Neurovõrgud. Praktikum 11.

29. aprill 2005. a.

1 Stohhastilised võrgud

Selles praktikumis vaatleme põhilisi stohhastilisi võrke ning nende rakendust kombinatoorsete optimiseerimisülesannete lahendamiseks. Stohhastiline võrk on süsteem mis koosneb n sõlmest. Iga sõlm saab olla seisundis $+1$ või -1, ning sõlme i seisundit s_i nimetame tema *spinniks*. Igal hetkel on seega terve süsteemi seisund kirjeldatav n-elemendilise spinnide vektoriga σ:

$$\sigma = (s_1, s_2, \ldots, s_n), \qquad s_i \in \{+1, -1\}$$

Iga kahe sõlme i ja j vahel on olemas teatud *vastasmõju* w_{ij}. Mida suurem on vastasmõju, seda rohkem ,,tahavad" sõlmed omama samasugust spinni või erisugust spinni sõltuvalt w_{ij} märgist. Süsteemi *siseenergia* on defineeritud valemiga

$$H(\sigma) = -\sum_{i,j} w_{ij} s_i s_j - h_{\text{ext}} \sum_i s_i, \tag{1}$$

kus w_{ij} on *vastasmõju kordajad* ning h_{ext} on *väline väli*. Funktsiooni H nimetatakse tihti süsteemi *hamiltoni funktsiooniks* või *hamiltoniaaniks*. Tavaliselt eeldatakse et $w_{ij} = w_{ji}$ ning $w_{ii} = 0$. Viimane eeldus ei ole kitsendav: tema rakendamine muudab süsteemi energiat ainult konstandi võrra (miks?).
Süsteem ise on dünaamiline ning üritab teatud seisundimuutuste kaudu saavutada energiafunktsiooni miinimumi. Sellise mudeliga saab füüsikas kirjeldada magnetismi, samas osutub et ka paljud rasked kombinatoorsed ülesanded on taandatavad sobiva hamiltoni funktsiooni minimiseerimisele.

2 NP-täieliku ülesande lahendamine

Vaatleme järgmist NP-täieliku ülesannet:

Partitioning: Jagada kivid kaaludega k_1, k_2, …, k_{2n} kahte hulka nii, et hulkade kogukaal oleks võrdne. Kui see pole võimalik, siis minimiseerida kaalude vahet.

Selle lahendamiseks vaatleme $2n$ sõlmedest koosneva võrku. Sõlme i spin s_i olgu $+1$ kui kivi i kuulub esimesse hulka, ning -1 vastasel juhul. Algse ülesande lahendamiseks peame me seega leidma sellist spinnide konfiguratsiooni σ, mis minimiseeriks funktsiooni:

$$H(\sigma) = \left(\sum_{i=1}^{2n} s_i k_i \right)^2$$

Selleks et viia süsteemi kujule (1) avame sulud ning saame et $w_{ij} = -k_i k_j$. Nagu ennem tähele pandud, võime panna $w_{ii} = 0$. See muudab energiat ainult konstandi võrra, ning seda on vaja edaspidises algoritmide toimimiseks. Nüüd jääb meil ainult realiseerida vastava võrgu seisundimuutmiste dünaamikat, mis viiks teda energiamiinimumi. Selleks on mitu võimalust.

2.1 Hopfieldi võrk

Hopfieldi võrgu puhul on seisundi muutmise reegel järgmine: valime igal sammul ühe sõlme k (juhuslikult või järjest), ning muudame selle spinni s_k. Uueks spinniks s'_k valime:

$$s'_k = \operatorname{sign}(h_k), \qquad \text{kus} \quad h_k = \sum_j w_{kj} s_j$$

Osutub et sellised sammud viivad võrgu energiamiinimumi.

Ülesanne 1 (1p): Näita et kui $w_{ij} = w_{ji}$, $w_{ii} = 0$, siis $H(\sigma') \leq H(\sigma)$, s.t. iga sammuga saab energia ainult kahaneda.

Seega on Hopfieldi võrgu algoritm lihtsalt ahne miinimumi otsing (*local search, hill climbing*) ning võib koonduda lokaalsesse miinimumi.

Ülesanne 2 (1p): Lahenda ülesannet **Partitioning** Hopfieldi võrgu abil. Olgu kivide arv 100, ning nende kaalud $(1, 2, 3, \ldots, 100)$ (s.t. $k_i = i$). Väljasta igal iteratsioonisammul võrgu energiat ja veendu, et see kahaneb. Jooksuta algoritmi mitu korda. Kas iga kord leitakse parim lahendus? Esita kood.
Vihjed:

1. Realiseeri funktsioon `function s = hopfield(W, numiterations)`
2. ```
   k = [1:100]';  // Algandmed
   W = -k*k';     // Vorgu kaalude maatriks
   for i = 1:100, W(i,i) = 0;, end; // Diagonaalis nullid
   s = hopfield(W, 400); // Simulerime vorku
   printf("Kuhjade kaalude erinevus: %f\n", k'*s);
   ```
3. Scilabi funktsioon `sign` võib tagastada nulli. Me ei tohi aga spinni väärtuseks nulli panna.
   ```

Selleks et algoritm ei peatuks lokaalses miinimumis, tuleb teha mõnikord ka energiat suurendavaid samme. See idee toob meid Boltzmanni võrguni.

2.2 Boltzmanni võrk

Kui Hopfieldi võrk vastab *hill-climbing*-ule, siis Boltzmanni võrk on sisuliselt libalõõmutamise algoritmi implementatsioon. Nagu Hopfieldi võrgu puhul kohendatakse igal sammul ühe sõlme spinni, kuid seekord on reegel tõenäosuslik. Tõenäosus et valitud sõlme k spin on 1 avaldub valemina:

$$P(s_k = +1) = \frac{1}{1 + \exp\left(-\frac{h_k}{T}\right)}$$

kus T on *temperatuur*. Mida suurem on temperatuur, seda suurem on tõenäosus et algoritm teeb energiafunktsiooni suurendavaid samme. Võrgu treenimise käigus alustatakse kõrge temperatuuri väärtustest, ning pärast tasapisi „jahutatakse". Temperatuuri on kasulik alandada faasides nii, et iga võrgu sõlme spinni uuendatakse vähemalt üks kord enne, kui võetakse uus T väärtus.

Ülesanne 3 (1p): Lahenda ülesannet Boltzmanni võrguga. Olgu esimeste 20 iteratsiooni käigus temperatuur fikseeritud 1000 peal, ning iga järgmises faasis korruta seda arvuga $\alpha = 0.9$ (faasipikkuseks olgu ka 20 iteratsiooni). Väljasta peale iga spinni uuendamist süsteemi energia ja temperatuur, koosta vastav graafik. Tee vähemalt 10000 sõlme uuendamist[1]. Kui tihti koondub Boltzmanni võrk mitteoptimaalseks lahenduseks? Esita kood ning saadud tulemused.
Vihjed:

1. Realiseeri funktsiooni `function s = boltzmann(W, numiterations)`. Ta ainult natuke erineb funktsioonist `hopfield`.

2. ```
   ...
   h_k = W(k, :)*s;
   p_k = 1/(1 + exp(-h_k/T)); // Toenaosus et s(k) = 1
   s(k) = sign(p_k - rand());
   if s(k) == 0 then, s(k) = 1;, end;
   // Alanda temperatuuri parast igat faasi
   if modulo(iter, 20) == 0 then, T = 0.9*T;, end;
   ```

[1] Reaalses elus peaks iteratsioonide arv olema veel suurem, ning faasid peavad olema pikem. Hea on kui igas faasis saab iga sõlm uuendatud mitu korda.
   ```

2.3 Metropolise reegel

Veel üks stohhastiline optimiseerimisalgoritm on *Metropolise reegel*. Selles algoritmis lihtsalt muudetakse igal sammul ühe juhuslikult või süstemaatiliselt valitud sõlme seisu. Siis vaadeldakse uue seisundi σ' ja vana seisundi σ energiate vahet:

$$\Delta H = H(\sigma') - H(\sigma)$$

kui $\Delta H < 0$, aktsepteeritakse uus seisund. Muidu aktsepteeritakse ta tõenäosusega $\exp(-\frac{\Delta H}{T})$.

Ülesanne 4 (1p): Lihtsusta ΔH avaldist nii, et see sisaldaks vaid suurusi h_k ja s_k. Võrdle Boltzmanni ja Metropolise reeglit.

Ülesanne 5 (1p): Programmeeri Metropolise reegel ning lahenda selle abil ülesannet **Partitioning**. *Cooling schedule* (ehk temperatuuri käitumine) olgu samasugune nagu eelmises ülesandes. Jooksuta 10000 võrgu iteratsiooni. Milline kolmest algoritmidest (Hopfield, Boltzmann, Metropolis) on parem?

Ülesanne 6 (2p): Vaatleme veel ühe ülesande lahendamist stohhastiliste võrkudega. Olgu meil antud graaf $\mathcal{G}$. Ülesandeks on jagada graafi tipud kahte hulka niimoodi, et hulkade vaheliste servade arv oleks minimaalne ning hulgad oleksid võimalikult võrdse suurusega.

- Sõnasta energiafunktsioon, mis jagab sõlmed kahte hulka nii, et vahepealsete servade arv oleks minimaalne. Olgu see funktsioon H_1.
- Sõnasta energiafunktsioon, mis jagab sõlmed kahte hulka nii, et need hulgad oleksid võimalikult võrdsed. Olgu see funktsioon H_2.
- Kokku kasutame minimiseerimiseks funktsiooni $H(\sigma) = H_1(\sigma) + \lambda H_2(\sigma)$, kus λ kontrollib kui oluline on saada hulgad võrdseteks. Vii seda funktsiooni kujule (1), s.t. leia võrgu kaalude maatriks.

Esita leitud kaalude maatriksit.

3 Assotsiatiivne mälu

Üks Hopfieldi võrgu rakendus on nn. *assotsiatiivne mälu*. Olgu $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$ mingid vektorid mida me tahame salvestada. Selleks võib konstrueerida Hopfieldi võrgu, mille kaalumaatriksiks on

$$\mathbf{W} = \frac{1}{k} \sum_{i=1}^{k} \mathbf{x}_i \mathbf{x}_i^T - \mathbf{I}.$$

On võimalik veenduda, et selle võrgu lokaalseteks miinimumideks (stabiilseteks seisunditeks) on suure tõenäosusega vektorid $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$. Seega kui sellise võrgu algseisundiks on midagi lähedast $\mathbf{x}_i$-le, siis suure tõenäosusega koondub võrk just seisundisse $\mathbf{x}_i$. See annab võimalust kasutada võrgu assotsiatiivmäluna. Loomulikult saab nii võrku talletada vaid lõpliku hulga mustreid. Selle piiri ületamisel tekivad esmalt lisamustrid ning seejärel hävivad ka kõik algsed mustrid. See efekt on tuntud üleõppimise nime all.

Ülesanne 7 (1p): Failis `pictures.data` on salvestatud neli 10×10 „pilti". Ülesandeks on salvestada pildid 1 ja 2 Hopfieldi võrku, ning uurida kuidas piltidest 3 ja 4 suudab võrk „meelde tuletada" salvestatud pilte. Salvestamiseks kasuta eeltoodud valemit. Meeldetuletamiseks jooksuta 1000 võrgu iteratsiooni, andes algseisuna pildi 3 või 4.
Vihjed:

- Muuda funktsiooni `hopfield` nii, et ta võtaks parameetrina algseisu.
- Pilte saab failist sisse lugeda ja vaadata järgmiselt:

```
X = read("pictures.data", -1, 100);
pic = X(1,:)'; // Votame esimest pilti
Matplot(matrix(pic + 1, 10, 10)); // Vaatame ekraanil
```

Ülesanne 8 (1p): Kas assotsatiivmälu jaoks saaks põhimõtteliselt kasutada ka Boltzmanni või Metropolise võrke? Milline peaks olema temperatuur? Mida võiks sel juhul tähendada lokaalsest miinimumist väljumine?