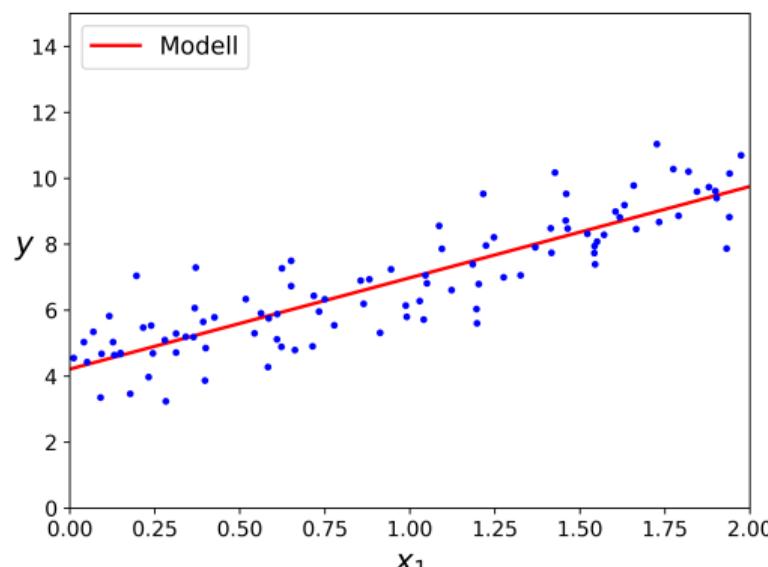
Az illesztett modell alakját és viselkedését a **paraméterei** határozzák meg, amelyek együtthatókként viselkednek a modell egyenletében. A lineáris modell egyenlete:

$$y_i = \theta_0 + \theta_1 x_i + \varepsilon_i$$

Ahol:

- $\theta_0$ : az  $y$  tengely metszéspontja, vagy eltolás
- $\theta_1$ : az egyenes meredeksége
- $\varepsilon$ : a véletlen hiba, amit a modell nem tud előre jelezni

$\theta = [\theta_0, \theta_1, \dots, \theta_n]$  a paraméterek vektora.

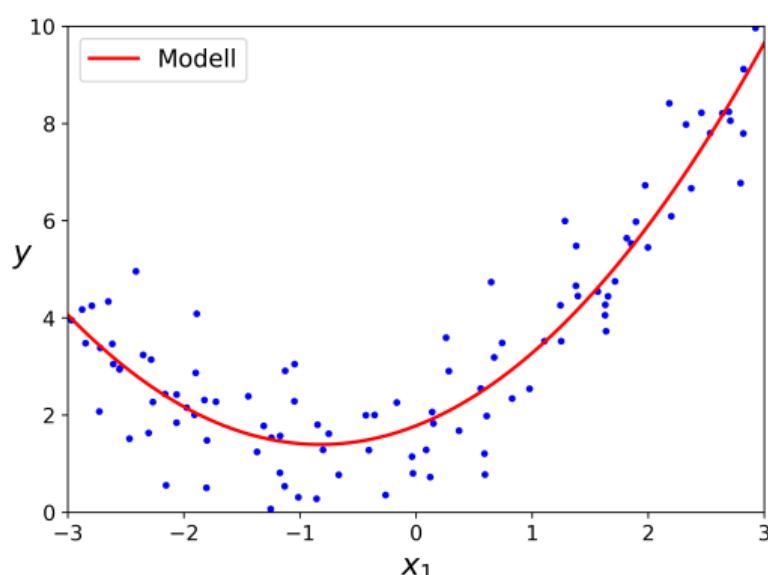


# Függvény illesztés több paraméterrel

Függvényt lehetséges nemlineáris adatokra is illeszteni. Ebben a példában a minta adatpontok kvadratikusak, **nem írhatók le egy lineáris egyenlettel**. A modellnek ebben az esetben 3 paramétere van:  $\theta_0$ ,  $\theta_1$  és  $\theta_2$ . Az illesztett modell egyenlete:

$$y = 1.78 + 0.93x + 0.56x^2$$

Tehát ebben az esetben  $\theta_0 = 1.78$ ,  $\theta_1 = 0.93$  és  $\theta_2 = 0.56$ .



1 Bevezetés

2 Gradiens ereszkedés

3 Neurális hálózatok

4 Tanítás

5 Konvolúciós hálózatok

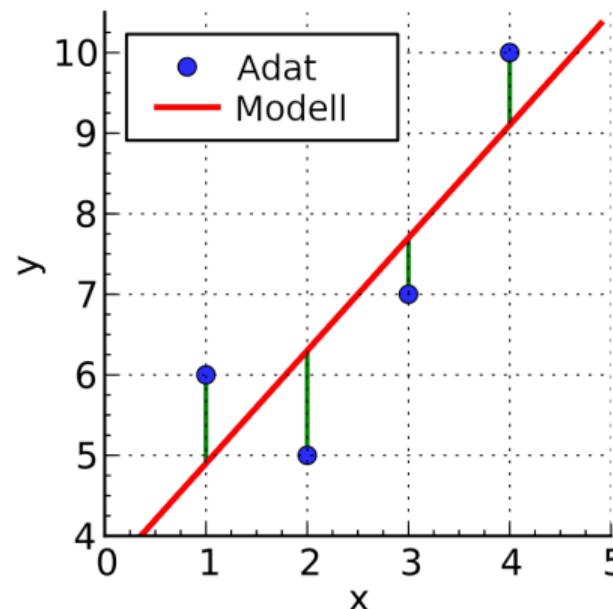
6 Önkódoló architektúrák

# A költségfüggvény

A modell illesztő algoritmusok mindegyike úgy találja meg az optimális függvényt, hogy valamilyen költségfüggvényt minimalizál. A leggyakoribb ilyen költségfüggvény az **átlagos négyzetes hiba**:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - f(x)_i)^2$$

- $y_i$ : Megfigyelt adatpont
- $f(x)_i$ : Modell által adott becslés



# A gradiens

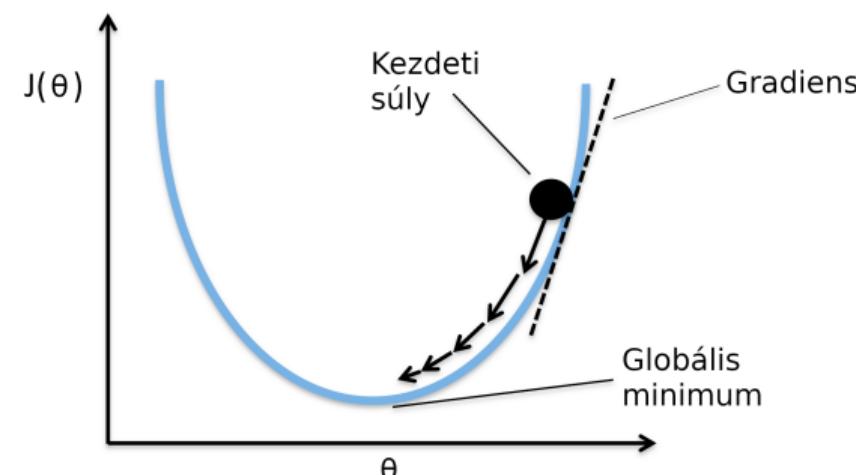
A függvény illesztés célja, hogy megtalálja azt a modellt, ami a legjobban illeszkedik az adatpontokra, tehát **minimalizálja a költségfüggvényt (MSE)**.

## Gradiens

Olyan vektor, amely megmutatja hogyan változik a függvény, és megadja a legnagyobb változás irányát minden dimenzióban.

$$df = \nabla f * dx$$

A gradiens segítségével meg lehet határozni, merre és mennyivel érdemes változtatni a paramétereket a célfüggvény értékének csökkentése érdekében.

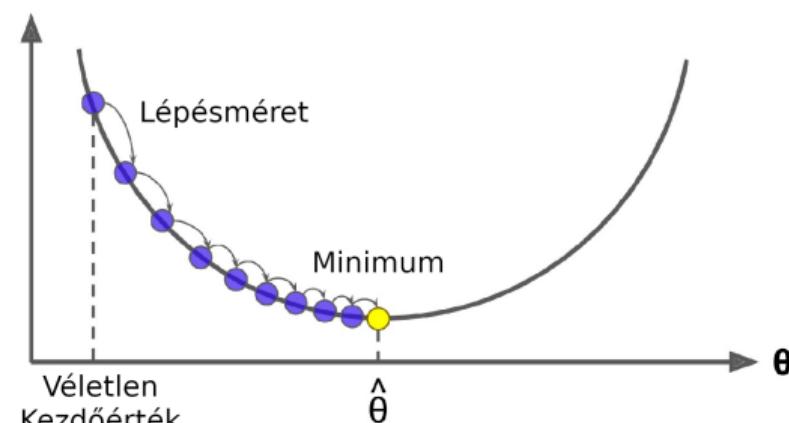


# Gradiens ereszkedés

A gradiens ereszkedés egy iteratív minimalizálási algoritmus egy tetszőleges függvény lokális minimum helyének megtalálására. Az algoritmus lépésről lépésre mozog a függvény értékének csökkentése érdekében.

## Tanulási sebesség ( $\alpha$ vagy $\eta$ )

A tanulási sebesség meghatározza, mennyire nagy lépésekkel tesz a gradiens ereszkedés az optimalizációs folyamat során.

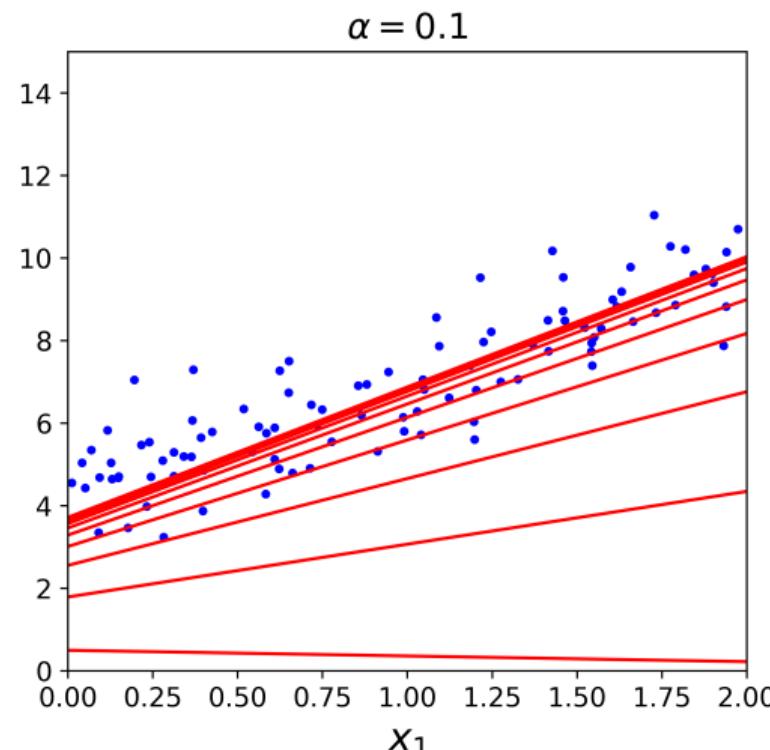


# Gradiens ereszkedés

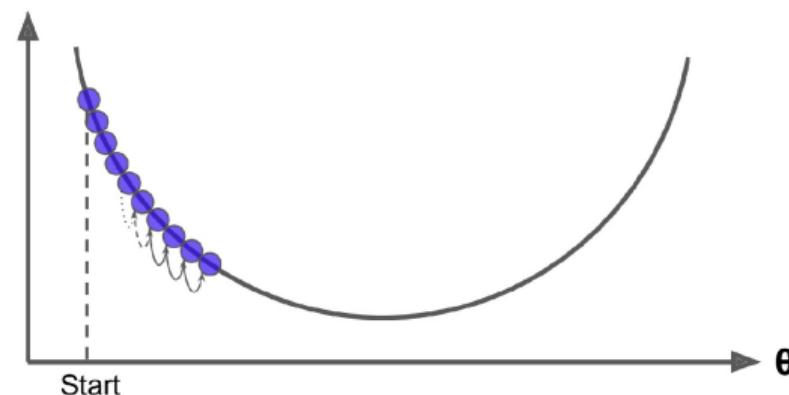
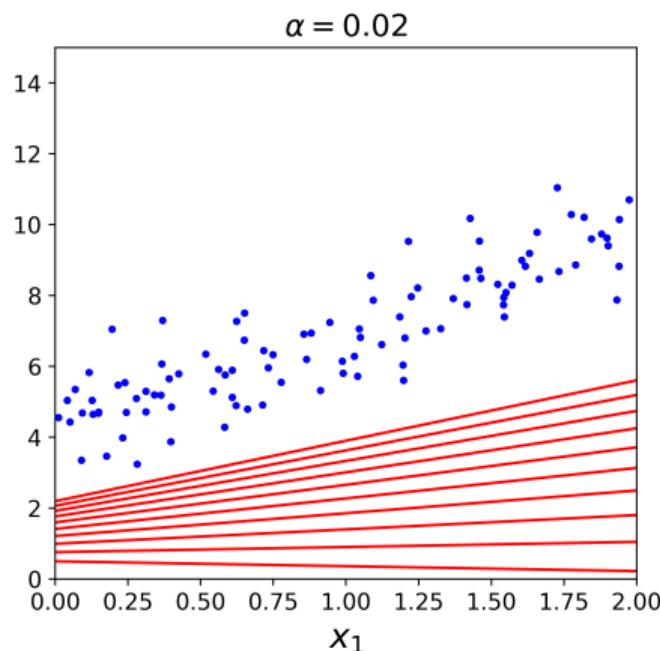
A gradiens ereszkedés paraméter frissítése:

$$\theta' \leftarrow \theta - \alpha \nabla_{\theta} J(\theta)$$

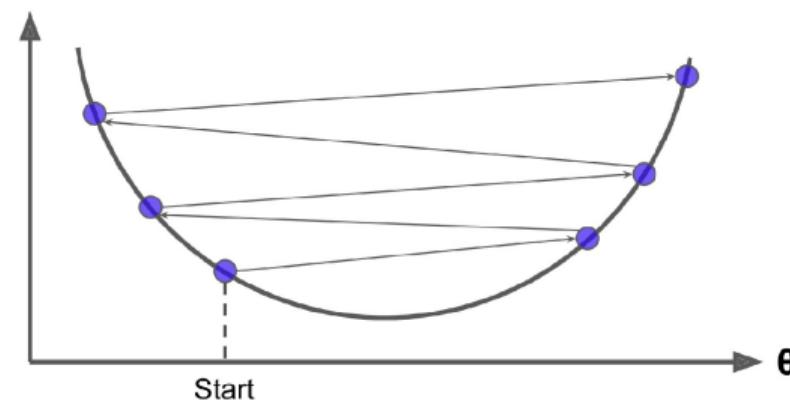
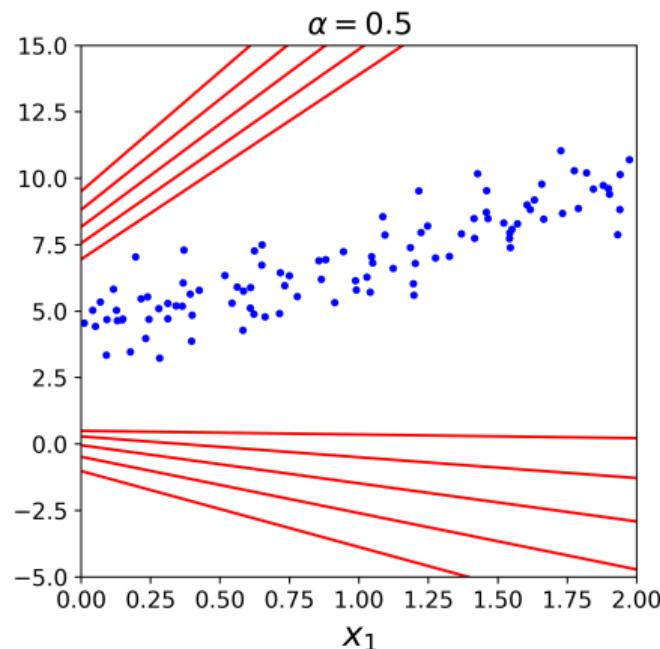
- $\theta = [\theta_0, \theta_1, \dots, \theta_n]$ : Paraméterek vektora
- $\alpha \in [0, 1]$ : Tanulási sebesség
- $\nabla_{\theta}$ : Költségfüggvény gradiense ( $\theta$  szerinti derivált)
- $J(\theta)$ : Költségfüggvény  $\theta$  szerint (pl. MSE)



# Túl alacsony tanulási sebesség



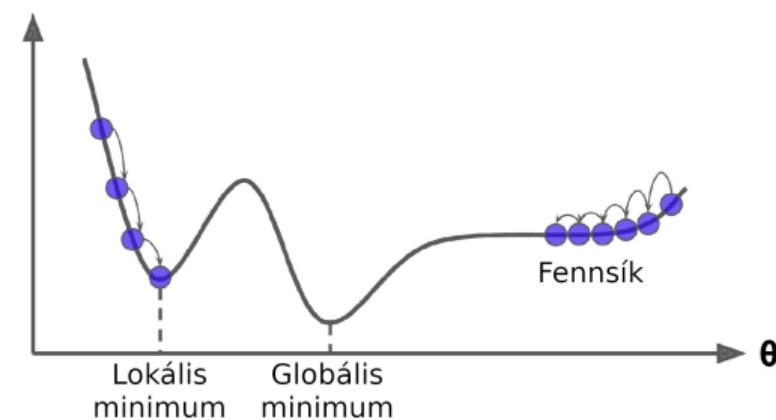
# Túl magas tanulási sebesség



## Egyéb problémák

A gradiens ereszkedés nem garantálja, hogy elér egy globális minimumot, hiszen a gradiens operátora csak lokális változásokat vesz figyelembe. Ebből adódóan az **algoritmus beragadhat egy lokális minimumon.**

Vannak esetek, amikor a gradiens nem meghatározható (szaturál), például amikor **elér egy fennsíkot a függvényen.** Ebben az esetben a futás vagy nagyon lassú lesz, vagy nem halad semerre.



1 Bevezetés

2 Gradiens ereszkedés

3 Neurális hálózatok

4 Tanítás

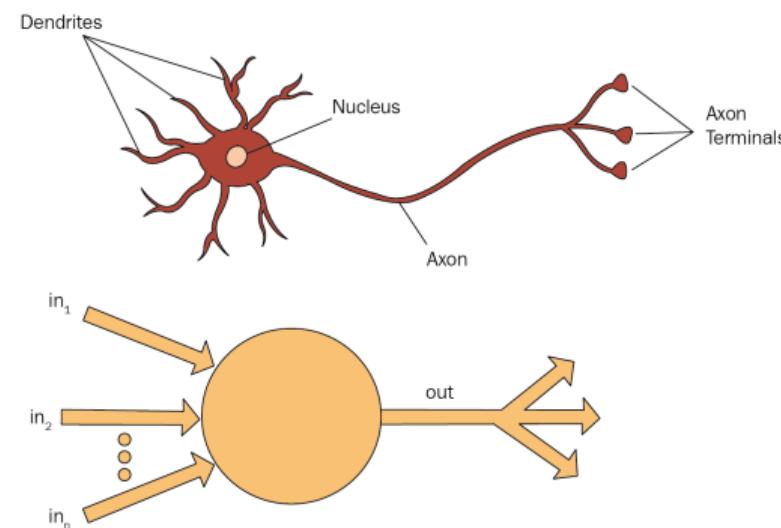
5 Konvolúciós hálózatok

6 Önkódoló architektúrák

# Biológiai és mesterséges neuronok

Az ember találmányait mindig a természet ihlette. A repülőgépeket a madarak mintájára, a gépkocsit a lovak inspirálták. A természetes lépés ezután az volt, hogy az emberi agyat is modellezik.

Az első perceptron modellt Warren McCulloch és Walter Pitts hozta létre, először pedig Frank Rosenblatt épített egy perceptron gépet. Ez egy képfelismerő gép volt, 400 véletlenszerűen kapcsolt fotocella volt az érzékelője. A súlyokat potenciometerek implementálták, és a súlyok frissítését elektromotorok hajtották végre.



# A neuron

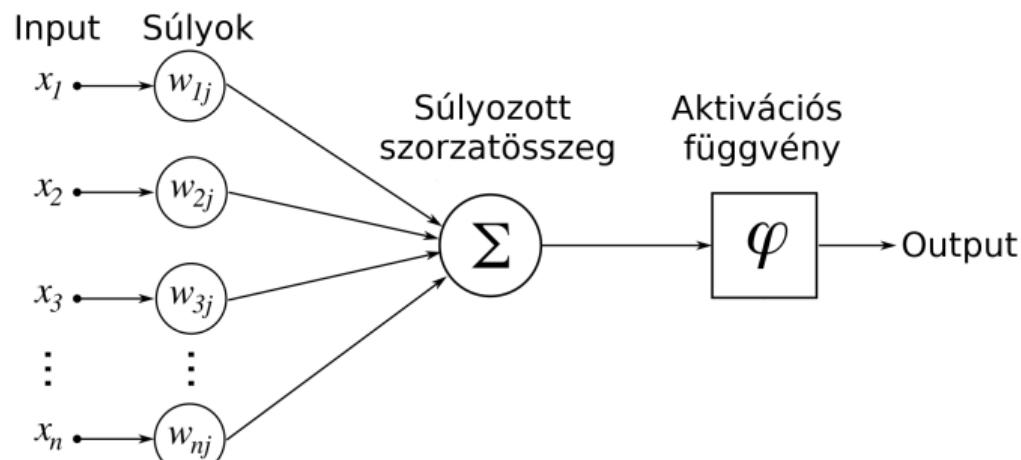
Az egyes neuronok egyszerű elemi műveleteket végeznek. A neuronnak több  $x$  inputja (kapcsolata) van, és mindegyikhez egy  $w$  súly tartozik.

A neuron kiszámítja az inputjainak a súlyozott szorzatösszegét:

$$z = x_1 w_1 + x_2 w_2 + \dots + x_n w_n$$

majd ezt az értéket behelyettesíti egy aktivációs függvénybe:

$$h = \varphi(z)$$

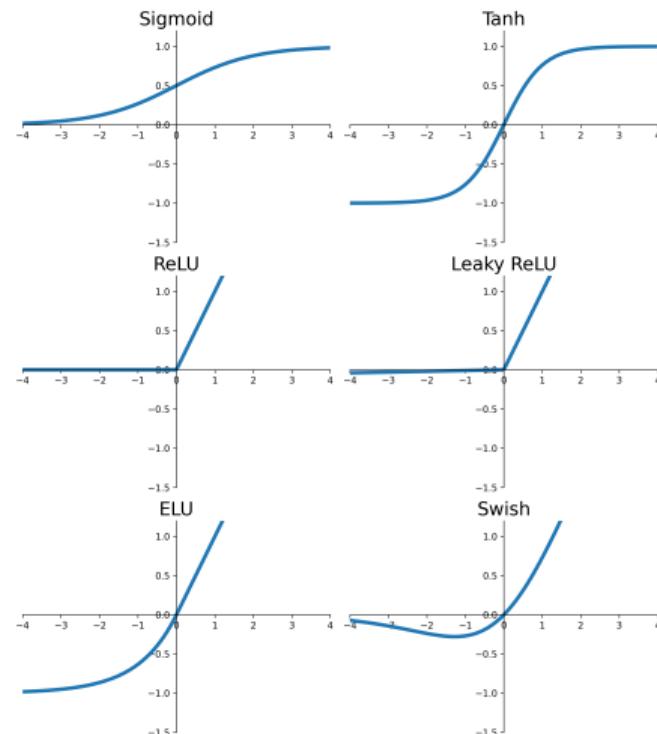


# Gyakori aktivációs függvények

A neuron által kiszámolt súlyozott szorzatösszeg egy aktivációs függvénybe kerül behelyettesítésre.

A neurális hálózatok az aktivációs függvények segítségével **sajátítanak el komplex mintázatokat**. Az aktivációs függvény vezeti be a neurális hálózatokba a **nemlineáris transzformációt**, enélkül csak egy lineáris transzformáció lenne.

A különböző alkalmazásokra külön aktivációs függvények használatosak.



# Gyakori aktivációs függvények

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

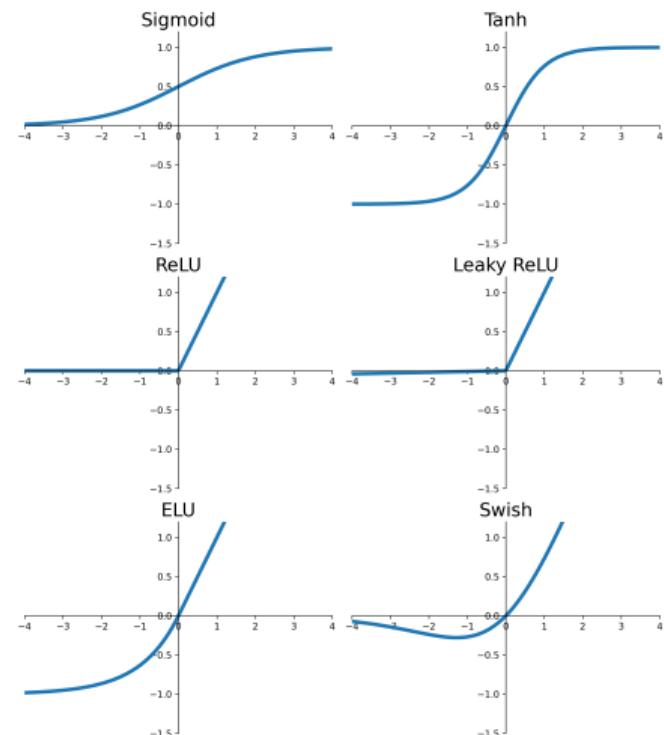
$$\text{ReLU}(x) = \max(0, x)$$

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

$$\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$$

$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{ha } x \geq 0 \\ \alpha \cdot x & \text{ha } x < 0 \end{cases}$$

$$\text{Swish}(x, \beta) = \frac{x}{1+e^{-\beta \cdot x}}$$



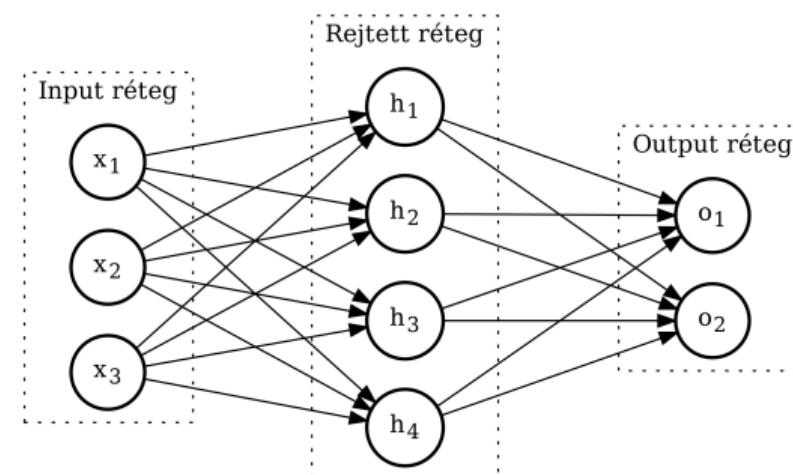
# Többrétegű hálózatok

Ebben az esetben a neuronok rétegekben foglalnak helyet. A kapcsolataik az előző réteg kimeneteivel állnak összeköttetésben. A legelső réteg neuronjai a bemeneti adattal állnak összeköttetésben. minden bemeneti jellemzőhöz egy neuron tartozik.

## Teljesen becsatolt neuronréteg kimenete

$$h_{W,b}(X) = \varphi(XW + b)$$

- $X$ : Input jellemzők mátrixa
- $W$ : Kapcsolati súlyok mátrixa
- $b$ : Torzítások vektorá
- $\varphi$ : Aktivációs függvény



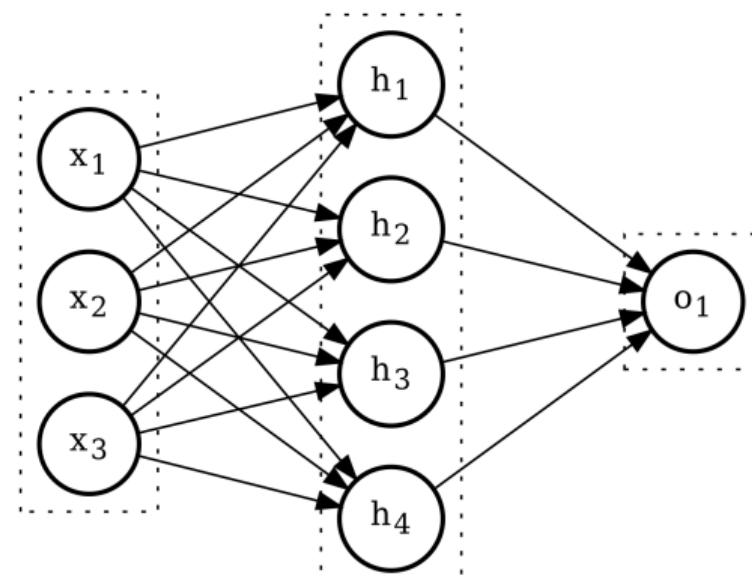
# Hálózati architektúrák

A regressziós problémák esetén a neurális hálónak **egyetlen output neuronja** van.

A regresszió **tárgya egy folytonos** változó. Ebben az esetben a neuron output értéke a neurális hálózat predikciója a célváltozóra vonatkozóan.

**Például:** hány fok lesz holnap este?

- 35.

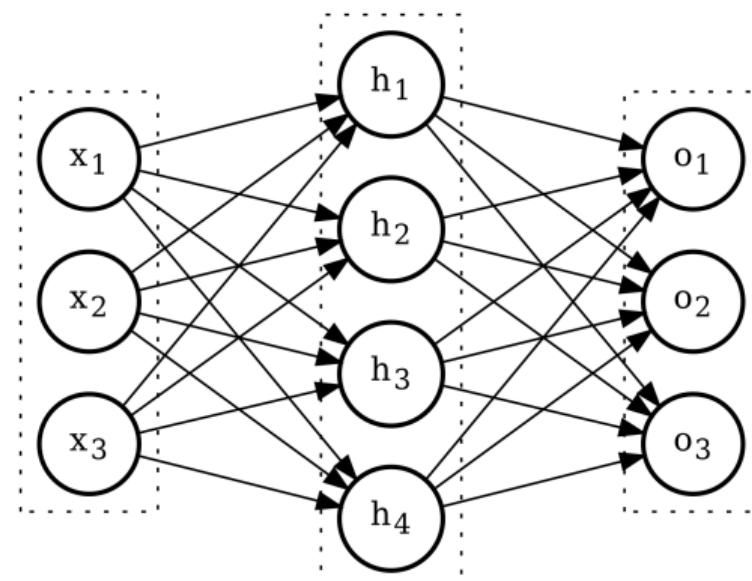


# Hálózati architektúrák

Az osztályozási problémák tárgya **egy diszkrét változó**, amely különálló kategóriáakra osztható.

Az osztályozó hálózatnak **annyi output neuronja van, ahány kategória lehetséges osztályozás esetén**. A predikció a mintaegyed adott osztályba esésének valószínűségét adja.

**Például:** meleg vagy hideg lesz az idő holnap este?  
- Hideg.



1 Bevezetés

2 Gradiens ereszkedés

3 Neurális hálózatok

4 Tanítás

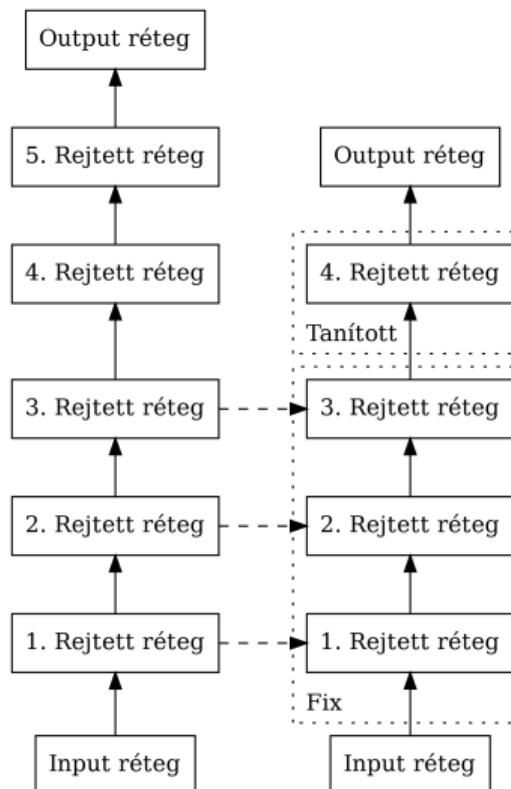
5 Konvolúciós hálózatok

6 Önkódoló architektúrák

# Transzfertanulás

Neurális hálózatok esetén lehetséges más neurális hálózatok **előretanított súlyainak felhasználása**. Ebben az esetben adott rétegek kímaradnak a tanításból.

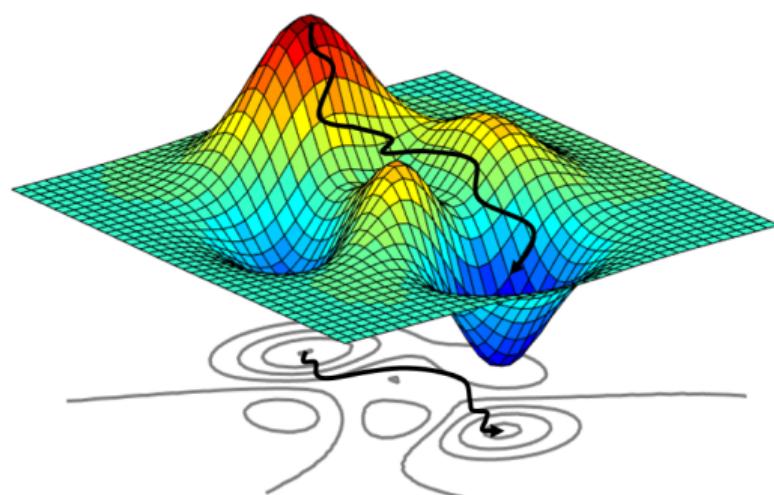
Jellemzően az alsó rétegek lesznek rögzítettek, és a felső rétegek pedig tanítottak. Ez lehetőséget ad a hálózatnak **új mintázatokat megismerni hasonló feladatok** esetén.



# Optimalizációs algoritmusok

A matematikában az optimalizációs algoritmusok célja megtalálni a legjobb megoldást egy olyan problémára, ahol valamilyen korlátosokkal kell **egy célfüggvényt minimalizálni**.

Mivel a neurális hálók által reprezentált nemlineáris függvények minimum helyei nem számíthatók ki zárt formájú függvénnyel mint pl. a másodfokú függvény esetén, iteratív optimalizációs algoritmusokra van szükség.



# Kötegelt gradiens ereszkedés

---

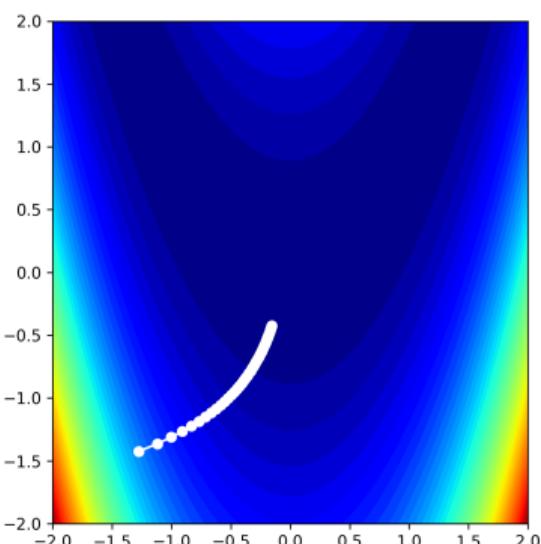
## Algoritmus 1: Kötegelt gradiens ereszkedés

---

**Input:**  $\alpha$  tanulási sebesség,  $f(\theta)$  célfüggvény  
 $\theta_0 \leftarrow 0$ ; /\* Paraméterek inicializálása \*/  
**for**  $t = 0 \rightarrow \max_t$  **do**  
    |  $\nabla f(\theta_t)$  gradiens kiszámítása;  
    |  $\theta_{t+1} = \theta_t - \alpha \cdot \nabla f(\theta_t)$ ;  
**end**

---

Ahol  $\theta$  a célfüggvényt meghatározó paraméterek vektora,  
 $\nabla f(\theta_t)$  a célfüggvény gradiense,  $\alpha \in [0, 1]$  a tanulási  
sebesség.



# Mini-kötegelt gradiens ereszkedés

---

## Algoritmus 2: Mini-kötegelt gradiens ereszkedés

---

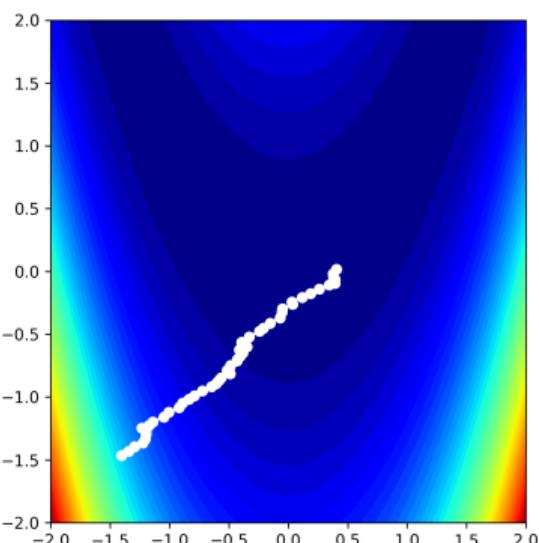
**Input:**  $\alpha$  tanulási sebesség,  $f(\theta)$  célfüggvény  
 $\theta_0 \leftarrow 0$ ; /\* Paraméterek inicializálása \*/  
**for**  $t = 0 \rightarrow \max_t$  **do**

$B$  miniköteg mintázása a tanító halmazból;  
 $\nabla f(\theta_t, B)$  gradiens kiszámítása a minikötegen;  
 $\theta_{t+1} = \theta_t - \alpha \cdot \nabla f(\theta_t, B)$ ;

**end**

---

Ahol  $B$  a teljes tanító halmazból vett véletlenszerű minta. A minta számoságát hiperparaméter szabályozza.



# Sztochasztikus gradiens ereszkedés

---

## Algoritmus 3: Sztochasztikus gradiens ereszkedés

---

**Input:**  $\alpha$  tanulási sebesség,  $f(\theta)$  célfüggvény

$\theta_0 \leftarrow 0$ ; /\* Paraméterek inicializálása \*/

**for**  $t = 0 \rightarrow \max_t$  **do**

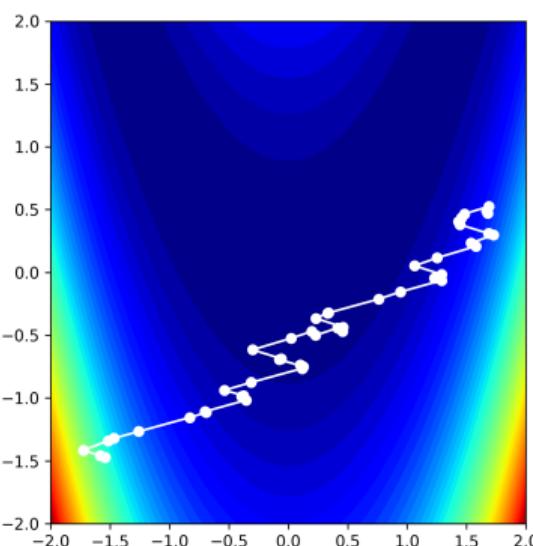
$p$  adatpont kiválasztása az adathalmazból;

$\nabla f(\theta_t, p)$  gradiens kiszámítása a pontban;

$\theta_{t+1} = \theta_t - \alpha \cdot \nabla f(\theta_t, p)$ ;

**end**

---



# Momentum gradiens ereszkedés

---

## Algoritmus 4: Momentum gradiens ereszkedés

---

**Input:**  $\alpha$  tanulási sebesség,  $\beta$  lendület komponens

$f(\theta)$  célfüggvény

$\theta_0 \leftarrow 0$ ; /\* Paraméterek inicializálása \*/

$v \leftarrow 0$ ; /\* Sebesség komponens inicializálása \*/

**for**  $t = 0 \rightarrow \max_t$  **do**

$\nabla f(\theta_t)$  gradiens kiszámítása;

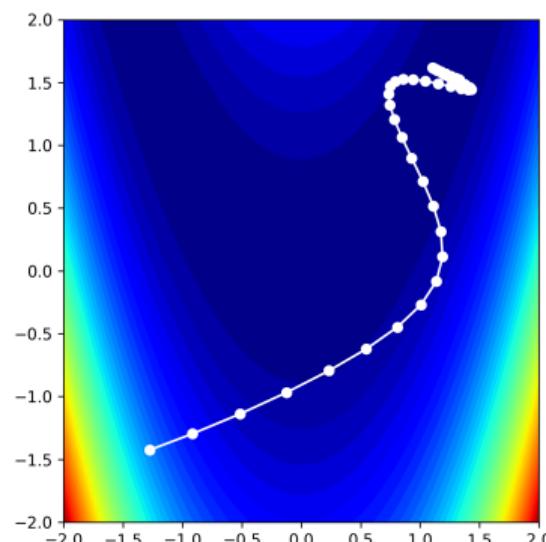
$v \leftarrow v \cdot \beta + (1 - v) \cdot \nabla f(\theta)$ ;

$\theta_{t+1} \leftarrow \theta_t - \alpha \cdot v$ ; /\* Paraméterek frissítése \*/

**end**

---

A  $v$  sebesség komponens akkumulálja a súlyozott múltbeli gradienseket, ezért fog hasonlítani az optimalizáció röppályája egy való életben elgurított golyóéra, amire érvényesek a newtoni mechanika elvei.



# Hiba-visszaáramoltató algoritmus neurális hálózat súlyok tanítására

- ① **Inicializáció:** súlyok és torzítások kezdőértékeinek véletlenszerű megadása
- ② **Előreáramoltatás:** rétegek output értékének kiszámítása:
  - Rejtett réteg:  $h_h = \varphi(W_h \cdot x + b_h)$
  - Output réteg:  $h_o = \varphi(W_o \cdot h_h + b_o)$
- ③ **Költség kiszámítása:** regresszió esetén pl.:  $MSE = \sum (h_o - y)^2$
- ④ **Visszaáramoltatás:**
  - Rejtett réteg hibája:  $\delta_h = (W_o \cdot \delta_o) \odot \varphi'(z_h)$
  - Output réteg hibája:  $\delta_o = (h_o - y) \odot \varphi'(z_o)$
- ⑤ **Gradiens ereszkedés a súlyokon és torzításokon:**
  - Rejtett réteg:  $W_h \leftarrow W_h - \alpha \cdot \delta_h \cdot x^\top$
  - Rejtett réteg:  $b_h \leftarrow b_h - \alpha \cdot \delta_h$
  - Output réteg:  $W_o \leftarrow W_o - \alpha \cdot \delta_o \cdot h_o^\top$
  - Output réteg:  $b_o \leftarrow b_o - \alpha \cdot \delta_o$
- ⑥ **Ismétlés a meghatározott lépésszámig**

1 Bevezetés

2 Gradiens ereszkedés

3 Neurális hálózatok

4 Tanítás

5 Konvolúciós hálózatok

6 Önkódoló architektúrák

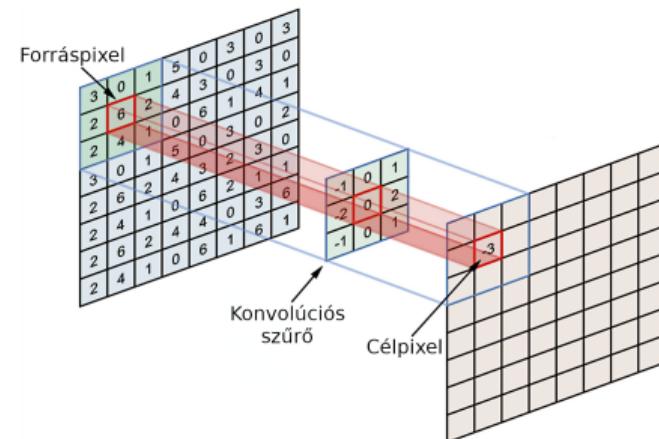
# A konvolúció művelete

A konvolúció egy matematikai művelet, ami két függvényt kombinál egy harmadikká. Egy **jelfüggvényt** valamelyen **magfüggvénnyel** vegyít egy outputtá, ami megadja, hogy a **mag mennyiben befolyásolja a jelet**.

## Konvolúció (2D, diszkrét)

A diszkrét konvolúció két 2D tömbön értelmezett, és jellemzők kivonatolására, szűrők alkalmazására használatos. Konvolúció  $f$  jelen és  $g$  magon,  $i, j$  képkordinátákra:

$$(f*g)[i, j] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} g[i - m, j - n]f[m, n]$$

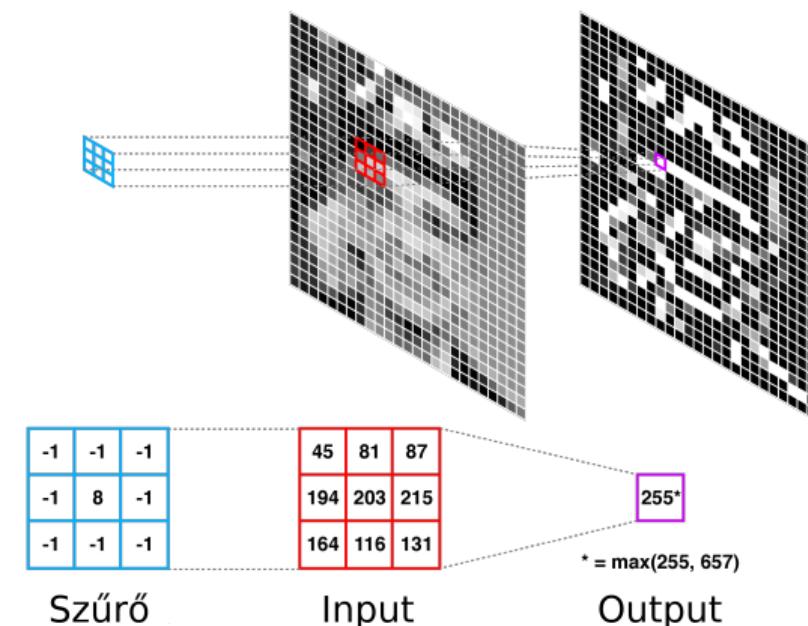


$$\begin{aligned} & (-1 \cdot 3) + (0 \cdot 0) + (1 \cdot 1) + \\ & (-2 \cdot 2) + (0 \cdot 6) + (2 \cdot 2) + \\ & (-1 \cdot 2) + (0 \cdot 4) + (1 \cdot 1) = -3 \end{aligned}$$

# A konvolúciós réteg

A neurális hálózatok konvolúciós rétegeiben a tanítható súlyok az egyes forráspixelekhez tartozó konvolúciós szűrőkben elhelyezkedő értékek. Ez minden forráspixelhez egy egyedi szűrőt jelent, de lehetséges tetszőleges számú szűrő is.

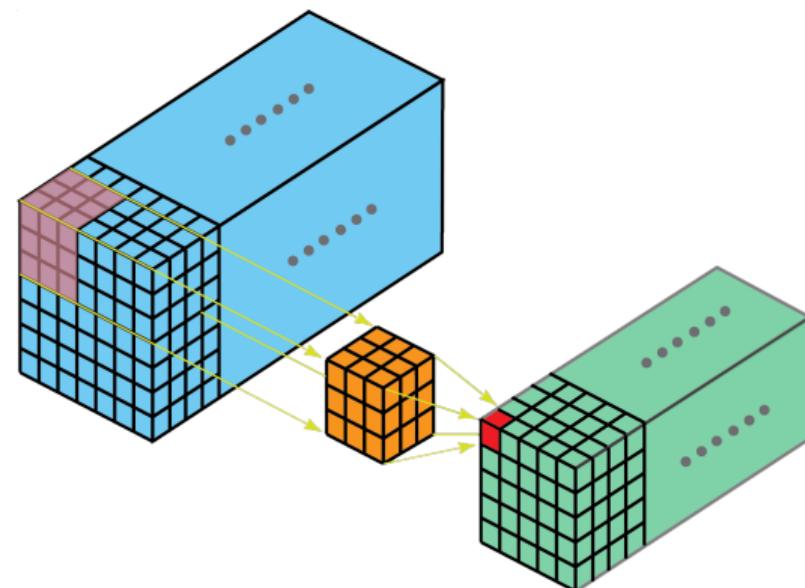
A konvolúciós magfüggvények szűrőként is alkalmazhatók. Például a képen egy élkereső szűrő látható, ami az outputra az input képen található élek másolódnak át.



# Konvolúció több dimenzióban

A konvolúció általánosítható több dimenzióra is. A gyakorlatban 3D konvolúció megy végbe a neurális hálózatokban, hiszen a színes képek 3D mátrixokban tárolódnak el, ahol az egyes dimenziók a *magasság*, *hosszúság* és *csatorna*. A csatorna három elemű és az R,G,B értékeket kódolja.

A konvolúció tetszőleges dimenzióra általánosítható művelet.



# Pooling réteg

A pooling vagy lazítás egy almintázási művelet, ami csökkenti a **bemenet térbeli dimenzióit**, miközben a fontos információt megtartja. A konvolúciós hálózatokban gyakran használatos a számítási igény csökkentésére, túltanulás elkerülésére és a hálózat segítésére abban, hogy fontos mintázatokat tanuljon meg.

## Pooling neuron

A vele kapcsolatban álló **neuronok output értékeit aggregálja** valamelyen függvény szerint. Leggyakrabban ez az **átlagolás** vagy a **maximum** kiválasztás.

|    |     |     |    |
|----|-----|-----|----|
| -2 | -4  | -4  | 5  |
| 14 | -11 | -2  | 10 |
| 14 | 11  | -11 | -3 |
| 13 | 14  | -9  | -9 |



|    |    |
|----|----|
| 14 | 10 |
| 14 | -3 |

Max pooling



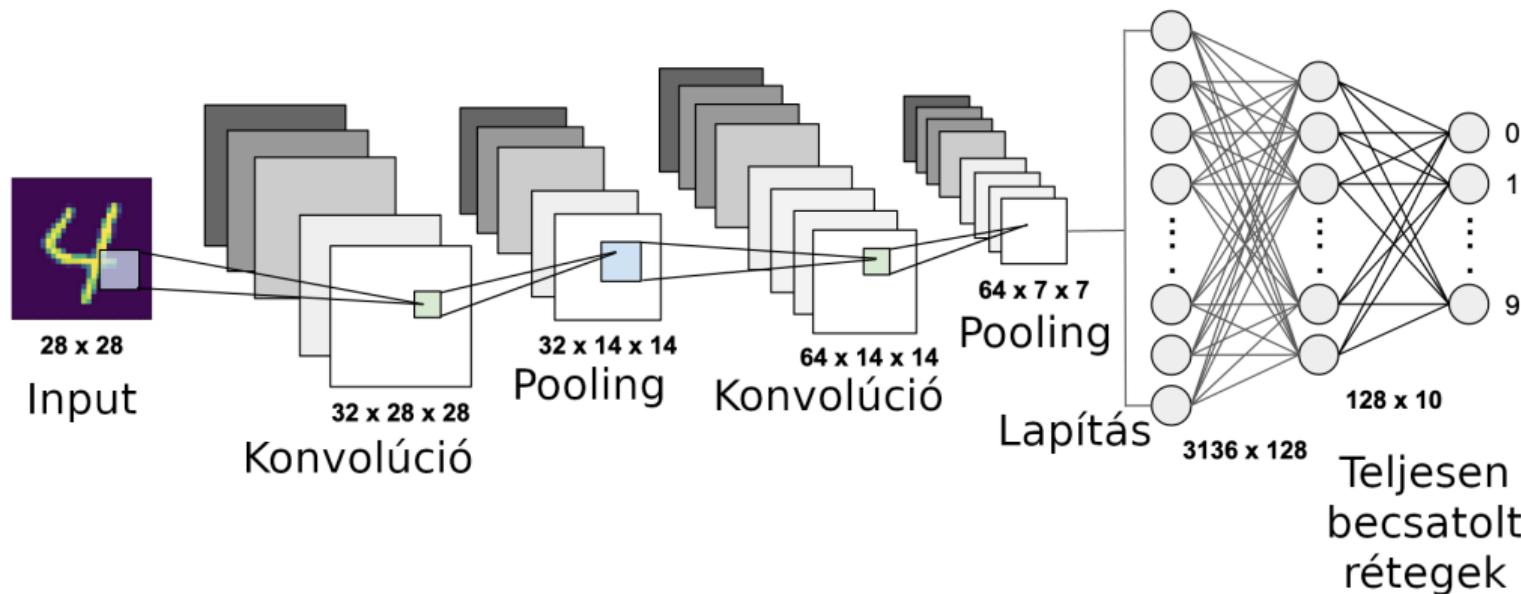
|    |    |
|----|----|
| -1 | 2  |
| 13 | -8 |

Avg pooling

$$\frac{1}{4}(-2 - 4 + 14 - 11) = -1$$

$$\max\{-2, -4, 14, -11\} = 14$$

# Teljes konvolúciós architektúra



## 1 Bevezetés

## 2 Gradiens ereszkedés

## 3 Neurális hálózatok

## 4 Tanítás

## 5 Konvolúciós hálózatok

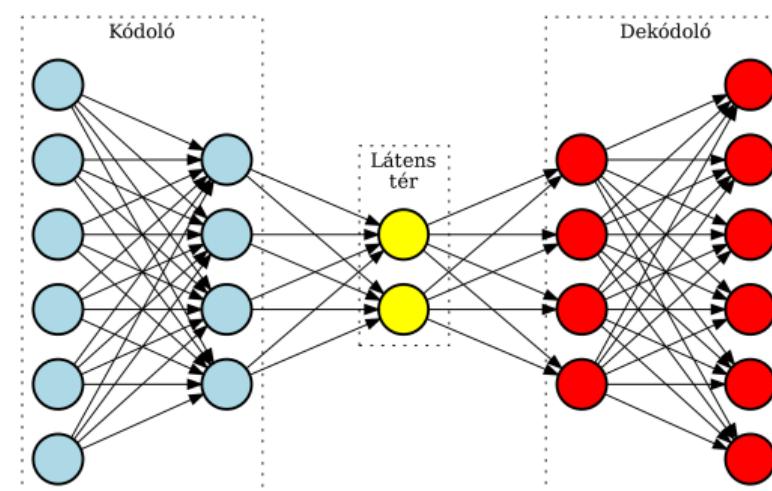
## 6 Önkódoló architektúrák

# Önkódoló architektúra

Az önkódoló architektúrák feladata megismerni az **inputnak egy tömörített reprezentációját**. Mivel nincs szükség címkére, az önkódoló egy **felügyelet nélküli tanítási módszernek** tekinthető. Viszont az **output az inputnak egy rekonstrukciója**, ezért tekinthető egy **önfelügyelt** tanítási módszernek is.

## Kódoló

A hálózat azon része, amelyik az inputot egy csökkentett méretű látens dimenzióba tömöríti az inputot. Egy  $h = f(x)$  kódoló függvénnyel reprezentálható.

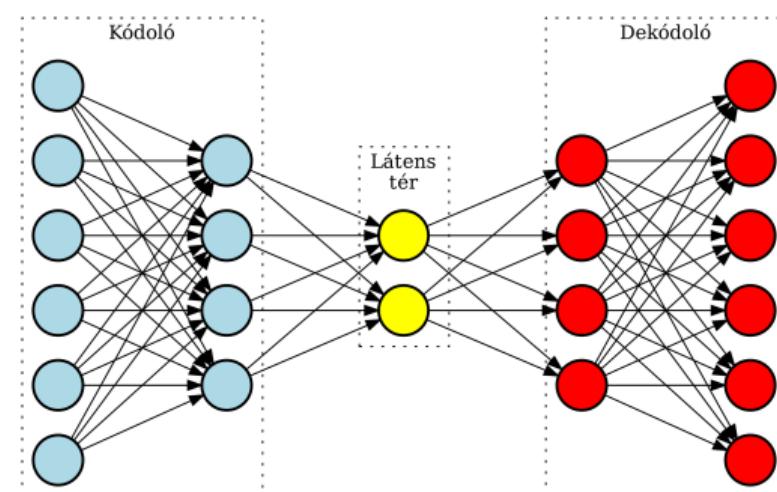


# Önkódoló architektúra

Az önkódoló architektúrák feladata megismerni az **inputnak** egy tömörített reprezentációját. Mivel nincs szükség címkére, az önkódoló egy **felügyelet nélküli** tanítási módszernek tekinthető. Viszont az **output** az **inputnak** egy **rekonstrukciója**, ezért tekinthető egy **önfelügyelt** tanítási módszernek is.

## Látens tér

A hálózat azon része, amelyik tartalmazza az input csökkentett dimenziósázmú reprezentációját.

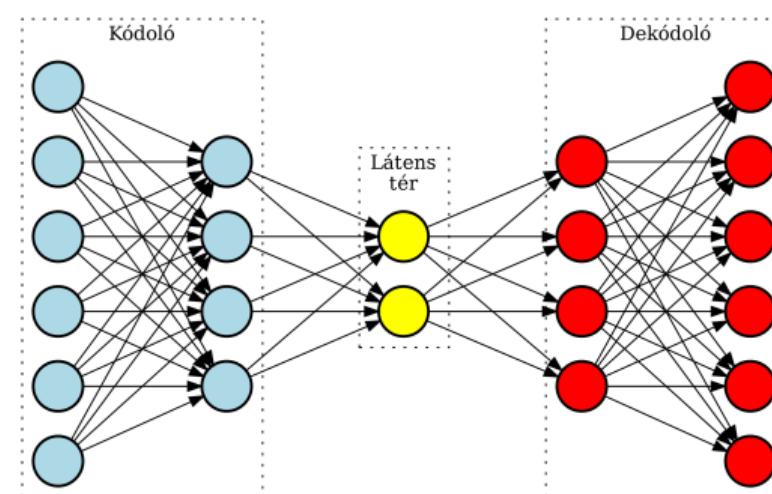


# Önkódoló architektúra

Az önkódoló architektúrák feladata megismerni az **inputnak egy tömörített reprezentációját**. Mivel nincs szükség címkére, az önkódoló egy **felügyelet nélküli tanítási módszernek** tekinthető. Viszont az **output az inputnak egy rekonstrukciója**, ezért tekinthető egy **önfelügyelt** tanítási módszernek is.

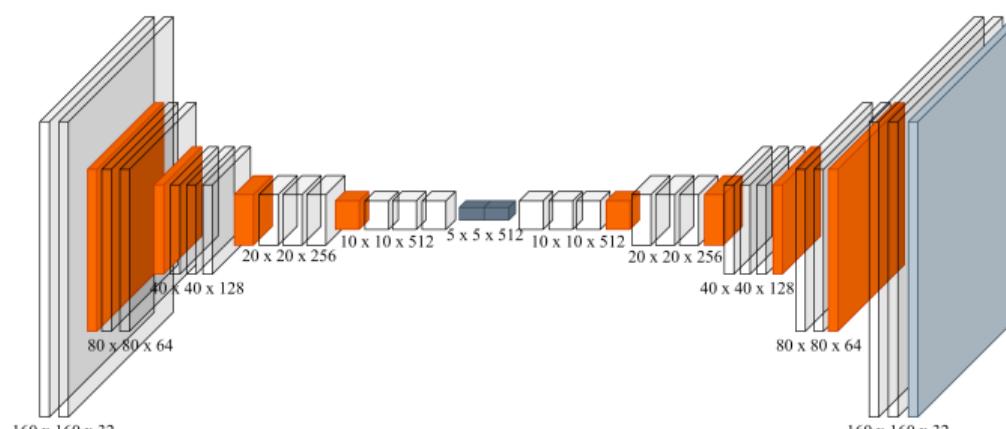
## Dekódoló

A hálózat azon része, amelyik a látens reprezentációt rekonstruálja. Struktúrája hasonló a kódolóéhoz. Egy  $r = g(h)$  függvénnyel jelölhető.



# Konvolúciós önkódolók

A konvolúciós önkódoló architektúrák kombinálják az önkódolók eredeti koncepcióját a konvolúciós rétegekkel. Az architektúra jól illeszkedik képfeldolgozó feladatokra, mint zajszűrés, tömörítés és mintázatok elsajátítása. Ebben az esetben a **hálózat inputja és outputja is egy kép**.



Konvolúció + ReLu

Max. Pooling

Kódolt adat

Konvolúció + ReLu

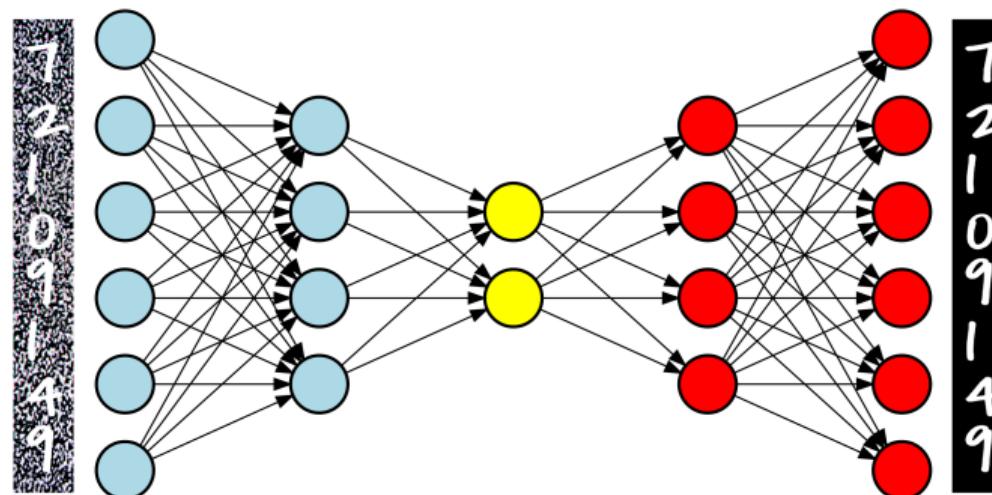
Max. pooling

Softmax

# Zajszűrő önkódolók

A **kódoló** zajos adatot fogad inputként, és átalakítja egy alacsonyabb dimenziós reprezentációjává. Ennek a reprezentációnak a feladata megragadni az input belső struktúráját és fontos mintázatait.

A **dekódoló** a kódolt reprezentációt fogadja inputként és a célja, hogy rekonstruálja az eredeti, zajmentes adatot.



# Variációs önkódolók

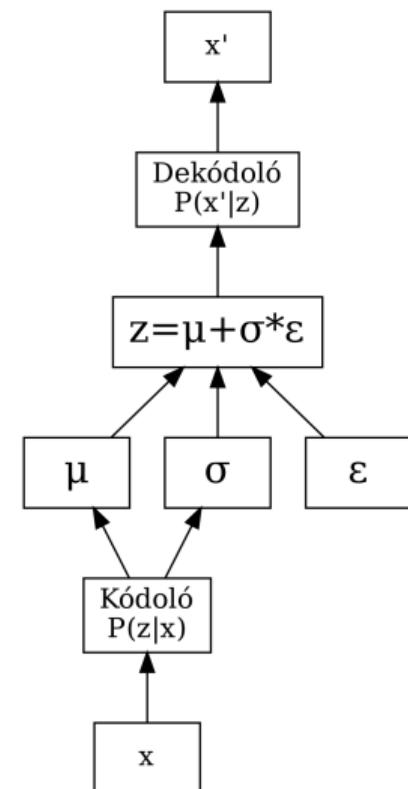
A variációs önkódolók az önkódoló mélytanuló architektúrák **probabilisztikus, generatív** változatai. A hagyományos önkódolóktól eltérően ahelyett, hogy egy input adat-szám leképező függvényt tanulnának meg, **az adathalmaznak a valószínűségeloszlását tanulják meg reprezentálni**.

A kódoló egy  $P(z|x)$  valószínűségeloszlásra képezi rá az input adatot. Ezt az eloszlást  $\mu$  várható érték és  $\sigma$  szóródás határozza meg.

A dekódoló mintát vesz a látens tér eloszlásából:

$z = \mu + \sigma \cdot \varepsilon$ , ahol  $\varepsilon$  a zaj és a  $P(x'|z)$  valószínűségeloszlás alapján állítja elő az outputot.

Mivel a variációs önkódolók megismerik az adatok alacsony szintű eloszlását, lehetséges új mintaadatokat generálni az eloszlásokból való mintavétellel.



# Szegmentációs önkódolók

Az önkódoló architektúrák használhatóak képszegmentálásra is. Ebben az esetben a hálózat feladata **felosztani a képet különálló régiókra szemantikai értelmezés szerint**.

Szegmentáció esetén a cél egy különböző képet előállítani mint az eredeti, amelynek az értelmezése is különbözik az eredetiétől. Ebben az esetben egy képből lesz egy szemantikai maszk, amely minden pixelről eldönti, melyik osztályba tartozik.

