PERCEPTRONUL cu un singur strat

Lucian Burja

Un calculator ce simulează gândirea umană este un vis vechi care în prezent nu poate fi încă realizat. Până și cele mai performante calculatoare și programe software sunt departe de a reproduce credibil procesele cognitive care se desfășoară la nivelul creierului. Dar un fapt îmbucurător este că bazele în acest sens au fost puse, sub forma rețelelor neuronale.

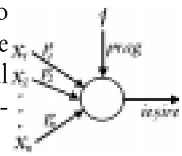
Rețelele neuronale încearcă să simuleze structura neurofiziologică a creierului uman. Creierul este alcătuit dintr-o mulțime de neuroni interconectați. Fiecare neuron primește informații de la alți neuroni prin intermediul dendritelor și transmite la rândul său un semnal prin intermediul axonului. Acest model poate fi implementat pe calculator, considerând legăturile dintre neuroni numere (ponderi) care se modifică în funcție de cât de des este activată legătura.

Primul model de rețea neuronală a apărut în 1958, când ***Frank Rosenblatt*** publică prima sa lucrare despre perceptron, denumire dată rețelei sale neuronale. Lumea oamenilor de știință a rămas fascinată de posibilitățile pe care le ofereau rețelele neuronale, care puteau fi folosite la recunoașterea unor structuri specifice electrocardiogramelor (recunoaștere de formă am spune astăzi).

Dar în 1969, ***Marvin Minsky*** și ***S. Papert*** au analizat lucid posibilitățile de învățare ale perceptronului și au ajuns la concluzii destul de sceptice, dovedind imposibilitatea rezolvării de către perceptronul cu un singur strat a unor probleme simple, cum ar fi învățarea funcției `XOR`. Ca urmare, interesul pentru acest domeniu de cercetare a scăzut dramatic. Relansarea interesului oamenilor de știință s-a produs în 1986, odată cu apariția unor modele și tehnici noi de instruire (metodele propagării înapoi, de exemplu).

În acest articol ne vom ocupa de cea mai simplă rețea neuronală posibilă, perceptronul cu un singur strat (având un singur neuron).

Neuronul are mai multe intrări și o singură ieșire. Valorile pe care le poate avea ieșirea sunt `+1` și `-1`, deci neuronul poate realiza o clasificare a datelor de intrare în două categorii.



El posedă și un prag de activare, adică ieșirea sa poate fi `+1` sau `-1` în funcție de semnalul de intrare și de valoarea *prag*. Dacă semnalul de intrare depășește valoarea *prag*, atunci ieșirea va fi `+1`, în caz contrar ea fiind `-1`. Semnalul de intrare se calculează folosind o sumă ponderată a valorilor de intrare:

$$I' = \sum_{i=1}^{n} x_i \cdot p_i$$

unde:

- $x_1, x_2, \ldots, x_n$ sunt datele de intrare;
- $p_1, p_2, \ldots, p_n$ sunt ponderile legăturilor;
- n este numărul intrărilor.

Dacă $I' > prag$, ieșirea va avea valoarea `+1`, iar în caz contrar va avea valoarea `-1`.

Pentru simplitate, putem considera pragul ca fiind o intrare separată care are întotdeauna valoarea semnalului de intrare egală cu `+1` și ponderea egală cu pragul. În acest caz semnalul de intrare al neuronului va fi $I = I' - prag$ și avem condiția de activare $I > 0$. Avantajul acestei abordări este acela că valoarea *prag* poate fi și ea modificată în procesul de instruire, împreună cu celelalte ponderi.

Procesul de instruire este cel care permite perceptronului să poată clasifica datele de intrare. Prin acest proces, perceptronul "învață" și aici găsim secretul succesului pe care îl au rețelele neuronale. De multe ori, în practică, este dificil de implementat un algoritm de rezolvare a unei anumite probleme (recunoaștere de forme, de imagini, de voce etc.) sau rezultatele sunt nesatisfăcătoare. Rețelele neuronale nu au nevoie să cunoască nici un algoritm, ci pot fi antrenate folosindu-se un set de exemple, ceea ce le face deosebit de utile pentru aplicații de genul celor de mai sus. Performanțele obținute prin folosirea rețelelor sunt adesea excelente. De exemplu, s-a creat programul *TD Gammon*, care joacă table pe baza unei rețele neuronale la nivel de maestru internațional. Pentru instruire, programul a pornit fără cunoștințe prealabile și a atins performanțe remarcabile după ce a jucat câteva sute de mii de partide contra lui însuși.

Revenind la perceptron, antrenarea acestuia se desfășoară în următorul mod: ponderile inițiale sunt generate aleator; apoi se prezintă rețelei date de intrare pentru care se cunoaște răspunsul dorit. Dacă răspunsul rețelei este diferit față de răspunsul dorit, atunci se ajustează ponderile, în așa fel încât perceptronul să clasifice corect datele de intrare. După ce rețeaua clasifică corect toate exemplele, putem considera că antrenarea s-a încheiat și ne putem aștepta ca în momentul introducerii unor seturi de date noi, perceptronul să răspundă corect.

Desigur, ne punem întrebarea dacă întotdeauna este posibil să găsim valori ale ponderilor, astfel încât perceptronul să clasifice corect datele de intrare. ***Teorema de convergență*** a perceptronului ne asigură că, pentru clase liniar separabile există întotdeauna un vector pondere astfel încât rețeaua să clasifice corect exemplele de instruire. Demonstrarea acestei teoreme presupune elemente de matematică superioară și poate fi găsită în resursele indicate în bibliografie. Prin ***clase liniar separabile***, înțelegem două clase ce pot fi despărțite printr-un hiperplan (o dreaptă în plan, un plan în spațiul 3D etc.) Acest concept va deveni mai clar puțin mai încolo, când vom apela la un exemplu. Pentru moment este suficient să reținem că ecuația hiperplanului de separație este furnizată de valorile ponderilor perceptronului.

O altă problemă care ne-o punem este cum să modificăm ponderile rețelei, în așa fel încât să ajungem la soluție, atunci când ea există.

Ideea de bază pentru antrenarea oricărui tip de rețea este folosirea unei ***funcții criteriu*** pe care încercăm să o minimizăm. În cazul nostru ***funcția criteriu*** este reprezentată de suma distanțelor până la hiperplanul de separație a punctelor greșit clasificate. Cu cât valoarea acestei funcții este mai mică (chiar zero), cu atât hiperplanul nostru separă mai bine cele două clase, deci este mai apropiat de soluție.

Minimizarea funcției criteriu se poate face prin ***metode de tip gradient***. Deoarece sunt necesare noțiuni de matematică ce depășesc programa de liceu, vom prezenta numai rezultatul final, demonstrația putând fi găsită în resursele indicate în bibliografie. Regula de corecție este:

$$p_{t+1} = p_t + \frac{c}{2} \cdot (d_t - o_t)$$

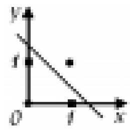
unde:

- p_{t+1} este vectorul de ponderi la pasul t+1;
- p_t este vectorul de ponderi la pasul t;
- d_t este rezultatul dorit la pasul t;
- o_t este rezultatul obținut la pasul t
- x_t este vectorul valorilor de intrare la pasul t;
- c este constanta de corecție ($0 < c \leq 1$)

Liniar separabilitatea este una din cele mai importante limitări ale perceptronului. De multe ori, exemplele de instruire în practică, nu constituie clase liniar separabile. Una dintre cele mai celebre probleme pe care un perceptron cu un singur strat nu o poate rezolva este problema funcției `XOR`. Tabelul de valori al funcției `XOR` este următorul:

x	y	x XOR y
1	1	0
1	0	1
0	1	1
0	0	0

Dacă reprezentăm funcția într-un sistem de axe, obținem următorul grafic:



unde prin cerculețe am notat valorile `0` ale funcției, iar prin pătrățele valorile `1`. Se observă că oricum am duce o linie, nu vom reuși niciodată să separăm cele două clase de puncte.

O modificare a algoritmului clasic de instruire a fost adusă de către ***Gallant*** în anul 1986, astfel încât acesta să poată fi aplicat și claselor liniar inseparabile. Ideea este de a antrena rețeaua printr-un număr mare de pași, și de a reține configurația care a clasificat corect cele mai multe date de intrare. De multe ori însă, acest algoritm nu prezintă aplicabilitate practică, deoarece sunt necesare extrem de multe iterații pentru a ajunge la o soluție acceptabilă.

Un alt algoritm de instruire a perceptronului cu un singur strat este algoritmul ***Widrow-Hoff***, cunoscut și sub numele de ***regula delta***. Acest algoritm alege drept ***funcție criteriu*** suma erorilor pătratice:

$$\sum_{k=1}^{n} (d_k - o_k)$$

unde n reprezintă numărul exemplelor de antrenare, d_k ieșirea dorită pentru exemplul k, iar o_k rezultatul obținut. Se urmărește minimizarea ei, folosind din nou ***metode de tip gradient*** și se ajunge la următoarea regulă de instruire:

$$p_{t+1} = p_t + \frac{c}{t} \cdot \delta_t \cdot x$$

$$\delta_t = d_t - o_t$$

unde simbolurile au semnificația cunoscută.

În acest caz, rețelei i se prezintă exemple de instruire, până când eroarea de clasificare este mai mică decât o eroare acceptabilă, moment în care încheiem procesul de antrenare.

Am realizat implementările celor doi algoritmi prezentați anterior. Acestea sunt disponibile pentru download la adresa www.ginfo.ro/revista/11_7.

În continuare vom prezenta o aplicație simplă în scopul de a demonstra funcționalitatea perceptronului.

Vom considera o dreaptă în plan (de exemplu prima bisectoare) și o mulțime de puncte situate deasupra sau sub dreaptă. Rețelei îi vom prezenta câteva exemple de puncte situate deasupra (ieșirea corectă `+1`) și dedesubt (ieșirea corectă `-1`), apoi o vom testa pentru puncte care nu sunt conținute în datele de antrenare. Datele de intrare se citesc din fișierul **`PERCEPT.TXT`**, unde pe prima linie avem numărul de puncte de antrenare, iar pe următoarele *n* linii triplete de numere ce reprezintă coordonatele punctului și răspunsul așteptat (`+1` sau `-1` în funcție de poziția punctului - deasupra sau sub linia considerată).

Programul declară două rețele care vor fi antrenate cu algoritmul clasic, respectiv cu ***algoritmul delta***. Parametrul din apelul constructorului indică numărul de intrări ale rețelei (două intrări în cazul nostru, corespunzător celor două coordonate ale unui punct în plan). Antrenarea se face prin apelul funcției membru `Antreneaza()` a fiecărei clase. Se observă că această funcție este redefinită în clasa `Perceptron_Delta`, deoarece algoritmul de instruire este, în mod evident, altul. Răspunsul se obține prin apelarea funcției `Iesire()`.

Codul sursă al programului este disponibil pentru download la adresa www.ginfo.ro/revista/11_7.

Desigur, această problemă se poate extinde în spațiul tridimensional, sau chiar mai departe, prin mărirea numărului de intrări ale neuronului.

Se observă că, în general, *algoritmul de instruire* al lui ***Widrow-Hoff*** produce o soluție mai bună decât algoritmul clasic, deoarece avem posibilitatea să controlăm eroarea hiperplanului de separație. În cazul algoritmului clasic, instruirea se oprește în momentul în care toate exemplele de instruire au fost corect clasificate, chiar dacă soluția obținută este acceptabilă "la limită". Dacă clasele nu sunt liniar separabile, *algoritmul delta* va produce cu siguranță o soluție mai bună, deoarece algoritmul clasic va oscila și nu va converge niciodată, și chiar și în varianta *Gallant* va produce rezultate mai puțin satisfăcătoare.

Concluzie

Gama de probleme pe care un perceptron cu un singur strat le poate rezolva este destul de limitată, dar el reprezintă punctul de plecare pentru rețele mult mai complexe cu aplicații practice. Exemplul prezentat mai sus are scop pur didactic, dar considerăm că reușește să dea o imagine asupra puterii ce se află în spatele unei rețele neuronale. Dacă un singur neuron a reușit să rezolve problema de mai sus, putem ușor să ne închipuim ce pot face mai mulți neuroni organizați pe mai multe straturi…

Bibliografie

1. D. Dumitrescu, Hariton Costin, *Rețele neuronale. Teorie și aplicații*, Editura Teora, București, 1996;
2. http://satirist.org/learn-game/systems/gammon/;
3. http://www.geocities.com/CapeCanaveral/1624/;
4. http://home.cc.umanitoba.ca/~umcorbe9/perceptron.html;
5. http://www.cs.bgu.ac.il/~omri/Perceptron/.

Lucian Burja este student în anul II la Universitatea Politehnica din București. Poate fi contactat prin e-mail la lucian_burja@email.ro.

Tasta "magică"

Sistemele de operare de la *Microsoft* au implementate, pe lângă specificațiile din documentații, și alte funcții care au diferite roluri.

De exemplu, tasta *Shift* folosită în diferite combinații, modifică funcția care ar fi fost efectuată dacă această tastă nu ar fi fost apăsată. În continuare, vom prezenta câteva dintre opțiunile la care avem acces folosind tasta "magică".

Pentru a reporni interfața grafică *Windows '95/'98* (GUI) și o parte din *drivere,* efectuați un click pe butonul *Start*, selectați opțiunea *Shut Down* și, în fereastra de dialog selectați *Restart computer* și țineți apăsată tasta *Shift* în timp ce apăsați pe butonul *OK*.

Majoritatea CD-urilor apărute în ultimul timp au un program care se execută la introducerea CD-ului în unitate. Pentru a evita pornirea acestui program, țineți apăsată tasta *Shift* din momentul în care introduceți CD-ul în unitate, până când acesta este accesat.

Dacă vreți să ștergeți unul sau mai multe fișiere selectate fără a le mai putea recupera (fără să mai treacă prin *Recycle Bin*) apăsați combinația de taste *Shift + Delete*.

Pentru a deschide un document folosind o altă aplicație (diferită de cea care este asociată cu tipul de document respectiv), țineți apăsată tasta *Shift* în timp ce efectuați un click dreapta pe fișierul selectat (sau apăsați *Shift + F10*, fără click de mouse). În acest fel va apărea opțiunea *Open With...* în meniul popup...

Dacă ați configurat *Explorer*-ul să deschidă pentru fiecare director o fereastră nouă, pentru a închide simultan toate ferestrele deschise, țineți apăsată tasta *Shift* în timp ce închideți ultima fereastră deschisă.

Pentru a intra în *Safe Mode*, tot tasta *Shift* este cea care vă salvează, chiar dacă meniul de boot este dezactivat. Mențineți tasta *Shift* apăsată în momentul încărcării sistemului de operare.