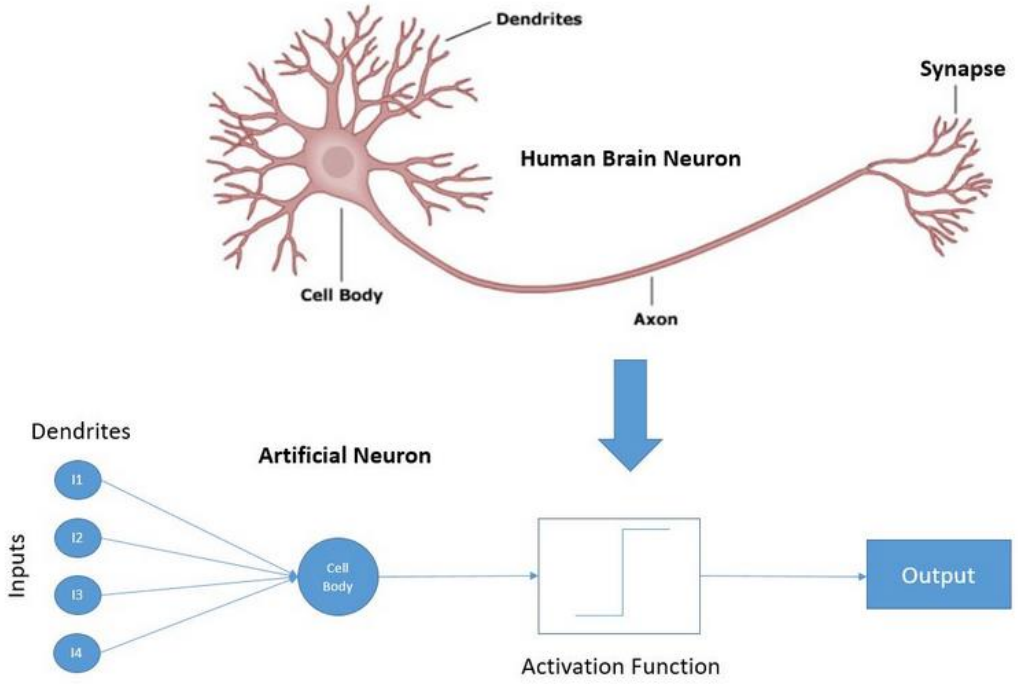
A PERCEPTRON NEURÁLIS MODELL ÉS TANÍTÁSA

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

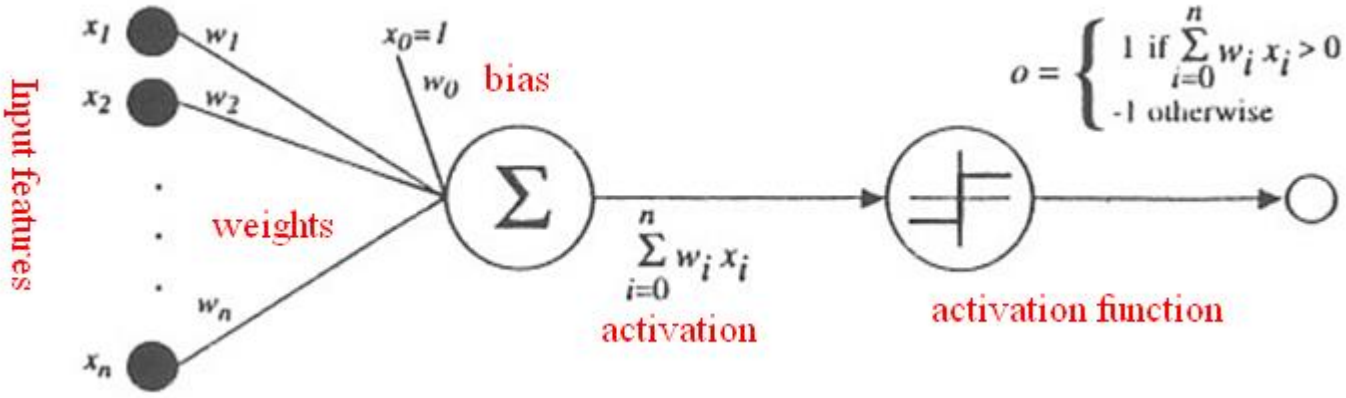
A „perceptron” neurális modell

- A legismertebb neurális modell a “perceptron” modell
- Rosenblatt találta fel 1957-ben
- A medell biológiai neuronokhoz való hasonlósága mellett érvelt
  - Sok input – “dendritek”
  - Egy output – “axon”
  - Ha az ingerek (súlyozott) összege elér egy küszöböt, a neuron tüzel, különben nem – a neuron kimenete bináris (az eredeti modellben), erre szolgál az aktivációs függvény



A perceptron matematikai modellje

- Az inputok az $x_1 \ldots x_n$ jellemzőknek felelnek meg
- Az egyes kapcsolatok erősségét a $w_1,..,w_n$ súlyok modellezik
- Ezek mellett van egy w_0 "bias" paraméter
- A neuront érő ingerek összességét aktivációnak hívjuk: $a = \sum_{1}^{n} w_i x_i + w_0$
- Egyszerűsítő jelöléssel, egy fix $x_0=1$ inputot felvéve: $\sum_{i=0}^{n} w_i x_i$
- Az *a* aktivációból az *o(a)* outputot az aktivációs függvény állítja elő
  - Ez eredeti modellben ez egy egyszerű küszöbölést végez:



- A modell tanulandó paraméterei a $w_0,..,w_n$ súlyok

A perceptron tanítása

- Rosenblatt nem csak feltalálta a perceptron modellt, de egy egyszerű tanítómódszert is megadott
  - Ez “perceptron tanulási szabály” néven ismert
- Két osztályt akarunk elválasztani
  - Az egyik osztály pontjain -1, a másikon 1 értékű kimenetet várunk
- A perceptron tanulási szabály
  - Iteratív algoritmus (végigmegy a tanítópéldákon, akár többször is)
  - Ha egy mintát helyesen osztályoz a rendszer, nem csinál semmit
  - Ha a minta osztályozása hibás, módosítja a súlyokat
  - Bizonyítható, hogy ha a két osztály lineárisan elválasztható, akkor véges lépésben jó megoldást talál
  - Azonban semmit sem garantál a lineárisan nem szétválasztható esetre
    - Emiatt inkább majd más módszereket fogunk preferálni

A perceptron tanulási szabály

- A Rosenblatt által javasolt tanítóalgoritmus:

inicializálás: véletlen súlyok választása
iteráció:
- engedjük be az egyes tanítópéldákat a perceptronba
- módosítsuk a súlyokat az alábbi szabály szerint:

$w_i = w_i + \Delta w_i \quad \Delta w_i = \eta(t-o)x_i$, ahol t a várt, o pedig a kapott output;

η : kis pozitív konstans (tanulási faktor)

az iterációt addig folytassuk, amíg szükségesnek látjuk (a már tanított mintákat akár újra felhasználva!!)

- Bizonyítjuk, hogy a módosítás után az aktiváció jó irányban változik (az adott példára):

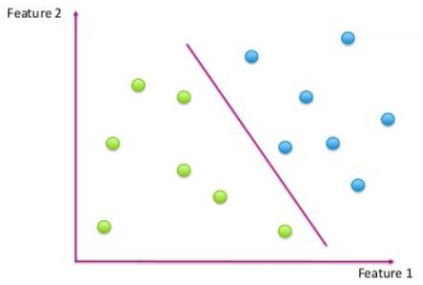
$$a_{uj} = \sum_{i=0}^{n}(w_i + \Delta w_i)x_i = \sum_{i=0}^{n} w_i x_i + \sum_{i=0}^{n} \Delta w_i x_i = a + \eta(t-o)\sum_{i=0}^{n} x_i^2$$

- Az aktiváció változásának előjelét (t-o) dönti el
  - Ha t=o, nincs változás
  - Ha t<o, az aktiváció csökken
  - Ha t>o, az aktiváció nő

A neuron kimenetének értelmezése

- Az $\sum_{i=0}^{n} x_i w_i = 0$ egy egyenes (több dimenzióban: hipersík) egyenlete
  - A neuron a súlyok által meghatározott hipersík mentén 0 aktivációs értéket ad
  - a hipersík egyik oldalán negatív, másik oldalán pozitív aktivációs értékeket ad
  - Így az aktivációs függvény az egyenes egyik oldalán -1-et, másik oldalán 1-et ad

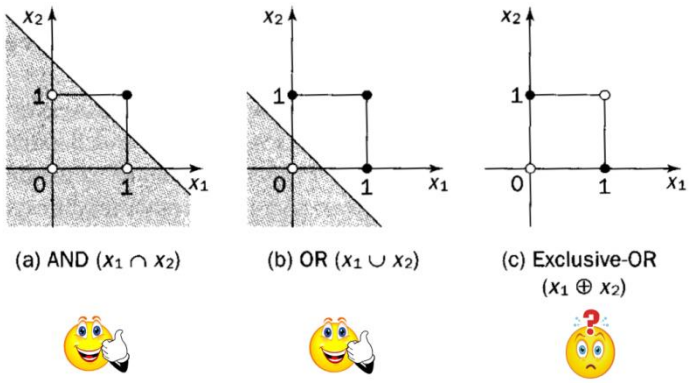
  - Azaz a perceptron a teret két résztérre képes osztani egy hipersík mentén
  - Vagyis két osztály pontjait csak akkor tudja elválasztani, ha azok egy hipersík két oldalára esnek



    - És a perceptron tanulási szabály is csak ebben az esetben működik (konvergál)

Egyetlen neuron reprezentációs ereje

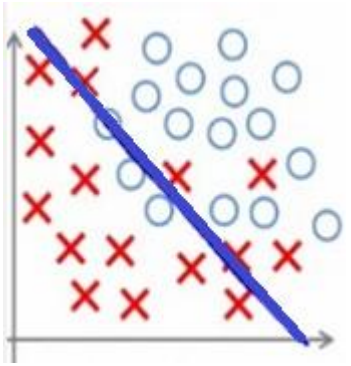
- Az egyenessel való elválasztás elég szerény reprezentációs erőt (tanulási képességet) jelent a gyakorlatban
  - Sajnos elég kevés valós feladat oldható meg ilyen egyszerűen…
  - Például az ÉS ill. VAGY művelet megtanulható, XOR nem:



  - Erre Minsky és Papert mutatott rá 1969-ben
  - Emiatt egész az 1980-as éveig hanyagolták a neuron-modell kutatását (ez volt az első „AI winter”)

Tanítás egy hibafüggvény optimalizálásával

- Tökéletes osztályozást egyetlen neuronnal sajnos csak akkor tudunk garantálni, ha a két osztály lineárisan elválasztható
  - És a perceptron tanulási szabály is csak ekkor működik
- Minket azért egy kis hibával való osztályozás és érdekelne…
  - Azaz amikor csak kevés pont esik a rossz oldalra…
  - Hogyan tudjuk a legkisebb hibájú lineáris osztályozást megtalálni?
  - Ehhez formalizálnunk kell, hogy mit értünk „kis hibán”



- Ehhez a módszercsaládhoz valahogy számszerűsítenünk kell az osztályozó hibáját egy függvény definiálásával
  - E függvényt többféleképp hívják: hibafüggvény, költségfüggvény, célfüggvény …
  - A függvény változói persze a neuron súlyai lesznek

A mean squared error (MSE) hibafüggvény

- A legegyszerűbb hibafüggvény az MSE hibafüggvény
  - Jelölje *t* a neuron kimenetén elvárt értéket („target”)
  - Jelölje *o* a neuron kimenetén kapott értéket („output”)
  - Ekkor *t* és *o* négyzetes eltérése $(t-o)^2$
- Természetesen a fenti hibaérték az össze tanítópéldára nézve érdekel bennünket, ezért vesszük a hibák átlagát a D tanító példahalmaz összes példája fölött
  - Ez lesz az átlagos négyzetes hiba (mean squared error, MSE):

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

  - Ne feledjük, hogy a hibafüggvény változói a neuron súlyai, hiszen az o output értéke a **w** súlyvektortól és az **x** inputvektortól függ (de az utóbbi adott, tehát csak az előbbin tudunk változtatni…)

Tanítás a hibafüggvény optimalizálásával

- Az optimális súlyértékeket a hibafüggvény optimalizálásával (jelen esetben minimalizálásával) fogjuk megtalálni
  - Ezzel a gépi tanulási feladat egy sokváltozós optimalizálási feladatba megy át
  - Vegyük észre, hogy a módszer nem csak a neuronhálók esetében működhet, hanem annál sokkal általánosabb
  - A legegyszerűbb esetekben a globális optimum zárt képlettel megadható
  - Bonyolultabb esetekben iteratív algoritmusokat fogunk használni
  - A legnehezebb esetekben ezek az iteratív algoritmusok is csak lokális optimumhelyet fognak garantálni, nem globálisat
- A hibafüggvény optimalizálásán alapuló módszer akkor is működik, amikor a perceptron tanulási szabály nem
- És kiterjeszthető lesz egyetlen neuronról neuronhálókra is
  - Ezért a továbbiakban ezt fogjuk használni, nem a perceptron szabályt

A “gradient descent” algoritmus

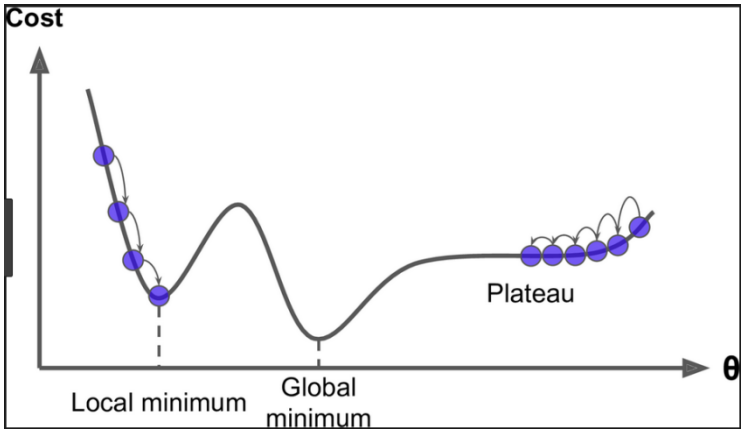
- A legegyszerűbb általános optimalizáló algoritmus sokváltozós függvényekre
- Tegyük fel hogy minimalizálni akarjuk az $E(\mathbf{w})$ sokváltozós függvényt
- A függvény deriváltja (sokváltozós esetben gradiensvektora):

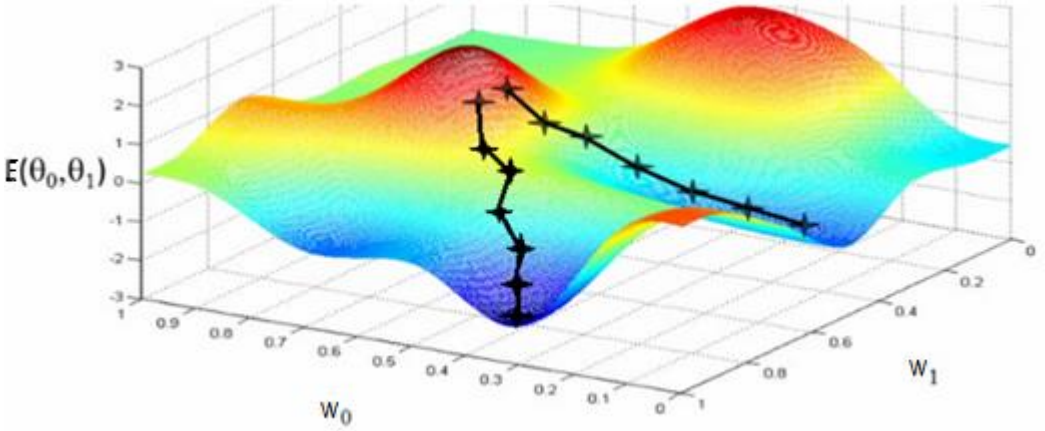
$$\nabla E(\mathbf{w}) = \left[\frac{\partial E}{\partial w_0}, \ldots, \frac{\partial E}{\partial w_n} \right]$$

  - Tudjuk, hogy a minimumpontban ez nulla
  - Tehát ki kellene számolnunk a gradienst, majd nullával egyenlővé tennünk
  - Sajnos sok gyakorlati esetben ennek megoldást túl nehéz formálisan levezetni
- A gradient descent algoritmus ennél egyszerűbb:
  - iteratívan csökkenti a célfüggvény értékét
  - mindig a legmeredekebb csökkenés irányába lép
  - amely irányt a gradiensvektor segítségével határoz meg

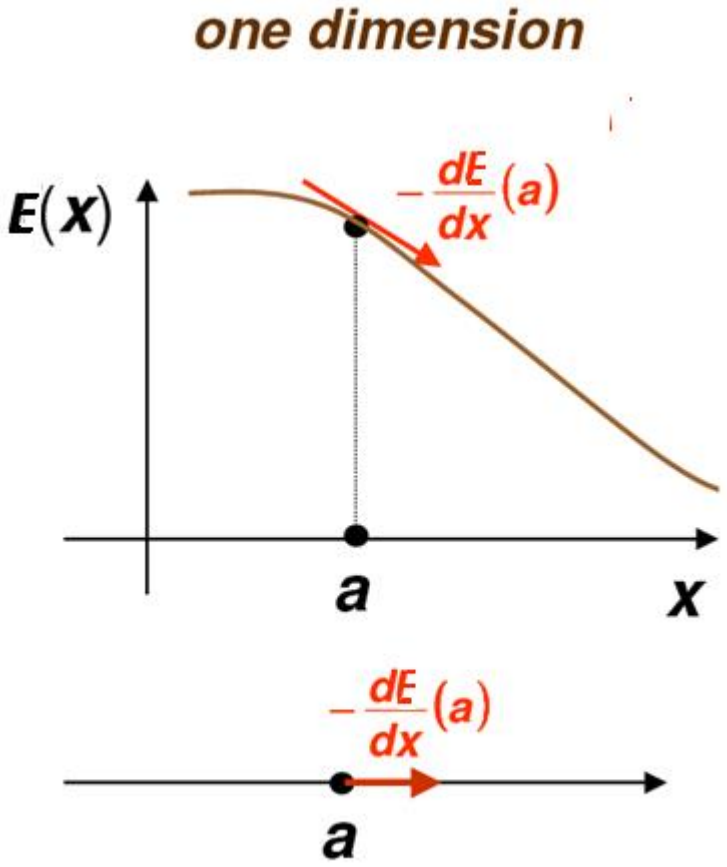
A gradient descent algoritmus (2)

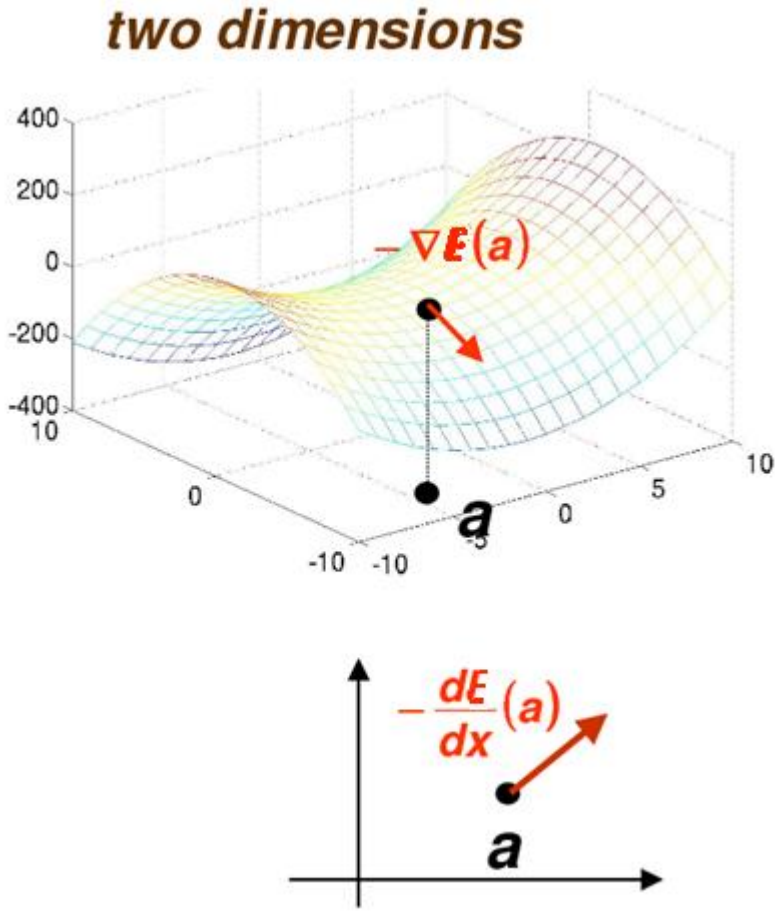
- A „hegymászó” algoritmus változata folytonos függvények esetére
  - A derivált alapján fogjuk eldönteni, hogy merre lépjünk a hibafelületen („gradiens módszer”) – gradiens-alapú csökkentés elve: a gradiensvektorral ellentétes irányba fogunk lépni
  - Véletlenszerű inicializálásból indulunk
  - A lépésközt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





A gradient descent szemléltetése





Gradient descent – az iteratív algoritmus

Gradient Descent for minimizing any function $J(\boldsymbol{x})$

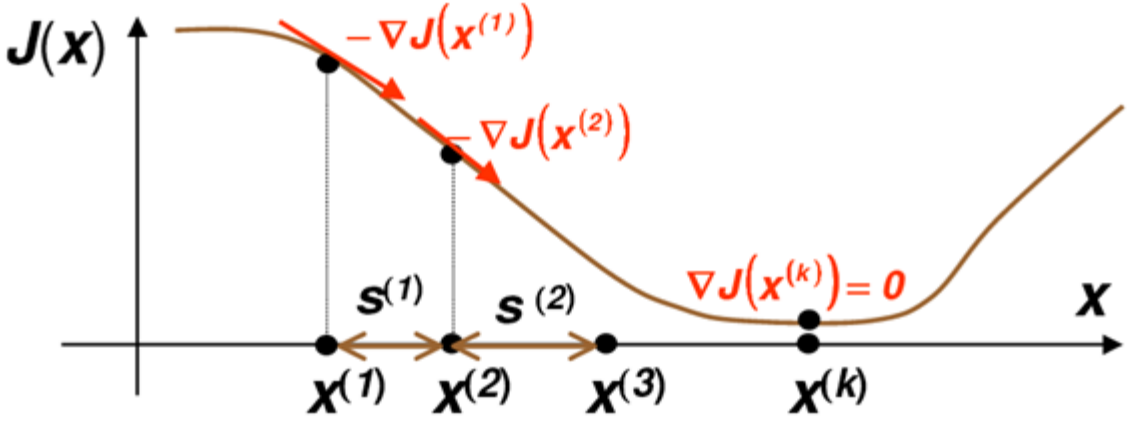
set $k = 1$ *and* $\boldsymbol{x}^{(1)}$ to some initial guess for the weight vector

while $\eta^{(k)} \left| \nabla J\left(\boldsymbol{x}^{(k)}\right) \right| > \varepsilon$

choose learning rate $\eta^{(k)}$

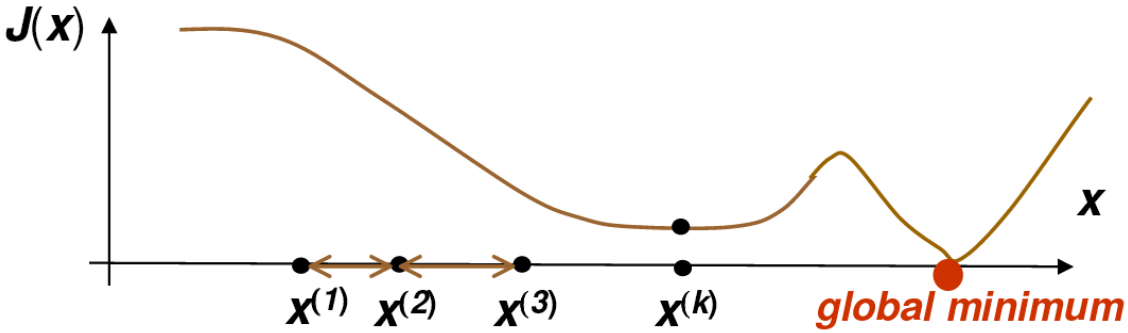
$$\boldsymbol{x}^{(k+1)} = \mathbf{x}^{(k)} - \eta^{(k)} \nabla J(\boldsymbol{x})$$ (*update rule*)

$$k = k + 1$$

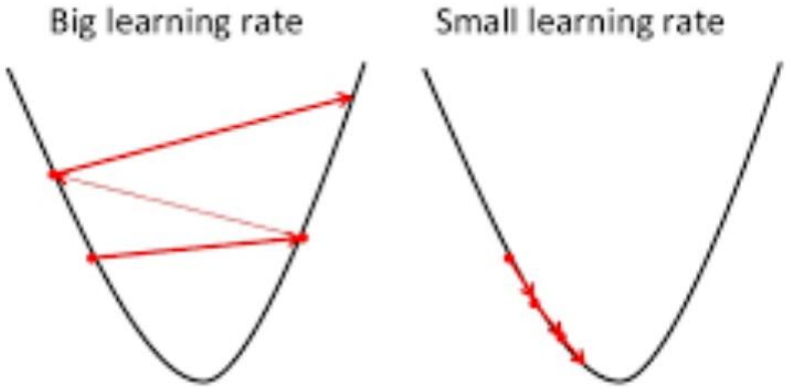


A gradient descent gyenge pontjai

- A gradient descent elakadhat lokális optimumban

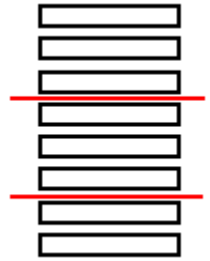


- Ha a tanulási ráta túl kicsi, az algoritmus nagyon lassan konvergál
- Ha a tanulási ráta túl nagy, nem találjuk meg az optimumot
  - A későbbiekben látunk majd heurisztikákat a tanulási ráta beállítására



A gradient descent változatai

- Az aktuális modell (paraméter-értékek) hibáját általában a hiba-függvény összes tanítópontban felvett értékének összegeként definiáljuk
  - Ezt hívjuk az algoritmus off-line avagy „batch” verziójának
- Azonban használhatjuk az algoritmus on-line vagy semi-batch verzióját is – ilyenkor az adatokat batch-ekre bontjuk
  - Ilyenkor a hibát egyetlen példán (on-line), vagy a példák egy részén (batch) számítjuk ki, majd frissítjük a paramétereket
  - Ez a változat a sztochasztikus gradient descent (SGD):
    - Mivel a hibát minden iterációban a példák más-más részhalmazán számoljuk, minden lépésben az eredeti hibafüggvény kicsit más verzióját optimalizáljuk (mivel az az *összes* példa fölött van definiálva)
    - Ez egy kis véletlenszerűséget visz az eljárásba
    - De ez a gyakorlatban nem árt, sőt inkább segíti a lokális optimumok elkerülését, és gyorsabb konvergenciát is eredményez

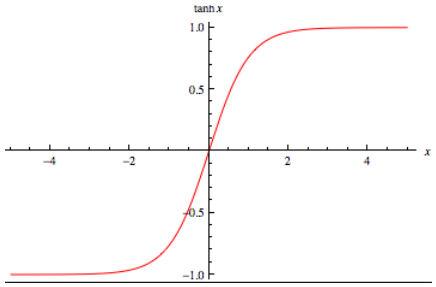


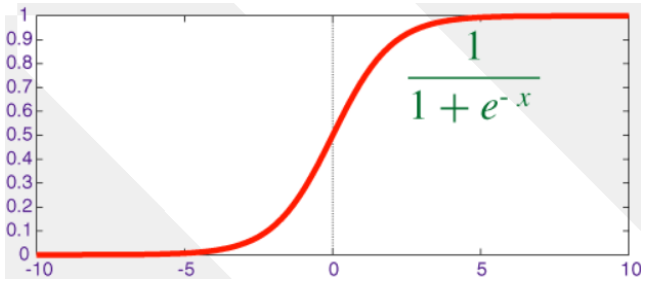
A sigmoid aktivációs függvény

- A gradient descent tanításhoz még egy problémát meg kell oldanunk:
  - Csak deriváltható függvényekkel működik, csakhogy az aktivációs függvényünk lépcsős, így nem deriválható:

$$o = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$$

  - Ezért lecseréljük a *tanh* függvényre, ami közelítőleg hasonló alakú:





  - A gyakorlatban inkább a sigmoid függvény terjedt el (jobb oldalon), ami ugyanolyan alakú, de értékkészlete (-1,1) helyett (0,1)
    - Mindkettő deriválható, így már nincs akadálya a gradiens alapú tanításnak

A sigmoid aktivációs függvény (2)

- Az MSE hibafüggvény tetszőleges valós *t* célértékek és o kimeneti értékek mellett értelmezve van, nem csak a -1, 1 értékek esetére
- A lépcsős aktivációs függvény lecserélésével a neuron kimenete nem csak a -1, 1 értékek egyike lehet, hanem a (0,1) intervallumból bármely érték
- Tehát – elvileg – célértéknek is megadhatunk a (0,1) intervallumból tetszőleges számot
- Folytonos célértékek esetén osztályozás helyett regressziós feladatról beszélünk
- Mivel a sigmoid aktivációs függvény a kimeneti értéket a (0,1) intervallumra korlátozza, regressziós feladat esetében inkább lineáris aktivációs függvényt használunk ($f(x)=x$)

Neuron tanítása gradient descent algoritmussal

- Egyetlen (sigmoid aktivációs) neuron MSE hibafüggvénye:

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

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

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

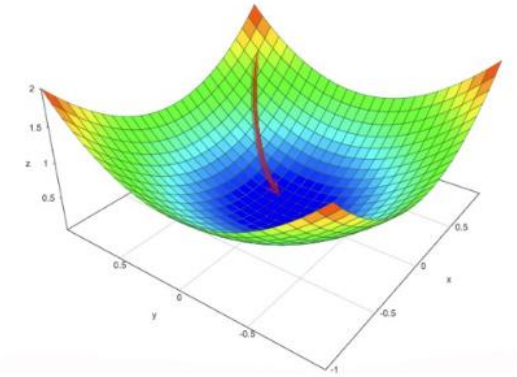
- Ezeket egymásba ágyazva kapjuk meg, hogyan függ E(w) egy konkrét w_j-től
- Deriválás: ezek deriváltjai láncszabállyal összerakva

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

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

Neuron tanítása gradient descent algoritmussal (2)

- Lineáris aktivációs függvény esetén az MSE hibafelület kvadratikus, egyetlen globális optimummal
  - A gradient descent algoritmus garantáltan megtalálja a globális optimumot (ha a learning rate fokozatos csökkentésére odafigyelünk, lásd később)



- Sigmoid aktivációs függvény esetén a hibafelület konvexitása nem feltétlenül áll fenn, tehát elvi garancia csak lokális optimum megtalálására van, de a gyakorlati tapasztalatok szerint az algoritmus ekkor is jó eredményeket ad

KÖSZÖNÖM A FIGYELMET!

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