TOVÁBBI TANÍTÁSI MÓDSZEREK

A tananyag az EFOP-3.5.1-16-2017-00004 pályázat támogatásával készült.

SZÉCHENYI 2020
MAGYARORSZÁG KORMÁNYA
Európai Unió
Európai Szociális Alap
BEFEKTETÉS A JÖVŐBE

Backpropagation és alternatívái

- Messze a legnépszerűbb, legelterjedtebb tanító algoritmus
  - Egyszerű alapelv, könnyű implementálni
  - Csak a derivált kell hozzá
  - Batch módban is használható, így nem muszáj az összes tanító adatot egyszerre betölteni, manipulálni
- Az alap backpropagation algoritmusnak van több finomítása
  - Alapvetően a derivált és a learning rate számítását próbálják finomítani
  - Régebben: Momentum módszer, Quickprop, resilient backprop, …
  - Újabban: Adagrad, Adadelta, Adam, …
- Fejlettebb optimalizálási technikák – mit várunk tőlük?
  - Globális optimum – ez nem fog menni…
  - Jobb lokális optimum
  - Gyorsabb konvergencia
  - Kevesebb kézileg állítandó meta-paraméter (pl. learning rate és felezése)

A momentum módszer

- Emlékezzünk vissza, hogy hogyan változtattuk a súlyokat:

$$\Delta w_{ij} = \eta * \frac{\partial E}{\partial w_{ij}}$$

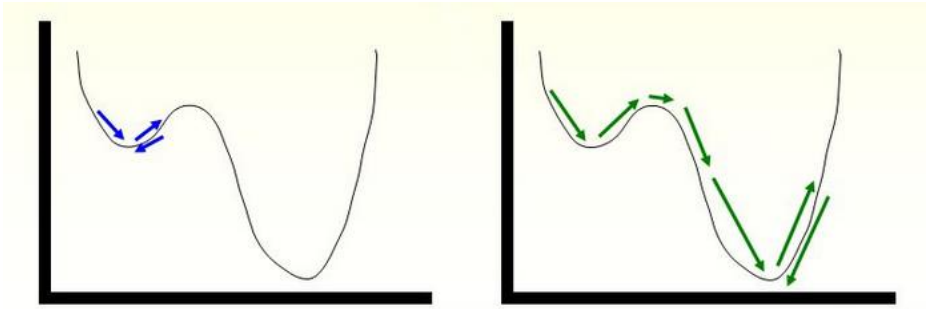
weight increment — learning rate — weight gradient

- Most ezt kiegészítjük egy „momentum” taggal:
  - Ahol a „momentum faktor” egy 0 és 1 közötti érték

$$\Delta w_{ij} = \eta * \frac{\partial E}{\partial w_{ij}} + (\gamma * \Delta w_{ij}^{t-1})$$

momentum factor — weight increment, previous iteration

- Ennek az a hatása, hogy időben elsimítja a Δ értéket, „lendületet” ad neki
- Átsegíthet kisebb lokális optimumokon



Adagrad

- Minden egyes paraméterhez *külön* learning rate
- A learning rate-et leosztja az eddigi deriváltak négyzetes átlagával
  - A nagyobb deriváltakat produkáló paraméterekhez kisebb learning rate
  - A kisebb deriváltúakhoz nagyobb learning rate fog tartozni
- Előnyök:
  - Kiegyensúlyozza az egyes paraméterek Δ értékének nagyságát
  - Nem kell menet közben állítani a learning rate-et, magát szabályozza
- Hátrány:
  - Mivel az összes korábbi deriváltat összeadja, a learning rate túl gyorsan elfogy
  - Gyakori tapasztalat, hogy lassabban konvergál, illetve kicsit rosszabb eredményt ad, mint egy jól beállított backprop momentummal

Adadelta, Adam, …

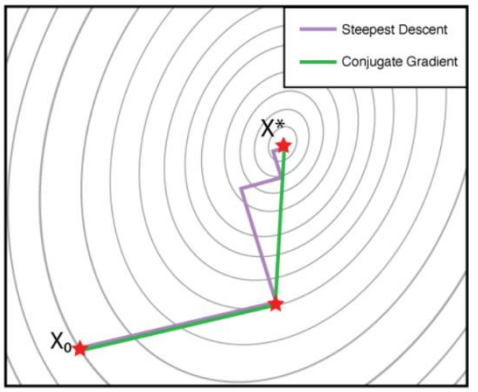
- Adadelta: az összegzésnél csak a k legutóbbi deltát, vagy az összeset, de exponenciálisan csökkenő súllyal veszi figyelembe
  - Ezzel kiküszöböli az Adagrad túl gyorsan elfogyó learning rate-jét
- Adam:
  - Szintén futó átlagot számol, de nem csak az átlagot, hanem a szórást is nézi
- Vannak még újabb finomítások is:
  - AdaMax, Nadam, Eve, …

Kifinomultabb optimalizálási módszerek

- Newton-módszer és rokonai
  - Kell hozzá a második derivált is → sokváltozós esetben a Hesse-mátrix
    - n változó estén ez *n*x*n*-es
    - Neuronhálók esetén ez túl nagy
    - Én invertálni is kellene minden iterációs lépésben…

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

- Kvázi Newton-módszerek – csak közelítik a Hesse-mátrixot
  - Pl. L-BFGS
- Konjugált gradiens módszerek – szintén elkerülik a Hesse-mátrix kiszámítását és invertálását
  - Adott irányban line search + mindig az előző irányokra merőlegesen lép
  - n-dimenzós kvadratikus felület optimumát n lépésben megtalálja
  - Nem kvadratikus felület – a fentit kell ismételgetni

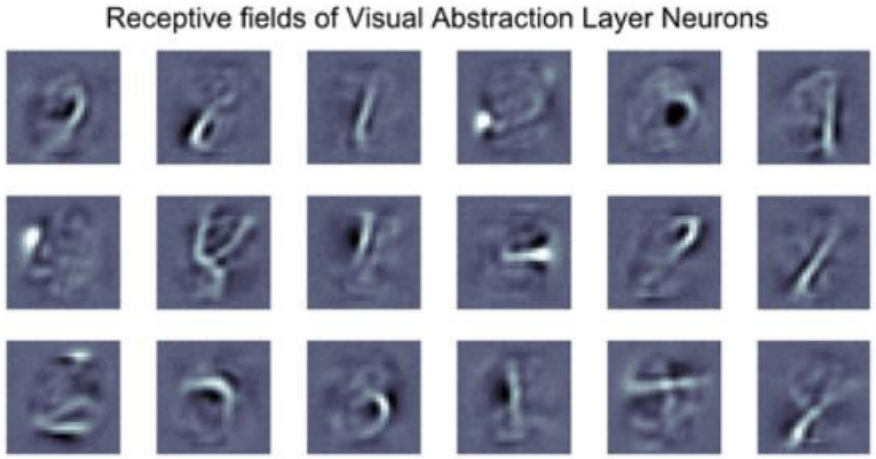


Kifinomultabb optimalizálási módszerek

- Összehasonlító kísérletekben általában a backpropagation hasonlóan, vagy csak kicsit rosszabbul teljesít
  - Érdemes lehet a két módszercsaládot kombinálni, pl. a tanulás elején a hiba gyorsan csökkenthető backpropagationnal, a végén pedig be lehet vetni a konjugált gradienst, vagy a kvázi-Newton módszereket
- Általános vélemény jelenleg:
  - Annyival bonyolultabbak matematikailag és implementációs szempontból, hogy nem éri meg az a kis javulás/gyorsulás, mit bizonyos feladatokban adnak

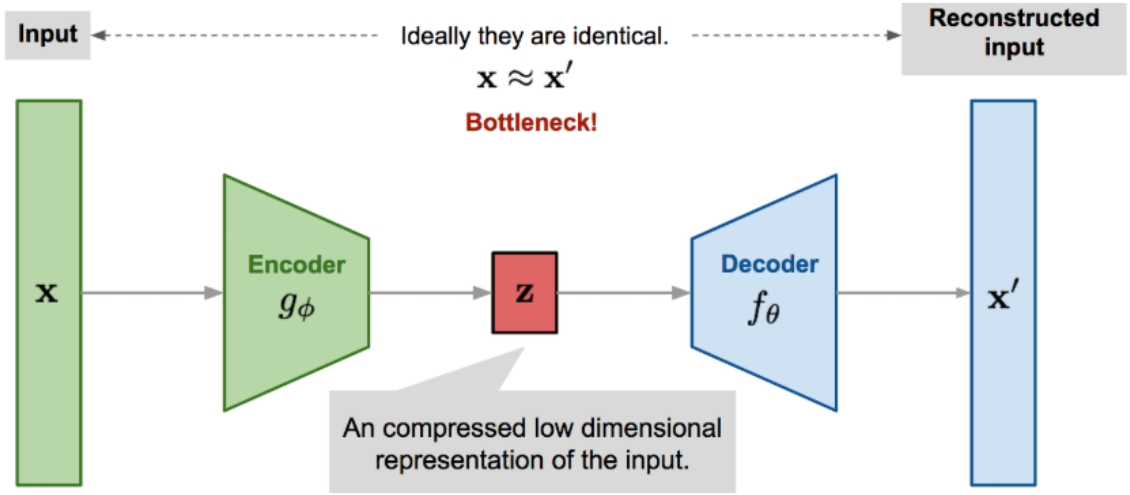
DBN-ek jelenlegi alkalmazása

- A DBN-ek CD-kritériummal való tanítása *felügyelet nélküli* tanítás
  - Továbbra is lehet értelme a DBN-előtanításnak, ha rengeteg címkézetlen adatunk van, és csak kevés címkézett
- Az RBM két rétege köz szimmetrikus a kapcsolat
  - Ezért a rejtett rétegből vissza lehet generálni az inputot
  - Könnyű vizualizálni, hogy milyen rejtett reprezentációt alakított ki



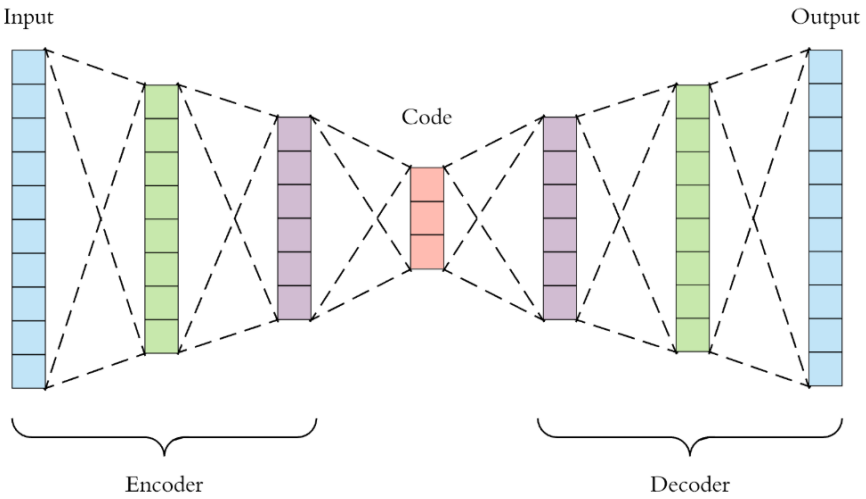
Mély hálók felügyelet nélküli tanítása

- A standard mély hálókat is taníthatjuk felügyelet (értsd: osztályokat megadó tanító címkék) nélkül
  - A leggyakoribb ilyen hálóstruktúra az ún. autoencoder háló, amikor a megtanulandó címkék megegyeznek az inputtal
  - A háló az inputját igyekszik rekonstruálni az outputon
  - Az autoencoder háló tipikusan homokóra alakú (az egyre kisebb rétegekkel próbáljuk rákényszeríteni a tömörítésre)
  - a középső, legkisebb réteget hívják „bottleneck” rétegnek



Autoencoder hálózatok

- Mire jó az autoencoder háló?
  - A tanulás során rákényszerítjük a hálót, hogy találjon egy tömör reprezentációt, amiből az input minél pontosabban rekonstruálható
  - Ezért használhatjuk jellemzőtér-redukcióra (a decoder részt eldobjuk, az eredetei jellemzők helyett a bottleneck réteg kimenetét használjuk)
  - De felügyelet nélküli előtanításra is jó (ha sok címkézetlen adatunk van, és kevés címkézett, betanítjuk vele az autoencodert, majd a decoder részt eldobjuk, és új kimeneti réteg(ek)et felvéve innen kezdjük a felügyelt tanítást)

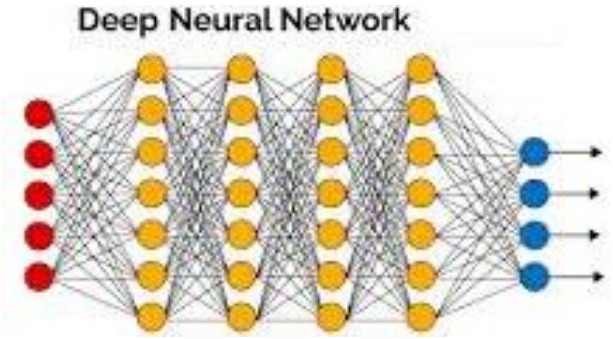


Batch normalizálás

- Azt mondtuk, hogy az input jellemzők normalizálása 0 átlagra és 1 szórásra segíti a tanulást

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

(Ez tkp. egy skálázás és egy eltolás)



- Mély háló esetén a k. réteg inputja a k-1. réteg outputja
- De akkor a k-adik réteg is könnyebben tanulna, a k. réteg kimenete 0 átlagú és 1 szórású lenne – *ehhez tudni kéne a skálázás és eltolás értékét*
- Megtehetnénk, hogy először csak az első réteget tanítjuk be, aztán kiszámolnánk a kimenet várható értékét és szórását, normalizálnánk, majd hozzáadnánk a hálóhoz egy új réteget, és így tovább…
  - De ez így lassú és nehézkes

Batch normalizálás 2

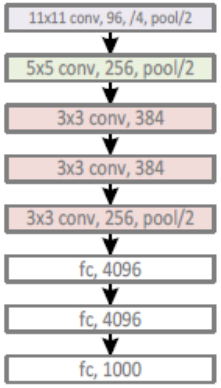
- Hatékonyabb megoldás, ha a normalizálást „online” végezzük, azaz mindig csak az adott batch-re
- De eközben a megfelelő skálázó és eltoló paramétereket megtanuljuk
  - Ugyanúgy tanulandó paraméternek tekintjük őket, mint a háló súlyait
  - Ezekre is kiterjesztjük a backpropagation algoritmust, és a tanulás során megtanuljuk
  - A tesztelés során már ezeket a tanult értékeket haszáljuk
- A batch normalizálás jótékony hatásai:
  - Gyorsabb konvergencia
  - Nagyobb kiindulási learning rate-et lehet használni
  - Mély hálók jobb eredménnyel taníthatók
  - A háló kevésbé lesz érzékeny a súlyok inicializálására

Nagyon mély neuronhálók

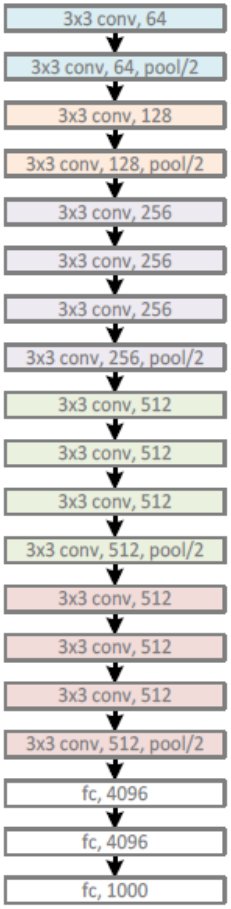
- A „mély háló” fogalma nincsen pontosan definiálva, de általában 5-10 rejtett réteggel rendelkező hálót jelent
- Újabban azonban megjelentek a „nagyon mély” (very deep) hálók is, tipikusan képfeldolgozási feladatokra
  - Ez sincs pontosan definiálva, de már 50-150 rejtett réteges hálókat is látni
- A nagyon mély hálók tanításánál már a korábban mutatott trükkök sem segítenek a hiba visszaterjesztésében
  - ReLU neuronok, ügyes inicializálás, batch normalization,…
- A mélyen levő rétegek hatékony tanításához új trükkök kellenek

Nagyon mély neuronhálók - példák

AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)



GoogleNet, 22 layers
(ILSVRC 2014)

ILSVRC (Large Scale Visual Recognition Challenge)

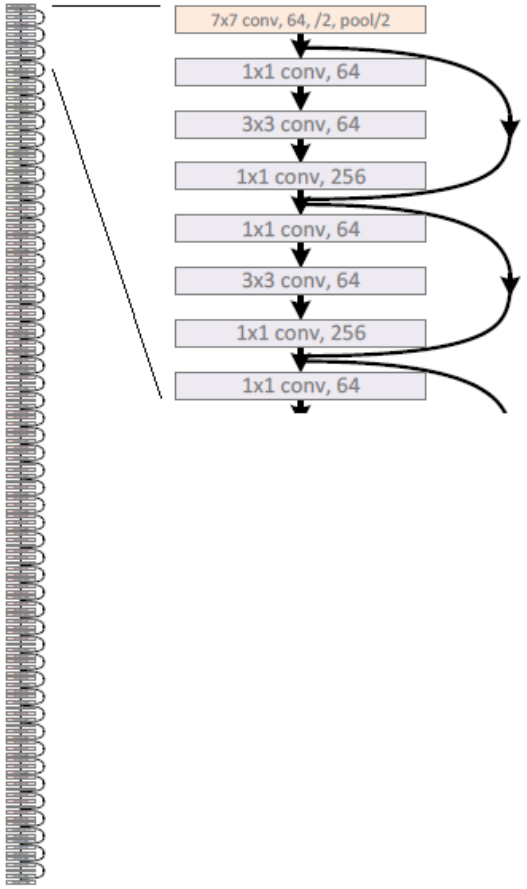
(slide credit: Jian Sun, MSR)

Nagyon mély neuronhálók - példák

AlexNet, 8 layers
(ILSVRC 2012)

VGG, 19 layers
(ILSVRC 2014)

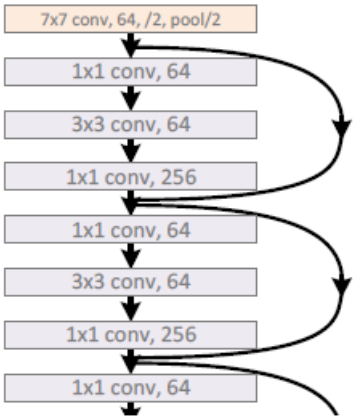
ResNet, 152 layers
(ILSVRC 2015)



ILSVRC (Large Scale Visual Recognition Challenge)

Nagyon mély hálók extra kapcsolatokkal

- A mély (távoli) rétegekbe visszaterjesztéskor a hiba sok rétegen megy át
  - Minden rétegben kicsit módosul – egyre nő a kockázata, hogy elvész vagy „felrobban”
- A javasolt megoldásokban közös, hogy új kapcsolatokkal egészítik ki a hálót, amelyek mentén a hiba kevesebb transzformáción átesve csatolódik vissza
- Néhány gyakori elnevezés ezekre a kapcsolatokra, hálókra:
  - „linear pass-through” vagy „jump” vagy „skip” connection
- És az ilyen hálók nevei:
  - „Highway network”
  - „Residual network”
  - „Linearly augmented network”

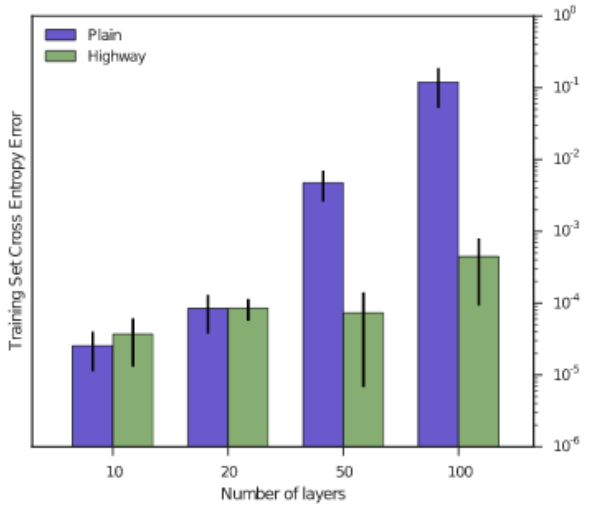


Highway Network

- Az új kapcsolatok egy ”information highway-t” alkotnak, ahol az információ transzformáció nélkül áramolhat
- Alapötlet: a rekurrens hálóknál is sok rétegen át kellett visszaterjeszteni a hibát, az LSTM kapuzós megoldása segített ebben
  - Hagyományos réteg kimenete: $\mathbf{y} = H(\mathbf{x}, \mathbf{W_H})$
  - Highway háló egy rétege: $\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot (1 - T(\mathbf{x}, \mathbf{W_T}))$
    - Ahol T a „transfer gate”: $T(\mathbf{x}) = \sigma({\mathbf{W_T}}^T \mathbf{x} + \mathbf{b_T})$
    - T így 0 és 1 közötti értékeket vehet fel
    - A kimenet *T*=0 ill. *T*=1 esetén:

$$\mathbf{y} = \begin{cases} \mathbf{x}, & \text{if } T(\mathbf{x}, \mathbf{W_T}) = \mathbf{0}, \\ H(\mathbf{x}, \mathbf{W_H}), & \text{if } T(\mathbf{x}, \mathbf{W_T}) = \mathbf{1}. \end{cases}$$

    - Azaz T szabályozza a transzformálatlan x és a transzformált H(x) arányát
    - Példa: beszédfelismerési hiba



Linearly Augmented Network

- A kapuzásnál egyszerűbb megoldást használ
- A hagyományos háló egy rétegének kimenete
  - (korábbi jelölésünkhöz képest V egy extra lineáris trafó):

$$f_i(x) = V_i \phi(U_i x + b_i)$$

- A lineárisan kiegészített réteg viszont ezt fogja számolni:

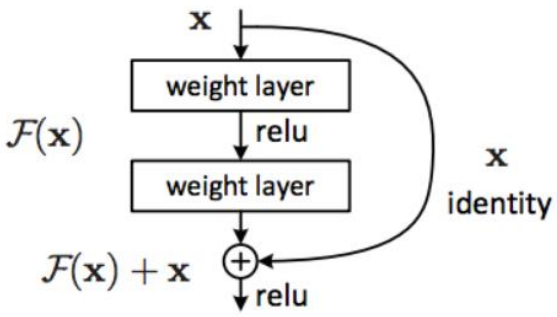
$$f_i(x) = V_i \phi(U_i x + b_i) + T_i x$$

  - Azaz az output egy hagyományos nemlineáris és egy lineáris transzformáció összegeként áll elő
  - T nem feltétlenül teljes transzformációs mátrixot jelent, diagonális mátrixszal, illetve rögzített (nem tanítható) egység-mátrixszal is jó eredményeket kaptak
  - Példa: beszédfelismerési eredmények

Num of H.Layers	Layers Size	# Params	PER %
3	1024X256	2.9M	**22.39**
12	512X256	3.8M	21.8
48	256X128	3.4M	**21.7**

Residual network

- Abból indul ki, hogy a kapuzás fölösleges túlbonyolítás
- Sőt, még a lineáris transzformáció sem kell
- Az extra kapcsolaton az input transzformáció nélkül jut el az outputig („identity mapping”)
- Ha a korábbi rétegek már megoldották a feladatot, az identitás leképezés elég, F(x)-nek már nincs mit tanulni
- Amit az eddigi rétegek még nem tanultak meg, azt a maradék („residual”) hibát kell visszacsatolni és megtanulni a nemlineáris F(x) leképezés segítségével



KÖSZÖNÖM A FIGYELMET!

A tananyag az EFOP-3.5.1-16-2017-00004 pályázat támogatásával készült.