Neurovõrgud. Praktikum 3.

25. veebruar 2005. a.

1 Adaline

Eelmine kord klassifitseerisime tähti pertseptroni abil. Seda saab teha ka adaptiivse neuroni algoritmiga (Adaline).
Traditsiooniline Adaline on lineaarse aktivatsioonifunktsiooniga neuroni ($y = \mathbf{w}^T\mathbf{x} + b$) treenimisalgoritm, mis minimiseerib stohhastilise kiirema languse meetodiga funktsiooni

$$\mathcal{E}(\mathbf{w}, b) = \sum_i e^2(i)$$

kus $e(i) = \|d(i) - y(i)\|$ ning $y(i) = \mathbf{w}^T\mathbf{x}(i) + b$ (kuna täna raagime natuke ka mitmekihilistest võrkudest, siis eristame treeningnäiteid indeksitega sulgudes, mitte alaindeksiga nagu ennem).
Stohhastilise kiirema languse meetodi samm selle funktsiooni minimiseerimiseks on:

$$\Delta\mathbf{w}_k = \eta e(i)\mathbf{x}(i) \qquad \Delta b = \eta e(i)$$

kus $\mathbf{x}(i) \to d(i)$ on antud sammul juhuslikult valitud treeningnäide ning η on väike arv mida nimetatakse *õppimisteguriks*.
Selleks et kasutada neuroni klassifitseerimiseks, peame valima tema aktivatsioonifunktsiooniks ϕ midagi, mis oleks tõkestatud 1 ja -1 vahel. Siis avaldub neuroni poolt realiseeritav funktsioon kujul $y = \phi(\mathbf{w}^T\mathbf{x} + b)$.
Tihti valitakse ϕ rolli hüperboolset tangensit:

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

Olgu nüüd andmetes soovitud väärtused d alati arvud -1 ja 1. Lineaarse klassifitseerija treenimise jaoks, nagu ikka, otsime $\mathbf{w}$ ja b sellised, mis minimiseeriksid funktsiooni

$$\mathcal{E}(\mathbf{w}, b) = \sum_i e^2(i)$$

kus $e(i) = \|d(i) - y(i)\|$, ning seekord $y(i) = \phi(\mathbf{w}^T\mathbf{x}(i) + b)$. Saab näidata, et sel juhul on stohhastilise kiirema languse algoritmi samm järgmine:

$$\Delta\mathbf{w} = \eta\phi'(\mathbf{w}^T\mathbf{x}(i) + b)e(i)\mathbf{x}(i) \qquad \Delta b = \eta\phi'(\mathbf{w}^T\mathbf{x}(i) + b)e(i)$$

ning juhul kui ϕ rollis on tanh

$$\Delta\mathbf{w} = \eta(1 - y^2(i))e(i)\mathbf{x}(i) \qquad \Delta b = \eta(1 - y^2(i))e(i)$$

Ülesanne 1 (1p): Tõesta seda.

Ülesanne 2 (1p): Realiseeri hüperboolse tangensiga adaline algoritmi funktsioonina
`function [w, b] = adaline(X, d, eta, num_iterations)`. Sisenditeks on treeningandmed, õppimistegur η, ning iteratsioonide arv. Väljundiks — leitud $\mathbf{w}$ ja b. Rakenda seda algoritmi eelmises praktikumis vaadeldud tähtede piltide klassifitseerimiseks. S.t. lae sisendandmed failidest `digits.data` ning `digits.class`, jaga neid juhuslikult treening- ja test hulkadeks (100/31), treeni adaptiivset neuronit treeninghulga andmete peal ning hinda täpsust testhulga andmete peal. Treenimisel võta $\eta = 0.05$ ning iteratsioonide arvuks 1000. Uuri kuidas sõltub täpsus iteratsioonide arvust. Milline on adaline algoritmi eelis pertseptroni ees? Esita kood.

Ülesanne 3 (1p): Treenitud neuroni kaalude vektor $\mathbf{w}$ on 28×28 komponendiline, ning samuti nagu treeningandmeid saab teda vaadata pildina:

```
c = [0:0.1:1]';
xset("colormap", [c c c]);
img = matrix(w, 28, 28);
grayplot(1:28, 1:28, img);
```

Seleta, miks see pilt just selline välja tuli.

2 Mitmekihiline pertseptron

Mitmekihiline pertseptron koosneb mitmetest pertseptronide kihtidest. Tähistame kihte kui $L_0, L_1, L_2, \ldots, L_k$. Kiht L_0 on *sisendkiht*, $L_1, \ldots, L_{k-1}$ on *peidetud kihid* ning L_k on *väljundkiht*. Kihi L_s igast neuronist lähevad järgmisse kihti sünaptilised ühendused, ning $w_{ji}^{(s+1)}$ tähistagu kihi L_s i-nda neuroni ja kihi L_{s+1} j-nda neuroni vahelise ühenduse kaalu. Iga kihiga L_s välja arvatud sisendkiht saab siis seostada *kaalude maatriksit* $\mathbf{W}^{(s)} = (w_{ij}^{(s)})$. Võrgu väärtust treeningnäite $\mathbf{x}$ puhul arvutatakse kiht kihilt:

- Alguses arvutatakse sisendkihi neuronite väljundid: $y_i^{(0)} = x_i$
- Edasi arvutatakse esimese peidetud kihi neuronite lokaalsed väljad: $v_j^{(1)} = \sum_i w_{ji}^{(1)} y_i^{(0)}$
- Lokaalsetest väljadest arvutatakse selle kihi iga neuroni väärtused: $y_j^{(1)} = \phi(v_j^{(1)})$.

- Viimased kaks sammu korratakse iga kihi jaoks, ning viimase kihi väljund ongi võrgu väljund.

Võrgu treenimine seisneb, nagu alati, vigade ruutude summa minimiseerimises ning klassikaline algoritm selleks on *vea tagastamise algoritm*, mis on sisuliselt kiirema languse meetod. Vea tagastamise algoritmis näidatakse võrgule treeningnäiteid järjest, ning iga näitega tehakse kaks võrgu läbimist — edaspidine ja äraspidine. Edaspidise läbimise käigus arvutatakse eelnevalt kirjeldatud viisil iga neuroni lokaalse välja ja väljundi, äraspidises läbimises aga arvutatakse iga neuroni jaoks tema *lokaalse gradiendi* $\delta_j^{(s)}$, kusjuures

- Väljundkihi L_k neuroni j lokaalne gradient on $\delta_j^{(k)} = \phi'(v_j^{(k)})e_j$ kus $e_j = d_j - y_j^{(k)}$ on vastava neuroni viga.
- Peidetud kihi L_s lokaalne gradient avaldub järgmise kihi neuronite lokaalsete gradientide kaudu:

$$\delta_i^{(s)} = \phi'(v_i^{(s)}) \sum_j w_{ji}^{(s+1)} \delta_j^{(s+1)}$$

Pärast seda kui edaspidine ja äraspidine läbimine on tehtud, kohendatakse kaalud:

$$\Delta w_{ji}^{(s+1)} = \eta \delta_j^{(s+1)} y_i^{(s)}$$

Ülesanne 4 (1p): Mõtle seda protseduuri korralikult läbi. Tähistagu $v^{(s)}$ kihi L_s kõigi neuronite aktivatsioonide vektorit, $\delta^{(s)}$ — kihi L_s kõikide neuronite lokaalsete gradientide vektorit, jne.

- Pane kirja maatrikskujul $v^{(s+1)}$ arvutamisreegli $y^{(s)}$ põhjal.
- Pane kirja analoogselt maatrikskujul $\delta^{(s)}$ arvutamisreegli $\delta^{(s+1)}$ põhjal. (siin tuleb kasutada ka vektorite korrutist elementidekaupa).
- Kuhu kadusid *bias* parameetrid?

Ülesanne 5 (1p): Mõnedes algoritmides läheb vaja võrgu veafunktsiooni teist tuletist, ehk hessiaani. Olgu meil kolme kihiga pertseptron, millel on 1000 neuronit sisendkihis, 100 neuronit peidetud kihis ning üks neuron väljundkihis. Kui suur on sellise võrgu hessiaan, s.t. kui palju elemente on selles maatriksis?

Puhtalt Scilabi vahenditega on mitmekihilise pertseptroni treenimist realiseerida ja katsetada väga ebamugav. Selle pärast teeme siin läbi ainult minimaalse näite. Failis `mlp.sce` on tehtud mitmekihilise pertseptroniga regressioonijoone otsimine eelmise praktikumi *Dow Jones* andmetele. Uuri seda koodi.

Ülesanne 6 (1p): Konstrueeri regressioonijoont võrguga milles on 1 neuroniga sisendkiht, 10 neuroniga peidetud kiht ning 1 neuroniga väljundkiht (1-10-1 võrk). Proovi seda teha ka 1-10-10-10-1 võrguga. Täienda koodi nii, et ta väljastaks treeningvea graafiku (*õppimiskõvera*, treeninguvea sõltuvust epohhist). Iga nimetatud võrgu puhul tekita vastava õppimiskõvera. Millest sõltub õppimiskõvera siledus antud juhul ja miks? Treeni nüüd 1-100-1 võrgu. See peaks minema suhteliselt halvasti — algoritm kas ei koondu või õppimiskõver on suurte kõikumistega. Kuidas seda parandada? Mõtle nüüd ka sellest, miks andmeid tuli eelnevalt skaleerida. Esita saadud pildid (regressioonijooned ja õppimiskõverad) ning oma mõted.