

Algoritmi de căutare aleatoare. Algoritmi de tip Simulated Annealing

octombrie 2003

1 Motivație

Algoritmii de învățare bazați pe minimizarea unui criteriu de eroare folosesc metode numerice de minimizare. Algoritmii tradiționali de învățare, cum este algoritmul "backpropagation", folosesc metode de descreștere de tip gradient. Aceste metode au caracter local conducând la minimul din vecinătatea căruia aparține aproximația inițială. O dată ajuns într-un minim local procesul iterativ rămâne blocat acolo, ceea ce din punct de vedere al unui algoritm de învățare reprezintă o învățare incompletă. Un alt dezavantaj pe care îl prezintă algoritmii bazați pe metode de tip gradient este că necesită ca funcția obiectiv să fie suficient de netedă în vecinătatea minimului căutat, ceea ce poate fi o restricție în anumite situații.

Algoritmii aleatori prezentați în continuare au avantajul că nu impun ipoteze de netezime asupra funcției obiectiv și datorită prezenței perturbațiilor permit uneori evitarea minimelor locale. Principalul dezavantaj al acestor metode este faptul că nu asigură o convergență în sens clasic ci doar convergență în sens probabilist (de exemplu, probabilitatea ca aproximația să fie într-o vecinătate oricât de mică a optimului global tinde către 1).

Să considerăm problema de minimizare:

Fie $F : A \subset R^n \rightarrow R$ o funcție. Să se determine $x^* \in A$ cu proprietatea că $f(x^*) \leq f(x)$ pentru orice $x \in A$.

Algoritmii care permit rezolvarea unei astfel de probleme necesită stabilirea a două elemente: *direcția de căutare* și *dimensiunea "pasului"* efectuat în direcția respectivă. Algoritmii aleatori de căutare se caracterizează prin utilizarea unor elemente aleatoare în stabilirea direcției de căutare.

2 Algoritmi simpli de căutare aleatoare

Ideea acestor algoritmi este de a stabili în mod aleator direcția de căutare. Ei sunt, la fel ca metodele de tip gradient, algoritmi de descreștere, în sensul că direcția nou generată este acceptată doar dacă asigură micșorarea funcției cost însă nu folosesc nici o informație asupra gradientului funcției f .

2.1 Structura unui algoritm aleator de descreștere.

Pas 1. *Inițializări.* Se alege aproximația inițială, x_0 ; se inițializează contorul de iterații, $k = 0$; se inițializează un contor de eșecuri, $e = 0$.

Pas 2. *Generarea noii direcții.* Se generează o valoare aleatoare ξ_k și se efectuează analiza:
dacă $f(x_k + \xi_k) < f(x_k)$ atunci $x_{k+1} = x_k + \xi_k$
altfel $x_{k+1} = x_k$; $e = e + 1$.

Pas 3. *Criteriul de oprire.* Dacă este satisfăcut criteriul de oprire atunci se oprește algoritmul, altfel se incrementează indicatorul iterației (k) și se reia de la Pasul 2.

Criteriul de oprire poate să fie $k = k_{max}$ (numărul maxim de iterații) sau $e = e_{max}$.

2.2 Exemple.

Variante ale algoritmului se obțin prin concretizarea repartiției valorilor variabilei aleatoare ξ_k :

1. *Algoritmul Matyas (1965)*: Pentru fiecare iterație k , ξ_k se generează în conformitate cu repartiția normală standard (medie 0 și dispersie 1) Valorile corespunzătoare unor iterații diferite vor fi independente.
2. *Algoritmul Solis-Wets (1981)*: ξ_k se generează în conformitate cu repartiția normală de medie m_k și dispersie 1. Se alege $m_0 = 0$ și pentru fiecare k se face analiza:
Dacă $f(x_k + \xi_k) < f(x_k)$
atunci $x_{k+1} = x_k + \xi_k$, $m_{k+1} = 0.4\xi_k + 0.2m_k$;
Dacă $f(x_k - \xi_k) < f(x_k)$
atunci $x_{k+1} = x_k - \xi_k$, $m_{k+1} = m_k - 0.4\xi_k$;
Dacă $f(x_k + \xi_k) \geq f(x_k)$ și $f(x_k - \xi_k) \geq f(x_k)$
atunci $x_{k+1} = x_k$ și $m_{k+1} = 0.5m_k$.

Pentru generarea de valori repartizate în conformitate cu repartiția normală standard se poate folosi metoda Box-Muller care conduce la următorul algoritm:

```
u:=Random;  
v:=Random;  
r:=sqrt(-2*ln(u));  
z1:=r*cos(2*pi*v);  
z2:=r*sin(2*pi*v);  
Return z1,z2
```

unde `Random` este o funcție ce generează valori uniform distribuite în $[0, 1]$. O variantă a acestui algoritm care evită folosirea funcțiilor trigonometrice și care se bazează pe generarea de puncte uniform distribuite în interiorul cercului unitate este:

```
Repeat  
  u:=2*Random-1;  
  v:=2*Random-1;  
  s=u^2+v^2;  
Until 0<s<1;  
r:=sqrt(-2*ln(s)/s);  
z1:=r*u;  
z2:=r*v;  
Return z1, z2
```

La fiecare apel al funcțiilor de mai sus se generează două valori normal repartizate. Dacă z este o valoare aleatoare având repartiția normală standard pentru a obține o valoare aleatoare repartizată în conformitate cu repartiția normală de medie m și dispersie d este suficient să se calculeze: $m + d \cdot z$.

2.3 Studiul convergenței.

Rezultate suficiente privind convergența în probabilitate a lui x_k sunt date în teorema următoare:

Teorema 2.1 *Fie x^* minimul global al funcției $f : A \subset \mathbb{R}^n \rightarrow \mathbb{R}$. Presupunem că*

1. $f : A \subset \mathbb{R}^n \rightarrow \mathbb{R}$ este continuă;
2. pentru fiecare $\delta > 0$ mulțimea $V_\delta = \{x \mid \|x - x^*\| < \delta\}$ este de măsură nenulă, $M(V_\delta) \neq 0$;
3. densitatea de probabilitate, q_k a fiecărei variabile aleatoare ξ_k are proprietatea: $q_k(x - y) > 0$ pentru orice $x, y \in A$.

Atunci $f(x_k) \xrightarrow{i.p.} f(x^*)$.

2.4 Aplicații la rețele neuronale:

- N. Baba et al (1989, 1994): combină algoritmul "backpropagation" cu algoritmul Solis-Wets și testează noul algoritm în probleme de predicție.
- Williams et Matsuoka (1992): utilizează algoritmi aleatori în antrenarea unei rețele destinate rezolvării unor probleme de control.

2.5 Deficiențe și soluții

Faptul că este o metodă de descreștere face ca blocarea în minime locale a procesului de căutare să nu fie întotdeauna evitată. Din același motiv rezultatul obținut este influențat esențial de aproximația inițială. Soluții pentru eliminarea acestor deficiențe sunt: (a) execuția repetată a algoritmului pentru diverse aproximații inițiale și folosirea rezultatelor obținute la rulările anterioare în stabilirea aproximației inițiale; (b) eliminarea caracterului de metodă de descreștere acceptându-se și perturbațiile care conduc la creșteri ale funcției obiectiv.

3 Algoritmi de tip "Simulated Annealing"

Algoritmii din această categorie nu sunt algoritmi de descreștere întrucât se pot efectua ajustări care conduc la creșterea funcției obiectiv. Aceste ajustări se fac însă cu o probabilitate mică.

Ideea acestor algoritmi provine de la analogia dintre căutarea soluției unei probleme de optimizare și evoluția stărilor unui solid supus unui tratament termic care începe printr-o încălzire bruscă și continuă cu o răcire lentă. La temperatură ridicată particulele solidului (aflat în fază lichidă) se aranjează aleator. Pe măsură ce temperatura scade particulele tind să ajungă în stări cu energie din ce în ce mai mică. Dacă temperatura scade *suficient de lent* atunci, pentru fiecare valoare a temperaturii, solidului îi este permis să atingă starea de așa-numit *echilibru termic*. Starea de echilibru termic este descrisă de distribuția de probabilitate:

$$P_T(S) = \frac{1}{Z(T)} \exp\left(-\frac{E(S)}{k_B T}\right) \quad (1)$$

unde $S \in \mathcal{S}$ este o stare a sistemului, T este temperatura la care se află sistemul, $Z(T)$ este un factor de normalizare, $E(S)$ este energia stării S , iar $k_B > 0$ este o constantă (în fizică este denumită constanta lui Boltzmann). Se observă că pentru un T dat, cu cât $E(S)$ este mai mică cu atât probabilitatea, $P_T(S)$, ca sistemul să se afle în starea S este mai mare. Prin urmare probabilitatea este maximă pentru stările de energie minimă.

Pe de altă parte, și T influențează distribuția staționară. Valori mari ale lui T ($T \rightarrow \infty$), fac ca $P_T(S) \simeq 1/\text{card}\mathcal{S}$ adică stările sunt aproape echiprobabile. Dacă T este mic ($T \rightarrow 0$) atunci vor avea probabilitate nenulă doar stările de energie minimă.

Folosind analogia dintre un sistem fizic și o problemă de optimizare, analogie bazată pe corespondențele: spațiul stărilor sistemului coincide cu spațiul configurațiilor problemei iar energia coincide cu funcția obiectiv rezultă că pentru a identifica configurații de cost minim ar fi suficient să le *generăm* în conformitatea cu repartiția (1). Acest lucru este dificil de realizat practic datorită necesității calculului lui $Z(T)$ (care este o sumă după toate configurațiile posibile ceea ce presupune o parcurgere exhaustivă a spațiului de căutare - imposibil de realizat pentru probleme de dimensiuni mari).

Pentru a evita acest calcul se poate simula evoluția unui proces stohastic (de tip lanț Markov) către distribuția staționară. Metropolis a propus o metodă de simulare de tip Monte-Carlo a cărei idee este următoarea: pornind de la starea curentă a sistemului este generată o perturbație aleatoare (mică) care este aplicată uneia dintre componentele (particulele) sistemului. Dacă această perturbație provoacă o descreștere a energiei sistemului atunci noua stare va fi considerată cea obținută prin aplicarea perturbației, altfel perturbația se va accepta numai cu o anumită probabilitate (care depinde invers proporțional de variația energiei și direct proporțional de temperatură). Astfel cu cât temperatura este mai ridicată, probabilitatea de a accepta schimbări care conduc la mărirea energiei este mai mare (particulele sistemului fluctuează aleator).

Aplicarea acestei idei pentru determinarea minimului unei funcții obiectiv se bazează pe următoarele analogii:

- funcția obiectiv, f , coincide cu energia, E ;
- vectorii din domeniul de definiție corespund unei configurații (stări) a sistemului;
- modificarea unei componente a vectorului corespunde perturbării unei particule;
- T este un parametru prin care se poate controla probabilitatea de acceptare a perturbațiilor care conduc la creșterea funcției obiectiv.

3.1 Structura unui algoritm de tip Metropolis

Pentru T constant, un algoritm de tip Metropolis are următoarea structură:

Pas 1. *Inițializări.* Se stabilește aproximația inițială, x_0 , și se inițializează indicatorul iterației, $k = 0$;

Pas 2. *Generare perturbație.* Se generează o nouă configurație $x' \in R^n$ prin modificarea uneia sau mai multor componente ale lui x .

Pas 3. *Acceptarea noii configurații.* Se calculează

$$\Delta f_k = f(x') - f(x_k).$$

Configurația x' se acceptă cu probabilitatea

$$P_a(x_{k+1} = x') = \min\{1, \exp(-\Delta f_k/T)\}, \quad T > 0. \quad (2)$$

Pas 4. *Criteriu de oprire.* Dacă $k < k_{max}$ atunci $k = k + 1$ și se trece la Pas 2, altfel algoritmul se oprește.

Generarea unei noi valori x' pornind de la x_k se face printr-o perturbație care depinde de natura problemei de rezolvat. Ideea generală este că fiecărei configurații x i se asociază o "vecinătate" de configurații, iar perturbația constă în selecția unui element din această vecinătate.

De exemplu, pentru problema comis voiajorului de dimensiune n , o configurație este o permutare de ordin n . Perturbarea unei configurații poate consta în inversarea a două elemente vecine (corespunde inversării ordinii de parcurgere a două orașe) sau în inversarea ordinii unei secvențe din permutare (corespunde inversării ordinii de parcurgere a unei succesiuni de orașe și este o transformare de tip 2-opt în cadrul metodelor euristice de rezolvare a problemei TSP).

În cazul rezolvării unor probleme de optimizare continuă perturbațiile pot fi obținute prin generarea de valori aleatoare corespunzătoare unei anumite distribuții de probabilitate. Exemple de perturbații aleatoare sunt:

1. perturbația $\Delta x = x' - x$ se generează aleator după repartiția normală de medie 0 și dispersie T (având densitatea de probabilitate $q(u) = \exp(-u^2/(2T^2))/(T\sqrt{2\pi})$;
2. perturbația $\Delta x = x' - x$ se generează aleator după repartiția Cauchy de parametru T a cărei densitate de probabilitate este:

$$q(u) = \frac{T}{\pi(u^2 + T^2)}$$

3. pentru fiecare componentă $x_i \in [A_i, B_i]$ deplasamentul va fi $\Delta x_i = y^i(B_i - A_i)$ unde y^i sunt variabile aleatoare independente având densitatea de repartiție:

$$q(u) = \begin{cases} \frac{1}{2(|u|+T)\ln(1+1/T)}, & u \in [-1, 1] \\ 0, & \text{altfel} \end{cases}$$

În toate situațiile tranziția de la o configurație la alta poate fi văzută ca un proces aleator caracterizat prin probabilitățile de tranziție asociate, numite *probabilități de generare* și notate cu P_g .

O altă formă a probabilității de acceptare este:

$$P_a(x_{k+1} = x') = 1/(1 + \exp(\Delta f_k/T)).$$

În această situație o descreștere a funcției obiectiv este acceptată cu o probabilitate mai mare decât 0.5 iar o creștere cu o probabilitate mai mică decât 0.5.

Funcționarea algoritmului Metropolis este controlată în mod esențial de valoarea parametrului T . Astfel, dacă $T \rightarrow 0$ atunci algoritmul tinde către unul determinist de descreștere (sunt acceptate doar configurațiile care asigură descreșterea funcției obiectiv), iar dacă $T \rightarrow \infty$ atunci toate valorile nou generate sunt acceptate, iar algoritmul realizează o căutare necontrolată în tot spațiul stărilor.

Problema cea mai dificilă în proiectarea unui algoritm de tip Metropolis este alegerea valorii potrivite pentru T . O soluție pentru această problemă o reprezintă modificarea valorii lui T în cadrul algoritmului. Pe această idee se bazează algoritmi de tip "Simulated annealing" (recoacere sau călire simulată) în care valoarea inițială a lui T este mare (căutarea se realizează pe întreg spațiul stărilor) după care T este micșorat astfel că domeniul de căutare este treptat limitat.

3.2 Structura unui algoritm de tip "Simulated Annealing"

Aplicând algoritmul Metropolis pentru valori ale lui T din ce în ce mai mici se obține:

Pas 1. *Inițializări.* Se inițializează indicatorul iterației ($m = 0$), aproximația x_0 și temperatura T_0 .

Pas 2. *Iterații Metropolis.* Se determină x_{m+1} utilizând algoritmul Metropolis pornind de la x_m și iterând de p_{max} ori (pentru temperatura T_m).

Pas 3. *Criteriu de oprire.* Dacă $m < m_{max}$ atunci se incrementează m , se stabilește noua valoare T_{m+1} a parametrului temperatură și se reia de la Pas 2, altfel algoritmul se oprește.

Eficiența algoritmului depinde de strategia de descreștere a temperaturii. Dacă descreșterea este prea rapidă algoritmul se oprește în minime locale (corespunde situației în care topitura fiind răcită prea brusc, solidul este "înghețat" într-o stare care conține și "defecte"). Pe de altă parte dacă temperatura descrește prea lent atunci apropierea de minimul global durează prea mult.

Strategii simple de descreștere a parametrului T sunt:

1. $T_m = T_0 / \ln(m + m_0 + 1)$, $m_0 > 1, T_0 > 0$;
2. $T_m = T_0 / (m + 1)$, $T_0 > 0$.

3.3 Proprietăți de convergență.

Condiții suficiente pentru a asigura convergența în probabilitate a șirului aproximațiilor x_m sunt date în [Laarhoven și Aarts, 1987]. Aceste rezultate pot fi sintetizate în:

Presupunem că sunt satisfăcute condițiile:

- (a) probabilitatea de trecere între oricare două stări (în faza de generare a perturbațiilor) este nenulă și simetrică, $P_g(x, x') = P_g(x', x) > 0$ pentru orice x și x' ;
- (b) probabilitatea de acceptare este de tip Metropolis (relația (1)).
- (c) $T_m = C_f / \ln m$ cu $C_f > 0$ o constantă suficient de mare (valoarea minimă pe care o poate lua C_f depinde de proprietățile lui f).

Atunci

$$\lim_{m \rightarrow \infty} P(f(x_m) = f(x^*)) = 1.$$

3.4 Aplicații pentru rețele neuronale

La rețele neuronale algoritmi de tip Simulated Annealing pot fi aplicați în două direcții:

- Minimizarea funcției de energie în cazul învățării supervizate la rețelele feedforward. În acest caz sunt utilizați pentru minimizarea unor funcții cu multe minime locale și sunt definite pe un domeniu continuu .
- Determinarea stării de energie minimă la rețelele recurente stohastice (modelul Hopfield pentru rezolvarea problemelor de optimizare sau mașina Boltzmann utilizată în rezolvarea problemelor de asociere). În acest caz sunt utilizați pentru determinarea unei configurații de cost minim dintr-un spațiu discret.

3.5 Observații privind implementarea.

Algoritmi de tip Simulated Annealing sunt algoritmi *aproximativi* de optimizare, dar prezintă avantajul că au o structură simplă și sunt cu caracter general. Eficiența lor este însă puternic influențată de modul de alegere a elementelor constitutive:

- *Metoda de perturbare.* Este elementul care depinde cel mai mult de problema concretă. În alegerea ei fie se pornește de la euristici specifice domeniului (cum este euristica 2-opt pentru problema comisvoiajorului) fie se caută distribuții P_g care să satisfacă condițiile teoretice privind convergența (cum sunt perturbațiile bazate pe repartiția normală sau pe repartiția Cauchy).
- *Strategia de acceptare.* Nu influențează în mod esențial eficiența astfel că în majoritatea aplicațiilor se poate folosi strategia clasică, de tip Metropolis.
- *Strategia de modificare a lui T - strategia de răcire.* Este partea cea mai dificilă din implementare și presupune:
 1. *Stabilirea valorii inițiale pentru T* - suficient de mare astfel încât la primele iterații aproape toate perturbațiile să fie acceptate ($\exp(-\Delta/T_0) \simeq 1$, cu Δ cea mai mare variație posibilă a funcției obiectiv).
 2. *Stabilirea numărului de repetări ale pasului Metropolis* (numărul de tranziții efectuate la o aceeași temperatură). Teoretic ar trebui ca procesul iterativ să continue până la atingerea "echilibrului". Caracterul aleator al procesului face ca verificarea să fie dificilă (necesită colectarea unor statistici pentru identificarea momentului în care se atinge distribuția staționară). Variante simple sunt: (i) se repetă de un număr de ori proporțional cu dimensiunea problemei (în felul acesta se oferă șansa fiecărei componente de a fi perturbată); (ii) se repetă până când un număr minim de tranziții (stabilit de către utilizator) au fost efectiv operate.
 3. *Determinarea regulii de descreștere a lui T_k .* Pe lângă cele două variante amintite deja, o modalitate simplă și frecvent utilizată este $T_{k+1} = \alpha T_k$ cu $0 < \alpha < 1$ dar *suficient* de apropiată de 1 (de exemplu, $\alpha = 0.95$). O altă variantă este: $T_{k+1} = T_k / (1 + \gamma T_k / U)$ cu γ un număr real mic, iar U o margine superioară a diferenței $f(x_k) - f^*$ (dezavantajul este că U depinde de "geometria" problemei și este dificil de estimat).
 4. *Alegerea criteriului de oprire.* Cea mai naturală variantă este cea în care T_k devine suficient de mic. Se mai poate opta și pentru determinarea ratei de acceptare și oprirea algoritmului când aceasta devine foarte mică (înseamnă că șansa de a găsi configurații mai bune este foarte mică).

3.6 Alte aplicații

Algoritmi de tip Simulated Annealing au fost aplicați cu succes în: prelucrarea imaginilor (eliminarea zgomotului și segmentare [Geman & Geman, 1984]), în rezolvarea problemelor de rutare, în modelarea structurilor amorfe, în analiza datelor obținute prin investigații bazate pe difracție cu raze X sau rezonanță magnetică nucleară, în planificarea activităților etc.

4 Bibliografie

1. N. Baba et al., A Hybrid algorithm for finding the global minimum of error function of neural networks and its applications, *Neural Networks*, vol. 7, no. 8, pg. 1253-1265, 1994.
2. R. Brunelli, Training neural nets through stochastic minimization, *Neural Networks*, vol. 7, no. 9, pg. 1405-1412, 1994.
3. V. Williams, K. Matsuoka, Learning of neural controllers by random search technique, *IEICE Trans. Inf. & Syst*, vol. E75 D, no. 4, 1992.
4. F.J. Solis, R.J-B. Wets, Minimization by random search techniques, *Mathematics of Operations Research*, vol. 6, no. 1, 1981.
5. P.J.M. van Laarhoven, E.H.L. Aarts, *Simulated annealing: theory and applications*, D. Reidel Publ. Company, 1987.
6. B. Rosen, *Function Optimization based on Advanced Simulated annealing*, preprint 1993.
7. L.Ingber, Very fast simulated re-annealing, *J. Math.Comp.Modelling*, 12, 967-973, 1989.
8. Geman S., Geman D.; Stochastic Relaxation, Gibbs Distributions and the Bayesian Restoration of Images, *IEEE Proc. PAMI-6*, p. 721-741, 1984.