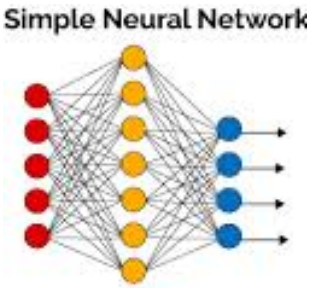
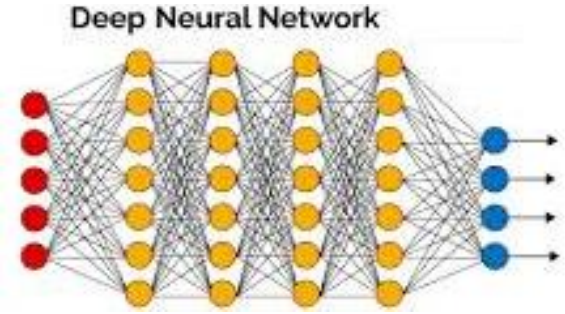
MÉLY NEURONHÁLÓK

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

Hagyományos és mély neuronhálók

- Miben különbözik a hagyományos és a mély neuronháló?
  - Strukturálisan annyi a különbség, hogy jóval több rejtett réteg van (1-2 helyett 5-10, de újabban akár 100-150 is)
- Egyszerűnek hangzik - miért csak most??
  - A mély hálók tanításához új algoritmusok kellettek
    - Legelső ilyen: DBN-előtanítás, 2006
  - A mély háló előnyei igazából csak sok tanító adat esetén mutatkoznak meg
    - Ez se volt meg a 80-as években
  - A mély háló tanítása számításigényes
    - Erre megoldás a GPU használata

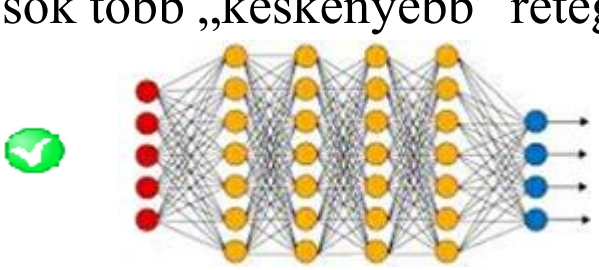


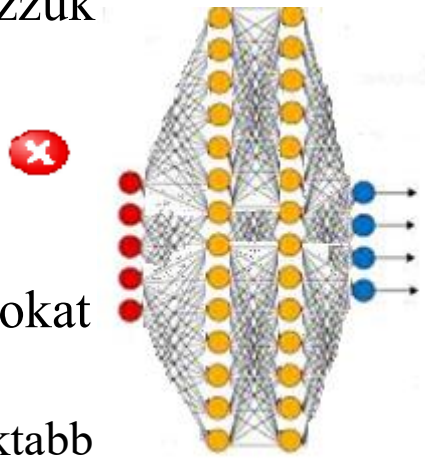


- A mély hálók jelenlegi sikeréhez az új algoritmusok, a sok tanító adat és a számítási kapacitás szerencsés együttállása kellett

Miért hatékonyabb a mély neuronháló?

- Korábban láttunk egy elvi bizonyítást arra, hogy két rejtett réteggel minden feladat megoldható
  - Azonban ez csak akkor igaz, ha végtelen nagy neuronhálónk, végtelen sok tanító adatunk és globális optimumot adó tanító algoritmusunk van
- Adott véges neuronszám mellett hatékonyabb, ha a neuronokat 1-2 „széles” réteg helyett sok több „keskenyebb” rétegbe rendezzük





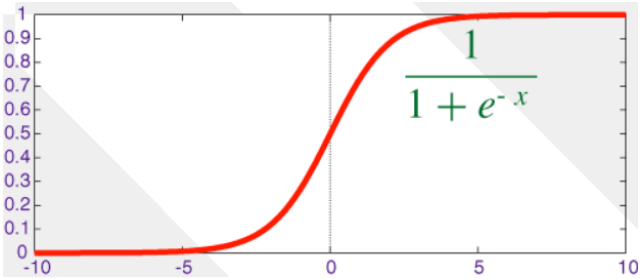
- Így a háló hierarchikusan tudja feldolgozni az adatokat
  - Képi alakfelismerési feladatokon jól látszik, hogy a magasabb rétegek egyre komplexebb, egyre absztraktabb fogalmakat tanulnak meg
    - Pont, él, → szem, orr→ arc→…

A mély háló tanítása

- A mély hálók tanítása sajnos nehezebb, mint a hagyományos hálóé
- Ennek fő oka a backpropagation algoritmus működési elve, illetve a sigmoid aktivációs függvény alakja
  - A backpropagation algoritmus a kimeneti rétegtől terjeszti vissza a hibát
  - Minél több réteget megyünk visszafele, annál nagyobb az esélye, hogy a gradiens „eltűnik”,
    - Ez az ún „vanishing gradient” effektus
  - A gyakorlatban ez azt eredményezi, hogy a mély háló egyre mélyebben levő rétegei egyre kevésbé tanulnak
  - Azaz hiába adunk újabb rétegeket a hálóhoz, az eredmények nem javulnak (sőt esetleg romlanak is)

Miért nehéz a mély háló tanítása?

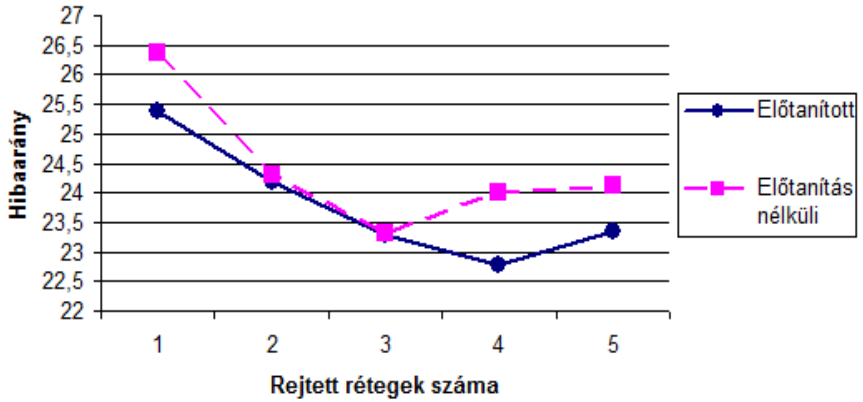
- Nézzük meg a sigmoid aktivációs függvény hatását:



  - A sigmoid függvény bemenete az aktiváció: $a = \sum_{i=0}^{n} w_i x_i$
  - Ha az aktiváció értéke nagyon nagy vagy kicsi, a bemenet a sigmoid „lapos” részeire esik
  - Itt a derivált nullához közeli, „eltűnik”, így a tanulás se fog működni
  - Ez ellen próbáltunk védekezni a súlyok inicializálása ill. az input normalizálása során is
    - A rejtett rétegek értékeinek „kordában tartására” azonban ez már kevés
    - Minél több az rejtett réteg, annál nehezebb a háló tanítása

Mély háló tanítása - szemléltetés

- Az ábra egy beszédfelismerési feladaton kapott eredményeket mutatja
  - Függőleges tengely: felismerési hiba
  - Vízszintes tengely: rejtett rétegek száma a hálóban



  - A lila görbe mutatja ez eredményeket backpropagation tanítás esetén
    - A rétegek hozzáadásával egyre kisebb a javulás, 4-5 rétegnél már romlás van
  - A kék görbe az egyik legkorábban javasolt megoldással (előtanítás) kapott eredményeket mutatja

Megoldások a mély neuronháló tanítására

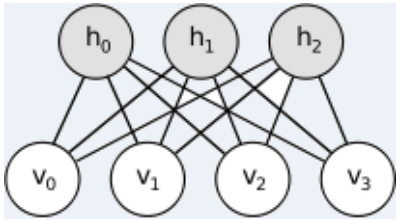
- A mély hálók hatékony tanításához módosítanunk kell a tanító algoritmust és/vagy az aktivációs függvényt
- A legismertebb módosítási lehetőségek az alábbiak
  - Előtanítás címkézetlen adatokkal (DBN-előtanítás kontrasztív divergencia (CD) hibafügvénnyel)
    - Ez volt az első ötlet, elég lassú és matematikailag bonyolult
  - A hálózat rétegről-rétegre való felépítése és tanítása
    - Jóval egyszerűbb, csak backpropagation kell hozzá
  - Újfajta aktivációs függvények használata
    - A legegyszerűbb megoldás, először ezt fogjuk megnézni
- A mély hálók tanításában további trükkök is segíthetnek (batch normalization, highway network, stb.) – ezekről később…

DBN-előtanítás

- Történetileg ez volt a legelső mélyháló-tanító algoritmus (2006)
- Egy előtanítási és egy finomhangolási lépésből áll, ahol a finomhangolás megfelel a már ismert backpropagation tanításnak
- Előtanítási lépés:
  - Konstruál egy ún. „deep belief” hálót
  - Ezt tanítja a backpropagationtól teljesen eltérő algoritmussal
- Tanítási lépés:
  - A betanított DBN hálót átkonvertálja hagyományos (sigmoidos) neuronhálóvá
  - Ezt már a backpropagation algoritmussal tanítja
  - Ezt a lépést „finomhangolásnak (fine-tuning)” nevezi

Korlátos Boltzmann-gépek

- A deep belief hálóhoz először a korlátos Boltzmann-gépet (RBM) kell megismernünk
  - Ez hasonlít a neuronháló egy rétegpárjára
  - De valós helyett bináris kimeneti értékeket ad
  - V: „visible” réteg: ez lesz egyszerre az input és az output réteg
  - H: „hidden” réteg: a rejtett réteg
  - A visible és hidden rétegek kapcsolata nem determinisztikus, hanem sztochasztikus
  - Ettől eltekintve a képlet eléggé hasonlít a standard neuronokéra (sigmoid aktivációs függvénnyel)
  - Vegyük észre, hogy standard neuronokkal szemben a V és H rétegek szerepe szimmetrikus, mindkét irányban tudnak menni az adatok

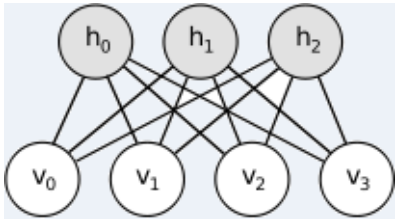


$$P(h_j = 1|v) = \sigma\left(b_j + \sum_{i=1}^{m} w_{i,j} v_i\right)$$

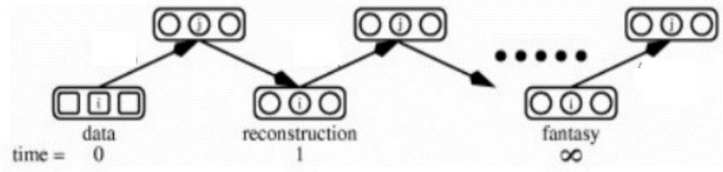
$$P(v_i = 1|h) = \sigma\left(a_i + \sum_{j=1}^{n} w_{i,j} h_j\right)$$

RBM-ek tanítása

- Az RBM-ek tanítására a kontrasztív divergencia (CD) algoritmust használhatjuk
  - Ez egy felügyelet nélküli módszer (nincsenek osztálycímkék)
  - Ugyanúgy iteratív algoritmus, mint a backpropagation
  - Cél az input rekonstruálása a rejtett réteg alapján (Hasonlít a Maximum Likelihood célfüggvényhez)



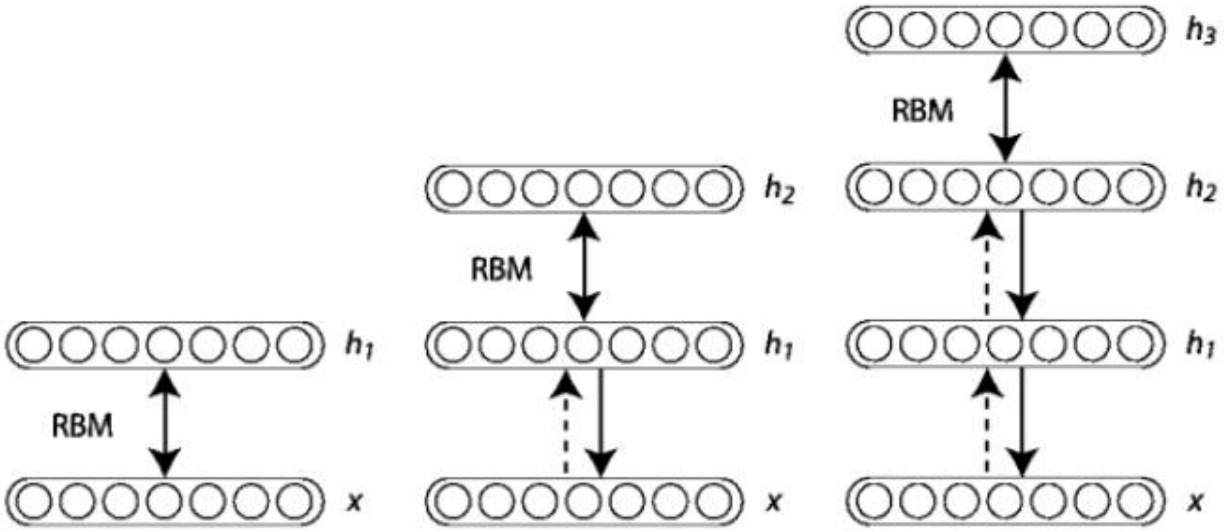
- Az algoritmus az eredeti inputból elindulva felváltva frissíti a rejtett, majd a visible réteget
  - Cél, hogy a kapott rekonstrukció minél jobban hasonlítson az eredeti inputhoz



  - Az ideális reprezentáció megtalálásához végtelen sok iteráció kellene
  - A gyakorlatban csak egyetlen iterációval közelítjük ezt, majd továbblépünk a következő adatvektorra
  - A fentieket iterálva áll elő a CD algoritmus

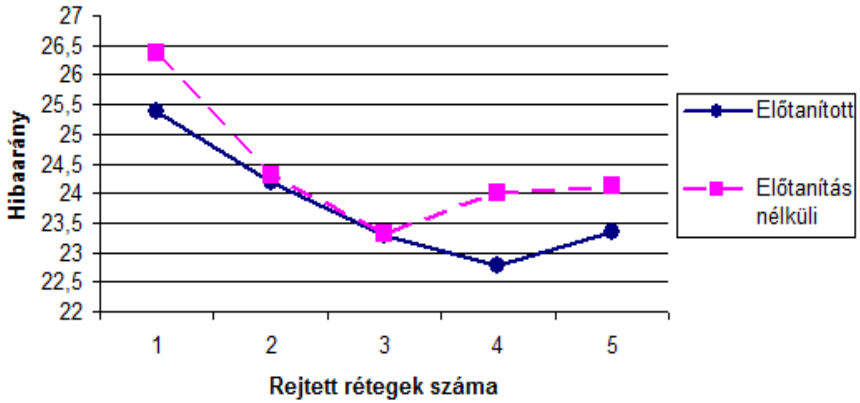
Deep Belief Network (DBN)

- A DBM-et RBM-ek egymásra pakolásával kapjuk meg
- A hálót nem rögtön egyben tanítja, hanem rétegenként építi fel
  - 1 rejtett réteg → tanítás
  - Újabb rejtett réteg hozzáadása → tanítás
  - …



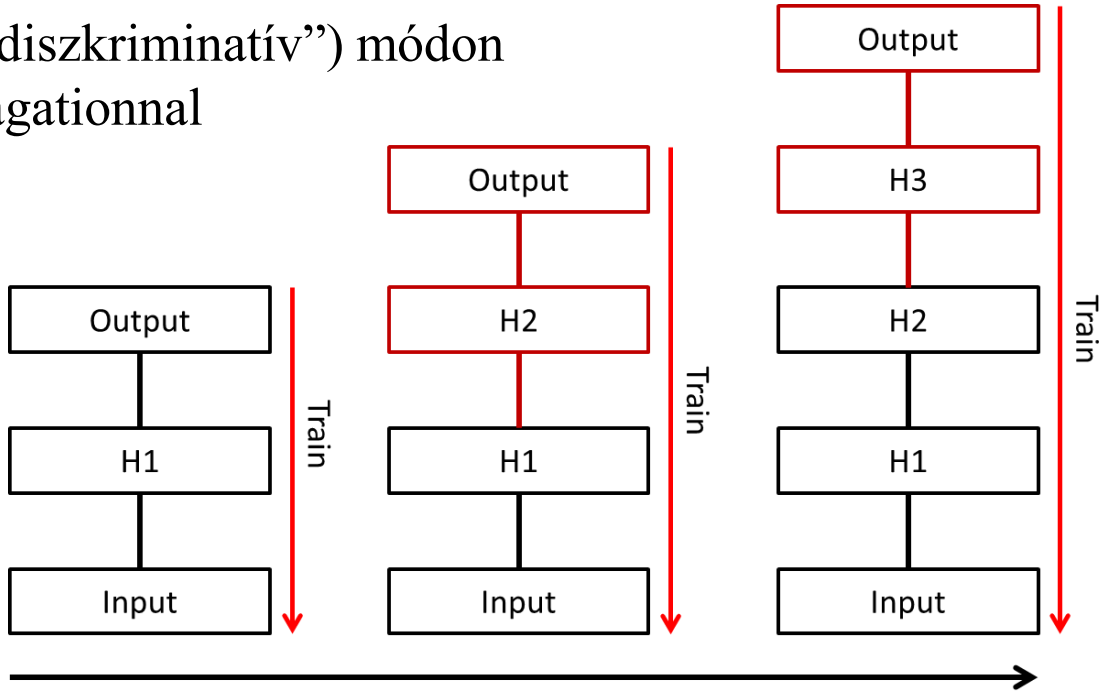
Konvertálás mély neuronhálóvá

- A DBN előtanítás után az RBM neuronjait hagyományos neuronokká konvertáljuk (nyilván a súlyok megtartásával)
- A tetejére rárakunk egy softmax réteget annyi neuronnal, ahány osztálycímkénk lesz
- És innentől elindíthatjuk a hagyományos, felügyelt backpropagation tanítást a címkézett példáinkat használva
- Az előtanítás szerepe tehát lényegében a súlyok inicializálása
- A random incializálásnál jobb eredményeket kapunk előtanítással (ld. korábbi példafeladat)



Diszkriminatív előtanítás

- Ez a módszer is a mély háló nehéz taníthatóságát igyekszik orvosolni
- Alapötletét a DBN-tanításból veszi: a hálót nem rögtön egyben tanítja, hanem a rétegeket egyenként adja hozzá
  - De: felügyelt („diszkriminatív”) módon tanít, backpropagationnal

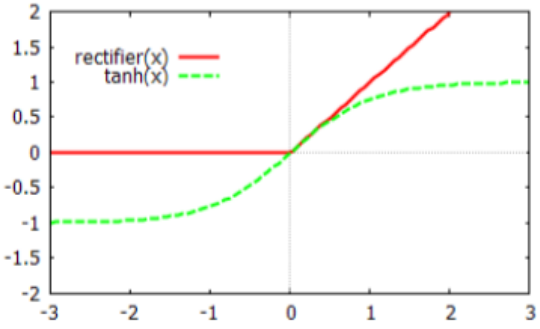


Diszkriminatív előtanítás (2)

- A módszer előnye: nagyon egyszerű implementálni, nem kell új tanítóalgoritmus, csak a backpropagation
- Hátrány: megnövekedett tanítási idő
  - Bár a tapasztalatok szerint az egyes rétegek hozzáadása után nem kell teljes konvergenciáig tanítani, elég pár iteráció („előtanítás”)
  - Teljes tanítást csak az utolsó rejtett réteg hozzáadása után végzünk
- A tapasztalatok szerint az „előtanítás” hatására a súlyok jobb kezdőértéket vesznek fel, mint ha véletlenszerűen incializálnánk őket
  - Ezért az utolsó, „teljes” tanítási lépés kisebb eséllyel akad el lokális optimumban

A rectifier aktivációs függvény

- Ez a módszer nem a tanításba nyúl bele, hanem lecseréli az aktivációs függvényt
- Az alábbi ábra a rectifier aktivációs függvényt mutatja
  - Képlete: $\phi(z) = max(z, 0)$
- Összehasonlítva a tanh aktivációs függvénnyel:



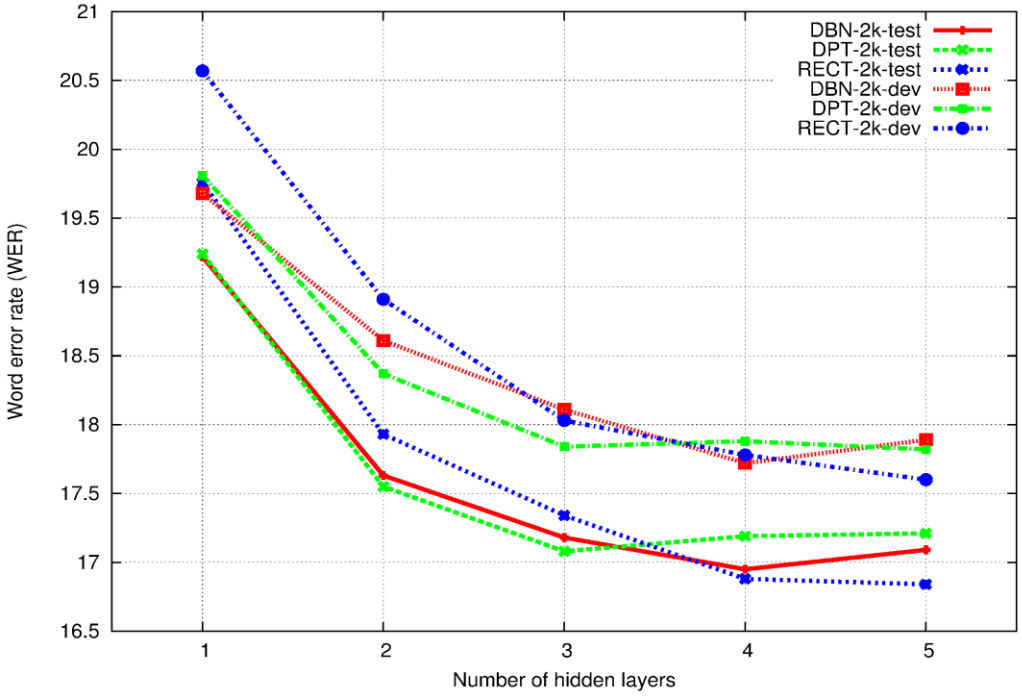
- Pozitív inputra
  - Megj: Az aktivációs függvénynek muszáj nemlineárisnak lenni!
- Pozitív inputra a deriváltja mindig 1, sehol sem tűnik el
- Negatív inputra a kimenet és a derivált is 0

A rectifier aktivációs függvény

- Előnye: pozitív inputra sosem 0 a derivált
- Hátrányok:
  - Negatív inputra viszont mindig 0
    - A gyakorlatban úgy tűnik, hogy ez nem okoz nagy problémát
  - Nem szimmetrikus (csak nemnegatív outputot ad)
    - A gyakorlatban ez se nagy probléma
  - Nem korlátos a kimenete, azaz végtelen nagy kimenetet is adhat
    - Vannak trükkök ennek a kezelésére, például a súlyok nagyságának időnkénti normalizálása

A három módszer összehasonlítása

- Kísérleti összehasonlítás beszédfelismerési adatokon a dev-test halmazokon
  - DBN: DBN-előtanítás, DPT: diszkriminatív előtanítás, RECT: rectifier háló



A tanítási idők összehasonlítása

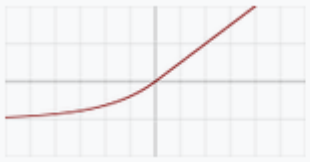
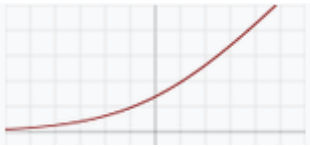
- A háromféle tanítási módszer futásideje a korábbi feladatra (öt rejtett réteg esetén):

Training method	Pre-training time	Fine-tuning training time
DBN pre-training	48 hours	14 hours
Discr. pre-training	9 hours	11 hours
Rectifier network	0 hours	14.5 hours

- Míg a három módszerrel kapott eredmények elég hasonlóak voltak, a rectifier háló tanítása sokkal kisebb időigényű, mivel nem kell előtanítani
- A „rectified linear” (ReLU) neuronokra épülő háló jelenleg a de facto standard a mély tanulásban

Még újabb aktivációs függvények

- Vannak javaslatok még újabb aktivációs függvényekre
  - Ezek a korábban említett hátrányokat próbálják korrigálni
  - Lásd még: https://en.wikipedia.org/wiki/Activation_function
- Példák: ELU, SELU, Softplus…

Exponential linear unit (ELU)[14]		$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
SoftPlus[18]		$f(x) = \ln(1 + e^x)$

- Ezekkel néha kicsit jobb eredmények jönnek ki, mint a rectifier függvénnyel, de általános áttörést egyik sem hozott eddig

KÖSZÖNÖM A FIGYELMET!

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