Algoritmy a struktury neuropočítačů
ASN – P5

Vícevrstvé sítě s učením zpětného šíření chyby

BPG

(*Error Back-propagation Algorithm*
Back-propagation of Gradient Algorithm)

- **dopředné propojení a šíření signálu**
- **heteroasociativní sítě s učitelem**
  (je požadována znalost cílových hodnot)

- **off-line učení – počítají se synaptické váhy a prahy**
  batch training **pro všechny tréninkové vzorky,**
  modifikace vah až po přivedení
  všech vzorků do sítě, počítá se
  aktuální gradient chyby *E* pro úplnou
  množinu tréninkových vzorků

- **on-line učení - váhy a prahy jsou modifikovány**
  incremental **bezprostředně po přivedení každého**
  training **tréninkového vzorku,**
  proces konverguje stochasticky
  k lokálním minimům a nezaručuje
  dosažení absolutního minima

Princip:

1) **Na vstup NS je přiváděn vektor resp. matice vstupních parametrů - číselné hodnoty pro konkrétní hodnoty fyzikálních veličin**
   - kategoriální data (přiřazení kategorie určité vlastnosti, číselná hodnota udává váhu dané vlastnosti vzhledem k ostatním).
2) **Po průchodu sítí je spočítán výstup z každého neuronu a výsledek je porovnán s požadovanou hodnotou.**
3) **Je spočítána chyba, ta se zpětně přepočítává do předchozích vrstev a synaptické váhy představující paměť jsou opraveny.**
4) **Do opravené sítě je znovu přiveden vstupní vektor resp. matice a proces se opakuje.**

Iterativní proces, hledání minima chyby mezi skutečnou (výstupní) hodnotou a požadovanou hodnotou.

Nevýhoda: velká citlivost na relevantnost vstupních dat a na inicializaci synaptických vah.

- **minimalizace rozhodovací funkce (energetická funkce, *cost function*):**

  → **gradientní metoda**

- **rozhodovací funkce - střední kvadratická chyba mezi požadovaným a skutečným výstupem**

- **aktivační funkce - spojitá diferencovatelná nelineární funkce (např. sigmoida nebo hyperbolická tangenta)**

Základní učení BPG

Modifikace vah ve směru negativního gradientu chybové funkce.

$$W_{k+1} = W_k - \alpha_k g_k$$

W_k je vektor vah a prahů (v daném časovém okamžiku – *current vector*),

g_k je gradient a α_k je rychlost učení

Učení je velmi pomalé.

1) **Vytvoření trénovací a testovací množiny vzorů.**
2) **Vytvoření množiny požadovaných (cílových) hodnot.**
   a) *Je-li UNS použita jako klasifikátor, pak všechny hodnoty požadovaných výstupů se blíží nule, vyjma hodnoty výstupu odpovídajícího třídě příslušného vstupu (hodnota výstupu je 1).*
   b) *Je-li UNS použita pro aproximaci funkcí, mohou být požadované hodnoty vyjádřeny analogovou hodnotou.*
3) **Volba topologie sítě.**
4) **Inicializace vah a prahů v jednotlivých vrstvách.**
   Může být explicitní (méně časté) nebo malými náhodnými čísly (vhodné zejména při malé znalosti problematiky).
5) **Trénování: výpočet výstupní hodnoty, adaptace vah a prahů.**

Pro každou uspořádanou dvojici (X_k, Y_k), $k = 1,2,\ldots,n$ platí, že

a) **hodnoty vzorů X_k vstupují do neuronů vrstvy L_1 a aktivují hodnotami vah a prahů výstupy z vrstvy w_{1ih} matice W_1. Spočítají se výstupy ze skryté vrstvy L_2**

$$h_i = \sigma\left(\sum_{h=1}^{n} x_h \; w_{ih}^1 + \Theta_i^1\right) \qquad i = 1,2,\ldots,S_1$$

h_i je aktivní hodnota i-tého neuronu ve vrstvě L_1,
Θ_i je i-tá prahová hodnota v i-tém neuronu ve vrstvě L_1,

$\sigma(u) = \left(1 + e^{-u/T}\right)^{-1}$ **je sigmoida,**
***1/T* je strmost funkce**

b) aktivní výstupy z vrstvy L_2, se filtrují prostřednictvím matice vah W_2 výstupní vrstvy L_3 podle vztahu

$$y_j = \sigma\left(\sum_{i=1}^{p} h_i \, w_{ji} + \Theta_j^2\right) \qquad j = 1,2,\ldots,S_2$$

c) chyba na výstupu mezi aktuální a požadovanou hodnotou se spočítá podle vztahu

$$E_j = y_j\left(1 - y_j\right)\left(t_j^k - y_j\right) \quad j = 1,2,\ldots,S_2$$

d) vypočítá se chyba každého neuronu ve skryté vrstvě z každé chyby ve výstupní vrstvě

$$e_i = h_i \left(1 - h_i\right) \sum_{j=1}^{S_č} w_{ji} E_j , \qquad i = 1,2,\ldots,S_1$$

e_i je i-tá chyba ve skryté vrstvě

e) adaptace vah mezi skrytou vrstvou L_2 a výstupní vrstvou L_3

$$\Delta w_{ji}^2 = \alpha\, h_i\, E_j \qquad i = 1,2,\ldots,S_1, \qquad j = 1,2,\ldots,S_2$$

Δw_{2ji} udává změnu vah mezi skrytou vrstvou L_2 a výstupní vrstvou L_3, α je kladná konstanta, která řídí rychlost učení

f) nastavení prahů na výstupu

$$\Delta\Theta_j^2 = \alpha\, E_j \qquad j = 1,2,\ldots,S_2$$

Θ_{2j} je přírůstek ze změny prahů ve výstupní vrstvě

g) adaptace vah mezi vstupní vrstvou L_1 a skrytou vrstvou L_2

$$\Delta w_{iS1}^1 = \beta\, x_h\, e_i \qquad i = 1,2,\ldots,S_1$$

Δw_{1ih} je přírůstek ze změny vah mezi vstupní vrstvou L_1 a skrytou vrstvou L_2, β je kladná konstanta řídící rychlost učení

h) nastavení prahů ve skryté vrstvě

$$\Delta \Theta_i^1 = \beta\, e_i \qquad i = 1,2,\ldots,S_1$$

6) Opakujeme předcházející krok tak dlouho, dokud není splněna podmínka pro dovolenou chybu.

Mírou naučenosti sítě je velikost globální chyby neboli hodnota energie

$$E_g = 0.5 \sum_{j=1}^{m} \left(y_j - t_j \right)^2$$

m **je počet výstupů ze sítě,** y_j **je j-tý výstup ze sítě,** t_j **je j-tý požadovaný výstup (cílová hodnota)**

Testování:

vypočet výstupních hodnot (v pořadí: skrytá vrstva, výstupní vrstva) pomocí matic vah W_1, W_2

Problémy:

- **nevhodná volba rychlosti učení (α, β) - může dojít k „přeskočení“ malých lokálních minim a k oscilacím**

- **není vždy možné dosáhnout lepšího průběhu chybové funkce náhodným výběrem pořadí vstupních trénovacích vektorů (např. při zpracování řečových signálů pořadí vstupních vektorů udává časovou posloupnost, která musí být dodržena)**

Řešení: **modifikace základního algoritmu**

Jak nastavit váhy a prahy NN ?

Pro NN, která dosud nebyla učena, je vhodné použít pro inicializaci funkce typu sigmoida malá náhodná čísla, aby derivace funkce nenabývala malých hodnot (při velkých hodnotách parametrů dochází k saturaci a síť se přestává adaptovat).

Naopak

- velké hodnoty derivací jsou vhodné pro aktivační funkce Gaussova typu.

Pro aktivační funkce typu hyperbolická tangenta se doporučuje (Nguyen a Widrow)

$$-\beta \leq w_{ij} \leq \beta, \qquad \beta = 0.7\,(p)^{1/n}$$

n … počet neuronů ve vstupní vrstvě,
p … počet neuronů ve skryté vrstvě

Jak volit počet učebních vzorů ?

viz sscC2_05

Lze odhadnout přibližně vztah mezi počtem vzorů N_P, počtem vah N_w a požadovanou přesností e:

$$e = \frac{N_w}{N_P}$$

Příklad: e = 0.1 , počet vah 80 , počet vzorů 800

Varianty rychlého učení

- **moment** - **snižuje citlivost učení na detaily chybové funkce**
  pomáhá vyhnout se uvíznutí v lokálním minimu
  dovoluje síti odpovídat na lokální gradient
  dovoluje sledovat trend chybové křivky

realizace momentu: přídavný člen v rovnici BPG

změna vah a prahů závisí na sumě podílu poslední a nové změny

mc
- **0 … změna vah je založena výhradně na gradientu**
- **1 … nová změna vah je rovna poslední změně vah, gradient je zcela ignorován**
- **0.95 … obvyklá hodnota**

- **doba trénování** - **sníží se pomocí adaptivního koeficientu učení**
  pomáhá udržet stabilitu učení

A) BPG s momentem

delta *input*

$$\Delta W(i,j) = mc \cdot \Delta W(i,j) + (1-mc)\, lr\, d(i)\, p(j)$$

moment vyvede parametry sítě z „hlubokého údolí“ v chybové křivce
k vyřazení vah dojde při poměru
nové chyby / staré chyby > maximální hodnota poměru chyb

často 1.04

Moment dovoluje sledovat tendence v chybovém prostoru

B) Rychlé BPG

- **základ - stejný jako u klasického BPG učení**
- **nová chyba > stará chyba (*více než dovoluje předepsaný poměr - obvykle 1.04*)**
- **nové váhy a prahy, výstup a chyba jsou zavrženy**
- **koeficient učení je snížen (*obvykle vynásobením 0.7*)**
- **výpočet nových vah, prahů, výstupu a chyby**
- **nová chyba < stará chyba**
- **koeficient učení je zvýšen (*obvykle vynásobením 1.05*)**
- **vždy pracují v dávkovém režimu (batch)**
- **10 až 100 krát rychlejší, než klasické BP**

- **heuristické metody**
  - **variabilní rychlost učení *lr***
  - **resilient BP *(„pružné“)***
- **standardní numerické optimalizační metody**
  - **Levenberg-Marquardt**
  - **conjugate gradient, quasi-Newton**

C) Levenberg-Marquardtova aproximace

$$\Delta W = (J^T J + \mu T)^{-1} J^T e$$

pro malé UNS (asi stovka vah) a velkou přesnost

J … *Jacobian derivací chyb v závislosti na vahách*
μ … *skalární hodnota* ***nevhodné pro rozpoznání***
e … *vektor chyb* ***vhodné pro aproximaci funkcí***

[HAM94] Hagan,M.T., Menhaj,M.: Training Feedforward Networks with the Marquardt algorithm. *In.: IEEE Trans.on Neural Networks*, vol.5, no.6, 1994.

Algoritmus s adaptivní rychlostí učení

během učení se mění rychlost učení *lr*

nová chyba > stará chyba o více než zvolený poměr

(max_perf_inc…v MATLABU)

→ váhy a prahy nejsou uvažovány

→ rychlost učení je snížena

(vynásobením lr_dec = 0.7)

nová chyba < stará chyba

→ rychlost učení je zvýšena

(vynásobením lr_inc = 1.05)

Lze kombinovat s momentovým učením.

Pružné učení *(Resilient Backpropagation)*

Vícevrstvé NS užívají často sigmoidy (aktivační funkce) ve skrytých vrstvách.

"squashing" funkce

- ***stlačí nekonečný rozsah vstupních dat do konečného rozsahu výstupních dat***
- ***sklon se blíží 0 pro velké vstupní hodnoty***

způsobuje malé změny vah a prahů (i když nejsou optimální)

- **Účelem pružného učení je odstranit tyto účinky hodnot parciálních derivací.**
- **Pouze znamínko derivace se užívá pro určení směru ukládaných vah.**
- **Velikost derivace nemá žádný vliv na uložení vah.**
- **Je mnohem rychlejší než standardní algoritmus.**
- **Vyžaduje jen malé zvýšení požadavků na paměť.**

Sdružené gradientní učení
(Conjugate Gradient Algorithms)

Základní BPG algoritmus - nastavuje váhy ve směru negativního gradientu.

Sdružený gradient - hledá směr s obecně rychlejší konvergencí.

Existuje několik variant.
Pro různé aplikace je třeba volit různé varianty.

Quasi-Newton Algorithms

- **Newtonova metoda je alternativou k metodě sdruženého gradientu.**
- **Je založena na výpočtu druhých derivací (Hessianovy matice).**
- **Vyznačuje se rychlejší optimalizací, ale obtížným výpočtem pro dopředné sítě.**
- **Quasi-Newtonovy metody jsou založeny na aproximaci Hessianovy matice v každé iteraci.**
- **Nové uložení vah je počítáno jako funkce gradientu.**

Tento algoritmus vyžaduje více výpočtů.
Aproximace Hessianu musí být uloženy, dimenze Hessianu je n x n (n se rovná počtu vah a prahů).
Pro velké sítě, může být lepší použít pružné učení (resilient) nebo jednu z variant sdruženého gradientu.
Pro menší sítě je možné použít jednu z variant Quasi-Newtonovy Metody.

Porovnání metod rychlého učení

LM - nejrychlejší konvergence pro aproximace funkcí
- pro velmi přesné trénování
- jen pro malé sítě
- nehodí se pro rozpoznání
- potřeba velké paměti

(redukce paměti možná za cenu zvýšení doby trénování)

resilient BP - nejrychlejší algoritmus pro rozpoznávací problémy
- není vhodný pro aproximace funkcí
- požadavky na paměť jsou relativně malé v porovnání s dalšími algoritmy

sdružené gradientní algoritmy - jsou vhodné pro nejrůznější problémy (zvláště pro sítě s velkým množstvím vah)
- jsou téměř tak rychlé jako LM pro aproximaci funkcí (rychlejší pro velké články)
- pro rozpoznání jsou téměř tak rychlé jako resilient učení
- mají relativně skromné požadavky na paměť

algoritmus s adaptivním *lr* - je obvykle mnohem pomalejší než další metody (může to být někdy užitěčné - někdy je vhodná pomalá konvergence při použití "early stopping")

Závěr

- **všechny varianty algoritmu BPG se užívají pro trénování vícevrstvé neuronové sítě s diferencovatelnými aktivačními funkcemi**
- **aplikace - aproximace funkcí**
  přiřazení vzorů
  klasifikace vzorů
  modelování některých parametrů signálů
- **počet vstupů do UNS závisí na řešeném problému**
- **počet výstupů z UNS je dán počtem požadovaných hodnot**
- **počet skrytých neuronů (resp. vrstev) záleží na návrháři**
- **80% - 90 % praktických aplikací užívá některou variantu učení BPG**
- **je obtížné určit, který algoritmus bude nejrychlejší pro daný problém**

Dřívější ukončení trénování (*early stopping*):

pro různé algoritmy učení
ne pro LM

trénovací data
validační data
testovací data

Porovnání algoritmů

- **hodnocení jednotlivých algoritmů je obtížné**
- **doba učení a úspěšná konvergence závisí na počtu vah a prahů**