Categoria de servicii: ***WP 3. Servicii de dezvoltare a competențelor și de formare profesională (Skills Development and Training Services)***

Subcategoria de servicii: ***Sesiuni de instruire pe soluții digitale avansate (Training sessions on high performance digital solutions)***

SESIUNE DE INSTRUIRE PENTRU DEEP LEARNING

- REȚELE NEURONALE PROFUNDE -

Parteneri WEH: Universitatea *Spiru Haret*

Trainer de coordonare a transformarii digitale:

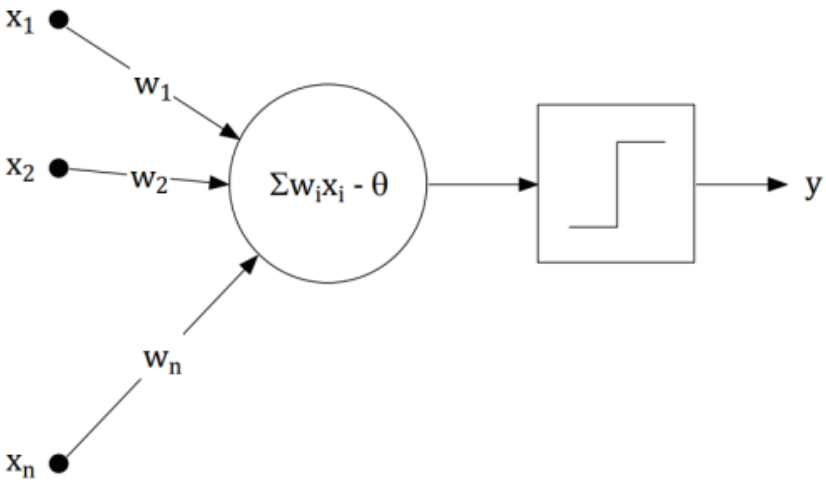
ALBEANU GRIGORE

Str. Italiană nr. 28, sect. 2; sect. 2; București - România
weh@spiruharet.ro weh.spiruharet.ro

Cuprins

1. Reţele Neuronale Artificiale (RNA)

Perceptronul standard este un model de neuron artificial în care semnalele de intrare sunt sumate, iar un semnal de ieșire este generat doar dacă suma depășește pragul.



$$y = F\left(\sum_{i=1}^{n} w_i x_i - \theta\right)$$

Perceptronul -sau perceptronul simplu, cum mai este el denumit- a fost introdus de Frank Rosenblatt şi reprezintă un neuron McCulloch-Pitts prevăzut cu un mecanism de învăţare, adică: cu un mecanism de stabilire "pe bază de exemple" a ponderilor sale sinaptice, în funcţie de comportamentul ce se doreşte a i se conferi.

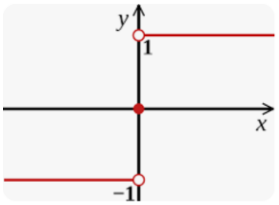
Iniţial, prin termenul perceptron a fost referit un neuron McCulloch-Pitts cu funcţie de răspuns la excitaţie discretă, instanţiată fie prin funcţia prag bipolară discretă –în cele mai multe cazuri-, fie prin funcţia prag unipolară discretă – destul de frecvent-. Ulterior, aria de acoperire a termenului s-a extins şi la neuroni McCulloch-Pitts cu funcţie de răspuns la excitaţie continuă.

Actualmente, un perceptron cu funcţie de răspuns la excitaţie discretă se numeşte perceptron discret, iar un perceptron cu funcţie de răspuns la excitaţie continuă se numeşte perceptron continuu.

Perceptronul are ponderi sinaptice ajustabile şi funcţie de activare semn sau treaptă:

$$F(a) = \begin{cases} -1, & dacă\ a < 0 \\ 1, & dacă\ a \geq 0 \end{cases}$$

$$F(a) = \begin{cases} 0, & dacă\ a < 0 \\ 1, & dacă\ a \geq 0 \end{cases}$$



Scopul perceptronului este să clasifice o instanţă într-una din două clase. Spaţiul intrărilor, n-dimensional, este împărţit în două de un hiperplan definit de ecuaţia:

$$\sum_{i=1}^{n} x_i w_i - \theta = 0$$

Pragul poate fi considerat o pondere suplimentară, cu intrarea tot timpul 1 sau –1. Acest fapt simplifică formula de calcul a ieşirii:

$$y = F\left(\sum_{i=1}^{n+1} w_i x_i\right)$$

Învăţarea are loc prin ajustarea succesivă a ponderilor pentru a reduce diferenţa dintre ieşirile reale (y) şi ieşirile dorite (y_d), pentru *toate* datele de antrenare.

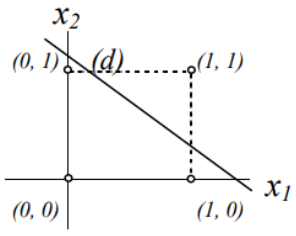
Regula de învăţare a perceptronului este:

$$\Delta w = \alpha \cdot x \cdot e$$

unde Δw este corecţia ponderii, α este rata de învăţare, x este intrarea, iar e este eroarea ($y_d - y$).

Perceptronul standard (cu un singur strat) poate învăţa tot ce poate reprezenta, dar nu poate reprezenta multe funcţii. Mai ales, nu poate reprezenta funcţii neseparabile liniar, de exemplu, XOR.

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1



(ŞI LOGIC)

Adaline este un tip de neuron asemănător perceptronului, dar are funcție de activare liniară și se antrenează diferențial.

$$y = \sum_{i=1}^{n+1} w_i x_i$$

Eroarea folosită este eroarea pătratică:

$$E_i = \frac{1}{2}\left(y_{di} - y_i\right)^2$$

Antrenarea presupune minimizarea erorii în raport cu ponderile. Regula de ajustare a ponderilor este aceeași ca la perceptron.

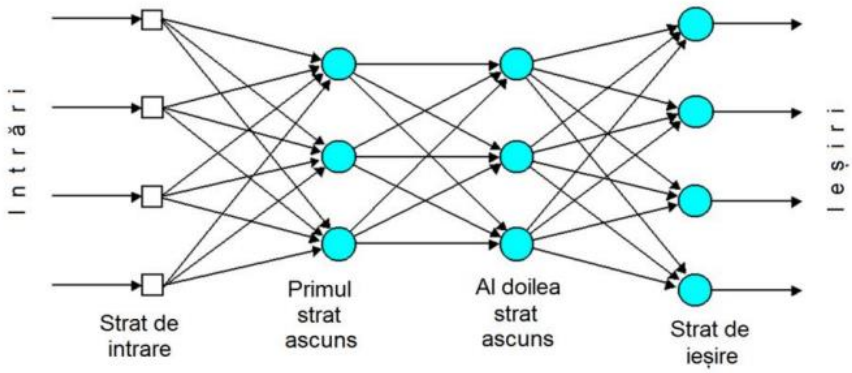
În cazul unei funcții de activare *f* neliniare, dar derivabile, *regula delta* este:

$$\frac{\partial E_i}{\partial w_j} = -x_{ij}\left(y_{di} - y_i\right) f'(y_i)$$

de unde se deduce:

$$\Delta w_j = \alpha \cdot x_{ij} \cdot e_i \cdot f'(y_i)$$

Perceptronul multi-strat este o rețea neuronală cu propagare înainte (feed-forward) cu unul sau mai multe straturi ascunse. El are: • Un strat de intrare (care nu face procesări); • Unul sau mai multe straturi ascunse (intermediare); • Un strat de ieșire



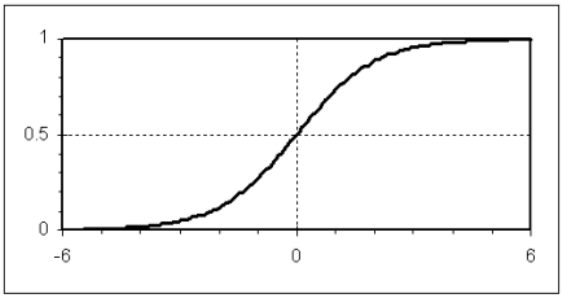
Proprietatea de aproximare universală. O rețea neuronală cu un singur strat ascuns, cu un număr posibil infinit de neuroni, poate aproxima orice funcție reală continuă.

Din punct de vedere practic, un strat nu poate avea un număr infinit de neuroni. Un strat suplimentar poate reduce foarte mult numărul de neuroni necesari în straturile ascunse.

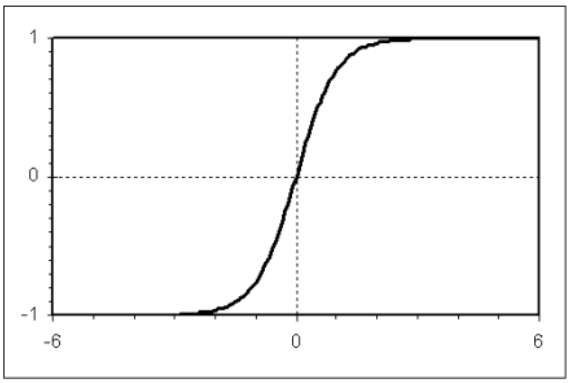
Str. Italiană nr. 28, sect. 2; sect. 2; București - România
weh@spiruharet.ro weh.spiruharet.ro

Funcțiile de activare pentru perceptronul multi-strat trebuie să fie neliniare. Cel mai des utilizate sunt *sigmoida unipolară* (sau logistică), respectiv *sigmoida bipolară* (tangenta hiperbolică):

$$f(x) = \frac{1}{1+e^{-x}}$$



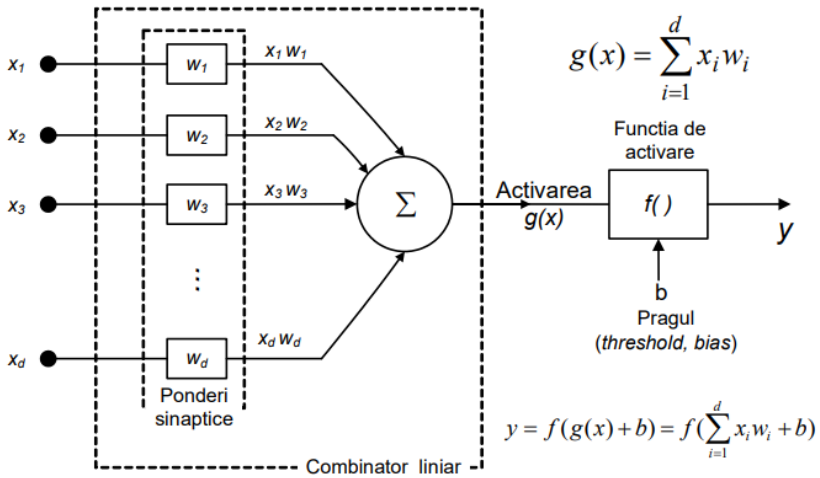
$$f(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$$



Un perceptron cu un singur strat are aceleași limitări chiar dacă folosește o funcție de activare neliniară.

Un perceptron multi-strat cu funcții de activare liniare este echivalent cu un perceptron cu un singur strat.

Structura fundamentală a unui neuron artificial:



Algoritmul de retro-propagare (back-propagation)

Are două faze:

- Rețeaua primește vectorul de intrare și propagă semnalul înainte, strat cu strat, până se generează ieșirea;
- Semnalul de eroare este propagat înapoi, de la stratul de ieșire către stratul de intrare, ajustându-se ponderile rețelei:

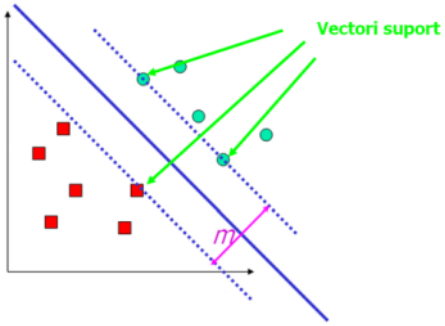
$$\Delta w_{jk} = \alpha \cdot y_j \cdot [y_k (1 - y_k)] \cdot e_k$$
$$\Delta w_{ij} = \alpha \cdot x_i \cdot [y_j (1 - y_j)] \cdot \sum_k \delta_k w_{jk}$$

Metoda momentului presupune adăugarea unui termen la regula delta, care accelerează și stabilizează căutarea:

$$\Delta w_{jk}(p) = \beta \cdot \Delta w_{jk}(p-1) + \alpha \cdot y_j(p) \cdot \delta_k(p)$$

Rata de învățare poate fi de asemenea modificată în mod adaptiv pe parcursul antrenării. Dacă variația sumei erorilor pătratice ΔE are același semn algebric pentru mai multe epoci consecutive, atunci rata de învățare α trebuie să crească. Dacă semnul lui ΔE alternează timp de câteva epoci consecutive, atunci rata de învățare α trebuie să scadă.

Pentru o problemă de clasificare liniar separabilă cu două clase, există multe limite de separație posibile. Teoria SVM consideră că suprafața care separă clasele trebuie să fie cât mai departe de datele din ambele clase, adică *marginea* determinată de limita de decizie trebuie *maximizată*. Instanțele de antrenare care se găsesc pe linia de demarcație a marginii se numesc *vectori suport*.



Pentru clasificarea unei instanțe **x**, se folosește următoarea funcție:

$$h(\mathbf{x}) = g(\mathbf{w}^T\mathbf{x} + b)$$

unde g este funcția semn:

$$g(z) = \begin{cases} 1, \text{ dacă } z \geq 0 \\ -1, \text{ dacă } z < 0 \end{cases}$$

Prin urmare, instanțele dintr-o clasă vor avea ieșirea $y = 1$, iar instanțele din cealaltă clasă vor avea ieșirea $y = -1$.

Acest efect de a împărți spațiul problemei în două este același ca la perceptron.

Maximizarea marginii se poate face rezolvând o problemă de optimizare (numită *problema primară*):

$$\begin{aligned} \min_{\gamma,w,b} \quad & \frac{1}{2}||w||^2 \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1, \quad i = 1, \ldots, m \end{aligned}$$

$$g_i(w) = -y^{(i)}(w^T x^{(i)} + b) + 1 \leq 0$$

Dificultatea acestei probleme constă în faptul că dimensionalitatea ei (numărul de atribute) poate fi foarte mare și există câte o constrângere pentru fiecare instanță. De aceea, soluția poate fi determinată transformând problema primară în *problema duală* cu ajutorul dualității Lagrange:

$$\begin{aligned} \max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle. \\ \text{s.t.} \quad & \alpha_i \geq 0, \quad i = 1, \ldots, m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0, \end{aligned}$$

Aici, trebuie determinați coeficienții α_i, câte unul pentru fiecare instanță de antrenare. Majoritatea α_i vor fi 0, cu excepția celor corespunzători vectorilor suport. Avantajul este că numărul vectorilor suport este mult mai mic decât numărul instanțelor de antrenare.

Există algoritmi speciali pentru rezolvarea problemei duale. Cel mai cunoscut este algoritmul *Sequential Minimal Optimization* (SMO), care este rapid și deci foarte potrivit pentru optimizarea problemelor de dimensiuni mari cu care lucrează de obicei SVM.

Rezolvând problema duală, se determină α_i, iar apoi se calculează **w** și b:

$$w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}$$

$$b = \frac{1}{|S|} \sum_{s \in S} \left(y_s - \sum_{t \in S} \alpha_t y_t (\mathbf{x}_t \cdot \mathbf{x}_s) \right)$$

unde S este mulțimea vectorilor suport, iar $|S|$ este numărul lor.

Atunci când clasele nu sunt liniar separabile, datele de antrenare se pot proiecta într-un spațiu cu mai multe dimensiuni, unde devin liniar separabile. Acest lucru se face cu o funcție $\phi(\mathbf{x})$ numită *transformare în trăsături* (engl. "feature mapping").

Se definește un *nucleu* (engl. "kernel") ca fiind o funcție $K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T \cdot \phi(\mathbf{z})$.

Un nucleu poate fi interpretat ca o măsură a similarității dintre cele două argumente.

2. Învăţarea Hebbiană

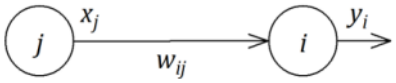
Învăţarea este procedura de extragere a informaţiei, din setul de date de antrenare, necesară modificării valorilor termenilor liberi ai unei RNA (a ponderilor) în vederea obţinerii fenomenului de generalizare – a obţinerii erorii minime de clasificare corecte a eşantioanelor din alt set de date ce nu a fost prezentat anterior RNA pentru învățare. Deci, într-o RNA şi, în general, în cazul sistemelor adaptive cel care construieşte reţeaua nu trebuie să specifice parametrii sistemului (valorile ponderilor fiecărui neuron în parte). Valorile acestor parametri sunt extraşi, în mod automat folosind ca mijloc pentru atingerea acestui obiectiv anumiţi algoritmi, denumiți de antrenare sau de adaptare. Scopul final fiind acela al reducerii unei anumite erori. Există la ora actuală trei paradigme de adaptare bazate pe: învățarea supervizată (supervised learning), învățarea nesupervizată (unsupervised learning) și învățarea întărită (reinforcement learning).

Utilizarea algoritmilor de învăţare superviyată pentru modificarea valorilor ponderilor în scopul obţinerii unei clasificări cât mai bune este unul din conceptele fundamentale ale unei RNA. Performanţele clasificării sunt cuantificate folosind pentru aceasta un anumit criteriu de eroare: diferența existentă între ieșirea RNA (răspunsul obținut la ieșirea RNA – dependent, în principal, de valorile aplicate intrărilor și de ponderile neuronilor – vectorul ponderilor w) și răspunsul dorit. Acest răspuns dorit este furnizat rețelei de către un "antrenor" ce etichetează fiecare intrare în parte conform unui criteriu extern știut doar de el – de aici derivă și denumirea de învățare supervizată.

Diferența dintre răspunsul obținut la ieșirea RNA și răspunsul dorit reprezintă eroarea de adaptare. Această eroare este cea care ghidează procesul de învățare și care, dacă algoritmul converge, se va diminua o dată cu imbunătățirea modelului neuronal. În cadrul algoritmilor de învățare nesupervizată (de adaptare) nu există un dorit al RNA. Deci, nu există așa zisul profesor sau antrenor prezentat anterior. Aceste rețele trebuie să aibe capacitatea de a forma singure reprezentări interne ale datelor de intrare și astfel de a organiza datele de intrare în „clase" sau clusteri printr-un mecanism de competiție internă. În final clusterii obținuți trebuie să respecte modul natural de grupare și distribuție a datelor de intrare – care sunt date de diferitele corelații existente în mulțimea acestora.

Învăţarea folosind un "critic" (reinforcement learning, prin întărire) are la bază principiul de învățare ce utilizează recompensa/pedeapsa (reward/punishment). Această metodă este utilizată în special în studiile efectuate pe animale, de unde altfel a fost și preluată conceptual. Aceste RNA, ce utilizează acest algoritm de învățare, nu beneficiază de semnal dorit ca în situația învățării suprervizte, ci de un răspuns binar. Acest răspuns oferă o informaţie calitativă RNA (de genul, răspuns bun sau greșit) funcție de cât de bine structura neuronală a putut să-și atingă sau nu obiectivul urmărit. Astfel, de exemplu, rețeaua va ști că a greșit, însă nu și cu cât a greșit. În acest mod atunci când RNA în urma unui proces de adaptare va genera rezultate eronate, ea va fi „pedepsită" printr-un feedback negativ și atunci prin algoritmul de adaptare o nouă tendință a RNA de a produce același rezultat sau unul similar va fi slăbită, redusă. În caz contrar această tendință va fi întărită (reinforced).

Învățarea hebbiană modelează dinamica sinapselor din creierele biologice. Modelul său computațional se bazează pe *legea lui Hebb*: Dacă doi neuroni conectați sunt activați în același timp, ponderea conexiunii dintre ei crește. Dacă doi neuroni sunt activați în contratimp, ponderea conexiunii dintre ei scade. În engleză: "Neurons that fire together, wire together".



$$\Delta w_{ij} = \alpha \cdot x_j \cdot y_i$$

unde α este rata de învățare, iar funcțiile de activare sunt de obicei liniare.

Pentru intrări multiple:

$$y = \sum_{i=1}^{D} w_i x_i$$

$$y = \mathbf{w}^\top \mathbf{x} = \mathbf{x}^\top \mathbf{w}$$

Produsul scalar poate fi interpretat ca o măsură de similaritate, prin urmare, neuronul hebbian implementează o măsură de similaritate în spațiul de intrare. O ieșire y mare înseamnă că intrarea curentă este similară cu vectorul de antrenare $\mathbf{x}$ care a creat ponderile. Rețeaua își amintește vectorul de antrenare, deci se comportă ca o *memorie asociativă*.

Folosind doar regula lui Hebb, ponderile pot crește la infinit. De aceea, se poate aplica normalizarea regulii hebbiene: noua valoare a unei ponderi se împarte la norma vectorului de ponderi. Dacă o pondere crește, celelalte trebuie să scadă.

Regula lui Oja este o formulă care aproximează această normalizare:

$$\Delta \mathbf{w} = \alpha \cdot y \cdot (\mathbf{x} - y \cdot \mathbf{w})$$

Ponderile nu mai cresc nelimitat, dar rețeaua poate uita asocierile vechi. Dacă un vector de antrenare nu este prezentat frecvent, el poate fi uitat.

Regula lui Sanger, care generalizează regula lui Oja pentru o rețea cu D intrări și M ieșiri ($M \leq D$, dar de obicei $M \ll D$) conduce la aproximarea componentelor principale ale datelor de intrare (din analiza componentelor principale, engl. "Principal Component Analysis", PCA):

$$\Delta w_{ij} = \alpha \cdot y_i \cdot \left(x_j - \sum_{k=1}^{i} w_{kj} y_k \right)$$

Ponderile rețelei reprezintă cei M vectori de dimensiune D. PCA este o metodă de reducere a dimensionalității sau de compresie a datelor. Prin selectarea proiecțiilor pe cele mai importante M dimensiuni, din cele D ale spațiului de intrare, cu $M \leq D$, se păstrează cele mai importante caracteristici ale datelor.

3. Reţele neuronale profunde

Din punct de vedere a topologiei RNA aceste se clasifică în două mari clase: (a) rețelele neuronale de tip feedforward în care informația se propagă doar de la intrare spre ieșire, iar ieșirea fiecărui neuron depinde doar de intrările sale care sunt conectate la ieșirile neuronilor din stratul anterior și (b) rețelele neuronale cu reacție sau recurente (recurrent neural network), la care prin intermediul conexiunilor existente se asigură bucle de reacție ce determină ca ieșirea diferiților neuroni să fie dependentă și de valori anterior calculate (de ex. de ieșirile altor neuroni din același strat sau dintr-un strat ulterior) – putem spune că aceste structuri neuronale au memorie în care păstrează diferite nivele de informație calculată anterior.

- Rețele clasice:
  - 1-2 straturi;
  - Funcții de activare sigmoide;
  - Funcții de cost bazate pe MSE;
  - Algoritmi de antrenare: BackPropagation, RProp, Levenberg-Marquardt etc.;
- Rețele profunde:
  - Mai multe straturi;
  - Funcții de activare mai simple: ReLU;
  - Funcții de cost bazate pe MLE;
  - Algoritmi de antrenare: SGD, RMSProp, Adam etc.;
  - Alte metode de inițializare a ponderilor, regularizare, pre-antrenare.

Evident, diferența principală este numărul de straturi. Dincolo de acesta, diferențele nu sunt stricte. De exemplu, și neuronii unei rețele profunde pot avea funcții de activare sigmoide. Dar o rețea clasică cu mai multe straturi poate avea probleme la antrenare. Primele straturi ale unui perceptron multi-strat clasic cu un număr mai mare de straturi ascunse nu se antrenează bine. Când ponderile sunt mici sau funcțiile de activare sigmoide sunt saturate (gradienți mici), corecțiile ponderilor Δw sunt mici, iar antrenarea este foarte lentă. De asemenea, ultimele straturi, cele apropiate de ieșire, pot învăța problema „destul de bine” și deci semnalul de eroare trimis către primele straturi devine și mai mic. De aceea, sunt necesare alte metode de antrenare pentru a pune în valoare primele straturi.

Tehnicile de învățare profundă pot acționa direct asupra datelor brute pentru a descoperi în mod automat reprezentările utile. Procesul este facilitat de existența unui mare volum de date disponibile pentru antrenare: imagini, documente etc.

Învățarea profundă a adus rezultate superioare metodelor tradiționale în multe domenii:

- Recunoașterea imaginilor
- Prelucrarea limbajului natural: clasificarea subiectelor, analiza sentimentelor, răspunsul la întrebări, traducerea automată
- Analiza efectelor medicamentelor
- Analiza exprimării genelor și a impactului asupra bolilor
- Analiza datelor din acceleratoarele de particule

Din punct de vedere istoric, cele mai relevante realizări în domeniu au fost: În1989, LeCun et al., aplicarea algoritmului backpropagation standard pentru recunoașterea cifrelor scrise de mână din codurile poștale. Tot LeCun propune ideea de autoencoder pentru reducerea dimensionalității. Apoi, Hochreiter & Schmidhuber, Long short-term memory (LSTM), 1997, un tip de rețea recurentă, utilizată pentru prelucrarea limbajului natural. De asemenea, Ng, 2011, proiect de învățare profundă la Google, inițial pentru comenzi vocale pe telefoanele Android și etichetarea imaginilor pe rețeaua socială Google+.Facebook folosește metode de învățare profundă pentru:

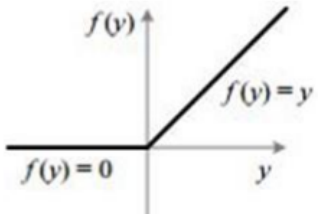
- Prelucrarea textelor (DeepText)
- Recunoaștere facială (DeepFace, 97% rata de succes la deosebirea a două persoane după figură, oamenii au 96% rata de succes)
- Direcționarea publicității.

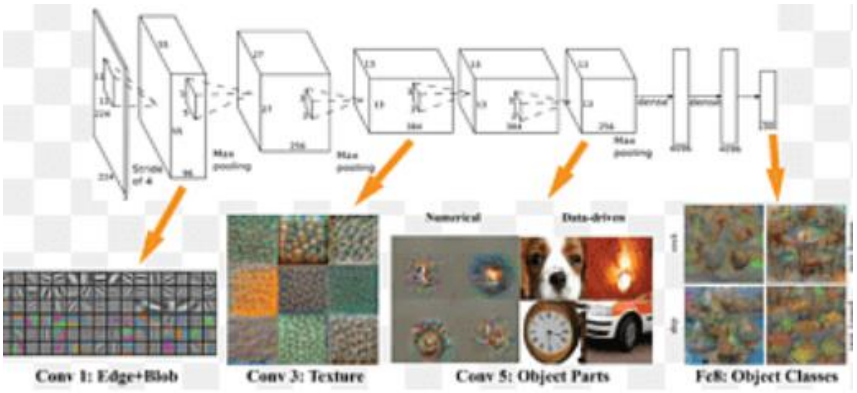
Proiectarea unor rețele neuronale cu performanțe foarte bune necesită foarte mult efort. În 2016, Folosirea rețelelor profunde pentru Google Translate a scăzut rata de eroare cu 60%. OpenAI (2020): Generative Pre-trained Transformer 3 (GPT-3) cu 175 miliarde de parametri pentru asistare în sistem Întrebare-Răspuns.

Funcții de activare

O funcție de activare tipică pentru arhitecturi profunde este ReLU (*Rectified Linear Unit*):

$$f(x) = \max(0, x)$$





Învățare profundă Rețea neuronală convoluțională AlexNet Viziune computer Rețea neuronală artificială, Învățare prof undă, alexnet, unghi png

Este o funcție neliniară cu gradienți foarte ușor de calculat:

$$\frac{df}{dx} = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x < 0 \end{cases}$$

Chiar dacă în partea negativă gradientul este 0, în practică funcționează bine. S-a observat că învățarea este de circa 6 ori mai rapidă față de funcțiile sigmoide.

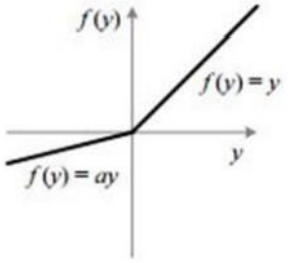
Totuși, pentru a evita problemele date de faptul că partea negativă nu contribuie la învățare, s-au propus unele variante alternative:

Leaky ReLU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$$

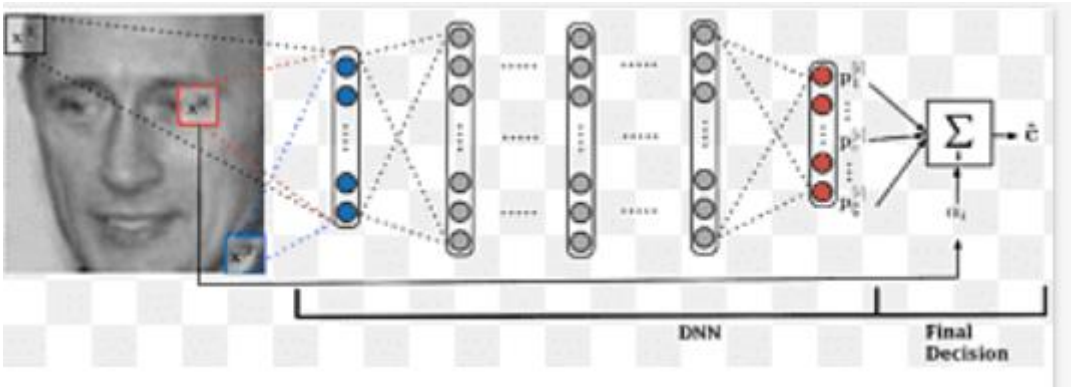
Parametric ReLU (PReLU), unde parametrul a poate fi învățat:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ ax & \text{otherwise} \end{cases}$$



Funcții de cost

Pentru rețele profunde, se preferă funcții de cost bazate pe MLE (*Maximum Likelihood Estimation*), de exemplu, *cross-entropy*, softmax etc.

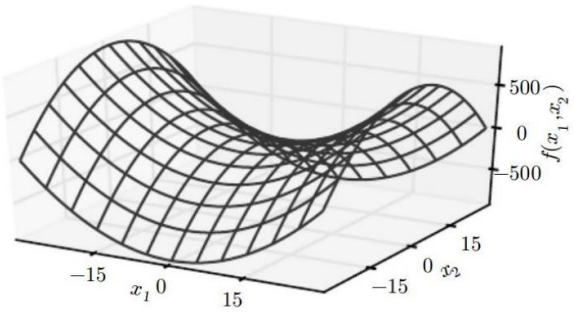


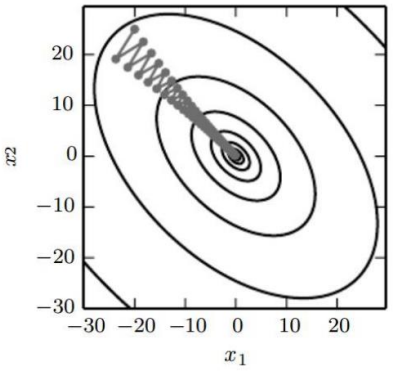
Algoritmi de antrenare

Algoritmii sunt bazați tot pe ideea de gradient descendent, la fel ca *backpropagation* clasic, dar au apărut variante specializate sau îmbunătățite:

- *Stochastic Gradient Descent:* Pentru problemele actuale, numărul instanțelor de antrenare poate fi foarte mare. SGD lucrează asemănător cu gradientul descendent, dar nu ia în calcul toate instanțele la calcularea gradienților, ci doar un mini-lot (*minibatch*). La fiecare iterație, se aleg aleatoriu alte instanțe. Gradienții sunt doar o aproximație a gradienților reali.

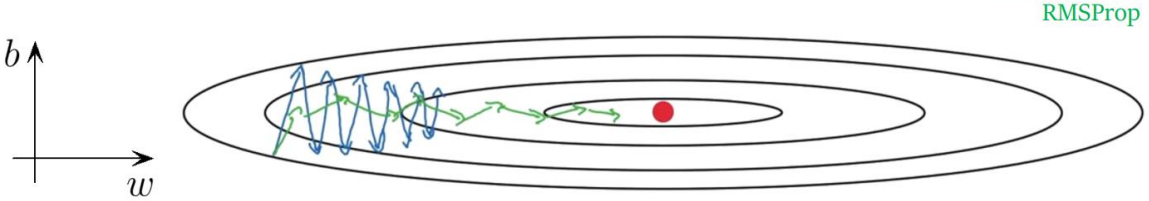
  Complexitatea de timp poate fi controlată prin dimensiunea mini-lotului. La algoritmul standard se pot adăuga variantele ratei de învățare descrescătoare și a momentului;
- *RMSProp (Root Mean Square Propagation):* Se bazează pe ideea adaptării ratei de învățare invers proporțional cu suma pătratelor gradienților. Rata de învățare scade rapid pentru parametrii cu gradienți mari și mai încet pentru cei cu gradienți mici. Nu se folosește direct suma istorică, ci o medie glisantă (*moving average*), care elimină efectele trecutului îndepărtat. Altfel, prin traversarea unor zone neconvexe ale spațiului problemei, rata de învățare ar putea deveni prea mică înainte ca algoritmul să ajungă într-o zonă convexă local, unde se găsește soluția. La fel ca metoda momentului, are ca efect atenuarea oscilațiilor și permite mărirea ratei de învățare, ceea ce accelerează învățarea;
- *Adam (Adaptive Moment):* Poate fi văzut ca o combinație între metoda momentului și algoritmul *RMSProp*.





$$f(\boldsymbol{x}) = x_1^2 - x_2^2$$

Gradient Descent
RMSProp



Problema: gradienții sunt prea mari în direcția *b* și prea mici în direcția *w*

Inițializarea ponderilor

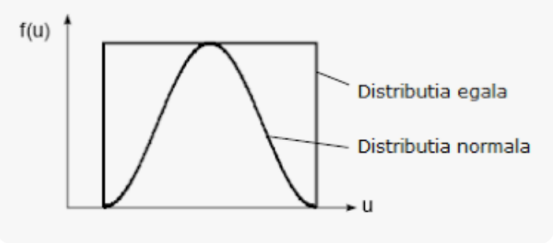
Ideea de bază pentru aceste metode de inițializare este că ponderile trebuie să aibă media 0 și o varianță specificată. La propagarea înainte într-o rețea profundă, se dorește ca varianța semnalului să rămână constantă.

Inițializarea Xavier (Glorot)

Fie n_i numărul de neuroni cu care este conectat la intrare un neuron considerat (*fan-in*), care are ponderile W_i , iar n_{i+1} numărul de neuroni cu care este conectat la ieșire (*fan-out*). N reprezintă distribuția normală, iar U cea uniformă. Așadar, ponderile se pot inițializa astfel:

$$W_i \sim N\left(0, \sqrt{\frac{2}{n_i + n_{i+1}}}\right)$$

$$W_i \sim U\left(-\sqrt{\frac{6}{n_i + n_{i+1}}}, \sqrt{\frac{6}{n_i + n_{i+1}}}\right)$$



Dropout

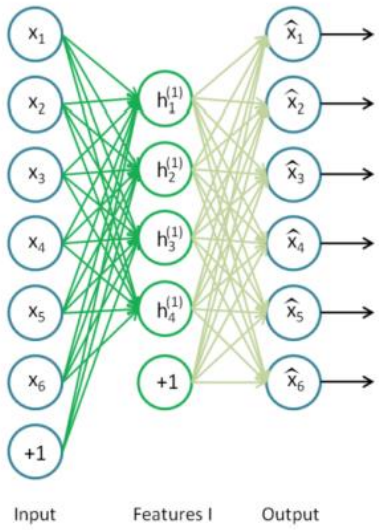
Dropout este o formă de regularizare. Efectul său este asemănător cu ideea de la *bagging*. Simulează antrenarea unor rețele diferite prin dezactivarea aleatorie a unor neuroni în timpul antrenării. Aceste subrețele sunt parte a rețelei originare, iar ponderile sunt comune. *Dropout*-ul previne suprapotrivirea (*overfitting*).

În mod ideal, fiecare neuron din rețea ar trebui să detecteze trăsături în mod independent. Dacă mai mulți neuroni detectează aceleași trăsături în mod repetat, fenomenul se numește co-adaptare. *Dropout*-ul previne și co-adaptarea.

Eliminarea unor neuroni de intrare are efecte foarte puternice. În lipsa unor intrări, rețeaua trebuie să compenseze prin găsirea altor trăsături importante pentru clasificare. De exemplu, pentru recunoașterea unei fețe, dacă din imagine se elimină nasul, rețeaua trebuie să fie capabilă să găsească alte indicii, de exemplu ochii sau gura.

Autoencodere

Un autoencoder învață funcția identitate: $h(\mathbf{x}) = \mathbf{x}$.



De multe ori, se folosesc autoencodere subcomplete (*undercomplete*), unde $|\mathbf{h}| < |\mathbf{x}|$. Stratul ascuns este o versiune comprimată a intrării. Autoencoderele reprezintă astfel o metodă de reducere a dimensionalității problemei, ceea ce ajută învățarea întrucât majoritatea modelelor de învățare presupun reținerea trăsăturilor esențiale ale datelor.

Denoising autoencoders sunt antrenate cu intrări corupte, pe care încearcă să le corecteze (*de-noise*). Antrenarea acestora se desfășoară precum urmează. Intrările sunt corupte prin adăugarea de zgomot. Intrările corupte intră în autoencoder. Funcția de cost se calculează prin compararea ieșirii autoencoderului cu intrările necorupte.

4. EXEMPLU: LENET

LeNet este o arhitectură de rețea neuronală convoluțională fundamentală dezvoltată de Yann LeCun și colegii săi, care a revoluționat recunoașterea imaginilor prin designul său inovator și principiile sale influente.

Concepută inițial pentru a recunoaște caracterele scrise de mână și tipărite de mașini, LeNet a fost un model revoluționar la momentul creării sale. Arhitectura sa, cunoscută sub numele de LeNet-5, constă din straturi convoluționale, urmate de straturi de subeșantionare și de straturi complet conectate, culminând cu un strat de ieșire softmax. Această dispunere a straturilor a fost concepută pentru a învăța automat caracteristicile din imaginile de intrare, în loc să se bazeze pe caracteristici create manual, stabilind un nou standard pentru aplicațiile de învățare automată.

Arhitectura LeNet constă din mai multe straturi care extrag și condensează progresiv informațiile din imaginile de intrare. Aici este descrierea fiecărui strat al arhitecturii LeNet:

Strat de intrare: Acceptă imagini de 32×32 pixeli, adesea completate cu zero dacă imaginile originale sunt mai mici.

Primul strat de convoluție (C1): Constă din șase filtre 5×5, care produc șase hărți de caracteristici de 28×28 fiecare.

Primul strat de punere în comun (S2): Aplică o medie de 2×2 pentru a reduce dimensiunea hărților de caracteristici la 14×14.

Al doilea strat convoluțional (C3): Utilizează șaisprezece filtre 5×5, dar cu conexiuni rare, producând șaisprezece hărți de caracteristici de 10×10.

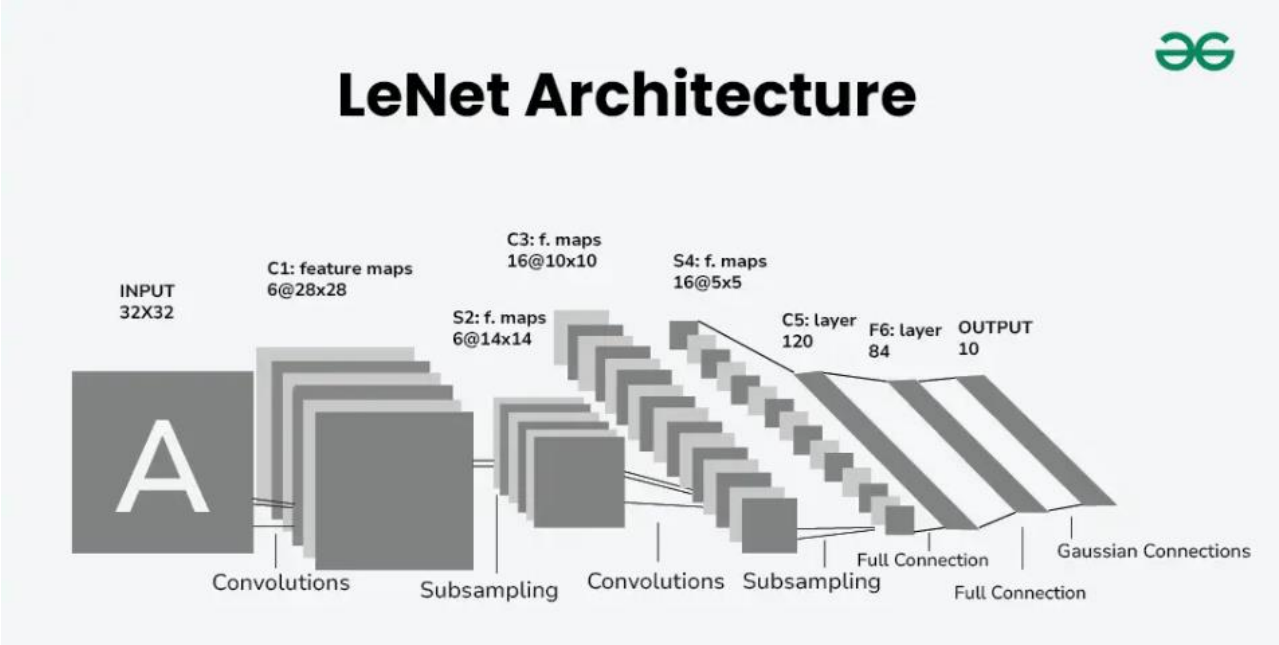
Al doilea strat de punere în comun (S4): Reduce și mai mult hărțile de caracteristici la 5×5, utilizând 2×2 medii de punere în comun.

Straturi complet conectate:

Primul strat complet conectat (C5): Complet conectat cu 120 de noduri.

Al doilea strat complet conectat (F6): Cuprinde 84 de noduri.

Strat de ieșire: Softmax sau activare gaussiană care produce probabilități pentru 10 clase (cifrele 0-9).



1. Stratul de intrare

Dimensiunea de intrare: 32×32 pixeli.

Intrarea este mai mare decât cel mai mare caracter din baza de date, care are cel mult 20×20 pixeli, centrat într-un câmp de 28×28. Dimensiunea mai mare a datelor de intrare garantează că trăsăturile distinctive, cum ar fi punctele finale ale trăsăturilor sau colțurile, pot apărea în centrul câmpului receptiv al detectorilor de trăsături de cel mai înalt nivel.

Normalizare: Valorile pixelilor de intrare sunt normalizate astfel încât fundalul (alb) să corespundă unei valori de 0, iar prim-planul (negru) să corespundă unei valori de 1. Această normalizare face ca media intrării să fie aproximativ 0, iar variația să fie aproximativ 1, ceea ce accelerează procesul de învățare.

2. Stratul C1 (strat convoluțional)

Hărți de caracteristici: 6 hărți de caracteristici.

Conexiuni: Fiecare unitate este conectată la o vecinătate de 5×5 în intrare, producând 28×28 hărți de caracteristici pentru a preveni efectele de limită.

Parametrii: 156 de parametri care pot fi antrenați și 117 600 de conexiuni.

3. Stratul S2 (strat de subeșantionare)

Hărți de caracteristici: 6 hărți de caracteristici.

Dimensiune: 14×14 (fiecare unitate conectată la o vecinătate de 2×2 în C1).

Funcționare: „Operațiune: Fiecare unitate adaugă patru intrări, înmulțește cu un coeficient care poate fi antrenat, adaugă un bias și aplică o funcție sigmoidă.

Parametrii: 12 parametri care pot fi antrenați și 5 880 de conexiuni.

Conectivitate parțială: C3 nu este complet conectat la S2, ceea ce limitează numărul de conexiuni și rupe simetria, forțând hărțile de caracteristici să învețe caracteristici diferite, complementare.

4. Stratul C3 (strat convoluțional)

Hărți de caracteristici: 16 hărți de caracteristici.

Conexiuni: Fiecare unitate este conectată la mai multe vecinătăți 5×5 în locații identice dintr-un subset de hărți de caracteristici S2.

Parametrii și conexiunile: Conexiunile sunt parțial conectate pentru a forța hărțile de caracteristici pentru a învăța diferite caracteristici, cu 1 516 parametri antrenabili și 151 600 de conexiuni.

5. Stratul S4 (strat de subeșantionare)

Hărți de caracteristici: 16 hărți de caracteristici.

Dimensiune: 7×7 (fiecare unitate este conectată la o vecinătate de 2×2 în C3).

Parametrii: 32 de parametri care pot fi antrenați și 2 744 de conexiuni.

6. Stratul C5 (strat convoluțional)

Hărți de caracteristici: 120 de hărți de caracteristici.

Dimensiune: 1×1 (fiecare unitate conectată la o vecinătate de 5×5 pe toate cele 16 hărți de caracteristici ale lui S4, conectată efectiv în totalitate datorită dimensiunii de intrare).

Parametrii: 48.000 de parametri care pot fi antrenați și 48.000 de conexiuni.

7. Stratul F6 (strat complet conectat)

Unități: 84 de unități.

Conexiuni: Fiecare unitate este complet conectată la C5, rezultând 10 164 de parametri care pot fi instruiți.

Activare: Folosește o funcție tangentă hiperbolică scalată.

8. Stratul de ieșire

În stratul de ieșire al LeNet, fiecare clasă este reprezentată de o unitate Euclidian Radial Basis Function (RBF). Iată cum se calculează ieșirea fiecărei unități RBF:

$$y_i = \sum_j x_j . w_{ij}$$

Unde

X[j] - reprezintă intrările în unitatea RBF.

W[I,j] - reprezintă ponderile asociate fiecărei intrări.

Suma se referă la toate intrările în unitatea RBF.

În esență, ieșirea fiecărei unități RBF este determinată de distanța euclidiană dintre vectorul său de intrare și vectorul său de parametri. Cu cât distanța dintre modelul de intrare și vectorul de parametri este mai mare, cu atât mai mare este rezultatul RBF. Această ieșire poate fi interpretată ca un termen de penalizare care măsoară potrivirea dintre modelul de intrare și modelul clasei asociat unității RBF.

De reținut că:

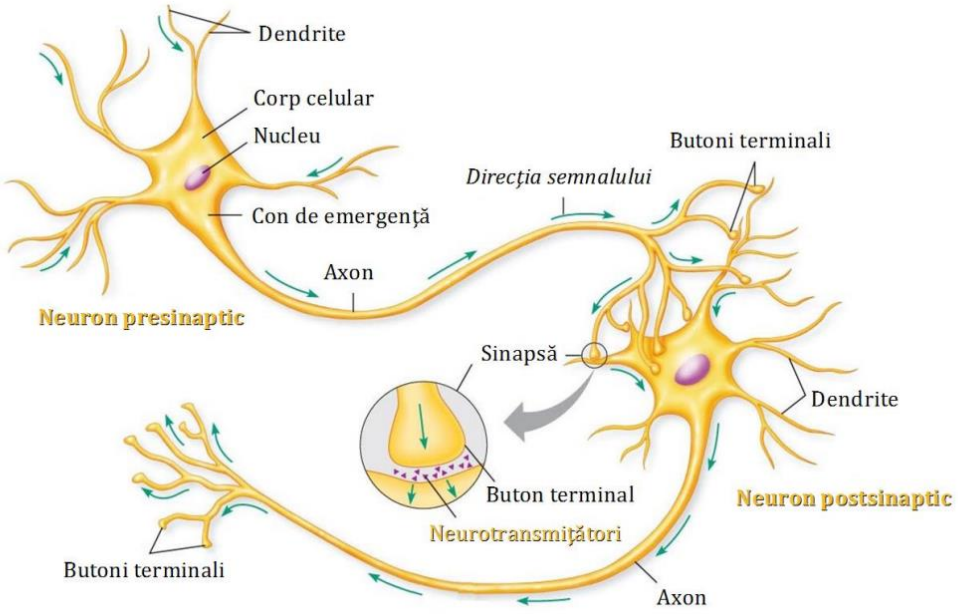
Straturi de convoluție (Cx): Aceste straturi aplică operații de convoluție la datele de intrare, utilizând mai multe filtre pentru a extrage diferite caracteristici. Filtrele alunecă pe imaginea de intrare, calculând produsul punctual dintre ponderile filtrelor și pixelii de intrare. Acest proces captează ierarhii spațiale de caracteristici, cum ar fi marginile și texturile.

Straturi de subeșantionare (Sx): Aceste straturi efectuează operațiuni de grupare (grupare medie în cazul LeNet-5) pentru a reduce dimensiunile spațiale ale hărților de caracteristici. Acest lucru ajută la controlul supraadaptării, la reducerea sarcinii de calcul și la o reprezentare mai compactă.

Straturi complet conectate (Fx): Aceste straturi sunt dens conectate, ceea ce înseamnă că fiecare neuron din aceste straturi este conectat la fiecare neuron din stratul anterior. Acest lucru permite rețelei să combine caracteristicile învățate în straturile anterioare pentru a face predicțiile finale.

Str. Italiană nr. 28, sect. 2; sect. 2; București - România
weh@spiruharet.ro weh.spiruharet.ro

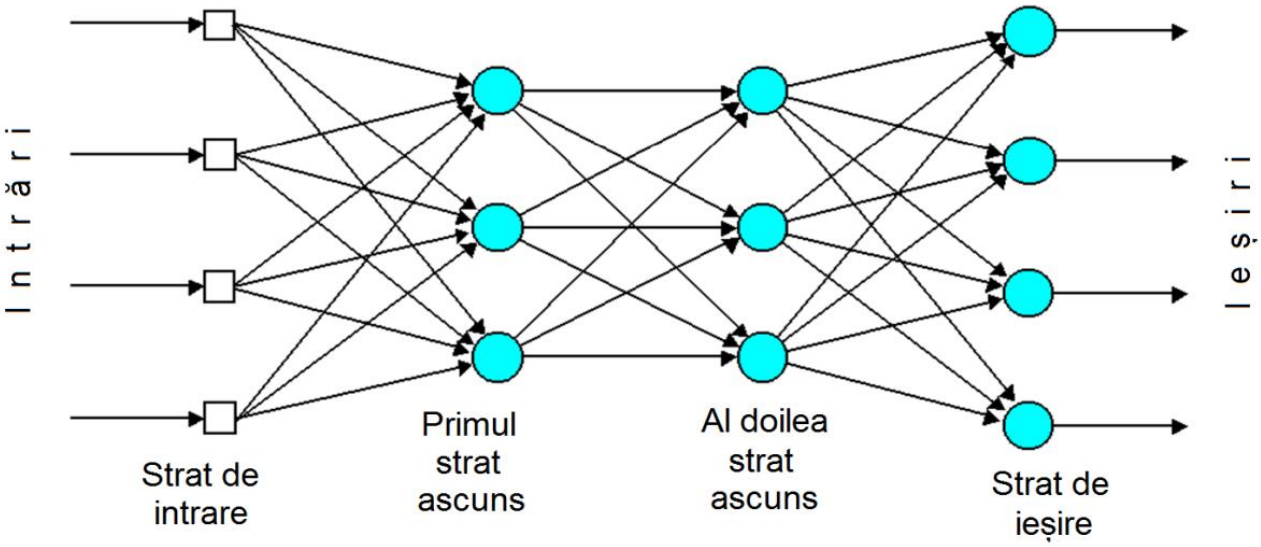
Complemente



Analogii

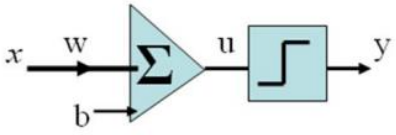
- **RN biologică**
  - Soma (corpul celulei)
  - Dendrite
  - Axon
  - Sinapsă
- **RN artificială**
  - Neuron
  - Intrări
  - Ieșire
  - Pondere (*weight*)

Str. Italiană nr. 28, sect. 2; sect. 2; București - România
weh@spiruharet.ro weh.spiruharet.ro

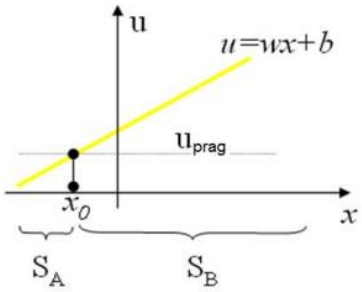


(Perceptronul multistrat)

Perceptron cu o singură intrare

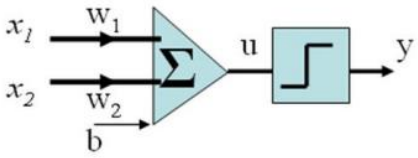


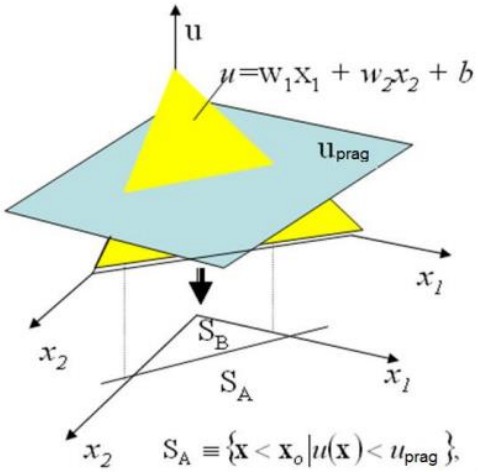
Interpretarea geometrică



$$S_A \equiv \{x < x_o | u(x) < u_{prag}\},$$
$$S_B \equiv \{x \geq x_o | u(x) \geq u_{prag}\},$$

Perceptron cu două intrări





$$S_A \equiv \{\mathbf{x} < \mathbf{x}_o | u(\mathbf{x}) < u_{prag}\},$$
$$S_B \equiv \{\mathbf{x} \geq \mathbf{x}_o | u(\mathbf{x}) \geq u_{prag}\},$$
$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix},$$

Str. Italiană nr. 28, sect. 2; sect. 2; București - România
weh@spiruharet.ro weh.spiruharet.ro

Tipuri de antrenare

- Învățarea incrementală (*online learning*)
  - Ponderile se actualizează după prelucrarea fiecărui vector de antrenare
  - Ca în exemplul anterior
- Învățarea pe lot (*batch learning*)
  - După prelucrarea unui vector de antrenare se acumulează gradienții de eroare în corecțiile ponderilor Δw
  - Ponderile se actualizează o singură dată la sfârșitul unei epoci, după prezentarea tuturor vectorilor de antrenare: $w \leftarrow w + \Delta w$
  - Avantaj: rezultatele antrenării nu depind de ordinea în care sunt prezentați vectorii de antrenare

Bibliografie

1. https://bura.brunel.ac.uk/bitstream/2438/14221/1/FullText.pdf
2. https://d2l.ai/d2l-en.pdf
3. https://cbmm.mit.edu/sites/default/files/documents/deep_neural_networks_tutorial.pdf