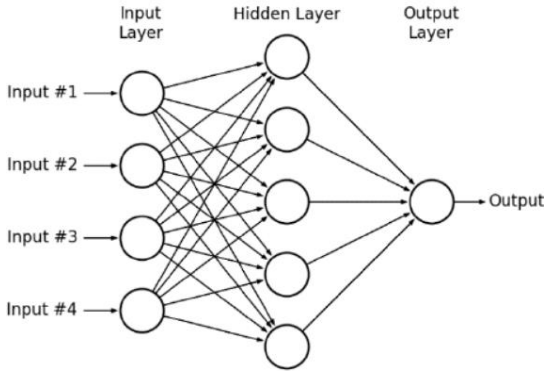
NEURONHÁLÓK ÉS TANÍTÁSUK A BACKPROPAGATION ALGORITMUSSAL

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

Neuron helyett neuronháló

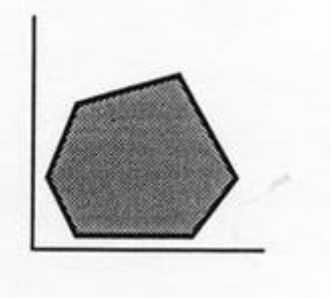
- Neuron reprezentációs erejének növelése: építsünk hálózatot!
- Klasszikus struktúra: Multilayer feedforward network
  - Minden réteg az alatta lévő rétegtől kapja az inputját
  - A rétegek között „full connection”: minden neuron minden neuronnal
  - Csak előremutató kapcsolatok



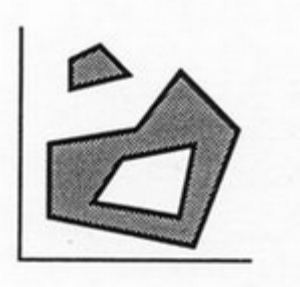
- Viszonylag ritkán szokás, de:
  - Fully connected helyett ritka („sparse”) hálót is lehet csinálni
  - Rétegek kihagyása szintén viszonylag könnyen megoldható
- Visszacsatolt (rekurrens) struktúra is lehetséges, azt viszont jóval bonyolultabb tanítani (ld. recurrent neural networks - később)
  - Ezek főleg időbeli sorozatok modellezésében hasznosak

A neuronháló reprezentációs ereje

- Konvex térrész körbekerítése: 1 rejtett réteg elég
  - A rejtett réteg egyes neuronjai reprezentálnak 1-1 határoló egyenest
  - A kimeneti neuron ÉS műveletet végez: ott jelez, ahol a rejtett neuronok mindegyike jelez



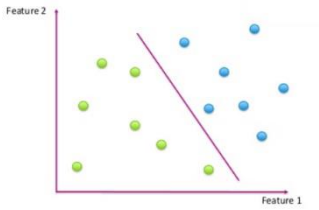
- Tetszőleges (akár non-konvex, nem összefüggő) térrész megtanulása: 2 rejtett réteg elég
  - Az első rejtett réteg neuronjai egyeneseket húznak
  - A második réteg neuronjai ÉS művelettel konvex térrészeket jelölnek ki
  - A kimeneti neuron VAGY művelettel ezeket non-konvex, nem összefüggő térrészeké is össze tudja kapcsolni



- Elvben 2 rejtett réteggel minden gyakorlati tanulási feladat megoldható
  - A tetszőleges pontossághoz végtelen sok neuron, végtelen sok tanító adat és tökéletes (globális optimumot adó) tanító algoritmus kellene…

Többosztályos tanulás

- Egyetlen neuron 2 osztályt tud csak elválasztani
  - Neuronháló esetén viszont már szóba jön a többosztályos tanulás is



- Többosztályos osztályozási feladat megoldása neuronhálóval
  - Minden c_i osztályhoz egy kimeneti neuront rendelünk
    - Azt várjuk a hálótól, hogy adott példa esetén a helyes osztályhoz rendelt kimeneten egyet adjon, a többin nullát
  - Tanító példák címkéje: 1-of-M kódolású (vagy „one-hot-encoded”) vektor
    - M méretű vektor (osztályok száma), benne egyetlen 1-es:

0 0 1 0 0 0
↑
correct class

  - Tanítási célfüggvény: maradhat az MSE error
    - Csak összegezni kell az összes kimenetre: $E(\mathbf{w}) = \frac{1}{N} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$
  - Hogyan soroljuk be az aktuális példát?
    - Arra tanítjuk a hálót, hogy a helyes osztályra 1-et adjon, máshol 0-t
    - Valójában - ha a hiba nem 0 - a sigmoid aktivációs fgv. 0-1 közti értékeket ad vissza
    - Válasszuk a maximális értéket adó kimeneti neuron osztályát!

Többosztályos tanulás a CE hibafüggvénnyel

- Egy korábbi ábrán szemléltettük, hogy a neuronháló hogyan képes osztályokat elszeparálni (geometriai szemlélet)
- Most azt látjuk, hogy többosztályos háló esetén egy-egy kimenetünk lesz minden c_i osztályhoz
  - És ezek közül a maximális értéket adóra tippelünk
  - Ezt pontosan megfelel a döntéselméleti szemlélet döntési sémájának!
- De értelmezhetjük-e a háló kimeneteit $P(c_i|\mathbf{x})$ valószínűségi becslésként?
- Igen, így a többosztályos háló a döntéselméleti szemléletnek is meg tud felelni, de ehhez az alábbiak kellenek még:
  - Módosítani kell a kimeneti neuronok aktivációs függvényét, hogy a kimenetek összege 1 legyen
  - Módosítani fogjuk a hibafüggvényt is: az MSE hiba helyett az ún. keresztentrópia (cross entropy, CE) hibafüggvényt fogjuk használni

Többosztályos tanulás a CE hibafüggvénnyel (2)

- A kimeneti neuronok értékét $P(c_i|\mathbf{x})$ becslésként akarjuk értelmezni
- Ezek együtt egy diszkrét valószínűségi eloszlást adnak meg
  - Az egyes kimenetek értékkészlete [0,1], összegük 1 kell legyen
  - Előbbit teljesíti a sigmoid függvény, de az utóbbit nem
- A kimeneti neuronokon a sigmoid helyett a softmax aktivációs függvényt fogjuk alkalmazni

$$\sigma(\mathbf{a})_j = \frac{e^{\mathbf{a}_j}}{\sum_{k=1}^{M} e^{\mathbf{a}_k}} \quad \text{for } j = 1, \ldots, M$$

  - A kimenetek értéke így garantáltan (0,1) közé esik, és az összegük 1

Többosztályos tanulás a CE hibafüggvénnyel (3)

- A keresztentrópia hibafüggvény:

$$E(\mathbf{w}) = -\frac{1}{N}\sum_{d\in D}\sum_{k\in outputs} t_{kd}\ln(o_{kd})$$

  - A keresztentrópiát diszkrét eloszlások eltérésének mérésére találták ki
  - Vegyük észre, hogy 1-of-N kódolás esetén t értéke csak 0 vagy 1 lehet, így a célfüggvény az alábbi módon egyszerűsödik:

$$E(\mathbf{w}) = -\frac{1}{N}\sum_{d\in D}\ln(o_{correct,d})$$

  - Ez akkor lesz minimális, ha a helyes osztályhoz tartozó kimenet minél inkább 1-hez közelít
  - Mivel a softmax függvény révén a kimenetek össze vannak kötve, ez csak úgy lehetséges, ha az összes többi osztályhoz tartozó kimenet értéke 0-hoz közelít

Kapcsolat a statisztikai alakfelismeréssel

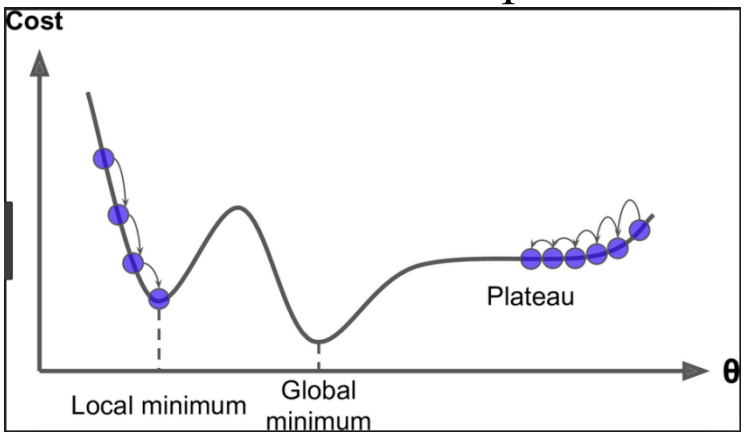
- A fenti módosításokkal a háló kimenetei értelmezhetők $P(c_i|\mathbf{x})$ –re adott becslésként, de vajon a tanulás során előálló értékeknek tényleg van közük $P(c_i|\mathbf{x})$ –hez?
- Bebizonyítható, hogy a korábban leírtaknak megfelelően felépített és tanított háló kimenetei tanítás során $P(c_i|\mathbf{x})$ valószínűségekhez tartanak!
  - A gyakorlatban nem szabad elfelejteni, hogy ez csak közelítés, a tökéletes modellezéshez végtelen nagy háló, végtelen sok tanító adat és globális optimumot garantáló tanítóalgoritmus kellene
- A statisztikai alakfelismeréssel, illetve a Bayes döntési szabállyal való összekapcsolás bizonyos alkalmazási területeken jelentősen megnövelte a neuronhálók iránti bizalmat

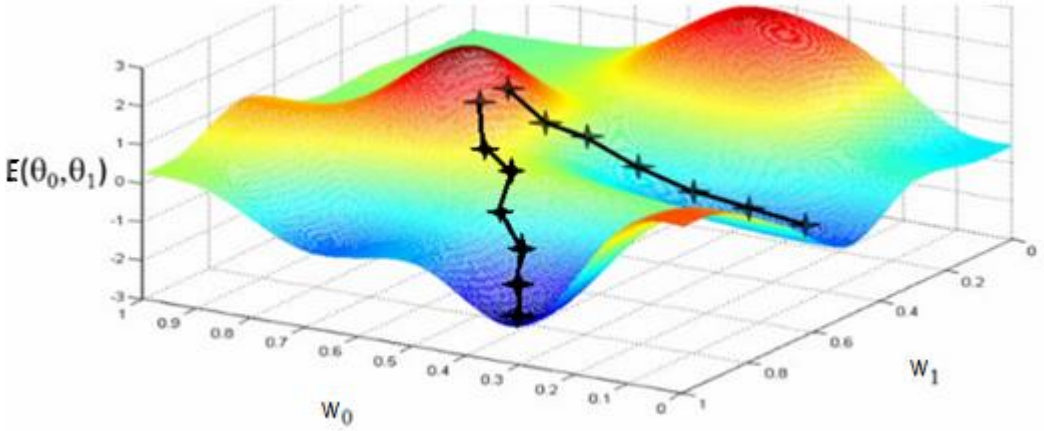
A neuronháló tanítása

- A hibafüggvények által definiáltuk, hogy hogyan mérjük a háló hibáját egy adott mintahalmazon
- A neuronháló úgy tekinthető, mint egyetlen nagy függvény, amelynek változói a súlyok
- A tanítás során a súlyokat akarjuk úgy beállítani, hogy a tanító példahalmazon a háló minél kisebb hibát adjon
- Ez egy sokváltozós optimalizálási feladat
  - Sokféle optimalizáló algoritmust lehetne használni a tanuláshoz
  - A legegyszerűbb és leggyakrabban használt megoldás a korábban már látott gradient descent algoritmus
  - Neuronhálók esetében ez hiba-visszaterjesztéses (backpropagation) algoritmus néven lett közismert

A backpropagation algoritmus

- A „hegymászó” ill. „gradent descent” algoritmus változata neuronhálók esetére
  - A derivált alapján fogjuk eldönteni, hogy merre lépjünk a hibafelületen („gradiens módszer”) – legmeredekebb csökkentés elve: a gradiensvektorral ellentétes irányba fogunk lépni
  - Véletlenszerű inicializálásból indulunk
  - A lépésközt heurisztikusan állítjuk be („learn rate”) –ld. később
  - Iteratív (addig lépkedünk, amíg el nem akadunk)
  - Csak lokális optimum megtalálását garantálja





A backpropagation algoritmus egyetlen neuron esetére (emlékeztető)

- Az MSE hibafüggvény egyetlen neuronra:

$$E(\mathbf{w}) = \frac{1}{N}\sum_{d\in D}(t_d - o_d)^2 \quad \text{- hibafüggvény}$$

$$o_d = 1/(1+e^{-a_d}) \quad \text{- aktivációs függvény}$$

$$a_d = \sum_i w_i x_i \quad \text{- lineáris aktiváció}$$

- Ezeket egymásba ágyazva kapjuk meg, hogyan függ E(w) egy konkrét w_j-től
- Deriválás: ezek deriváltjai láncszabállyal összerakva

$$\frac{\partial E}{\partial w_i} = \frac{1}{N}\sum_{d\in D} 2(t_d - o_d)o_d(1-o_d)x_{id}$$

- Súlyok frissítése minden lépésben: $w_i = w_i - \eta\frac{\partial E}{\partial w_i}$
  - Ahol η egy kis pozitív kontans („learn rate”)

A backpropagation algoritmus általánosan

- Általános eset: több kimenő neuron, többrétegű hálózat

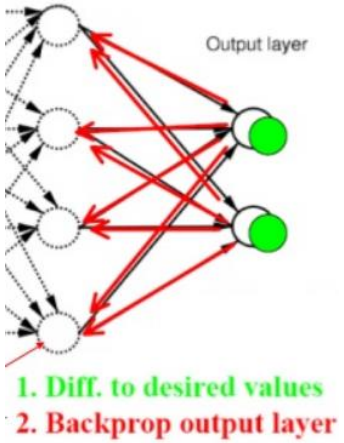
$$E(\mathbf{w}) = \frac{1}{2N} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

  - Ugyanúgy a láncszabályt kell alkalmazni, csak több lépésen keresztül (a felső neuronok bemenete nem x, hanem az alatta levő réteg kimenete)
    - *(az egyszerűség kedvéért a D-re összegzést kihagyjuk a levezetésből)*
    - A deriváltat (röviden: „hibát”) először levezetjük a kimeneti rétegre
    - Majd rétegről rétegre terjesztjük vissza (backpropagation!) az alsóbb rétegekre

- a k. kimenőegység hibája: $\delta_k = o_k(1-o_k)(t_k - o_k)$
- a h. rejtett egység hibája (rétegenként visszafele):

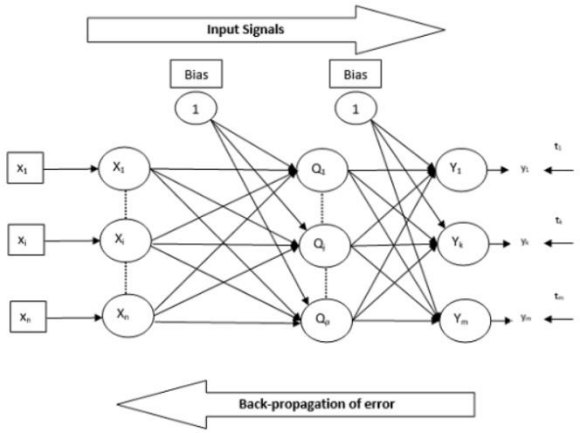
$$\delta_h = o_h(1-o_h) \sum_{k \in Uses(h)} w_{kh} \delta_k$$

    - Értelmezés: adott rejtett neuron hibája az ő kimenetét felhasználó neuronok hibájának súlyozott összegével arányos
      - Ahol a súlyok megegyeznek az adott kapcsolat súlyával



A backpropagation algoritmus - Összegzés

- Összegezve, a backpropagation tanítás fő lépései:
  - Inputtól az output felé haladva kiszámoljuk az egyes neuronok kimeneteit a D tanítópéldákon
    - *A hibafüggvényhez kell tudnunk a kimeneteket!*
  - Az outputtól az input felé haladva kiszámoljuk az egyes neuronok hibáit
  - Frissítjük a súlyokat: $w_{ji} = w_{ji} + \eta \delta_j x_{ji}$



- Megjegyzés: a hiba-visszaterjesztés megengedi a ritka (“sparse”) hálózatot, vagy a rétegek közötti ugrást tartalmazó hálózatot is. Az egyetlen lényeges megkötés a hálózati struktúrára, hogy ne legyen visszacsatolás (kör)
  - Ha nincs, akkor topológiai rendezés sorrendjében lehet frissíteni
  - Ha van, akkor visszacsatolt (rekurrens) hálózatot kapunk – ezek tanítása jóval komplikáltabb

Sztochasztikus backpropagation

- A hibafüggvényeink a hibát az összes tanítópéldára átlagolják:

$$E(\mathbf{w}) = \frac{1}{2N} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

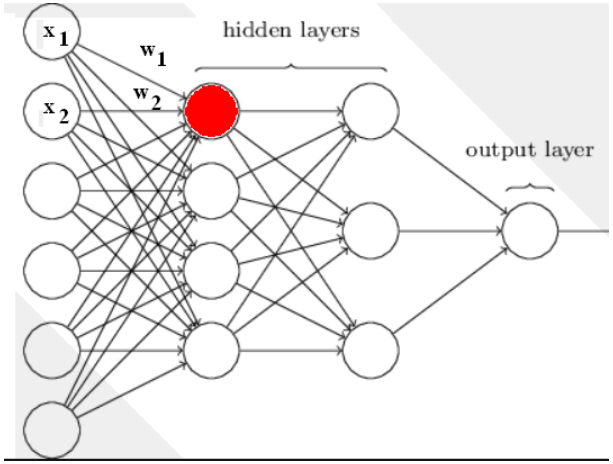
  - Ennek kiértékeléséhez az összes példát össze kell várnunk. Ez kizárja pl. az online tanulást
- On-line backpropagation: a hibafüggvényből kihagyjuk a D-re összegzést. Ehelyett minden egyes beérkező példán elvégezzük a kiértékelést, majd rögtön a hibaszámolást és súlyfrissítést is
- Semi-batch backpropagation: nem egy, de nem is az összes példa alapján frissítünk, hanem egy blokknyi („batch”) adat alapján
  - Tulajdonképpen minden frissítéskor kicsit más hibafüggvényt optimalizálunk! (az adott batch-ből számoltat)
  - Ez nem ront, hanem javít: kis véletlenszerűséget visz a rendszerbe – csökkenti a lokális optimumban való elakadás esélyét
    - Innen a név: „sztochasztikus” gradient descent (SGD)

Semi-batch backpropagation

- A gyakorlatban mindig ezt használjuk
- A batch alapú tanítás előnyei:
  - Csökken a lokális optimumban elakadás kockázata
  - Gyorsabb konvergencia
  - Gyorsabb végrehajtás (nem kell az összes adatot feldolgozni egyszerre)
  - Egyszerűbb implementáció (egy batch-nyi adat befér a memóriába)
  - Lehetővé teszi a (szemi) online tanítást
- Tipikus batch-méret: 10-100-1000 példa
- Mint a teljes adathalmazon tanításnál, itt is általában többször végigmegyünk az összes adaton
  - Egy „tanítási kör” az adatokon angolul: egy “training epoch”

Backpropagation GPU-n

- Miért hatékonyabb a neuronhálókat GPU-n tanítani?
  - Egy neuron aktivációjának kiszámítása: $a = \sum_{i=0}^{n} w_i x_i$
    =vektor-vektor szorzás
  - De az adott réteg neuronjainak aktivációját párhuzamosan is kiszámolhatjuk!
    =mátrix-vektor szorzás
  - De ugyanezt párhuzamosan elvégezhetjük egy batch-nyi input vektorra is!
    =mátrix-mátrix szorzás



  - A GPU-k a szorzatmátrix egyes celláinak értékét párhuzamosan tudják számolni – 30-40-szer gyorsabb, mint ha egyetlen CPU-n végeznénk

Tanítási tippek és trükkök

- A backpropagation algoritmus megadja a matematikai alapot a neuronhálók betanításához
  - Viszont csak lokális optimum megtalálását garantálja, így jó a tanítási eredmény elérése sajnos különféle gyakorlati trükkökön is múlik
  - Ezek többnyire inkább csak heurisztikák, nem precízen megalapozott elvek
- A legfontosabb gyakorlati fortélyokat mutatjuk be a továbbiakban
  - Adatok normalizálása, randomizálása
  - Súlyok inicializálása
  - A learning rate hangolása

Adatok randomizálása

- Semi-batch tanításnál sokat segíthet, ha az adatvektorokat véletlenszerű sorrendbe rakjuk
  - Nélkülözhetetlen, ha pl. a példák osztálycímkék szerint nincsenek jól összekeverve (pl. először az 1. osztály példái, aztán a 2. osztály, stb.)
  - Akkor is segít, ha az osztálycímkék ugyan keverednek, de valami más szempont szerint nem véletlenszerű a példák sorrendje (pl. beszédfelismerés: a felvételek beszélők szerint sorban jönnek)
  - Ugyanis mindig az aktuális batch hibafüggvényére optimalizálunk, így az újabb adatok nagyobb hangsúlyt kapnak, mint a régiek – a rendszer rátanul a későbbi adatok speciális tulajdonságára, a korábbiakat elfelejti
- Az adatokon többször végig kell menni → elvileg érdemes minden epoch előtt újra randomizálni az adatokat
  - Sajnos a randomizálás nem egyszerű: ha a memóriában csináljuk, akkor lassul az adatelérés, ha fájlban, akkor még macerásabb.

Adatok normalizálása (standardizálása)

- Tanítás előtt érdemes az egyes jellemzők értéktartományát egységes skálára hozni

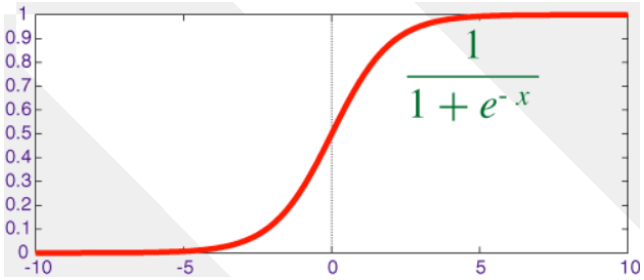
Láz	Ízületi_fájdalom	Köhögés	Influenzás
38,2	Van	Nincs	Igen
36,7	Van	Száraz	Nem
41,2	Nincs	Nyákos	Igen
38,5	Van	Száraz	Igen
37,2	Nincs	Nincs	Nem

  - min=-1, max=1: nem annyira jó ötlet, egyetlen kilógó adat („outlier”) hazavághatja
  - Szokásos megoldás: az egyes jellemzők átlaga 0, szórása 1 legyen
- Miért segít az egységes skála?
  - A súlyokat egységes tartományban inicializáljuk. Ha a jellemzők más-más skálára esnének, egyes jellemzők elnyomnának másokat az aktivációban:

$$a = \sum_{i=0}^{n} w_i x_i$$

Adatok normalizálása (2)

- Miért 0 környékére normalizálunk?
  - Idézzük fel a sigmoid aktivációs fügvényt:



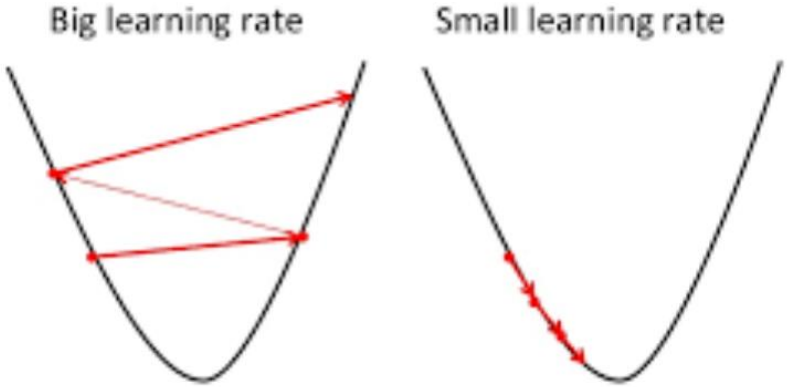
- Ha $a = \sum_{i=0}^{n} w_i x_i$ túl nagy van túl kicsi, akkor a sigmoid „lapos” részére esik → itt a derivált lényegében 0, a rendszer nem fog tanulni („elhaló gradiens” problémája)
  - Erre még visszatérünk az újabb aktivációs függvények, illetve a regularizációs módszerek bemutatásakor

Súlyok inicializálása

- A súlyokat kis random értékekkel inicializáljuk
  - Pl. [-0.1, 0.1] közötti véletlenszámokkal
- Különösen mély (sok rejtett réteget tartalmazó) hálók esetén fontos, hogy hogyan inicializálunk
  - A jó inicializálás segíti a hiba visszaterjesztését a legalsó rétegig
  - Különböző stratégiák léteznek az inicializálásra, ez a használt aktivációs függvénytől is függhet
    - Pl: a súlyok legyenek Gauss-eloszlású véletlenszámok 0 várható értékkel és 1/inputszám szórással
    - A magyarázat ugyanaz, mint az input normalizálásánál: szeretnénk az $a = \sum_{i=0}^{n} w_i x_i$ szorzatot egységesen ugyanabban a tartományban tartani

A learning rate beállítása

- A learning rate értékét általában tapasztalat alapján lőjük be
  - Bár vannak automatikus optimalizálási próbálkozások is
  - Túl nagy learn rate: nincs tanulás, ugrálás a hibafelületen
  - Túl kicsi learn rate: lassú tanulás, könnyebb elakadás lokális optimumban



  - Szokásos általános heurisztika:
    - Indítsunk olyan nagy learn rate-tel, amekkorával csak lehet (van tanulás)
    - Egy idő után (ha már nem csökken a hiba) kezdjük el csökkenteni (pl. felezgetni)

A learning rate beállítása (2)

- A túltanulás elkerülése érdekében érdemes a hiba alkaulását nem csak a tanítóadatokon, hanem egy független validációs halmazon is figyelni
- A learning rate frissítése validációs példahalmaz segítségével
  - A tanítás előtt tegyük féle az adatok egy részét → validációs halmaz
  - Valamennyi tanítási lépés (pl. 1 epoch) után értékeljük ki a hibát a validációs halmazon
  - Ha az előző kiértékeléshez képest a hiba csökkent: újabb iteráció ha nőtt, vagy csökkenése kisebb, mint egy küszöb → learning rate felezése
  - Teljes leállás: ha a learning rate lement egy küszöb alá

KÖSZÖNÖM A FIGYELMET!

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