

10. Előadás

Neurális hálózatok

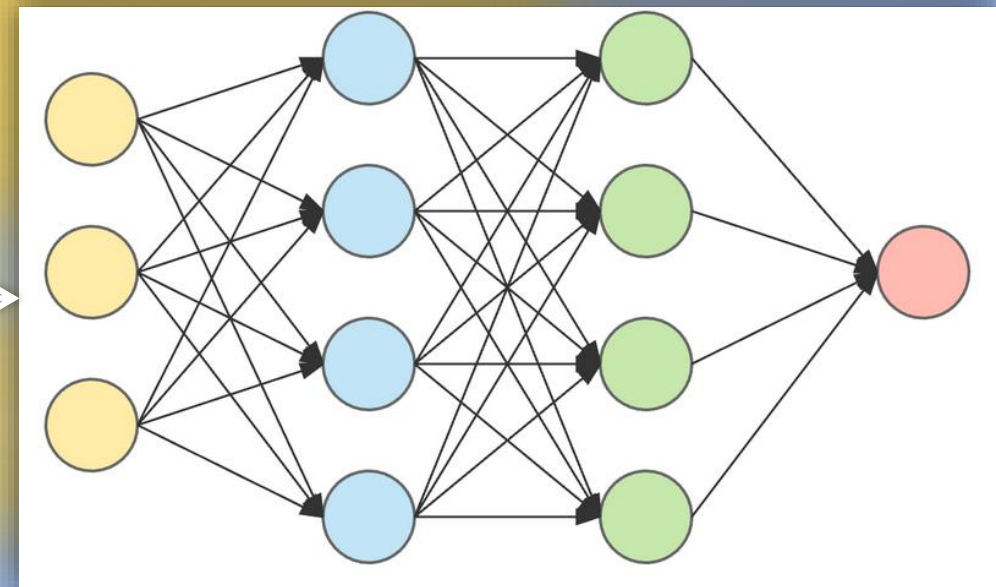
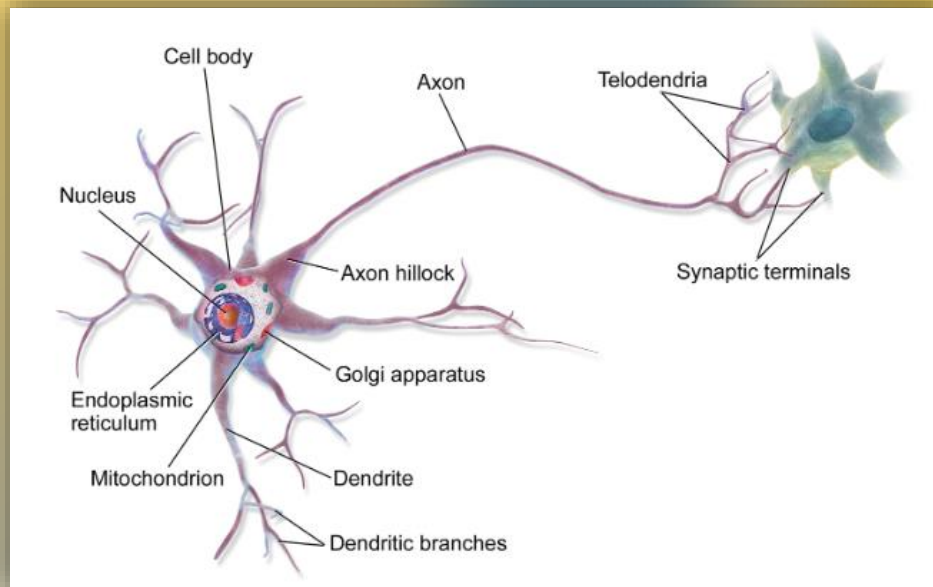
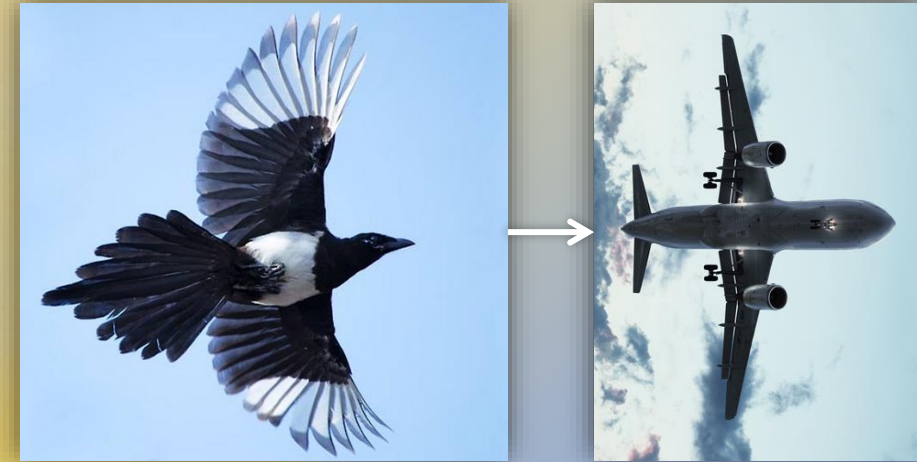
Deep Learning

Képfeldolgozás

A biológiai neurontól a mélytanulásig

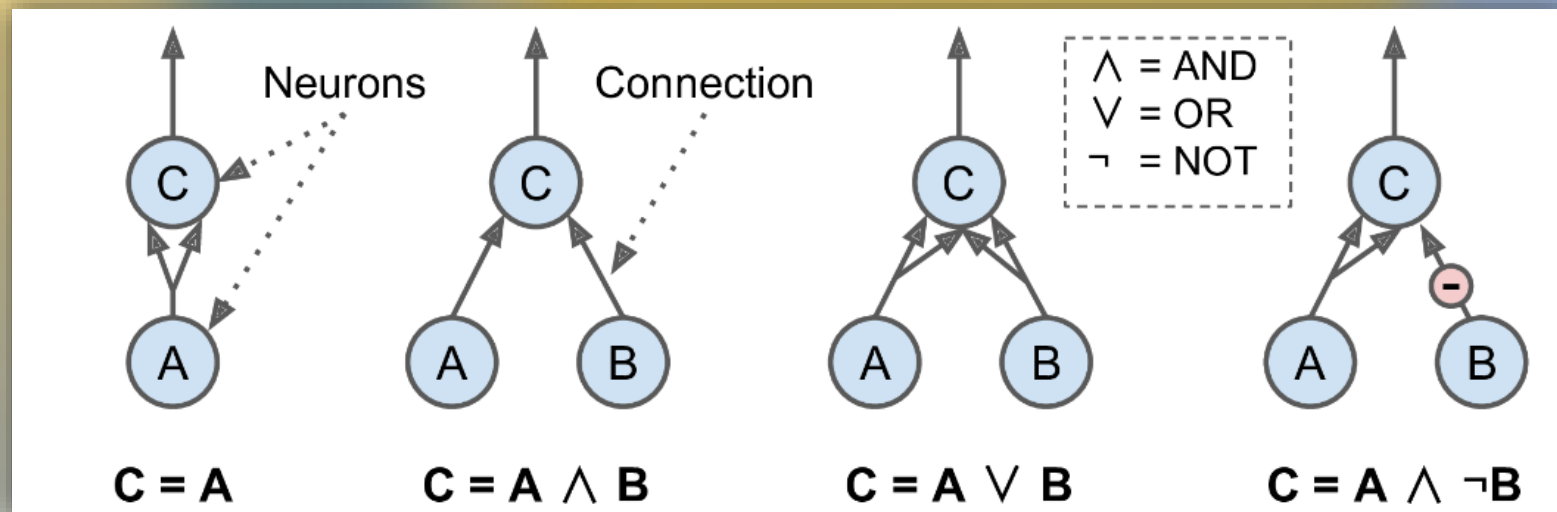
🐍 A madarak repülésre inspirálták az embert, a bogáncsok a tépőzárat ihlették. A logikus lépés az volt, hogy az emberi agy által inspirálódva megpróbálunk intelligens gépeket létrehozni.

🐍 Ahogy repülők sem csapkodnak a szárnyaikkal, a mesterséges neuronok is meglehetősen különböznek a biológiai unokatestvéreiktől.



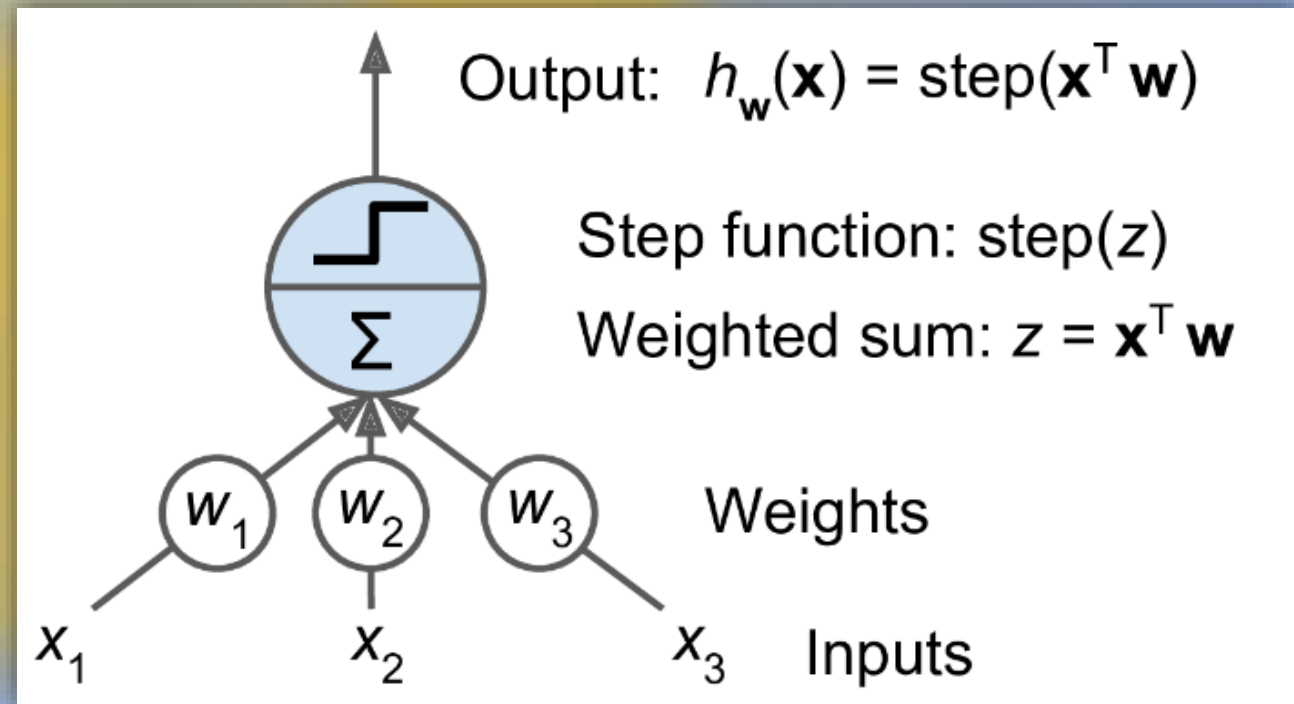
Logikai számítások neuronokkal

- 🐍 Warren McCulloch és Walter Pitts modellezték először a biológiai neuront, ami később a mesterséges neuronként vált ismertté.
- 🐍 Egy bináris kimenete, és több bináris bemenete van.
- 🐍 A mesterséges neuron akkor aktiválja az outputját, ha az inputjai meghatározott számban aktiválódnak.
- 🐍 Bármilyen összetett logikai kifejezés kifejezhető ilyen neuronokkal: lentebb az identitás, ÉS, VAGY, XVAGY kifejezések konfigurációi láthatók.



A perceptron

- 🐍 Az egyik legegyszerűbb ANN architektúra, Frank Rosenblatt nevéhez fűződik.
- 🐍 Alapja egy kissé különböző neuron: a **küszöblogikai egység** (TLU: Threshold Logical Unit).
- 🐍 Az inputok és outputok itt már valós számok. Minden inputhoz tartozik egy súly.
- 🐍 A TLU kiszámolja a bemenetek súlyozott szorzatösszegét:
$$z = w_1x_1 + w_2x_2 + \dots w_nx_n = \mathbf{X}^T \mathbf{W}.$$
- 🐍 Majd az eredményt behelyettesíti egy aktivációs függvénybe:
$$h_w(x) = \text{step}(z).$$
- 🐍 Egy perceptron képes elvégezni a lineáris szeparálás feladatát, amennyiben az adathalmaz erre alkalmas.



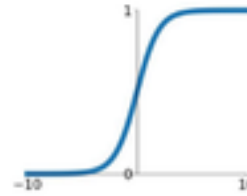
Aktivációs függvények

- 🐍 Leggyakoribbak az olyan aktivációs függvények, amelyek monoton növeők, jobbról folytonosak, határértékük a $-\infty$ -ben 0, a $+\infty$ -ben pedig 1.
- 🐍 Az ilyen φ tekinthető mint egy ξ valószínűségi változó $\varphi(x) = P(\xi \leq x)$ eloszlásfüggvénye. (Balról folytonosság esetén $\varphi(x) = P(\xi < x)$ lenne.)
- 🐍 Leggyakrabban az aktivációs függvények szigmoid, azaz S alakú függvények.
- 🐍 A TLU-k esetében ez a lépésfüggvény.
- 🐍 Komplex hálóknál esetén ReLu, Sigmoid stb...

Activation Functions

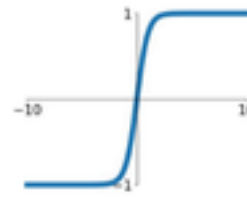
Sigmoid

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



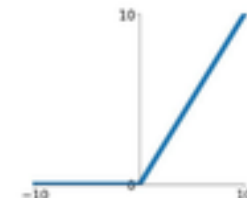
tanh

$$\tanh(x)$$



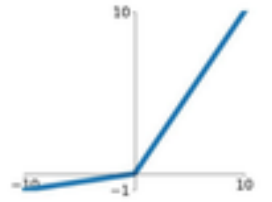
ReLU

$$\max(0, x)$$



Leaky ReLU

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

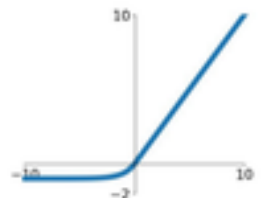


Maxout

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

ELU

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

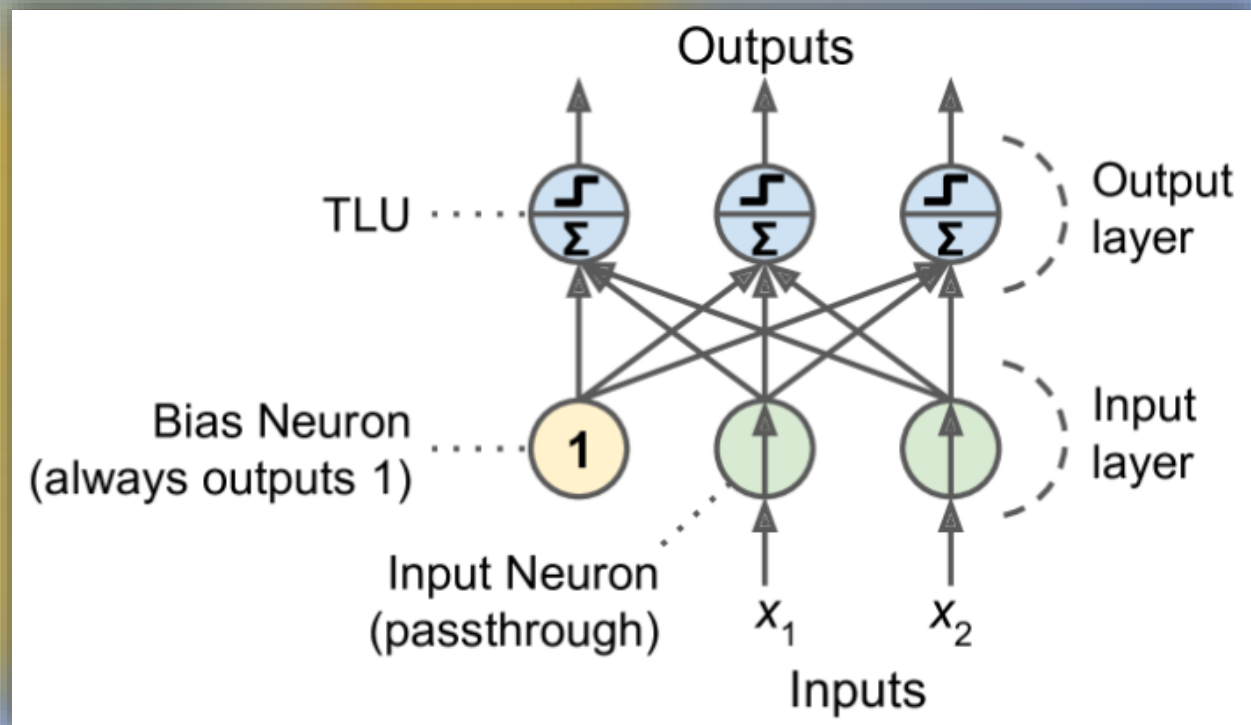


A többrétegű perceptron

- 🐍 A nagy áttörést az jelentette, amikor rájöttek, hogy a perceptronokat egymással összehangolva is lehet működtetni.
- 🐍 Ebben az esetben a TLU-k rétegekben foglalnak helyet, és a bemeneteik az előző réteg kimeneteivel állnak összeköttetésben.
- 🐍 A legelső réteg a bemeneti adathalmazzal áll összeköttetésben, minden bemeneti jellemzőhöz egy neuron tartozik.
- 🐍 Egy teljesen becsatolt réteg kimenete:

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

- 🐍 X : a tanuló adathalmaz mátrixa
- 🐍 W : a súlyok mátrixa
- 🐍 b : a torzításokat tartalmazó vektor
- 🐍 ϕ : az aktivációs függvény

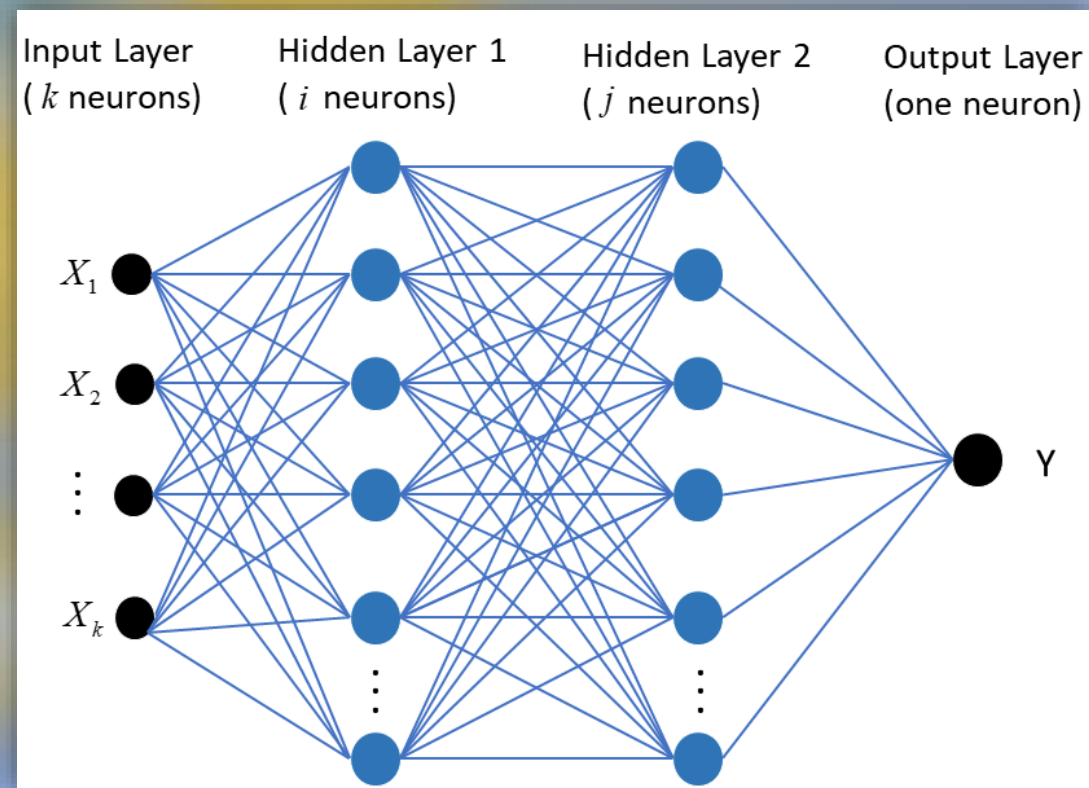


A regressziós architektúra

🐍 Egy többrétegű hálózat egy **input** réteget, tetszőleges számú, TLU-kból álló **rejtett réteget**, és egy szintén TLU-kból álló **output réteget** tartalmaz. Az output rétegen található TLU-k száma felhasználástól függ. Mikor egy neurális hálózat sok rejtett réteget tartalmaz, mély háló válik belőle (**DNN**).

🐍 Regressziós problémák esetén egy output neuron van, melynek kimenete a végleges becsült érték. Egy RNN tipikus hiperparaméter-konfigurációja:

Hyperparameter	Typical Value
# input neurons	One per input feature (e.g., $28 \times 28 = 784$ for MNIST)
# hidden layers	Depends on the problem. Typically 1 to 5.
# neurons per hidden layer	Depends on the problem. Typically 10 to 100.
# output neurons	1 per prediction dimension
Hidden activation	ReLU (or SELU, see Chapter 11)
Output activation	None or ReLU/Softplus (if positive outputs) or Logistic/Tanh (if bounded outputs)
Loss function	MSE or MAE/Huber (if outliers)



Az osztályozó architektúra

🐍 Ez egy kicsit különböző konfigurációt igényel. Minden lehetséges osztályhoz egyetlen output neuron tartozik.

🐍 Az aktivációs függvény a kimeneti rétegben a softmax (vagy log loss) függvény.

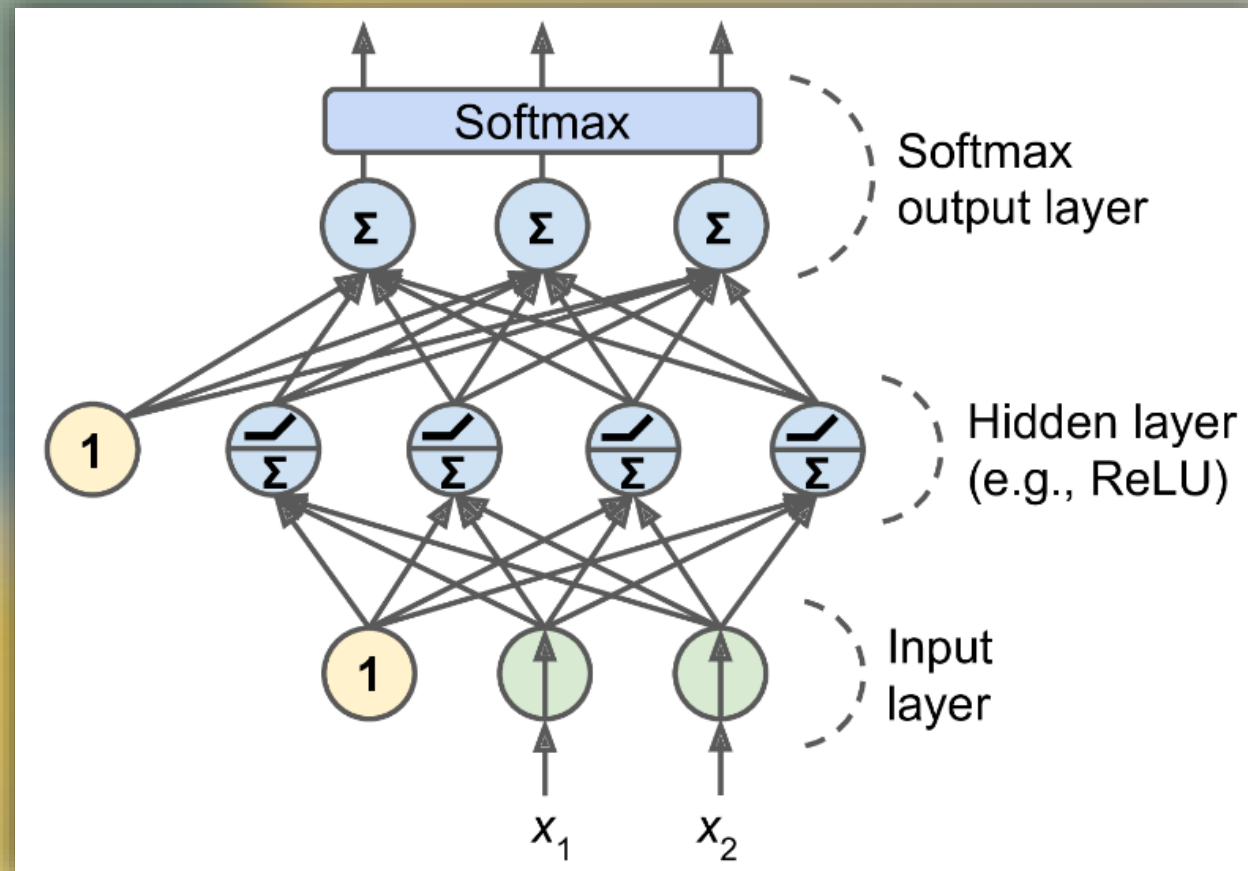
🐍 Ez biztosítja, hogy a becsült valószínűségek 0 és 1 közötti értékek legyenek, és az összes osztályhoz tartozó predikció összege 1 legyen.

🐍 Osztályozási fajták:

🐍 Bináris

🐍 Multilabel

🐍 Multiclass



Hyperparameter	Binary classification	Multilabel binary classification	Multiclass classification
Input and hidden layers	Same as regression	Same as regression	Same as regression
# output neurons	1	1 per label	1 per class
Output layer activation	Logistic	Logistic	Softmax
Loss function	Cross-Entropy	Cross-Entropy	Cross-Entropy

Nevesebb neurális hálózat topológiák

🐍 Bizonyos neurális hálózati architektúrák jobban teljesítenek adott feladatokon, mint a többi.

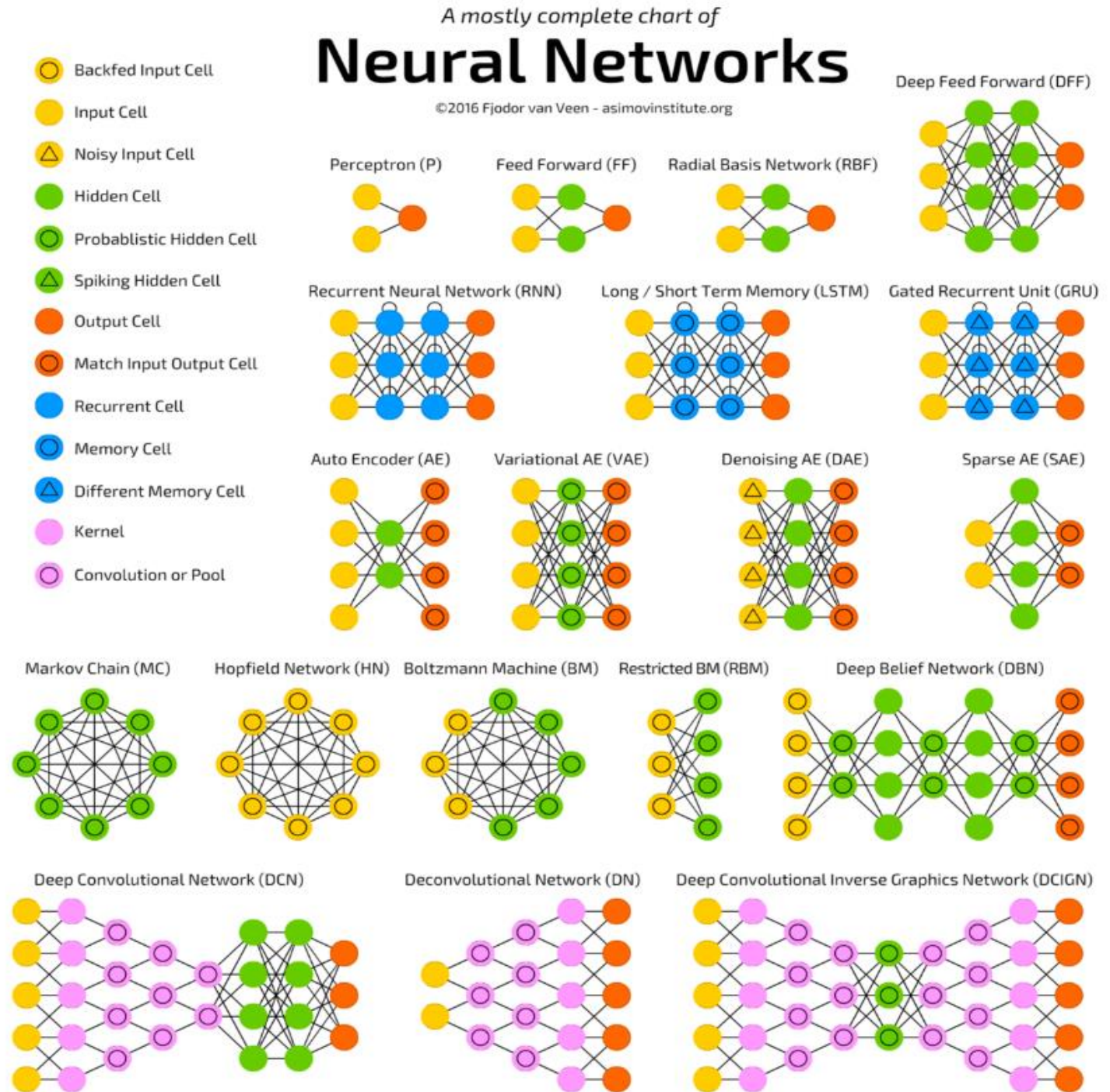
🐍 Ennek megfelelően az idő során többféle elrendeződés kialakult eszköz- és célfüggően.

🐍 Leghíresebb architektúrák:

🐍 VGGNET

🐍 RESNET50

🐍 GPT3

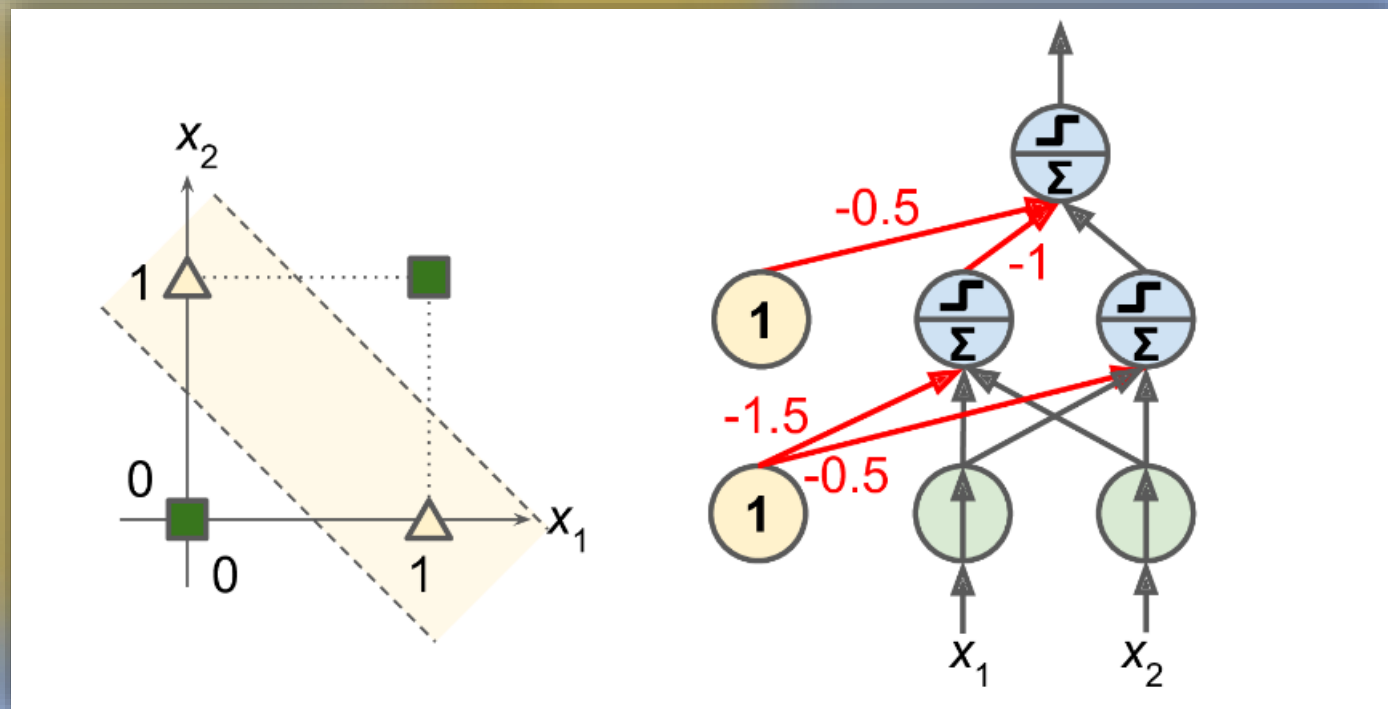


A perceptron tanítása

- A tanítás során egyszerre egy mintaegyed áramlik át a hálózaton. A modell a mintaegyedre készít egy predikciót.
- Minden output neuronnak, ami rossz predikciót ad, megerősíti a súlyát abból a neuronból, ami a jó predikciót generálta.
- A tanítás szabálya:

$$w_{i,j}^{(\text{next step})} = w_{i,j} + \eta(y_j - \hat{y}_j)x_i$$

- $w_{i,j}$: a kapcsolati súly az i-edik input neuron és a j-edik output neuron között
- x_i : az aktuális mintaegyed i-edik input értéke
- \hat{y}_j : a j-edik output neuron aktuális mintaegyedre adott predikciója
- y_j : a j-edik output neuron valós cél output értéke az aktuális egyedre
- η : a tanulási sebesség



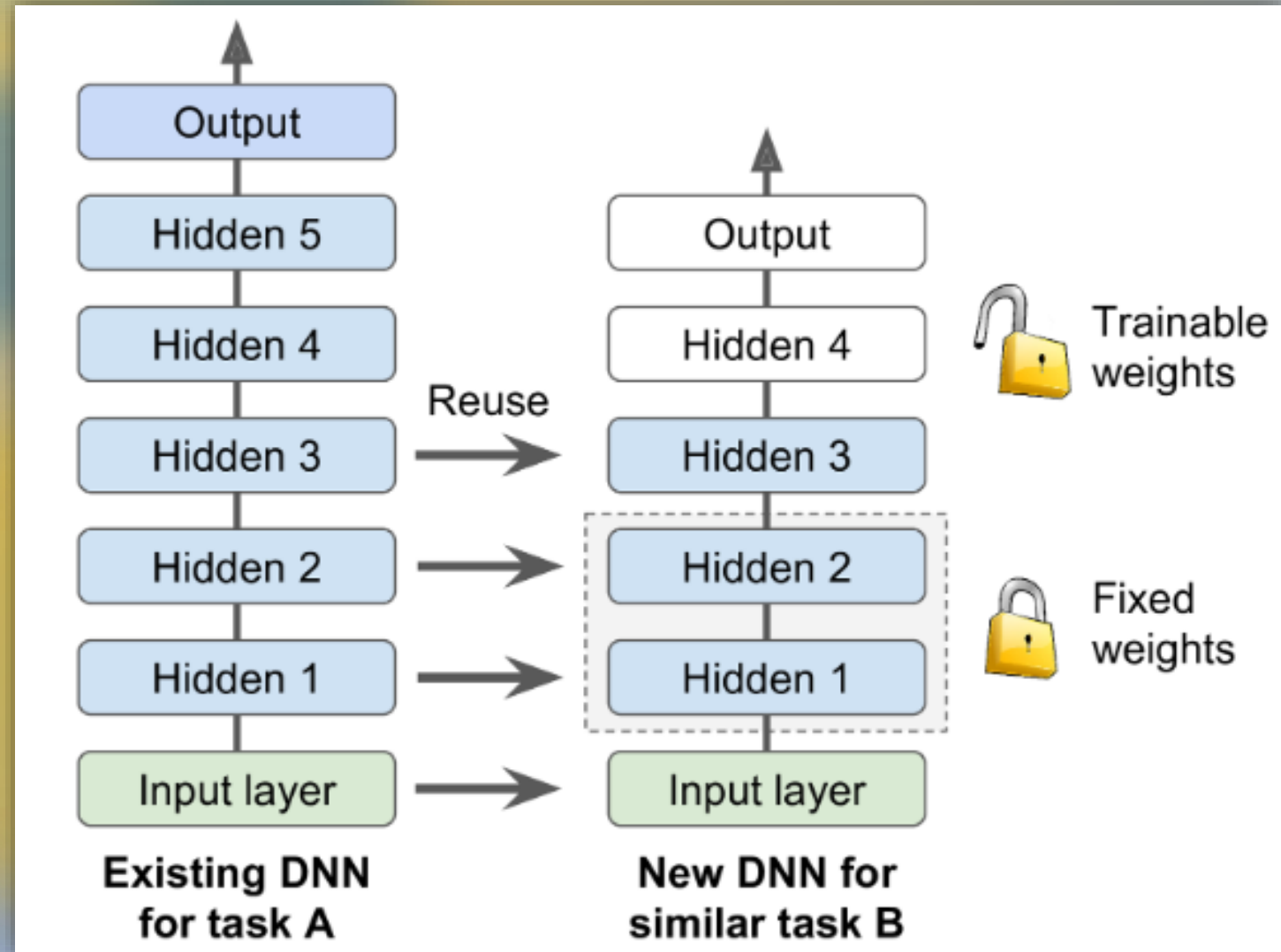
Transzfertanulás

🐍 Általában nem jó ötlet egy komplex mélyhálózatot a nulláról tanítani.


🐍 Ehelyett hasznos lehet egy olyan hálózat megkeresése, ami egy hasonló feladatot végez el, mint amit szeretnénk, majd ezeket a komponenseket felhasználni.

🐍 Minél hasonlóbbak a feladatok, annál több réteget fel lehet használni az előretanítottak közül.

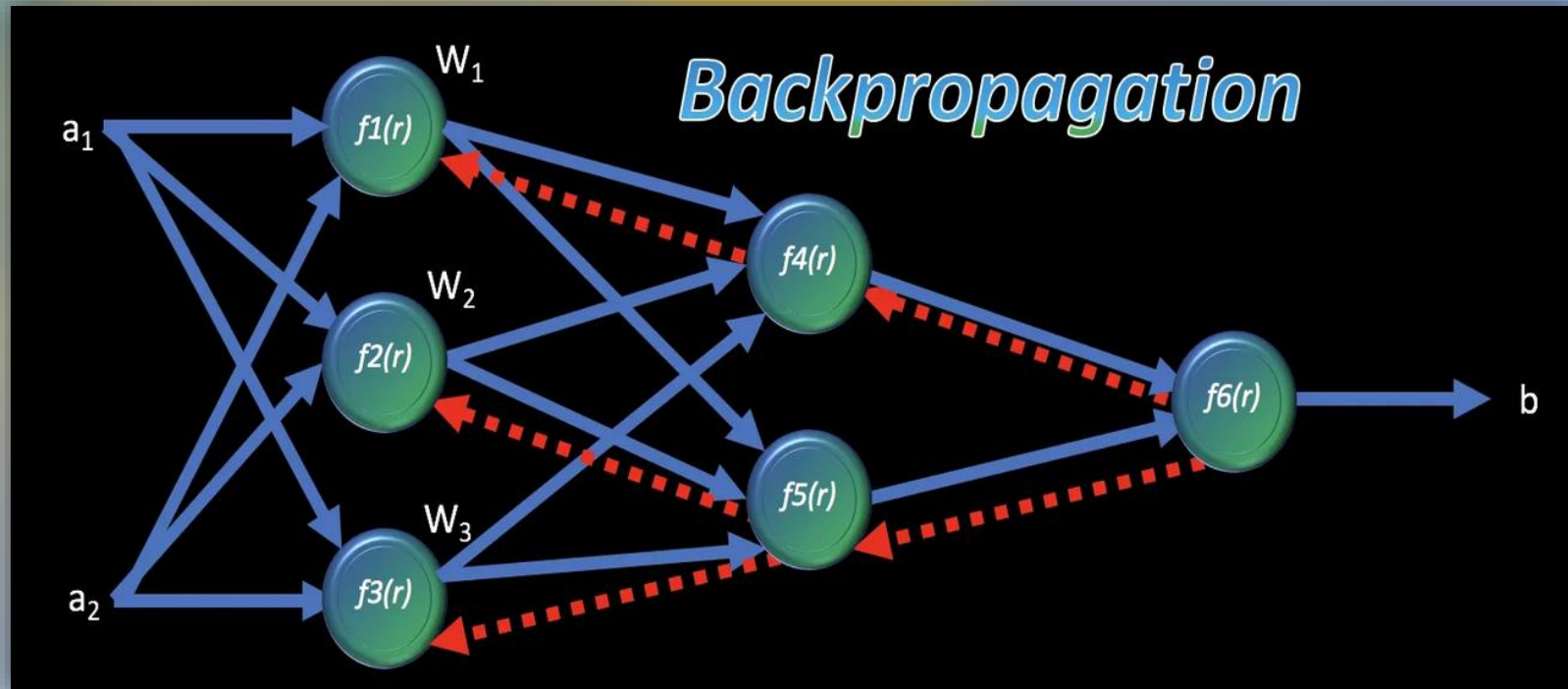
🐍 Ha a forma (méret, dimenzió...) amit a rétegek elvárnak nem ugyanolyan mindkét esetben, előfeldolgozásra van szükség.



A hiba visszaáramoltatása a hálózatba

 A hiba-visszaterjesztő algoritmus egy hatékony módszert ad a gradiensek automatikus kiszámítására. Két iteráció alatt (egy előre, egy hátra) minden paraméterhez tartozóan ki tudja számolni a hálózat hibáját:

1. Egy mini-köteget kezel egyszerre, és többször megy át a tanító pontokon (**epoch**).
2. Minden mini-köteget átenged a hálózaton, és közben minden réteg predikcióját elmenti, hogy felhasználja a visszaáramoltatás közben (**forward pass**).
3. Megméri a hálózat output hibáját.
4. Kiszámolja, hogy az egyes kimeneti kapcsolatok mennyiben járultak hozzá a hibához.
5. Kiszámolja, hogy a kimeneti kapcsolatok hibái mennyiben származtathatók az előző kapcsolat hibájából (**backward pass**)
6. Gradiens ereszkedéssel kiszámolja a kapcsolatok súlyait.



Tanulási sebesség és optimalizálók

A frissítések mérete a tanulási sebességtől és az optimalizálótól függ. Az optimális tanulási sebességre az epoch szerinti költségből lehet következtetni.

A paraméterek frissítéseinek megtalálására több eljárást is kifejlesztettek. Ezek közül mindegyiknek egy célja van: közelíteni a minimum helyet. A fejlettebbek a kötegmérettel is számolnak (**kötegnormalizálás / batch normalization**).

A különbség csak abban áll, hogyan viselkednek eltérő típusú terekben:

[Gradiens ereszkedés](#)

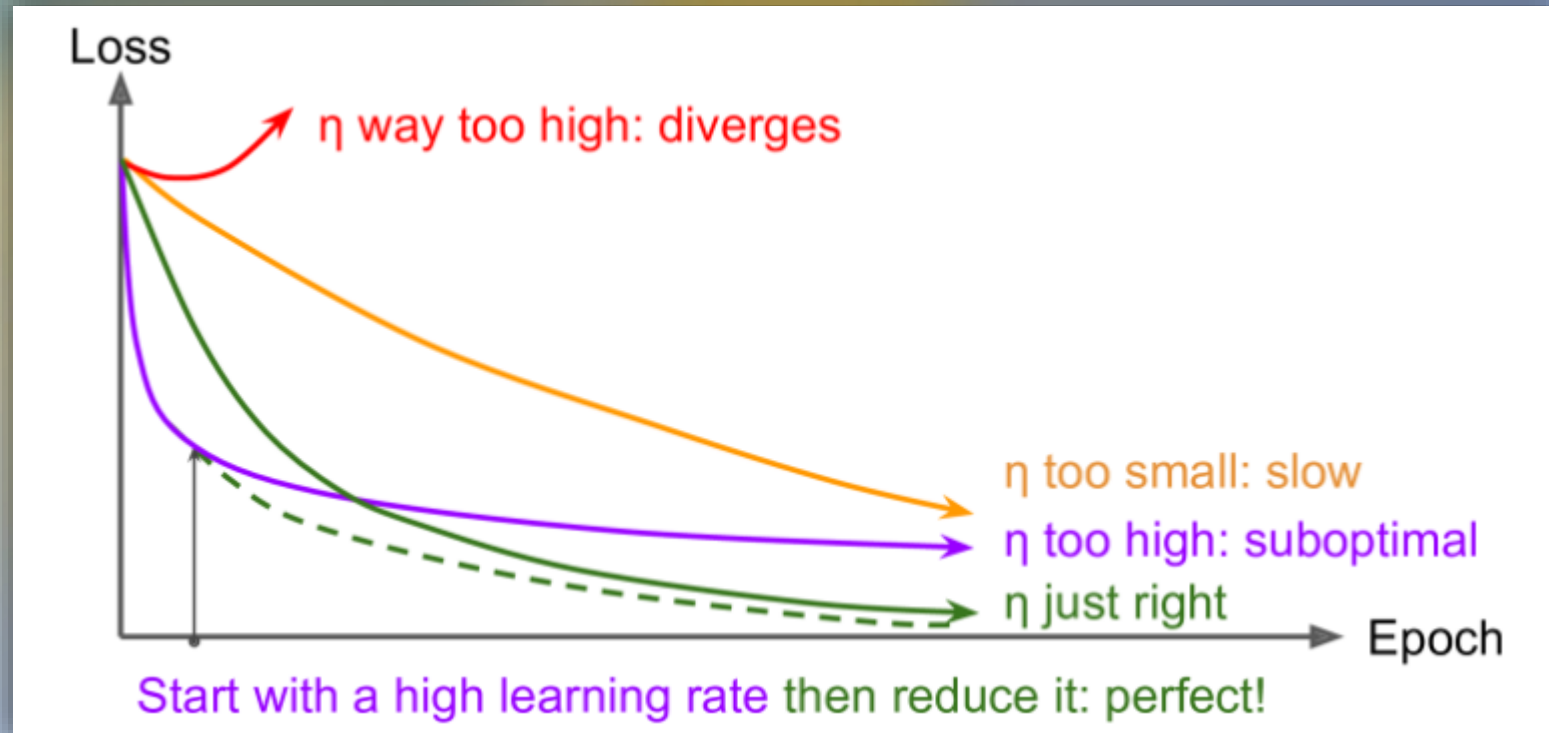
[Momentum](#)

[AdaGrad](#)

[RMSProp](#)

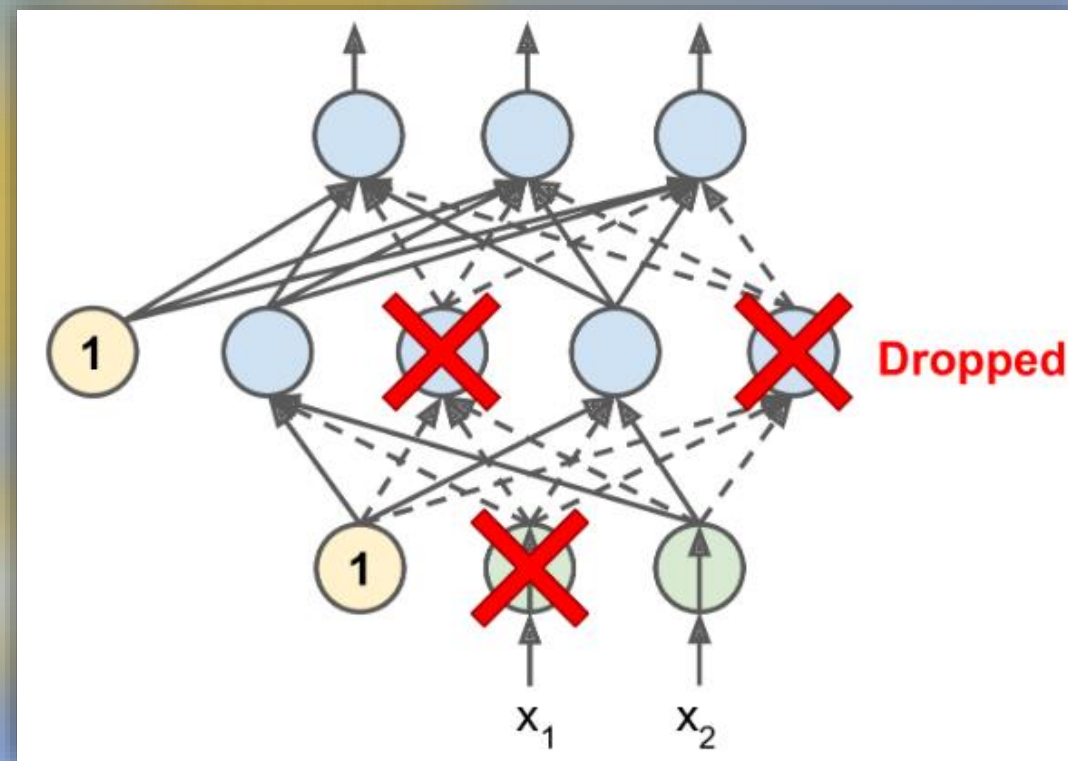
[AdaDelta](#) és a többiek

[Másik](#) vizualizáció



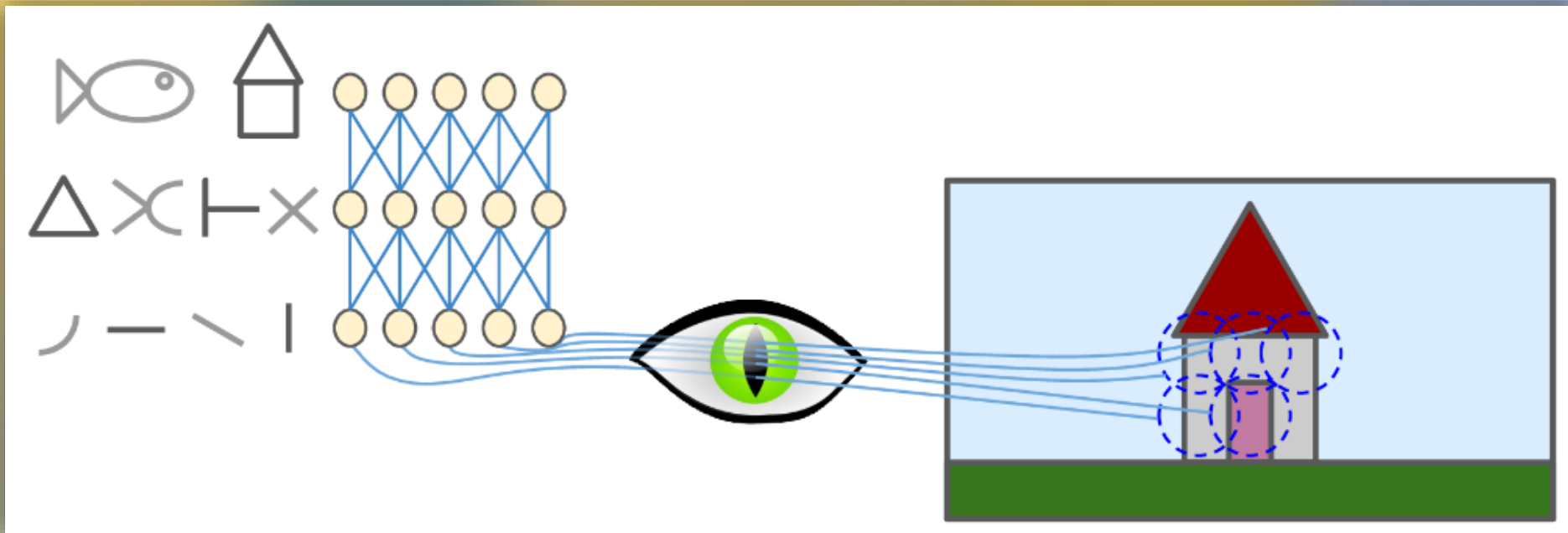
Regularizáció neurális hálók esetén

- 🐍 A neurális hálózatok is fogékonyak a túltanulásra, ha túl magas a szabadságfok.
- 🐍 A paraméterek optimalizálásában fel lehet használni az ℓ_1 és ℓ_2 normát. Ez teljesen úgy működik, mint a Lasso vagy Ridge regresszió esetén.
- 🐍 Egy jobb regularizációs módszer a **dropout**: minden tanító lépésben minden neuron (beleértve az inputot, de sosem az outputot) kap p valószínűséget arra, hogy ne legyen figyelembe véve a tanítás során.
- 🐍 Ez a p hiperparaméter a **dropout rate** (kiesési ráta), általában 50%.
- 🐍 Vegyük észre, hogy minden tanítási lépésben egyedi neurális háló architektúrát tanítunk. Ez összesen 2^N lehetőség!



A biológiai látás alapjai

- David H. Hubel és Torsten Wiesel macskákon mutatták meg, hogy a vizuális cortexben sok neuronnak van egy kis helyi befogadó mezője, azaz csak a vizuális térnek egy bizonyos szegmensén elhelyezett ingerületet képesek befogadni.
- Vannak olyanok, amelyek csak vertikális, vagy horizontális alakokat fogadnak be.
- A neuronok különböző befogadó mezői átfedésben állhatnak egymással, és együtt alakítják ki a vizuális teret.



A konvolúciós réteg

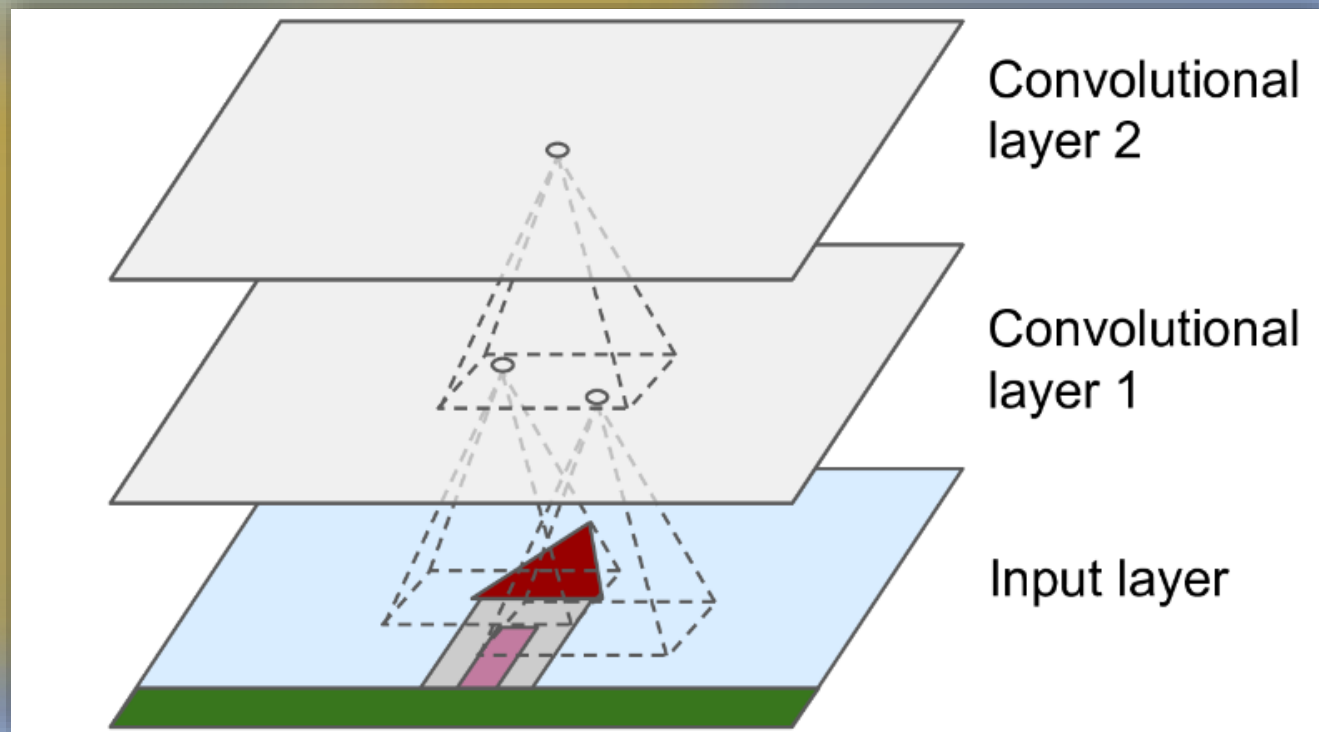
🐍 A CNN legfontosabb építő eleme a **konvolúciós réteg**.

🐍 Az első konvolúciós rétegben lévő neuronok nincsenek összekötve a bemeneti kép minden pixelével, csak azokkal, amik az ő befogadó mezőikben vannak.

🐍 A második konvolúciós rétegben lévő neuronok csak a saját téglalapjukon belüli neuronokkal vannak összeköttetésben (az eggyel lentebbi rétegben).

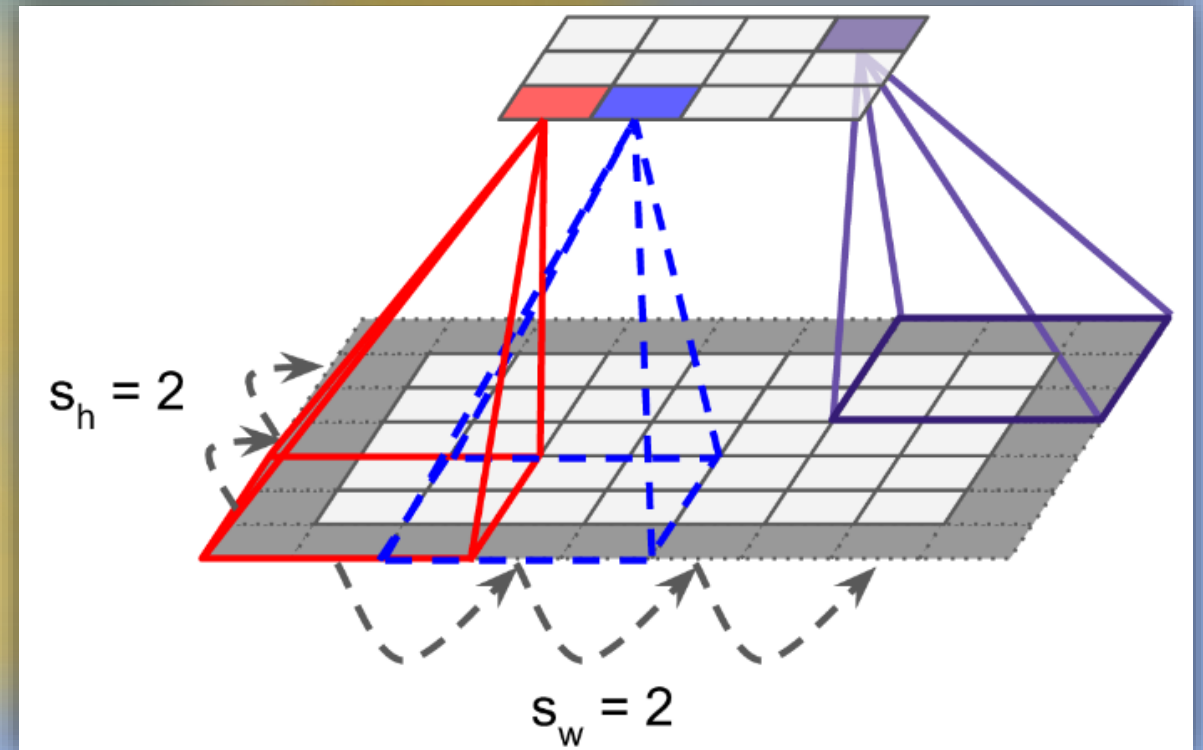
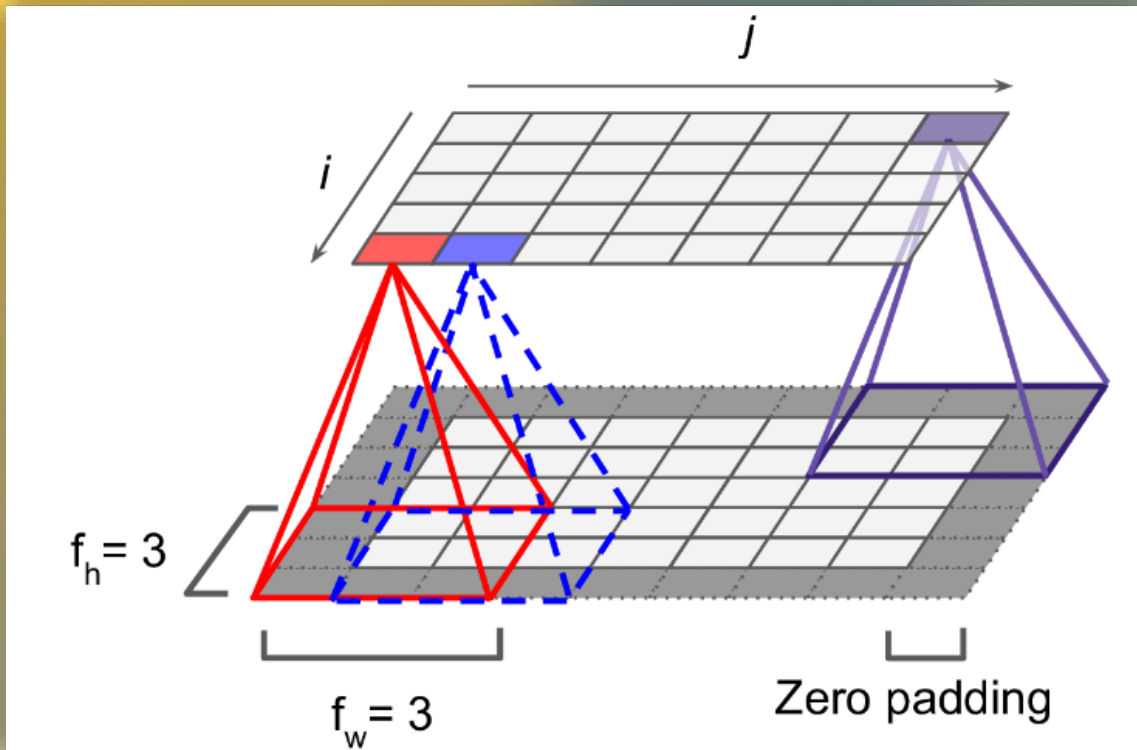
🐍 Ez az architektúra lehetővé teszi a kicsi, alacsony szintű jellemzők leképezését az első, majd a magasabb szintű jellemzők kivonatolását a második rétegben.

🐍 Ez a hierarchikus struktúra gyakori a való életben, ezért is működik olyan jól a CNN képfelismerésre.



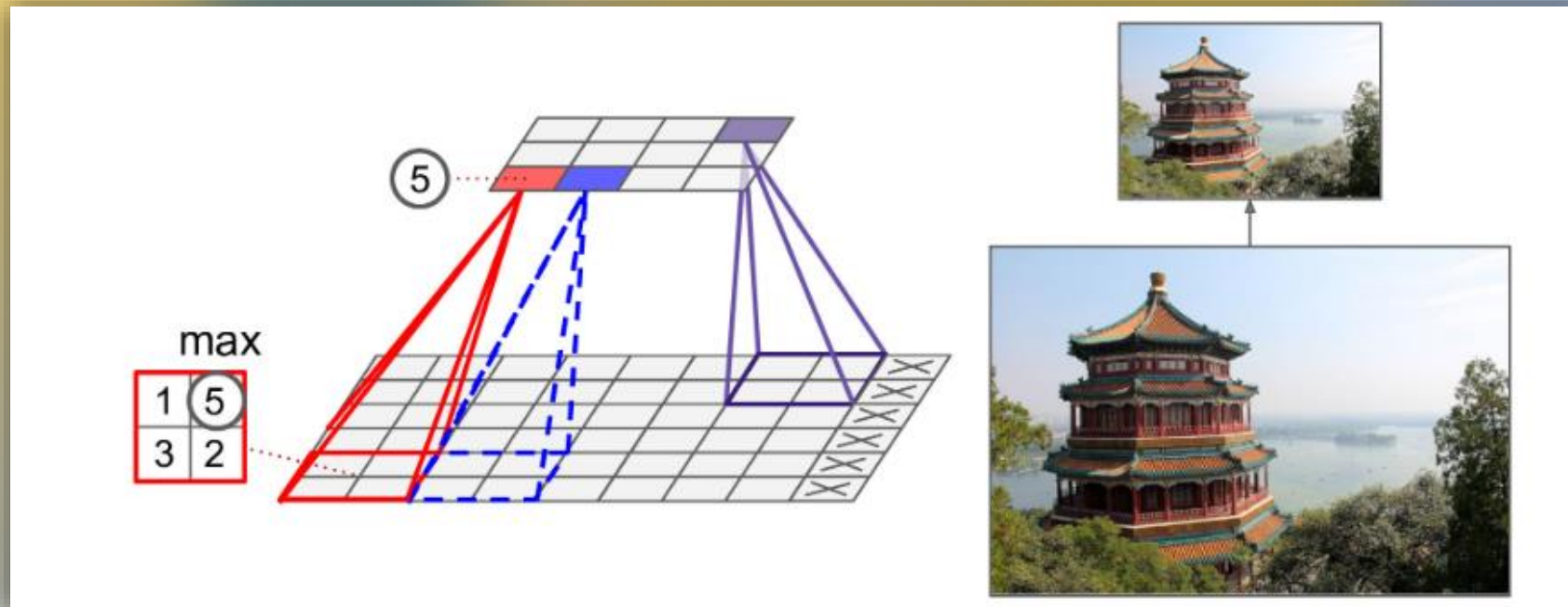
Konvolúciós rétegek kapcsolatai

- 🐍 Az i -edik sorban és a j -edik oszlopban lévő neuron az előző réteg i és $i + f_h - 1$ sorai, illetve j és $j + f_w - 1$ neuronok által határolt mező neuronjaival áll összeösszeköttetésben.
- 🐍 Gyakran 0 értékekkel körbeveszik a képet, ez a **zero-padding** (bal).
- 🐍 A befogadó mezők kihúzásával kisebb kimeneti kép is elérhető (jobb).



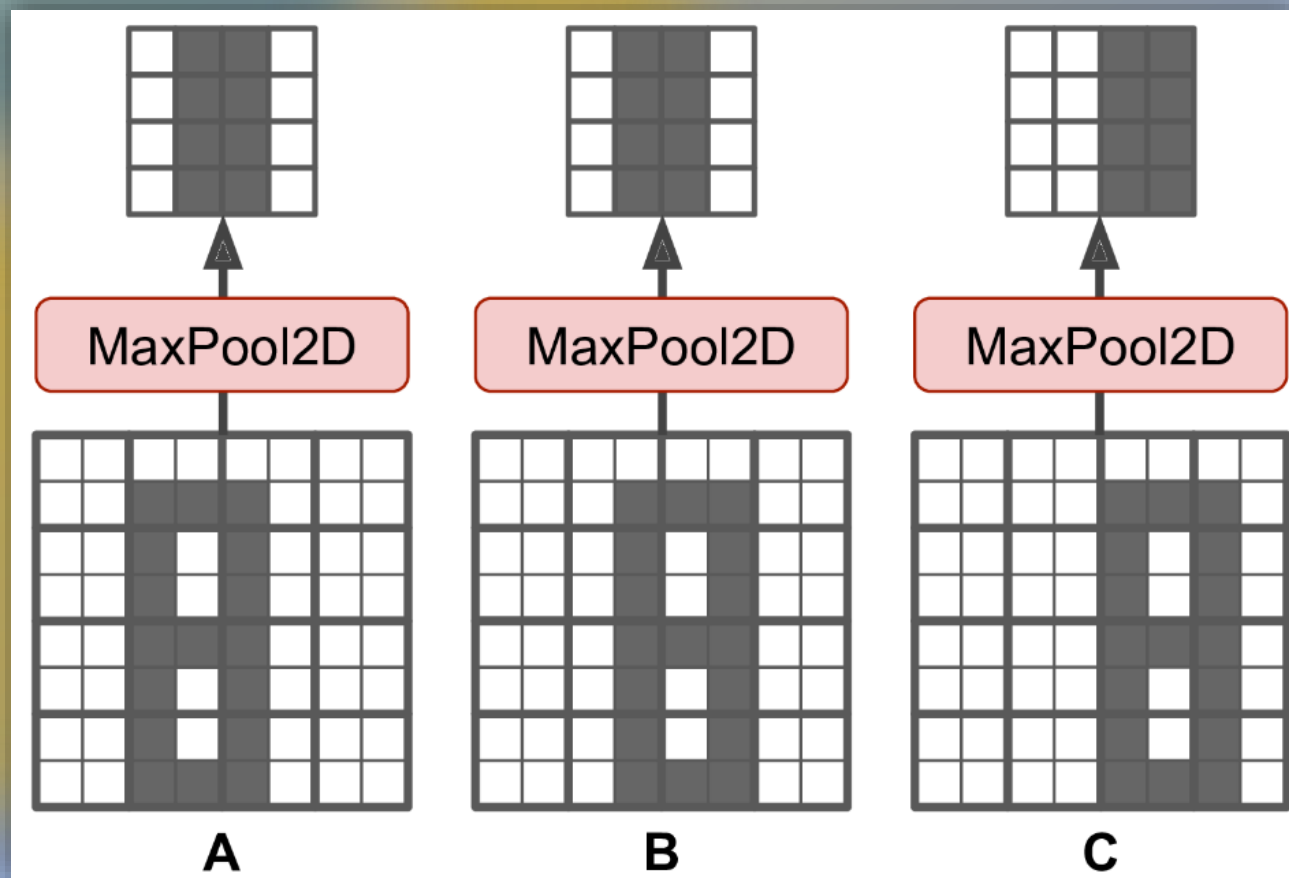
A pooling réteg

- 🐍 A pooling rétegek feladata, hogy almintázzák (csökkentsék) a bemeneti képet annak érdekében, hogy kisebb legyen a memória igény és a paraméterek száma.
- 🐍 Csakúgy, mint a konvolúció esetén, a neuronok egy befogadó mezőhöz vannak csatolva, viszont nincsenek súlyaik.
- 🐍 Az egyetlen amit csinál egy aggregáló művelet: kiválasztja a befogadó mezőjének max, min, átlag értékeit. A képen egy 2×2 MaxPooling réteget látunk.



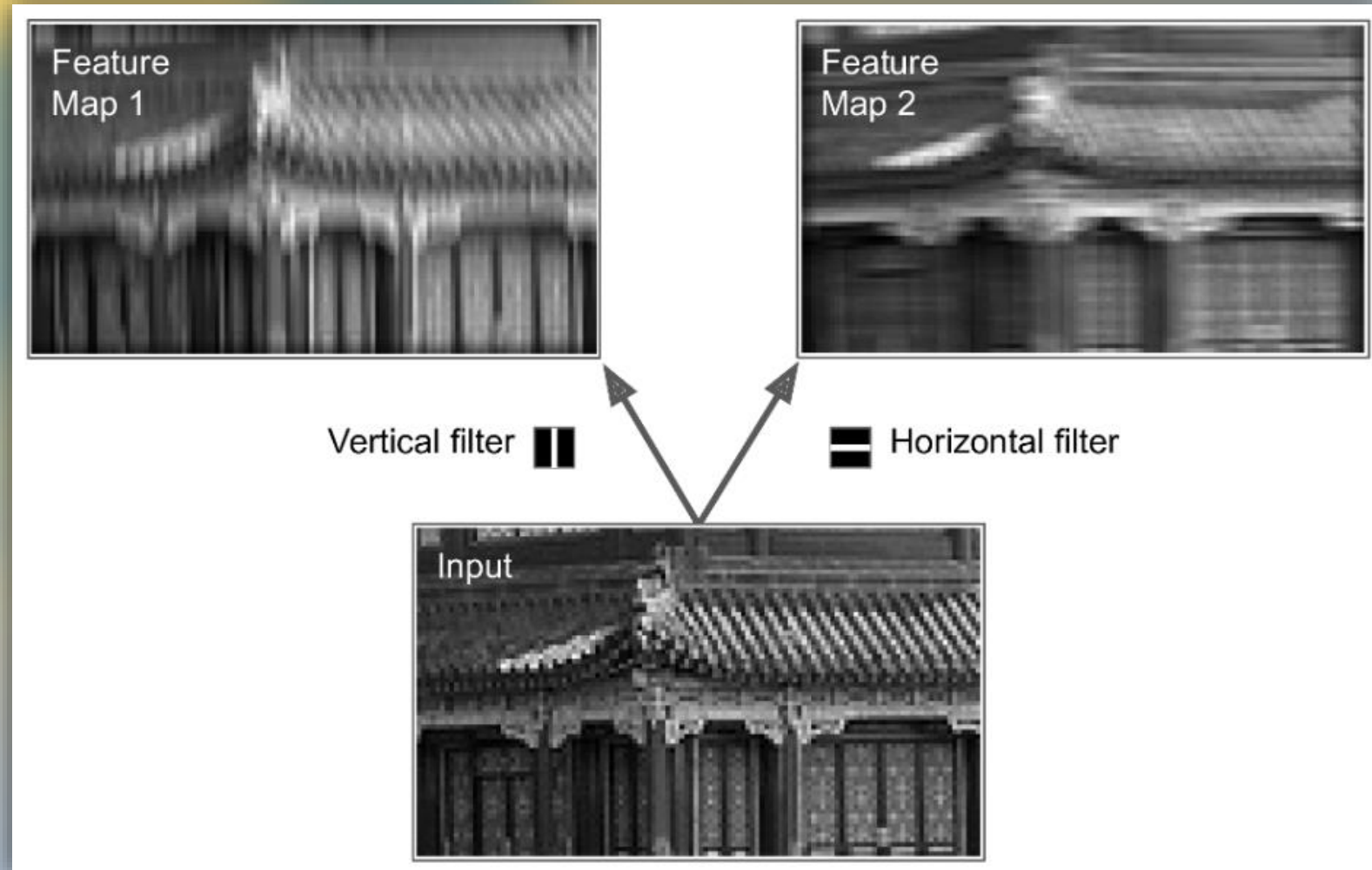
Pooling gyakorlati alkalmazásban

- A memória- és paraméterszám csökkentése mellett a pooling bevezet egy alacsony fokú invarianciát kisebb fordításokkal szemben.
- Itt a három képet áteresztettük egy MaxPooling rétegen. A , B és a C kép ugyanaz mint az A , csak 1, illetve 2 pixellel jobbra eltolva.
- A MaxPooling eredménye az A és B képek esetén megegyezik: ez az **eltolási invariancia** a gyakorlatban.
- Ez kép osztályozásnál nagyon hasznos tud lenni, viszont vannak olyan alkalmazási területek, ahol az invariancia nem kívánatos, mint a szemantikus szegmentáció. Ekkor ekvivarianciára törekszünk. Miért?



Szűrők

- A neuron súlyai reprezentálhatók egy kis képpel, aminek a mérete megegyezik befogadó mezőjével.
- A képen két különböző súlyhalmazzt (filtert) látunk.
- Az első egy fekete négyzet, benne egy függőleges fehér vonallal a közepén.
- Azok a neuronok, amelyek ezt a szűrőt használják, minden súlyt figyelmen kívül fognak hagyni, kivéve a középső vonalon lévőket.



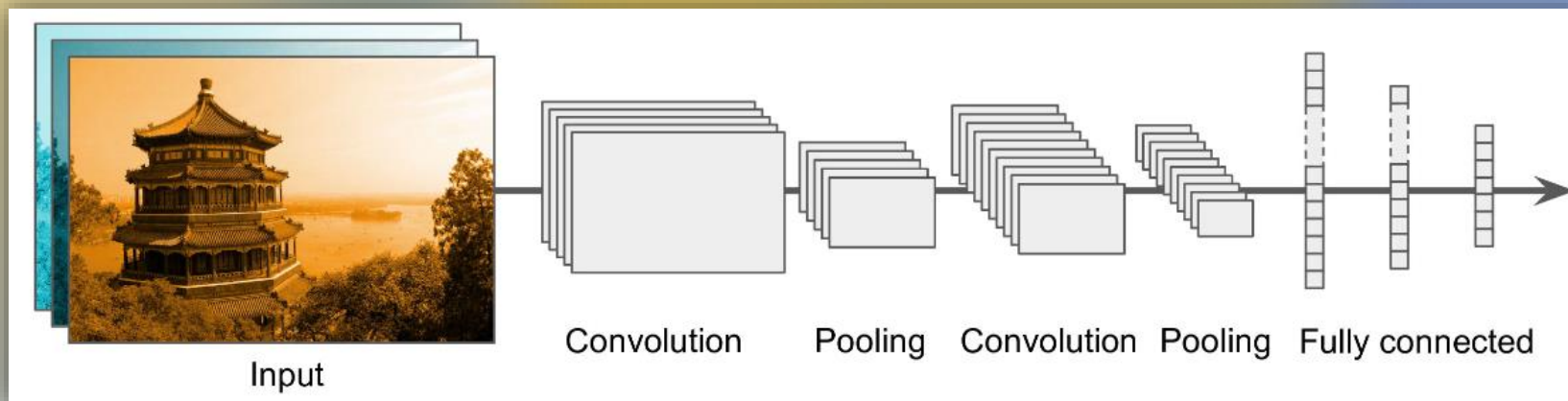
Képfeldolgozó architektúrák felépítése

- 🐍 A tipikus CNN néhány konvolúciós réteget használ, majd pooling, majd megint konvolúció és így tovább.
- 🐍 A kép, ahogy halad előre a hálóban egyre kisebb és absztraktabb lesz.
- 🐍 Ez a CNN 92% pontosságot ért el az MNIST Fashion dataseten.

```
from functools import partial

DefaultConv2D = partial(keras.layers.Conv2D,
                        kernel_size=3, activation='relu', padding="SAME")

model = keras.models.Sequential([
    DefaultConv2D(filters=64, kernel_size=7, input_shape=[28, 28, 1]),
    keras.layers.MaxPooling2D(pool_size=2),
    DefaultConv2D(filters=128),
    DefaultConv2D(filters=128),
    keras.layers.MaxPooling2D(pool_size=2),
    DefaultConv2D(filters=256),
    DefaultConv2D(filters=256),
    keras.layers.MaxPooling2D(pool_size=2),
    keras.layers.Flatten(),
    keras.layers.Dense(units=128, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(units=64, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(units=10, activation='softmax'),
])
```

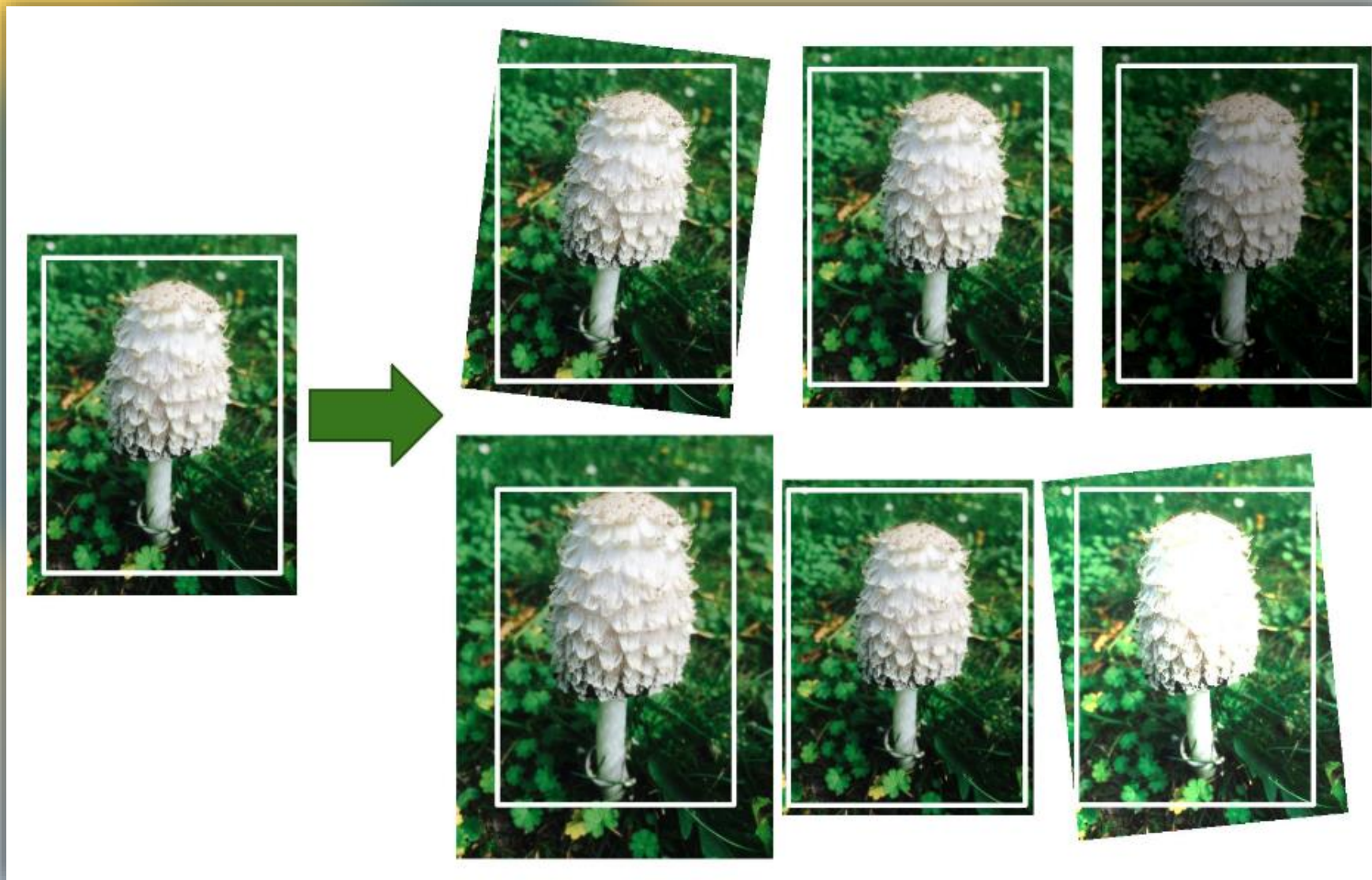


Adataugmentáció

🐍 Új tanító egyedek létrehozása a meglévőkön végzett apróbb módosítások segítségével:

- 🐍 eltolás
- 🐍 forgatás
- 🐍 sötétítés
- 🐍 döntés
- 🐍 világosítás.

🐍 Eredménye egy nagyobb tanító halmaz.



Kapcsolódó tartalmak neurális hálókról

 3Blue1Brown Neurális háló sorozata:

 [Első rész](#)

 [Második rész](#)

 [Harmadik rész](#)

 [Negyedik Rész](#)

 [Statquest Neurális háló sorozata](#) (13 rész)

 Hands-on Machine Learning (Aurélien Geron)