



Suntem „dotati” cu aproximativ 100 de miliarde de celule numite neuroni, dintre care doar 11 miliarde se afla in cortexul cerebral. Neuronii au o mare abilitate de a receptiona si a transmite semnale electro-chimice. Acest sistem este asemanator cu portile logice dintr-un calculator. Caracteristicile neuronilor permit trimiterea unor semnale pe distante mari, chiar si de pana la cativa metri, trimitand mesajul de la unul la altul.

Santiago Ramon Cajal a fost anatomistul care a introdus conceptul de neuron, ca unitate principala a sistemului nervos. Santiago Cajal a aratat ca neuronii sunt capabili de a comunica intre ei. O contributie fundamentala la cunoasterea celulei nervoase in stare normala si patologica a constituit-o la vremea sa, monografia lui Gheorghe Marinescu, „La cellule nerveuse (Paris, 1909)”.

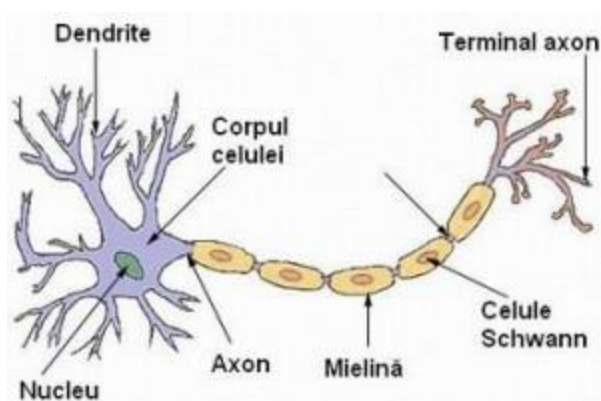


Neuronul

Neuronii sunt celule remarcabile si au proprietatea de a propaga semnale extrem de rapid pe distante mari. Fac acest lucru prin generarea unor pulsuri electrice numite potential de actiune. Neuronii isi schimba starea trimitand secvente de pulsuri electrice in diverse sabloane temporale in prezenta stimulilor exteriori precum lumina, sunet sau

gust. Este cunoscut faptul ca informatia stimulilor este codificata in sablonul potentialului de actiune si este trimisa in tot creierul. S-a descoperit ca muschii sunt pusi in miscare de acest potential de actiune si de neuronii motori ce servesc pentru a transforma potentialul de actiune generat de creier in contractii ale muschilor ce permit animalelor sa interactioneze cu mediul inconjurator, adesea ca un raspuns la stimulii pe care ii percep.

Neuronii au diferite forme si marimi, dar toti au aceeasi structura de baza. Au un nucleu central situat intr-o portiune aproximativ sferica a neuronului denumita corp celular. Din corpul celular se desprind un numar de prelungiri fine, ramificate. Acestea sunt denumite dendrite. Din celula se desprinde o fibra unica, lunga, denumita axon, principala fibra care asigura conducerea semnalului intr-un nerv.



Pentru a mari viteza de transmitere a semnalelor, axonii au un invelis de mielina, care are rol si de izolator. Fara invelisul de mielina neuronul nu poate functiona corespunzator. Acest lucru este demonstrat de efectele devastatoare de afectiunea numita scleroza. Cand un semnal atinge butonii axonului, acesta poate, in anumite imprejurari, sa traverseze sinapsele catre dendritele unui alt neuron adiacent si astfel sa se propage in continuare.

Neuronii nu sunt singurele tipuri de celule care se intalnesc in sistemul nervos. Celulele denumite nevroglii sau celulele gliale sunt prezente in numar mare in sistemul nervos central, iar celulele Schwann in sistemul periferic. Ambele tipuri leaga, protejeaza, hranesc si ofera suport neuronilor.

Neuronii sunt formati din trei regiuni, una receptoare, una conducatoare, si una efectoare.

Regiunea receptoare receptioneaza si proceseaza informatia. Aceasta este formata din dendrite si soma. Regiunea conductoare leaga regiunea receptoare de cea efectoare si este formata din portiunea axonului de la locul in care acesta iese din corpul celular. Aici au loc si potentialele de actiune prin sumarea potentialelor locale. Si in final, regiunea efectuare, informatia, sau potentialul de actiune, este recodificata aici sub forma chimica prin neurotransmitatori si transmisa prin sinapsa regiunii receptoare a urmatorului neuron.

Semnalele care vin de la alti neuroni pot face – sau nu – un neuron sa transmita mai departe semnalul. Neuronul calculeaza semnalul pe baza semnalelor inhibatoare si excitatoare si decide daca va „actiona” sau nu. Neuronii au de obicei doua stari. Cand se afla in stare pasiva acesta are o frecventa redusa, insa cand acesta primeste un anumit semnal, frecventa creste. Un singur neuron are mai multe intrari excitatorii si inhibitorii.

## Proprietatile neuronilor

Neuronii au proprietati de conductibilitate, excitabilitate, degenerescenta si regenerare.

Conductibilitatea este proprietatea de a conduce impulsurile. Aceasta conducere se realizeaza diferit in fibrele mielinice si amielinice, datorita diferentei de grosime a lor.

Regenerarea este o alta proprietate a neuronului, fiind capabil sa se refaca dupa anumite lezari, in timp ce degenerescenta se refera la degradarea neuronului in conditii de lezare serioasa a axonului.

## Conectivitatea neuronilor

Neuronii comunica între ei prin sinapse, putând avea peste 1000 de ramificații, făcând conexiuni cu alte zeci de mii de celule. Sinapsele neuronilor pot fi de două tipuri, excitatorii sau inhibitorii.

## Potentialul de acțiune

Potentialul de acțiune apare într-un neuron și reprezintă modificarea potentialului de repaus, după stimularea supraliminală a celulei. În „Neurofiziologia sistemelor senzitivo-senzoriale”, de A. Olteanu, este explicat faptul că potentialul de acțiune se produce datorită creșterii rapide a permeabilității pentru  $\text{Na}^+$  de aproximativ 5000 de ori. Membrana plasmatică prezintă canale ionice voltaj-dependente și canale ionice ligand-dependente.

La potentialul de  $-70\text{mV}$  canalul de  $\text{Na}^+$  este închis, iar când potentialul crește la  $-65\text{mV}$  el se deschide și ioni de  $\text{Na}^+$  patrund în celulă, rezultând în depolarizarea neuronului.

Diferența de potential electric dintre interiorul și exteriorul celulei se reduce până când la un moment dat se inversează polarizările: pozitivă în interior și negativă în exterior.

Valoarea care trece de  $0\text{mV}$  se numește overshoot ( $+35\text{mV}$ ). Creșterea și scăderea rapidă a potentialului se numește spike potential sau potential de varf. După aceasta se revine la potentialul de repaus. Postpotentialul negativ (sau postdepolarizare) este peste valoarea de repaus. După ce s-a atins valoarea de repaus, potentialul scade și se află puțin sub valoarea de repaus, timp de 40-50 ms, ceea ce constituie postpotentialul pozitiv (sau posthiperpolarizare).

Trecerea ionilor de  $\text{Na}^+$  prin membrana se face pasiv și este dependentă exclusiv de gradientul de concentrație. În concluzie, geneza impulsurilor nervoase nu consumă energie.

Cand membrana este stimulata subliminal, neaparand acea diferenta de 15mV, nu se produce un potential de actiune dar creste sensibilitatea membranei, ceea ce rezulta intr-un potential local.

## Cum se retin informatiile?

Creierul inregistreaza un eveniment intarind conexiunile intre grupuri de neuroni care participa in „codificarea” experientei. Acest sablon de conexiuni constituie inregistrarea evenimentului, cunoscut ca engrama. O engrama este un fel de urma lasata de un excitant asupra sistemului nervos. Timpul joaca un rol important in procesul memoriei. Engramele care nu le mai folosim se slabesc si se incetoseaza. Experientele din trecut se sterg, unele rapid, altele imperceptibil. Uitarea, desi un lucru frustrant, este importanta. Engramele care nu le folosim niciodata sunt primele care se uita. Uitarea este un raspuns economic al memoriei, influentat de mediul inconjurator in care traim. Insa exista si memorie pe termen lung, unde pastram informatii pe care nu le uitam niciodata, de exemplu: numele propriu, anul de nastere, adresa, etc. Modul cum functioneaza nu este inteles in totalitate.

## Introducere în rețele neuronale – Teorie și aplicații



Domeniul rețelelor neuronale este unul foarte activ la ora actuală, lucru oarecum surprinzător, ținând cont că unele idei au peste 60 de ani. Chiar și așa, doar în ultimii ani, acest domeniu a devenit cu adevărat atractiv, asta în special datorită puterii crescute de calcul ce-i la îndemâna tuturor. E o un domeniu cu strânse legături în deep learning, big data și machine learning așa că unele aspecte tratate aici se regăsesc, într-o formă sau alta, și în domeniile menționate anterior.

## Cuprins

1. [Ce este un neuron?](#)
2. [Ce este o rețea neuronală?](#)
3. [Formulare matematică](#)
4. [Tipuri de rețele](#)
5. [Modalități de învățare](#)
  1. [Învățarea rețelelor neuronale](#)
    1. [Regresia liniară](#)
    2. [Clasificatorul liniar \(perceptronul\)](#)
    3. [Învățarea competitivă](#)
  2. [Retele Kohonen](#)
6. [Entropia încrucișată](#)

# 1. Ce este un neuron?

În biologie, neuronul este o celulă adaptată la recepționarea și transmiterea informației. Un număr suficient de mare de astfel de celule pot crea structuri de complexitate deosebită, exemplul pertinent fiind creierul nostru. De aici a pornit ideea rețelelor neuronale artificiale (ANN). Deși mult mai simple, ele oferă o serie de rezultate remarcabile ce sunt similare cu procesele aparent naturale.

Warren McCulloch și Walter Pits au creat în anii '40 prima abstractizare al acestui mecanism complex. La început, ei au privit acest neuron ca pe-o funcție. Deși e o abordare foarte simplistă, măcar oferă o primă abstractizare matematică a neuronului. Astfel:

1. **Sinapsele** pe care un neuron le făcea prin dendrite cu alți neuroni au devenit **INTRĂRI**;
2. **Corpul neuronului** a devenit un **SUMATOR + o FUNCȚIE DE ACTIVARE**(fc. de TRANSFER)
3. **Axonul** a devenit **IEȘIREA**
4. Pentru a reprezenta caracteristici neliniare simple s-a introdus noțiunea de **bias**

## Despre bias

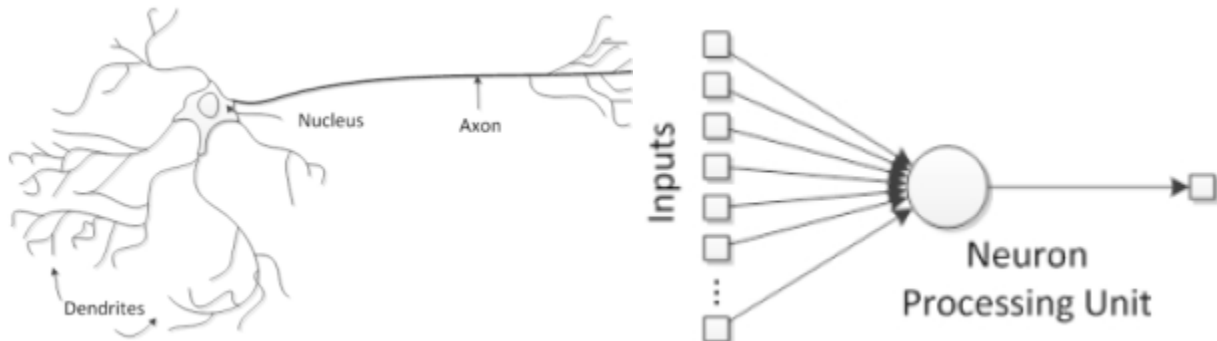
Ca o paranteză, în română, bias s-ar traduce ca parțialitate, care urmărește un anumit scop (tendențiozitate). Un exemplu e pareidolia – acea trăsătură a psihicului nostru ce ne face să vedem fețe umane...peste tot: vedem un craniu pe Lună, o față în spuma de cafea etc. Este un soi de reacție instinctivă, de bază, la tiparele pe care noi le găsim în structuri aleatoare.

Mai concret și mai tehnic înseamnă să ai un răspuns de bază la o anumită intrare. Să dau câteva exemple:

- Electroniștii rar vorbesc de componenta constantă a unui semnal (ei rezolvă de la început asta cu niște condensatoare). Pentru aceștia e mai importantă partea ce variază tot timpul. De aceea, în calcule, se scade componenta pasivă și rămânem cu o serie de sinusoidale (facem transformata Fourier și e de bine).
- Dacă lucrezi în Data Science, prietenii matematicieni o să îți spună să scazi media dintr-o distribuție pentru a o translați în origine.
- Băieții ce fac memorii DRAM (cele volatile cu condensator) îți vor spune că celula de memorie nu trage la 0 sau la 1 linia de citire atunci când citim bit-ul ci numai modifică puțin valoarea medie de 0.5 ... un soi de +0.5 ori -0.5 pe motiv ca acel condensator e microscopic și prăpădit și nu poate influența prea mult decât linia (-:

În cazul rețelelor neuronale am vrea să rezolvăm situația când nu avem intrări ori valorile variază, dar foarte important, variază în jurul unei valori, să-i zicem **b**.

Un exemplu simplu ar fi ca la intrările 1, 2, 3, 4 funcția noastră să returneze doar valoarea  $\beta$ . O funcție simplă ar fi  $f(0 * x + b) = \beta$ . Pentru noi,  $\beta$  reprezintă o probabilitate, iar  $b$  intrarea standard.



## 2. Ce este o rețea neuronală?

- Analogie cu creierul uman – mai mulți neuroni interconectați formează o rețea neuronală.
- Este compusă dintr-un număr mare de elemente interconectate (modelul matematic al neuronului) care lucrează împreună pentru rezolvarea unei anumite probleme
- Poate fi îmbunătățită prin învățare. Poate învăța funcții noi pe baza unor informații primite.

Tocmai aceste caracteristici care derivă și de la o structură simplă ca cea a lui McCulloch permite o serie de aplicații foarte variate precum:

- Pot fi antrenate pentru recunoașterea scrisului de mână
- Google folosește un algoritm bazat pe rețele neuronale pentru returnarea unor imagini pe baza unor cuvinte date ca intrare la căutare
- Microsoft a dezvoltat un sistem de rețele neuronale care ajută la convertirea limbii engleze în chineză
- Largă aplicabilitate în medicină, afaceri, matematică, știință, business etc
- O rețea neuronală poate fi învățată pe baza unui volum mare de date și îmbunătățită iterativ pentru realizarea altor funcții. (Folosirea algoritmilor bazați pe rețele neuronale în domeniul BigData)

## 3. Formalismul matematic

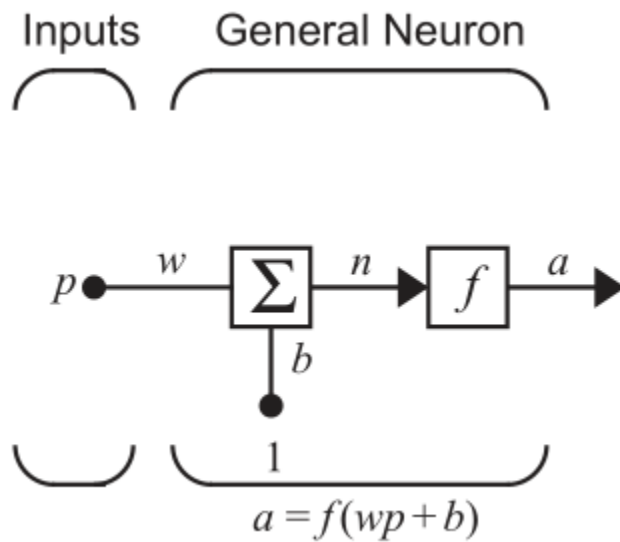
Înainte de a prezenta o aplicație a acestei rețele cred că e mai bine să punem în lumină aparatul matematic necesar. Voi încerca să nu intru în detalii și voi pune accent pe intuiție în detrimentul specificității matematice. De asemenea, cred că adesea, lumea trece cu vederea aceste "detalii" și astfel se pierde înțelegerea per ansamblu a sistemului. Chiar nu îmi doresc să popularizez ideea că **"Machine Learning e**



***simplu...îi dai niște date, run și gata*** – acesta e doar efectul a 70 de ani de muncă coagulați într-o serie de librării care ne permit să scriem 20 de linii de cod și să recunoaștem câte degetele sunt în vreo poză(acesta e exemplul istoric). Faptul că sunt necesare librării relativ mari pentru lucruri așa simple are o serie de implicații filozofice...surpinse bine de paradoxul lui Moravec.

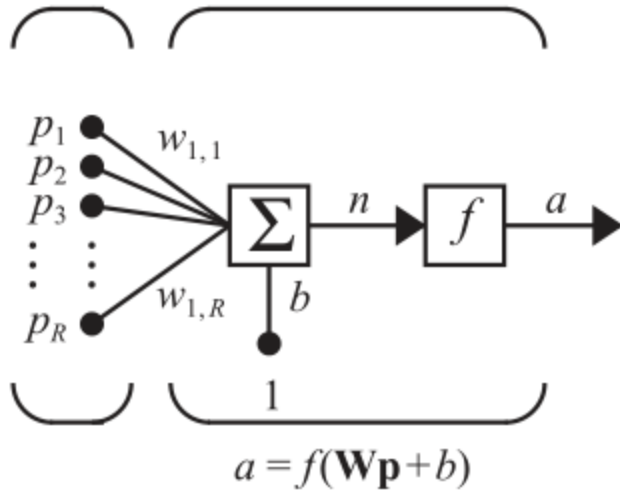
### 3.1 Tipuri de rețele

#### 1. Neuronul cu o singură intrare

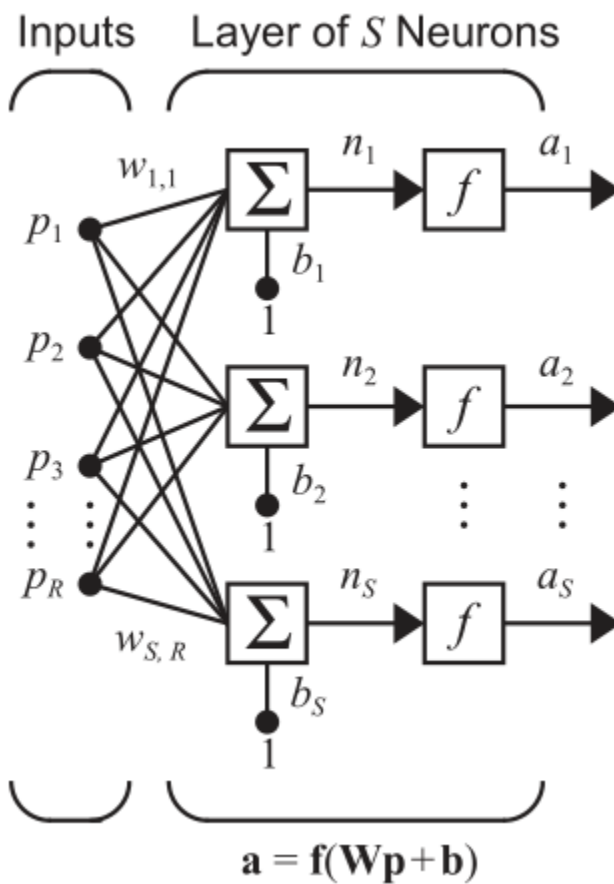


2. **Neuronul cu intrări multiple.** Aici  $\mathbf{p}$  este un vector, iar  $\mathbf{W}$  este o matrice cu un singur rând.

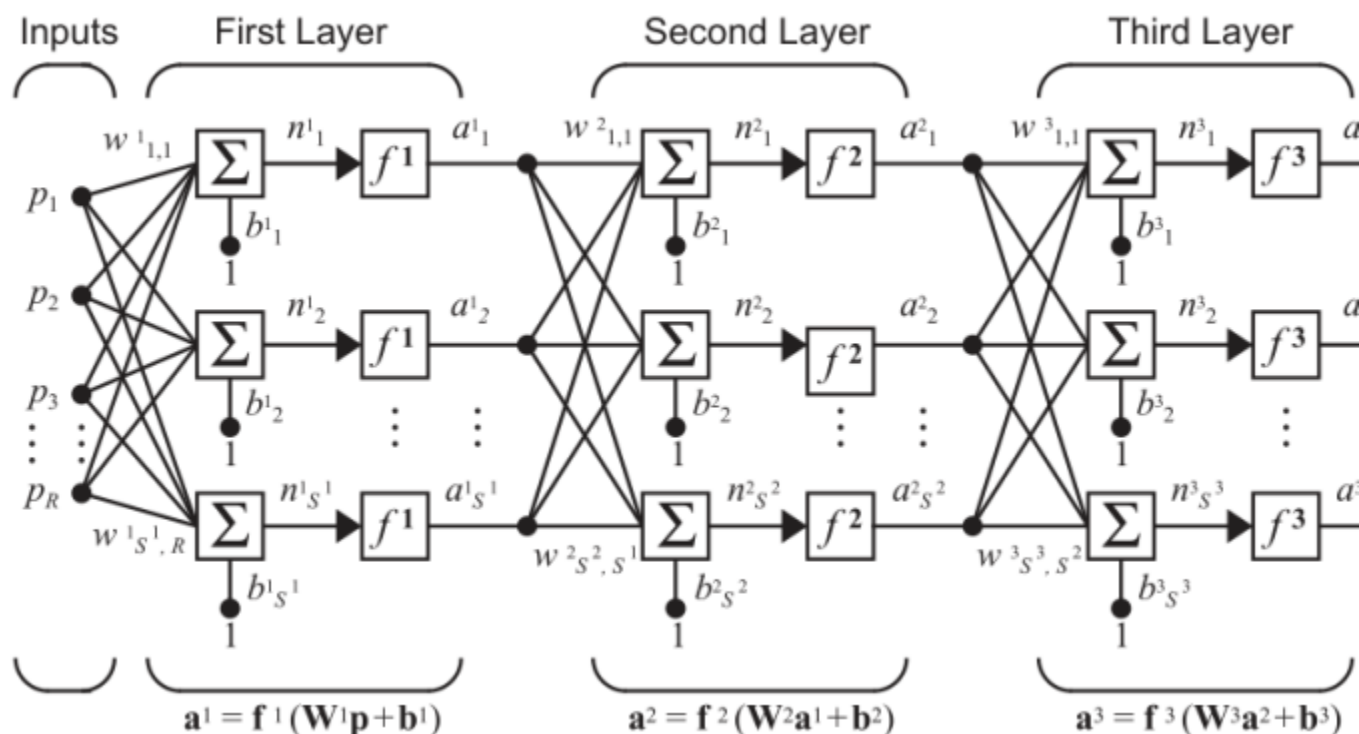
### Inputs Multiple-Input Neuron



3. **Rețea cu un singur nivel de neuroni.** Aici  $\mathbf{p}$  este un vector, iar  $\mathbf{W}$  este o matrice cu  $S$  linii și  $R$  coloane.



#### 4. Rețea cu mai multe nivele de neuroni



5.

$$y_o = f_o \left( \sum_{i=1}^{n_{h_l}} w_{li} f_i \left( \sum_{j=1}^{n_{h_{l-1}}} w_{lj} f_j \left( \sum_{k=1}^{n_{h_{l-2}}} w_{jk} f_k(\dots) + b_j \right) + b_l \right) + b_o \right)$$

Observăm că deși poate părea complicat, formula finală e doar un efect combinat al rețelelor cu un singur nivel extins la mai multe.

### 3.2. Funcții de activare

Plecând de la ideea că un neuron poate fi reprezentat de un sumator și o funcție, atunci o rețea neuronală va deveni o mulțime de funcții interconectate. Este important totuși să definim tipul de funcții folosite.

Aceste funcții reprezintă filtrele prin care va trece informația. Vrem totuși un set de funcții cu proprietăți specifice, fiindcă de ele depinde cum trebuie să modificăm ponderile, modificarea acestora reprezentând învățarea rețelei neuronale. Modificarea funcțiilor implică să le variem în timp, deci să ne jucăm și cu derivatele lor. Deși ruptă puțin din context, ideea învățării rețelelor neuronale prin metoda gradientului chiar este o metodă consistentă și este folosită adesea la antrenare.

Voi detalia acum două funcții de activare foarte populare, cea mai importantă fiind **sigmoida**. Înainte de a vorbi de sigmoidă, voi încerca să surprind modul în care,

istoric, această funcție a luat naștere, ca o generalizare naturală a unei funcții mult mai simple și anume funcția lui Heavyside (folosită la perceptroni). Vom vedea care sunt limitările ei, pentru a remarca apoi, ce frumos se astupă acel gol cu sigmoida.

**Bucata gri ce începe aici poate fi omisă și este mai mult țintită spre pasionații de mate.**

Să facem o paranteză și să ne gândim la populații... așa s-a gândit și un tip pe nume **Pierre François Verhulst** care a vrut să găsească modelul matematic la aceste creșteri. La ce s-a gândit e chiar interesant...

Să zicem că avem o populație cu **K** membri. Copiii care apar în această populație sigur depind de numărul de indivizi din ea (logic nu-i așa). Cu cât **K** e mai mare... mai mulți copii și vice versa. Dar cine sunt copiii? Dacă presupunem că nimeni nu moare atunci copiii reprezintă chiar variația populației **K**. Deci:

Nr. de copii  $\propto$  Populația totală **K**

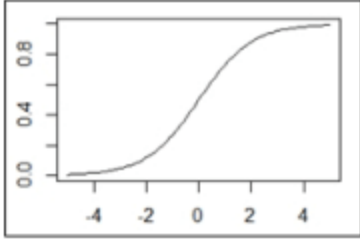
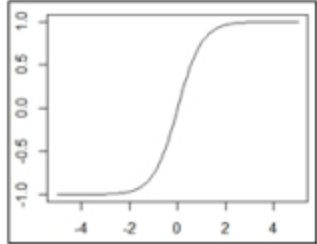
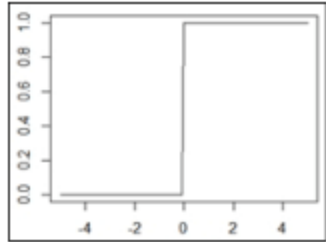
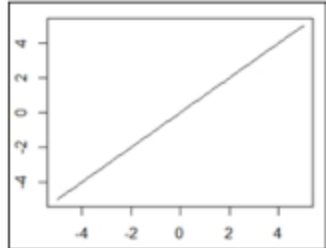
Cu soluția  $K = Ce^{\alpha K}$

Deși destul de bun, acest model nu prea reflectă realitatea. Populația în acest caz ar crește imediat spre valori foarte mari, iar asta nu prea pare să fie în concordanță cu realitățile. Să schimbăm puțin problema și să ne gândim altfel... să zicem că populația noastră va fi **P**, iar populația maximă ce poate fi suportată de mediu este **K** (ca și cum am zice că pe Pământ nu pot să trăiască mai mult de 10 miliarde de oameni și basta). Pare o idee simplă și chiar este... da cum rămâne cu variația populației și implicit cu copiii? Păi, intuitiv, când populația e mică... numărul de copii contează foarte mult și ne așteptăm ca populația să crească semnificativ (o rată mare de creștere). Pe măsură ce ne apropiem de limita mediului... populația ar trebui să nu mai poată crește, adică rata de creștere a populației să scadă, iar populația să rămână la valoarea maximă (10 miliarde în cazul nostru). Matematic:

Variația populației  $\propto K - P \Rightarrow$

$$-\frac{dP}{P^2 dt} + \tau K \frac{1}{P} - \tau = 0 \quad (1)$$

Astfel de funcții (se mai numesc funcții logistice) îs tare bune și pentru rețele neuronale fiindcă mărginesc rezultatele – le pun într-o gamă cât de cât controlabilă de valori.

Function	Equation	Chart
Sigmoid	$f(x) = \frac{1}{1 + e^{-x}}$	
Hyperbolic tangent	$f(x) = \frac{1 - e^{-x}}{1 + e^{-x}}$	
Hard limiting threshold	$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	
Linear	$f(x) = x$	

După cum spuneam, să vedem de ce funcția lui Heaveside(funcția treapta) de mai jos poate avea anumite limitări, în special în problemele cu un anumit grad de incertitudine (cum sunt majoritatea de altfel).

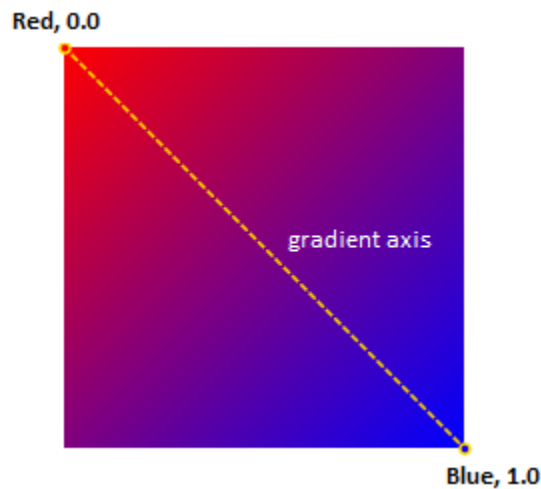
$$\text{iesirea} = \begin{cases} 0, & \text{daca } wx + b \leq 0 \\ 1, & \text{daca } wx + b > 0 \end{cases}$$

Exemplul constă într-o rețea neuronală capabilă să recunoască cifre. Să presupunem că folosim un MLP, iar la un moment dat ne vom dori să recunoaștem cifra 6. Rețeaua cu perceptroni simpli va învăța modificând ponderile până când răspunsul rețelei e cel așteptat – 6. Problema va fi cu celelalte numere, rețeaua probabil fiind destabilizată. Astfel de rețele MLP sunt rigide, tocmai datorită modului de funcționare a perceptronului. Nefiind decât 2 stări distincte, trecerea de la 0 la 1 s-a făcut la o schimbare probabil mică a ponderilor. Din acest motiv se folosesc neuronii sigmoizi, ce au sigmoida ca funcție de activare. Ea poate acoperi întreg intervalul  $[0,1]$ , iar astfel se

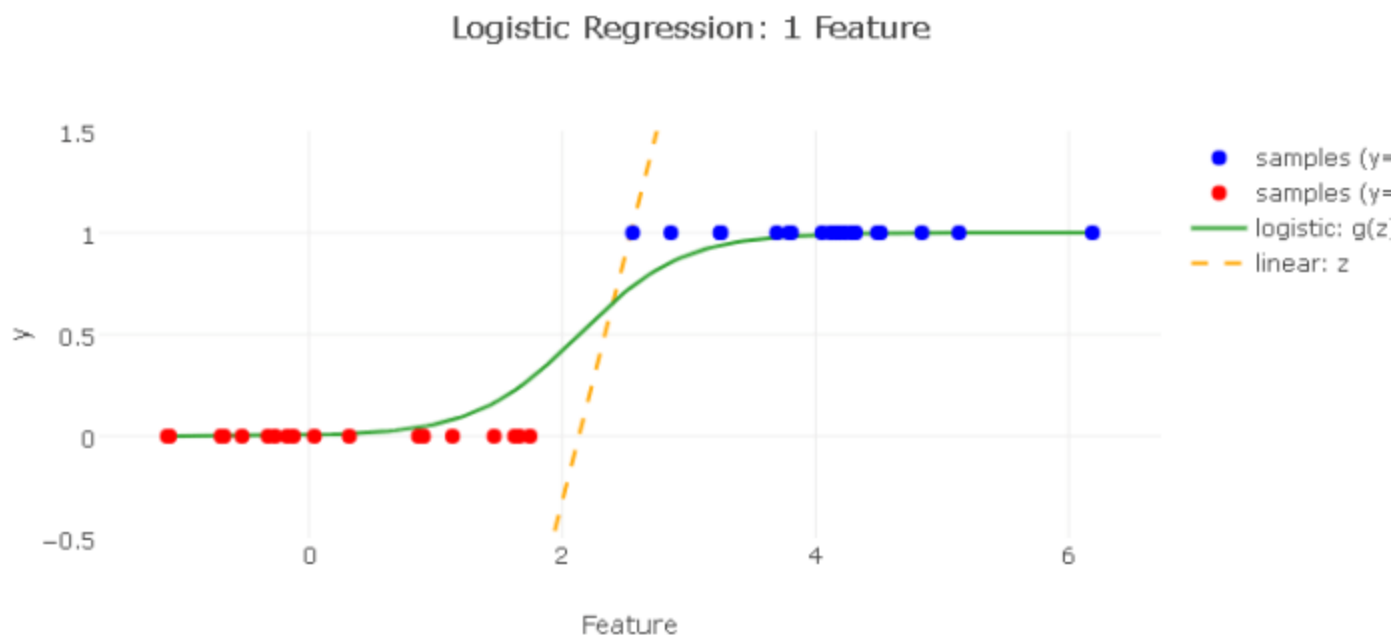
asigură o relație reciprocă între ponderi și intrări – variații mici la ponderi vor induce variații mici la ieșiri.

Să fim acum puțin mai specifici. Vom dezvolta puțin rezultatul obținut în căsuța matematică, cel cu distribuția populațiilor.

Se poate observa din graficul funcției sigmoide de mai sus că am putea s-o folosim cu succes pentru a defini o probabilitate. Asta, fiindcă funcția e mărginită între 0 și 1, exact ca și în cazul unei probabilități. Pentru a vedea mai bine, să luăm următoarea poză:



Am vrea o funcție care să ne spună, pe axa gradientului de culoare, cu ce probabilitate avem o anumită culoare. Deci suntem precum un calculator, orbi, dar mergem pe acea axă și vrem ca la fiecare pas ce-l facem să zicem, cu o anumită probabilitate, culoarea. Funcția sigmoidă e numai bună pentru asta. Dacă luăm acei pași și îi punem pe funcția noastră, am obține ceva de genul:



Putea fi și invers, roșu la 1 și albastru la 0, dar așa am vrut eu. Adesea, lumea preferă totuși să zică următoarea chestie: Care e probabilitatea ca acest punct să fie de culoare roșie? (sau albastră). Ce e important aici e că menționez o singură culoare. În poza de mai sus, sunt cumva ambele acolo, pe același grafic. Vreau să le separ în două grafice (o să înțelegem imediat de ce). Soluția e simplă: facem simetricul funcției de mai sus, față de y, și obținem probabilitatea pentru cealaltă clasă.

$$Pr(Y_i = \text{rosu}) = \frac{e^{-\beta X_i}}{1 + e^{-\beta X_i}}$$

$$Pr(Y_i = \text{albastru}) = 1 - Pr(Y_i = \text{rosu}) = \frac{1}{1 + e^{-\beta X_i}}$$

Se merită să scriem clasificatorul binar cu două funcții, fiindcă putem intui cum să generalizăm pentru mai multe clase. Această nouă funcție se va numi **softmax**, iar în cazul particular cu două clase, **softmax** se va reduce la **sigmoid**. Funcția softmax arată astfel:

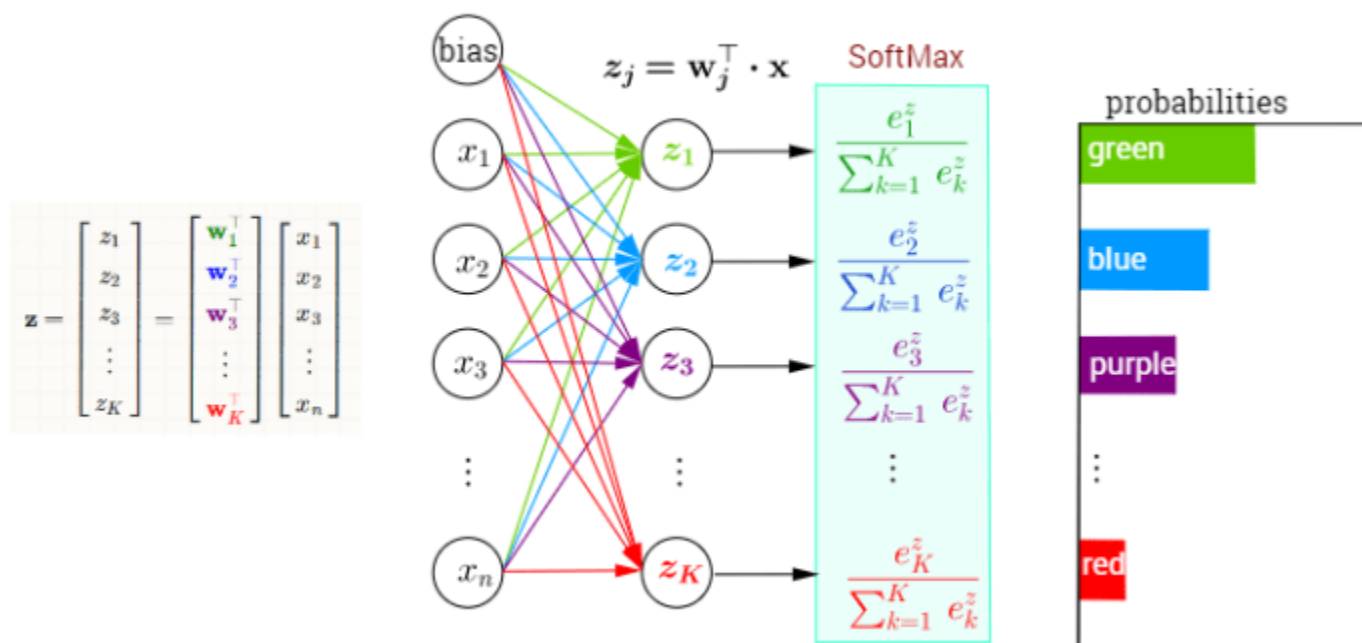
$$Pr(Y_i = k) = \frac{e^{\beta_k X_i}}{\sum_{c=0}^K e^{\beta_c X_i}}$$

Vedem că pentru cazul particular  $n=2$  ajungem la sigmoidă:

Vedem avantajul scrierii de mai sus...acum putem lucra cu mai multe clase, deci mai multe culori...gata cu clasificatorii binari.

Funcția ar avea acum ieșiri de genul.

## Multi-Class Classification with NN and SoftMax Function



Ba mai mult de atât, fiind la mijloc niște exponențiale, pe măsură ce rețeaua devine mai bună o clasă va deveni din ce în ce în ce mai probabilă, iar celelalte vor fi exponențial mai mici. “Efectul” învățării seamănă mult cu efectul micșorării deviației standard din distribuția lui Gauss – clopotul devine mai ascuțit, iar un set tot mai redus de puncte din jurul mediei chiar ajung să fie apropiate de medie => crește PRECIZIA. La clasificatori vorbim de acuratețe, dar

## 4. Tipuri de rețele

Taxonomiile din toate cărțile de specialitate se fac după tipul conexiunilor sau după direcția semnalului în rețea.

- Tipul conexiunilor făcute de neuroni
  - Monolayer – aici avem de obicei doar intrări și ieșiri ce formează un strat
  - Multilayer – avem Input, Hidden și Output Layer, precum am putut vedea la modelul matematic, secțiunea 3.1., punctul 4.
- Direcția semnalului
  - Rețele feedforward – care sunt rețele multilayer unde informația “curge” într-o singură direcție (în teoria sistemelor, analogia ar fi sisteme fără reacție).
  - Rețele feedback – unde există reacție, adică legături dintr-un strat superior la altele anterioare.



## 5. Modalități de învățare a rețelelor neuronale

Învățarea automată, unul din sub-domeniile de bază ale Inteligenței Artificiale, se preocupă cu dezvoltarea de algoritmi și metode ce permit unui sistem informatic să învețe date, reguli, chiar algoritmi.

Privite în ansamblu, mecanismele de învățare a rețelelor neuronale se pot împărți în două clase:

- Supervised – Fiindcă vorbim de un tip de învățare inductivă ce pleacă de la un set de exemple de instanțe ale problemei și formează o funcție de evaluare (șablon) care să permită clasificarea (rezolvarea) unor instanțe noi. Învățarea este supervizată în sensul că setul de exemple este dat împreună cu clasificarea lor corectă.
- Unsupervised – În acest caz setul de antrenare conține doar date de intrare
  - Exemplu: gruparea datelor în categorii în funcție de similitudinea dintre ele
  - Se bazează pe proprietățile statistice ale datelor
  - Nu se bazează pe conceptul de funcție de eroare ci pe cel de calitate a modelului extras din date, care trebuie maximizat prin procesul de antrenare.

### 5.1. Învățarea rețelelor neuronale

Înțelegem arhitectura neuronului, am vorbit despre funcții de activare, dar nu vedem cum putem obține din tot acest ansamblu rezultate concrete. Aici intervine cel mai important proces al întregului mecanism – metoda de învățare. În timp, veți realiza că aproape toată complexitatea din spatele rețelelor neuronale rezidă aici, iar tratarea riguroasă ar ieși cu mult din sfera acestui articol.

Totuși voi aminti de o tehnică mai generală, folosită cu predilecție în regresii. În statistică, prin regresie înțelegem procedeul prin care se pun în legătură variabilele dintr-un spațiu. În practică, legătură între variabile ar putea fi destul de ascunsă. TU ești cel care decizi gradul de legătură. Dacă te chinui multă vreme e posibil să găsești o legătură între orice tip de variabile, chiar și cele care logic sunt necorelate. Problema este delicată fiindcă de fapt faci **overfitting**, adică ai găsit o legătură, dar care nu poate generaliza deloc.

Doi iepuri dintr-o lovitură

Voi încerca să împusc doi iepuri dintr-o lovitură, prezentând același concept cu o minoră schimbare pentru a vedea cum putem face miracole pe-aici fără să ne dăm seama.

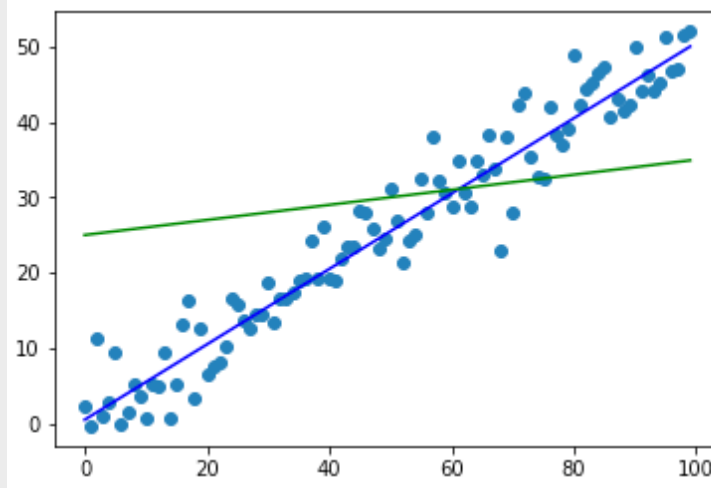
#### 5.1.1. Regresia liniară și funcția de cost

Rețelele neuronale pot fi privite ca un mecanism general prin care putem aproxima funcții, de orice fel ar fi ele. E natural deci să încep cu cele mai simple funcții, cele liniare.

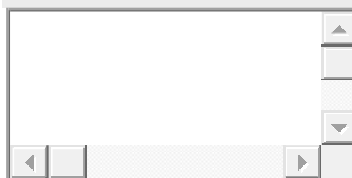
În imaginea de mai jos vedem o dependență liniară între două variabile – timp și număr de utilizatori (nu știu în ce context, dar nu-i important...e o dependență liniară). Ne-am dori un algoritm care să găsească acea linie albastră. Probabil nu va putea s-o nimerească din prima(dacă nu mă crezi leagă-te la ochi și încearcă). Nu am zis-o chiar ironic, fiindcă același lucru o să-l facă algoritmul nostru, la primul PAS.

În esență vrem să găsim  $(w^*, b^*)$  astfel încât  $y = w^*x + b^*$  să fie cea mai bună linie pentru punctele de mai sus.

Pasul 1 constă, așa precum am spus, în alegerea a doi parametri random,  $w_0$  și  $b_0$ . De comoditate, să zicem că am ales  $(0.1, 25)$ . Ce-am obținut? Linia cea verde...



Regresia liniara



```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 np.random.seed(5)
5 x = np.arange(0,100)
6 y = 0.5 + 1/2 * x + np.random.normal(0, 4, 100)
7 plt.plot(x, y, "o")
8 plt.plot(np.arange(0,100), eval('1/2 + 1/2*x'), "b")
9 plt.plot(np.arange(0,100), eval('25 + 0.1 * x'), "g")
10 plt.show()
```

Începem să vedem că am avea nevoie de un algoritm care să modifice la fiecare pas cele două valori astfel încât după câteva iterații linia verde să fie, ideal, suprapusă peste cea albastră. Ne trebuie un mecanism prin care să penalizăm algoritmul dacă e departe de parametri ideali...să înțeleagă că erorile au în viață un **COST**. Ce cost ar fi aici bun?

O soluție rezonabilă este să măsurăm distanța de la valoarea așteptată până la cea obținută. Dacă numărul de puncte rămâne constant, am putea liniștiți să găsim un cost mediu fiindcă nu ne afectează. Definim atunci pe **C**:

$$C(w, b) = \frac{1}{2n} \sum_{i=0}^n (z(x_i, w, b) - y_i)^2$$

$z(x_i, w, b) = wx_i + b$  – are treabă cu modelul generat de algoritmul nostru, adică Y-ul de pe linia verde corespunzător lui  $x_i$ . Valoarea așteptată e punctul albastru care s-a găsit la acel  $x_i$ , ce corespunde lui  $y_i$ .

Trebuie să găsim un mecanism prin care să minimizăm acest C. El este o funcție care depinde de 2 parametri reali, deci este o pătură. Problema se reduce la a găsi minimul. Din liceu, știm că locul în care derivatele unei funcții continue se anulează sunt punctele de extrem (de maxim sau de minim). Aici știm sigur că sunt de minim, fiindcă funcția e strict mai mare ca 0, deci are ca minim pe 0. Ne folosim atunci de gradient și vom merge împotriva sensului său de creștere.

$$w_{i+1} = w_i - \eta \frac{\partial C}{\partial w} = w_i - \eta \frac{1}{n} \sum_{i=0}^n (z(x_i, w, b) - y_i) x_i$$

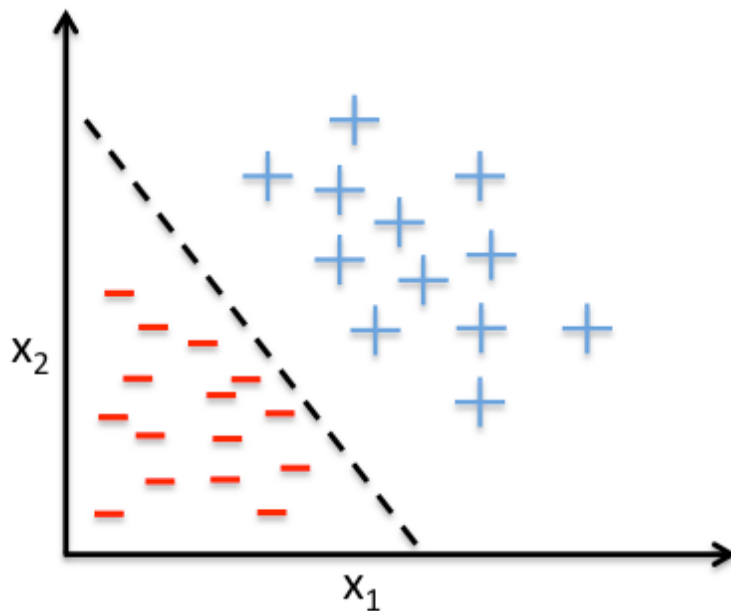
$$b_{i+1} = b_i - \eta \frac{\partial C}{\partial b} = b_i - \eta \frac{1}{n} \sum_{i=0}^n z(x_i, w, b) - y_i$$

$\eta$  – reprezintă o rată de învățare ce dictează cât de repede mergem spre minim. Dacă e prea mare poate vom țopăi în jurul minimului, iar dacă e prea mică durează mult să ajungem la el. Mai rău de atât, în absența unei rate de învățare adecvate am putea rămâne blocați într-un minim local.

Actualizând parametrii  $w$  și  $b$  după regulile de mai sus, o să vedem că după câteva iterații linia verde se va apropia de cea albastră.

### 5.1.2. Clasificatorul liniar (perceptronul)

Mai sus am discutat de problema regresiei liniare. Vom vedea cum, făcând o modificare foarte subtilă putem să ne creăm un clasificator liniar. Trebuie o modificare simplă la modelul nostru prin introducerea unui nou element și anume **funcția de activare**. Pentru a înțelege rostul ei să ne uităm la imaginea de mai jos.



**Example of a linear decision boundary for binary classification.**

Clasificatorul liniar

Spre deosebire de exemplul cu regresia liniară, unde în funcția de cost adunam diferențele de la rezultatul generat până la cel așteptat, aici trebuie să cuantificăm faptul că linia probabil nu clasifică datele cunoscute de antrenament corect. Linia model(cea punctată), vedem că are următoarea proprietate:

$$Ax + by + C = 0$$

$$(1) \quad \forall P(x, y) \wedge tip(P) = - \rightarrow Ax_P + By_P + C < 0$$

$$(2) \quad \forall P(x, y) \wedge tip(P) = + \rightarrow Ax_P + By_P + C > 0$$

Revenim acum la funcția de activare:

$$g(x) = \begin{cases} 1, & x \geq 0 \\ -1, & x < 0 \end{cases}$$

Combinând (1) și (2) cu funcția de activare **g**, putem folosi din nou funcția de cost C, exact ca și la regresia liniară, numai că acum vom avea:

$$C = \frac{1}{2n} \sum_{i=0}^n (g(z) - a_i)^2$$

Aici  $a_i$  poate fi doar  $\pm 1$  și reprezintă clasa reală a instanței din setul de training.

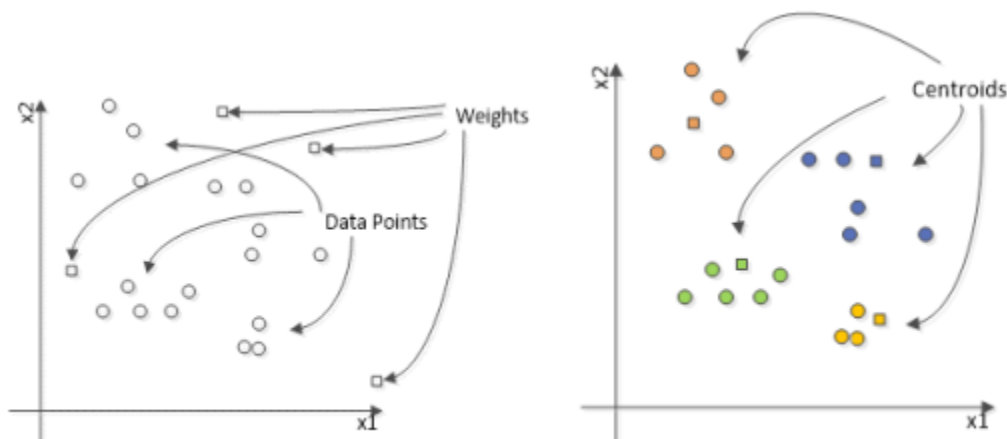
Mai am o singură remarcă: Cele două metode pe care le-ați putut vedea în acțiune mai sus folosesc în ambele cazuri un set de date cu valori cunoscute, adică un set de

training. De aceea, ele fac parte din clasa de algoritmi supravegheați (supervised learning).

Pentru cei dornici să învețe mai multe, vă recomand să urmăriți secțiunea cu referințe, de la finalul acestui articol.

### 5.1.3. Învățarea competitivă

Învățarea competitivă se bazează pe un proces concurențial între neuronii din stratul de ieșire, existând un neuron “câștigător”, în detrimentul celorlalți neuroni. Dacă în ceilalți algoritmi de învățare prezentați până acum, toți neuronii din stratul de ieșire puteau genera valori de ieșire mai mari sau mai mici, în algoritmul de învățare competitivă, doar neuronul de ieșire “câștigător” este activ (generează o valoare), ceilalți neuroni de ieșire devenind inactivi (generează valoarea zero).



Să luăm imaginea de mai sus. Vom presupune, din varii motive, care țin de specificitatea problemei, că trebuie să clasificăm datele în 4 clase distincte. Vom avea o rețea neuronală generală, care poate avea foarte multe intrări (chiar în poza noastră e simplu că ar fi două numere, pentru cele două coordonate de pe plan), dar știm sigur că are doar 4 ieșiri. Comportamentul dorit al rețelei va fi, ca după antrenare, la o combinație de intrări să obținem la ieșire o singură clasă, iar restul 0. Spunem că acest comportament este competitiv, fiindcă doar o clasă poate fi câștigătoare => va fi o rețea cu perceptroni (unii zic perceptroni la neuronii sigmoizi, eu zic la cei cu funcția treaptă).

La finalul procesului de învățare vedem de fapt și ce a făcut rețeaua: a adus modificări la ponderile rețelei, până când acestea au clasificat corect toate datele de intrare. Cum știm câte modificări ar trebui să facem la acele ponderi? Până când funcția de cost ajunge la o valoare mai mică decât pragul definit de noi,  $\epsilon$  să zicem.

Deci matematic:

– ponderea  $j$  a neuronului  $i$ , adică  $w_{ij}$  se actualizează iterativ astfel:  $w_{ij}^{k+1} = w_{ij}^k + \Delta w_{ij}^k$ .  
 $\eta$  este un factor de învățare, de care depinde acuratețea învățării. Un factor  $\eta$  mare implică o acuratețe mai scăzută, dar viteză foarte mare și vice-versa. Deci variația va fi de forma:

$$\Delta w_{ij} = \begin{cases} \eta(x_i - w_{ij}) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

### 5.3. Rețele Kohonen (hărți cu auto-organizare)

Peste tot o să vedem noțiunea de SOM(Self Organizing Map) – o denumire pompoasă, pentru o serie de acțiuni pe care oamenii le fac toată ziua fără să se gândească.

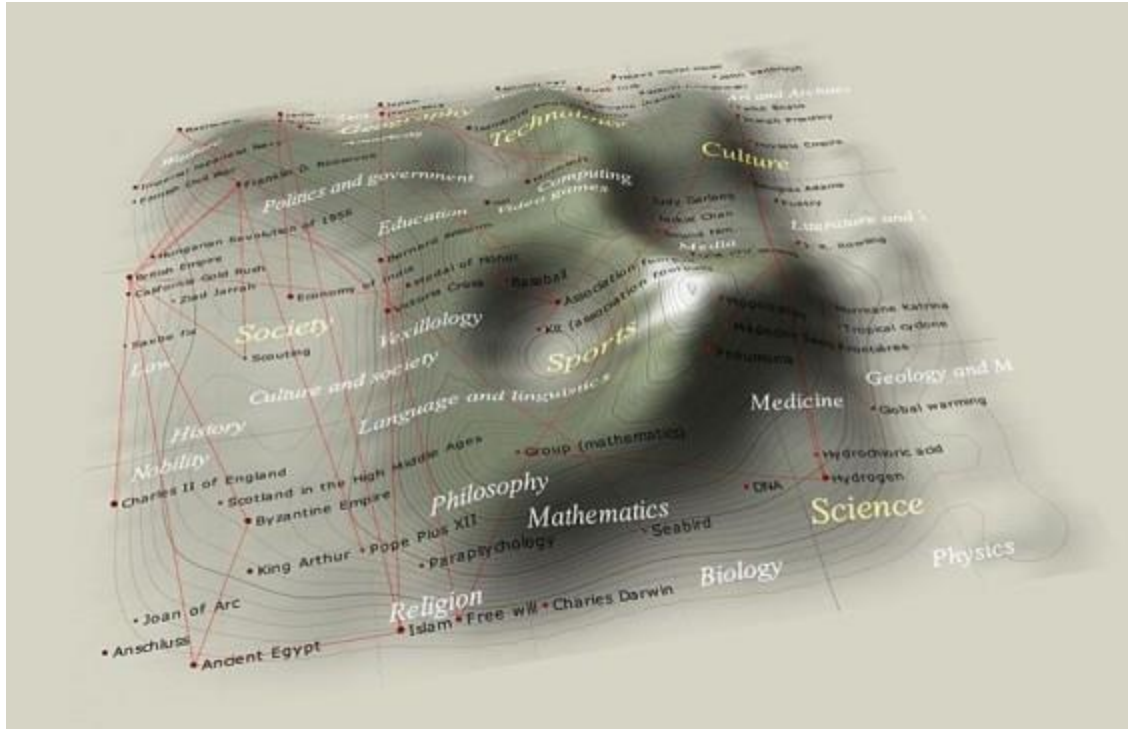
#### Joc – ghicește culoarea bilei fără s-o vezi

Să ne imaginăm c-avem o masă unde sunt biluțe de diferite culori, în așa fel încât am putea distinge unde se află cele mai multe bile de-o culoare. Pare super simplu, dar să presupunem că suntem un calculator – deci nu avem ochi, și putem face doar calcule. Ce-am vrea să facem? Am vrea să ne *învățăm* pentru a putea spune cu mare exactitate de ce culoare este o bilă într-o anumită zonă a mesei FĂRĂ a o vedea...

Dacă am avea ochii închisi ce-am face? Inițial am zice la noroc o culoare. Dacă e greșită am zice o altă culoare...și tot așa până ce-am nimeri culoarea adevărată. Următoarea bilă, dacă e chiar lângă cea pe care i-am determinat culoarea(numa pe ghicite (-:)) va fi probabil de aceeași culoare. Repetăm procesul până când dăm de altă culoare. Atunci știm că am ajuns la o zonă de graniță și culoarea s-a schimbat. Repetăm asta de mii și mii de ori și descoperim fiecare cluster cu bile de o anumită culoare. Exact așa funcționează și o hartă Kohonen și de aceea ele sunt folosite în probleme de categorizare(clustering).

Pentru un calculator asta se rezumă la modificarea ponderilor(care or fi ele, nu ne interesează cum este creată rețeaua, vrem doar să ne facem o idee asupra terminologiei de rețea cu auto organizare) asociate acestei rețele neuronale, ce are ca intrări coordonate pe planul mesei și ca ieșire ce culoare ar fi pe-acolo.

În concluzie, ele sunt folosite cu precădere la gruparea datelor. Rețelele Kohonen sunt destinate în principal clasificării nesupervizate a datelor vectoriale. Ele asigură însă un plus față de celelalte rețele competitive și anume conservarea relațiilor de vecinătate din domeniul datelor de intrare. Aceasta înseamnă că datele de intrare similare vor fi fie în aceeași clasă fie în clase diferite dar care sunt "reprezentate" de către unități funcționale vecine.



## 6. Modalități de îmbunătățire a învățării

### 6.1. Entropia încrucișată

Entropia încrucișată apare în contextul învățării rețelelor neuronale. Atunci când vorbim de învățare supravegheată, învățarea se rezumă la modificarea ponderilor și bias-urilor atașate neuronilor pentru ca întreaga rețea să mapeze rezultatele dorite. Entropia încrucișată s-a dezvoltat pornindu-se de la optimizarea funcției de cost prin metoda gradientului, atunci când s-au văzut primele limitări a acesteia.

$$\frac{\partial C}{\partial w} = (a - y)\sigma'(z)x$$

$$\frac{\partial C}{\partial b} = (a - y)\sigma'(z)$$

Unde  $\sigma$  este sigmoida, iar  $z$  este intrarea ponderată,  $\sum_k w_{kj}^L a_k^{L-1} + b_j^L$

Din ecuațiile de mai sus, se poate intui problema care a dat naștere entropiei încrucișate. Funcția  $\sigma(z)$  este sigmoida, iar această funcție variază foarte puțin în afara intervalului  $[-1, 1]$ . Acest lucru implică că și derivata va tinde spre 0 când eroarea este mare, făcând învățarea mult mai grea.

Pentru a rezolva această problemă, va trebui introdusă o funcție de cost diferită, care să nu țină cont de comportamentul sigmoidei, pentru a facilita învățarea rapidă, chiar și

atunci când variația sigmoidei este foarte mică. Intuitiv, ne-am putea întreba dacă există o astfel de funcție. În acest caz, am putea transforma ecuațiile de mai sus în:

$$\frac{\partial C}{\partial w_j} = x(a_j - y_j) \quad (1)$$

$$\frac{\partial C}{\partial b} = (a - y) \quad (2)$$

Folosind regula produsului se obține

Apoi, ne folosim de proprietatea sigmoidei  $\sigma'(z) = \sigma(z)(1 - \sigma(z)) = a(1 - a)$

Introducând identitatea de mai sus în (3) obținem:

$$\frac{\partial C}{\partial b} = \frac{\partial C}{\partial a} a(1 - a)$$

Folosindu-ne de ecuația (2) obținem

$$\frac{\partial C}{\partial a} = \frac{a - y}{a(1 - a)}$$

Integrăm pe ambele părți cu  $a$  și obținem forma finală a costului, pe un singur neuron

$$C = -[y \ln a + (1 - y) \ln(1 - a)] + Const$$

Pentru a obține forma finală a funcției de cost, va trebui să luăm media pe toate exemplele de training:

$$C = -\frac{1}{n} \sum_x [y \ln a + (1 - y) \ln(1 - a)] + Const$$