VYTAUTO DIDŽIOJO UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS INSTITUTAS

Sergėjus IVANIKOVAS

LYGIAGREČIŲ SKAIČIAVIMŲ TAIKYMO DAUGIAMAČIAMS DUOMENIMS VIZUALIZUOTI PROBLEMOS

Daktaro disertacija

Fiziniai mokslai (P 000)
Informatika (09 P)
Informatika, sistemų teorija (P 175)

Vilnius, 2009

Disertacija rengta 2005–2009 metais Matematikos ir informatikos institute.

Darbo mokslinis vadovas

prof. habil. dr. Gintautas Dzemyda (Matematikos ir informatikos institutas, fiziniai mokslai, informatika – 09 P).

Reziumė

Disertacijos tyrimų sritis yra daugiamačių duomenų vizuali analizė, vizualizavimo algoritmų tyrimas bei tų algoritmų lygiagrečiųjų versijų kūrimas. Duomenų suvokimas yra sudėtingas uždavinys, ypač kai duomenys atspindi sudėtingą objektą, kuris aprašytas daugeliu parametrų. Disertacijoje nagrinėjami dirbtiniais neuroniniais tinklais grindžiami algoritmai daugiamačiams duomenims vizualizuoti. Pagrindinis darbo tyrimų objektas yra dirbtiniai neuroniniai tinklai, skirti daugiamačiams duomenims vizualizuoti. Su šiuo objektu yra betarpiškai susiję daugiamačių duomenų vizualizavimas, dimensijos mažinimo algoritmai, projekcijos paklaidos, naujų taškų atvaizdavimas, vizualizavimui skirto neuroninio tinklo mokymo strategijos, lygiagretieji skaičiavimai. Pagrindinis disertacijos tikslas yra sukurti ir tobulinti metodus, kuriuos taikant būtų efektyviai minimizuojamos daugiamačių duomenų projekcijos paklaidos bei pagreitinamas dirbtinio neuroninio tinklo mokymas.

Disertaciją sudaro šeši skyriai ir literatūros sąrašas. Bendra disertacijos apimtis 104 puslapiai.

Tyrimų rezultatai publikuoti 6 moksliniuose leidiniuose (trys iš jų periodiniuose): 1 straipsnis leidinyje, įtrauktame į Mokslinės informacijos instituto pagrindinį (ISI Web of Science) sąrašą, 5 straipsniai leidiniuose, įtrauktuose į Mokslinės informacijos instituto konferencijos darbų (ISI Proceedings) sąrašą. Tyrimų rezultatai buvo pristatyti ir aptarti 4-iose tarptautinėse ir vienoje respublikinėje konferencijoje.

Abstract

The research area of this work is the analysis of multidimensional data, the investigation of visualization algorithms and the development of parallel realizations of these algorithms. Data apprehension is rather a complicated problem especially if the data refer to a complex object or phenomenon described by many parameters. The research object of the dissertation is artificial neural networks for multidimensional data projection. General topics that are related with this object: multidimensional data visualization; dimensionality reduction algorithms; data projection errors; the projection of the new data; strategies for training the neural network that visualizes multidimensional data; parallel computing. The key aim of the work is to create and develop methods to minimize the visualization errors of multidimensional data projection by using artificial neural networks efficiently.

The work is written in Lithuanian. It consists of 6 chapters and the list of references. There are 104 pages of the text, 49 figures, 7 tables and 98 bibliographical sources.

The main results of this dissertation were published in 6 scientific publications: 1 article in the periodical scientific issue from the ISI Web of Science list; 2 articles in the periodical scientific issues from the ISI Proceedings list; 1 chapter in the rewiewed book (Springer) and 2 articles in the proceedings of scientific conferences included into the ISI Proceedings list.

The main results of the work have been presented and discussed at 4 international and 1 national conferences.

Padėka

Nuoširdžiai dėkoju moksliniam vadovui prof. habil. dr. Gintautui Dzemydai už vertingas mokslines konsultacijas, nuoseklų vadovavimą, pagalbą ir kantrybę rengiant šią disertaciją.

Ačiū disertacijos recenzentams prof. Kaziui Kazlauskui ir dr. Olgai Kurasovai, atidžiai perskaičiusiems disertaciją ir pateikusiems vertingų patarimų bei kritinių pastabų.

Dėkoju Matematikos ir informatikos instituto Sistemų analizės skyriaus darbuotojams ir kolegoms už naudingus patarimus ir draugišką pagalbą.

Nuoširdžiai dėkoju savo artimiesiems ir draugams už jų paramą, moralinį palaikymą, kantrybę ir supratingumą.

Dėkoju Lietuvos valstybiniam mokslo ir studijų fondui už suteiktą finansinę paramą disertacijos rengimo metu.

Taip pat dėkoju visiems kitiems, kurie tiesiogiai ar netiesiogiai prisidėjo prie šio darbo.

Sergėjus Ivanikovas

Turinys

1

Įvadas

1.1. Tyrimų sritis

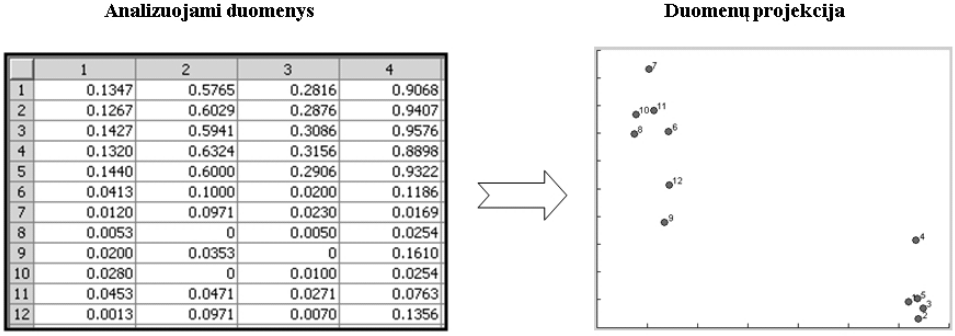
Šiuolaikinis gyvenimas priverčia mus apdoroti didžiulius duomenų kiekius, įvairiais būdais valdyti informaciją ir žinias. Duomenys – tai objektyviai egzistuojantys faktai, vaizdai arba garsai, kurie gali būti naudingi tam tikram uždaviniui spręsti. Informacija – tai duomenys, kurių forma ir turinys yra pateikti tinkamiausiu būdu priimti sprendimą. Duomenys virsta informacija, kai jiems suteikiamas kontekstas ir jie priskiriami tam tikrai problemai ar sprendimui. Žinios yra gebėjimas spręsti problemas, atnaujinti arba kitaip sukurti reikšmes remiantis ankstesne patirtimi, įgūdžiais ar išmokimu (Piatetsky-Shapiro, 1996).

Realus pasaulis pateikia daugybę duomenų, kuriuos tenka analizuoti, vertinti. Labai svarbu iš turimų duomenų gauti reikiamą informaciją, atskiriant menkaverčius faktus. Duomenų suvokimas yra gana sudėtingas uždavinys, ypač kai duomenys charakterizuoja sudėtingą objektą, reiškinį, kuris aprašytas daugeliu parametrų. Tokie duomenys vadinami *daugiamačiais duomenimis.* Šiuolaikinės technologijos leidžia pagreitinti daugiamačių duomenų apdorojimą naudojant didelio našumo skaičiavimo įrangą bei taikant lygiagrečiuosius skaičiavimus. Šio darbo tyrimų sritis yra daugiamačių duomenų analizė, vizualizavimo algoritmų tyrimas bei tų algoritmų lygiagrečiųjų versijų kūrimas.

1.2. Darbo aktualumas

Medicinoje, technikoje, ekonomikoje ir daugelyje kitų sričių nuolat susiduriama su daugiamačiais duomenimis. Sparčiai besivystančios informacinės technologijos ir tobulėjanti kompiuterinė technika leidžia apdoroti vis didesnius duomenų kiekius, tačiau žmogaus suvokimas nepasikeičia. Žmogui sunku analizuoti daugiamačius duomenis, jei jų dimensija didesnė už 3. Dimensija paprastai suprantama kaip parametrų, charakterizuojančių tiriamą objektą ar reiškinį, skaičius.

Šiuo metu duomenų analizė yra labai aktuali įvairiose veiklos srityse: moksle, versle, ekonomikoje, medicinoje, sociologijoje ir kt. Duomenų pateikimas žmogui suprantama forma dažnai būna efektyvi priemonė, norint geriau suvokti daugiamačius duomenis, t.y. nustatyti jų struktūrą, tarpusavio ryšius, susidariusias grupes ir pan. Vienas iš galimų būdų yra grafinis informacijos pateikimas – vizualizavimas. Pagrindinė vizualizavimo idėja – duomenis pateikti tokia forma, kuri leistų vartotojui suprasti duomenis, daryti išvadas ir tiesiogiai įtakoti tolesnį jų srautą. Vizualią informaciją žmogus pajėgus suvokti daug greičiau negu tekstinę.



	1	2	3	4
1	0.1347	0.5765	0.2816	0.9068
2	0.1267	0.6029	0.2876	0.9407
3	0.1427	0.5941	0.3086	0.9576
4	0.1320	0.6324	0.3156	0.8898
5	0.1440	0.6000	0.2906	0.9322
6	0.0413	0.1000	0.0200	0.1186
7	0.0120	0.0971	0.0230	0.0169
8	0.0053	0	0.0050	0.0254
9	0.0200	0.0353	0	0.1610
10	0.0280	0	0.0100	0.0254
11	0.0453	0.0471	0.0271	0.0763
12	0.0013	0.0971	0.0070	0.1356

1.1 pav. *Duomenų vizualizavimas*

Daugiamačių duomenų vizualizavimo metodai vystomi gana intensyviai siekiant lengvinti duomenų interpretavimą, efektyviai pateikti ir įvertinti duomenų gavybos rezultatus. Taip pat plačiai naudojami projekcijos metodai. Dažnai iškyla būtinybė nustatyti duomenų struktūrą: susidariusias grupes (klasterius), žymiai išsiskiriančius objektus (taškus-atsiskyrėlius), atstumus tarp objektų ir pan. Transformavus daugiamačius duomenis į dvimatę arba trimatę erdvę, darosi daug paprasčiau suvokti duomenų struktūrą ir sąryšius tarp jų. Pavyzdžiui, 1.1 paveiksle pavaizduota daugiamačių duomenų projekcija plokštumoje. Turint tokį daugiamačių duomenų vizualizavimą, jau galime daryti tam tikras išvadas apie analizuojamą duomenų aibės struktūrą, gauname papildomą informaciją apie duomenis. Paveiksle aiškiai matosi du nagrinėjamų duomenų klasteriai, daugiamačių duomenų tarpusavio išsidėstymas.

Tačiau duomenis transformuojant į mažesnės dimensijos erdvę, duomenų vizualizavimo iškraipymai, paklaidos yra neišvengiamos. Šių paklaidų minimizavimas

išlieka aktualia problema. Ši problema yra pagrindinė šioje disertacijoje sprendžiama **problema**.

Dažnai tenka dirbti su didelės apimties duomenų aibėmis, kurios pastoviai papildomos naujais duomenimis. Todėl naujų taškų atvaizdavimas, jų įterpimas tarp anksčiau atvaizduotų taškų – dar viena aktuali problema.

Vienas populiariausių metodų daugiamačiams duomenims vizualizuoti yra taip vadinamas daugiamačių skalių metodas (MDS – *angl. Multidimensional Scaling*). Pastaruoju metu stebima tendencija, kad MDS tyrimus vykdantys mokslininkai dažnai atsiriboja nuo kitų metodų tyrimų, ar net ignoruoja tuos metodus. Antra vertus, kitų vizualizavimo metodų tyrimuose nėra lyginimų ar sąsajų su MDS tipo metodais. Šiame darbe siekiama praplėsti MDS tipo metodų realizacijas dirbtinių neuroninių tinklų taikymu, tuo būdu stiprinant ryšį tarp skirtingų vizualios duomenų analizės krypčių.

Darbe nagrinėjami dirbtinių neuroninių tinklų algoritmai daugiamačiams duomenims vizualizuoti. Pavyzdžiui, Mao ir Jain (Mao, 1995) pasiūlyta specifinė „klaidos sklidimo atgal“ mokymo taisyklė, pavadinta SAMANN, kuri leidžia įprastam tiesioginio sklidimo neuroniniam tinklui realizuoti populiarią Sammono projekciją mokymo be mokytojo būdu.

1.3. Darbo tikslas ir uždaviniai

Pagrindinis disertacijos tikslas yra išvystyti neuroniniais tinklais grindžiamus daugiamačių duomenų vizualizavimo metodus užtikrinant efektyvų daugiamačių duomenų projekcijos paklaidos minimizavimą bei pagreitinant neuroninio tinklo mokymą.

Norint pasiekti šį tikslą, reikėjo išspręsti tokius uždavinius:

1) analitiškai apžvelgti daugiamačių duomenų vizualizavimo metodus,
2) ištirti galimybes paspartinti dirbtinių neuroninių tinklų mokymąsi atliekant daugiamačių duomenų vizualizavimą,
3) ištirti neurono aktyvacijos funkcijos parametrų įtaką neuroninio tinklo mokymuisi,
4) ištirti galimybes SAMANN tinklą mokyti analizuojamos duomenų aibės dalimi arba specialiai konstruojama sumažinta mokymo aibe,
5) išanalizuoti SAMANN algoritmo veikimo principus ir šio algoritmo lygiagretinimo galimybes,
6) ištirti technologines lygiagretinimo galimybes (klasteriai, SMP kompiuteriai, Hyper-Threading technologija),
7) sukurti ir ištirti SAMANN algoritmo lygiagrečiąją versiją.

Tyrimų **metodikos** pagrindą sudaro naujų SAMANN neuroninio tinklo mokymo strategijų kūrimas ir jų eksperimentinis tyrimas.

1.4. Tyrimo objektas

Analizuojant daugiamačius duomenis, klasikinių vizualizavimo metodų (tokių kaip pagrindinių komponenčių analizė, daugiamačių skalių metodas) dažnai nepakanka norint atskleisti duomenų struktūrą. Disertacijos tyrimų objektas – dirbtiniais neuroniniais tinklais grindžiami daugiamačių duomenų vizualizavimo algoritmai. Su šiuo objektu betarpiškai susiję dalykai:

1) daugiamačių duomenų vizualizavimas,
2) dimensijos mažinimo (projekcijos) algoritmai,
3) tiesioginio sklidimo dirbtiniai neuroniniai tinklai,
4) saviorganizuojantys neuroniniai tinklai,
5) klasterizavimo algoritmai,
6) daugiamačių duomenų projekcijos į mažesnės dimensijos erdvę paklaidos,
7) naujų daugiamačių taškų atvaizdavimas,
8) lygiagretieji ir paskirstyti skaičiavimai.

1.5. Mokslinis naujumas ir ginamieji teiginiai

Sukurtas lygiagretusis SAMANN algoritmas, leidžiantis greičiau atlikti didelės dimensijos daugiamačių duomenų vizualizavimą. Šis algoritmas taip pat leidžia vizualizuoti didesnės apimties duomenų aibes.

Darbe nagrinėta Hyper-Threading technologija leidžia efektyviau išnaudoti kompiuterio resursus, tačiau jos naudojimas dirbant su lygiagrečiuoju SAMANN algoritmu yra neefektyvus.

Eksperimentiškai nustatyta, kaip parinkti SAMANN tinklo neuronų aktyvacijos funkcijos nuolydžio parametro reikšmę, kad algoritmas veiktų efektyviai.

Vizualizuojant didelės apimties duomenų aibes, net dirbant su lygiagrečiuoju algoritmu, SAMANN tinklo mokymas trunka ilgai. Darbe yra pasiūlyta strategija, kuri leidžia sudaryti tokią neuroninio tinklo mokymosi aibę, kad tinklas mokytųsi žymiai greičiau, o gaunamos per tą patį laiką projekcijos paklaidos būtų neblogesnės už projekcijos paklaidas, mokant neuroninį tinklą visais analizuojamos duomenų aibės taškais.

1.6. Darbo rezultatų aprobavimas

Tyrimų rezultatai publikuoti 6 moksliniuose leidiniuose (trys iš jų periodiniuose): 1 straipsnis leidinyje, įtrauktame į Mokslinės informacijos instituto pagrindinį (ISI Web of Science) sąrašą, 5 straipsniai leidiniuose, įtrauktuose į Mokslinės informacijos instituto konferencijos darbų (ISI Proceedings) sąrašą.

Tyrimų rezultatai buvo pristatyti ir aptarti šiose nacionalinėse ir tarptautinėse konferencijose Lietuvoje ir užsienyje:

1. The 8th International Conference ICANNGA 2007, Varšuva, Lenkija. 2007 m. balandžio 11–14 d.

   (Pranešimas įvertintas ICANNGA 2007 Best Young Researcher Paper Award in the area of Neural Networks)

2. Lietuvos jaunųjų mokslininkų konferencija „Operacijų tyrimas ir taikymai“ (LOTD – 2007), Vilnius, Lietuva, 2007 m. gegužės 18 d.

3. EURO Mini Conference Continuous Optimization and Knowledge-Based Technologies EurOPT-2008, Neringa, Lietuva, 2008 m. gegužės 20-23 d.

4. Twelfth East-European Conference on Advances in Databases and Information Systems ADBIS’2008, Pori, Suomija, 2008 m. rugsėjo 5-9 d.

5. XIII International Conference Applied Stochastic Models and Data Analysis ASMDA 2009, Vilnius, Lietuva, 2009 m. birželio 30-liepos 3 d.

1.7. Disertacijos struktūra

Disertaciją sudaro šeši skyriai ir literatūros sąrašas. Disertacijos skyriai: Įvadas, Daugiamačių duomenų projekcijos metodai, Dirbtinių neuroninių tinklų taikymas daugiamačiams duomenims vizualizuoti, SAMANN neuroninio tinklo mokymo spartinimas, Lygiagrečios technologijos ir jų taikymas SAMANN tinklo mokymui, Bendrosios išvados. Disertacijos apimtis 104 puslapiai.

2

Daugiamačių duomenų projekcijos metodai

Daugiamačių duomenų vizualizavimas – svarbus duomenų analizės įrankis, padedantis geriau suvokti daugiamačių duomenų struktūras, grupavimosi tendencijas, nustatyti atstumus tarp objektų. Grafinis duomenų pateikimas leidžia perduoti informacijos turinį natūraliu ir labiau žmogui suprantamu būdu. Įvairiose žmogaus veiklos srityse nuolat susiduriama su daugiamačiais duomenimis. Vystantis technologijoms, tobulėjant kompiuteriams ir programinei įrangai, kaupiamų duomenų apimtis sparčiai didėja. Didėja ir poreikiai tuos duomenis analizuoti.

Daugiamačiais duomenimis vadinami duomenys, charakterizuojantys objektą (tašką) $X^j = (x_1^j, x_2^j, ..., x_n^j)$, parametrai $x_1, x_2, ..., x_n$ vadinami komponentėmis, parametrų skaičius n – dimensija (matavimų skaičius). Galima suformuoti analizuojamų duomenų aibę $\mathbb{X}$, sudarytą iš taškų $X^j \in R^n$, ($X^j = (x_1^j, x_2^j, ..., x_n^j) = \{x_i^j\}$, $i = 1, ..., n$, $j = 1, ..., m$), čia m – taškų skaičius. Kiekvienas parametras turi tam tikras skaitines reikšmes. Iš analizuojamos duomenų aibės taškų galima sudaryti skaičių lentelę. Kuo didesnės dimensijos duomenys, tuo sunkiau iš lentelės išgauti informacijos apie santykius tarp atskirų objektų.

Daugiamačių duomenų analizės ir vizualizavimo metodai padeda analizuoti ir atvaizduoti žmogui suprantamesne forma sudėtingais ir dažnai nežinomais tarpusavio ryšiais susietus daugiamačius duomenis. Pagrindinis daugiamačių duomenų

vizualizavimo tikslas – tai duomenų pavaizdavimas mažesnės dimensijos erdvėje, išsaugant jų tarpusavio panašumo struktūras.

Daugiamačių duomenų (kai dimensija daugiau nei trys) tiesioginis vaizdavimas yra sudėtingas, todėl tokių duomenų pateikimui naudojami įvairūs vizualizavimo metodai. Vizualizuojant duomenis (mažinant duomenų dimensiją) dalis informacijos prarandama, todėl būtina ieškoti būdų, leidžiančių tuos praradimus minimizuoti. Vizualizavimo metodų įvairovė yra pakankamai plati, todėl, renkantis tinkamiausią metodą konkrečiai problemai, tenka nagrinėti įvairius metodus.

Šiame skyriuje yra nagrinėjami *projekcijos metodai*, kurie leidžia daugiamačius duomenų taškus pateikti mažesnės dimensijos erdvėje.

Skyriuje daugiamačių duomenų vizualizavimui buvo naudojama vyno atpažinimo duomenų aibė (*angl. Wine recognition dataset*):

Vyno duomenų aibė (Asuncion, 2007) yra klasikiniai testiniai duomenys, naudojami daugiamačių duomenų analizėje. Vyno duomenų aibė yra sudaryta iš 178-ių trylikamačių taškų sudarančių tris klases. Kiekvienas taškas atspindi vyno savybes. Duomenys buvo gauti atliekant trijų gamintojų tos pačios rūšies vyno tyrimus. Buvo tiriama vyno cheminė sudėtis atkreipiant dėmesį į 13 parametrų.

Projekcijos metodai (*angl. projection techniques, dimension reduction techniques*) *n*-mačius taškus transformuoja į *d*-mačius taškus, kai $d \le n$. Šių metodų tikslas – pateikti daugiamačius duomenis mažesnės dimensijos erdvėje taip, kad būtų kiek galima tiksliau išlaikyta tam tikra duomenų struktūra.

Duomenų projekcija gali būti suformuluota, kaip atvaizdavimas ψ iš *n*-matės erdvės į *d*-matę erdvę $\psi : R^n \to R^d, d \le n$, toks, kad tam tikras kriterijus $\boldsymbol{E}$ būtų optimizuotas. Šis formulavimas panašus į funkcijos aproksimavimo uždavinį. Tačiau, skirtingai nuo funkcijos aproksimavimo, kur atvaizduojančioji funkcija apskaičiuojama iš mokymo taškų, kurie yra įėjimo-išėjimo poros (kai išėjimai yra žinomi), projekcijos metoduose norimi išėjimai dažnai nėra žinomi. Duomenų projekcijai dydžio *d* reikšmė paprastai lygi 2 arba 3.

Tarkime, kad yra daugiamačiai duomenys, kurie išreikšti *n*-matės erdvės taškais $X^j = (x_1^j, x_2^j, ..., x_n^j)$, ($X^j \in R^n$), $j = 1,...,m$; čia x_i^j yra duomenų *j*-tojo taško X^j *i*-toji komponentė, atitinkanti *i*-tąjį parametrą; *n* žymi taško koordinačių skaičių, t.y. erdvės, kuriai priklauso taškas X^j, dimensiją; *m* – analizuojamų taškų skaičius. Tikslas – rasti daugiamačio taško $X^j = (x_1^j, x_2^j, ..., x_n^j)$ atvaizdavimą $Y^j = (y_1^j, y_2^j, ..., y_d^j)$, kai *d*<*n*, mažesnės dimensijos erdvėje R^d.

Egzistuoja didelis projekcijos metodų skaičius. Šie metodai skiriasi vienas nuo kito atvaizduojančios funkcijos ψ charakteristikomis, kaip ji gaunama ir koks optimizavimo kriterijus $\boldsymbol{E}$ yra naudojamas. Atvaizduojančioji (projekcijos) funkcija gali būti tiesinė arba netiesinė. Pagal atvaizduojančios funkcijos pobūdį projekcijos metodai yra skirstomi į dvi dideles grupes: *tiesiniai* ir *netiesiniai*.

Projekcijos metoduose yra naudojamas formalus matematinis kriterijus, pagal kurį minimizuojamas projekcijos iškraipymas. Vizualizuojant daugiamačius taškus skirtingais projekcijos metodais gaunami skirtingi vaizdai. Skirtingų vaizdų palyginimas tarpusavyje yra sudėtingas uždavinys. Todėl, vertinant daugiamačių duomenų projekcijos kokybę dažnai naudojami objektyvūs kriterijai (pvz., projekcijos paklaidos dydis), o ne gaunamų vaizdų palyginimas. Žinant visų turimų metodų detales ir charakteristikas galima pasirinkti tinkamiausią metodą.

Tiesinės projekcijos metodai skirti analizuojamų duomenų dimensijos mažinimui, atliekant duomenų tiesines transformacijas. Šie metodai yra labai efektyvūs, kai duomenys pasiskirsto po tam tikrą poerdvį. Tikslesnė duomenų struktūra išlaikoma naudojant *netiesinės projekcijos metodus*. Bet ir čia duomenų vizualizavimo iškraipymai yra neišvengiami.

Projekcijos metodai:

1) **Tiesinės projekcijos metodai**:
   - Pagrindinių komponenčių analizė (*angl. principal component analysis*),
   - Projekcijos paieškos metodas (*angl. projection pursuit*),
   - Faktorinė analizė (*angl. factor analysis*),
   - Tiesinė diskriminantinė analizė (*angl. linear discriminant analysis*),
   - Nepriklausomų komponenčių analizė (*angl. independent component analysis*);

2) **Netiesinės projekcijos metodai**:
   - Daugiamatės skalės (*angl. multidimensional scaling*),
   - Sammono projekcija (*angl. Sammon mapping*),
   - Trianguliacijos metodas (*angl. triangulation*),
   - ISOMAP metodas,
   - Lokaliai tiesinis atvaizdavimas (*angl. locally linear embedding*).

Šiame darbe nagrinėjami dažniausiai naudojami projekcijos metodai. Yra daug bandymų apjungti kelis skirtingais principais grindžiamus vizualizavimo metodus. Tai leidžia gauti daugiau naujų žinių apie analizuojamus duomenis, lyginant su atskirų metodų taikymu.

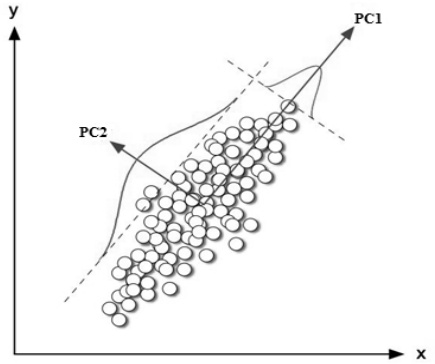
2.1. Tiesinės projekcijos metodai

Tarkime, kad yra duomenų matrica $\mathbb{X}$, sudaryta iš daugiamačių taškų $X^j = (x_1^j, x_2^j, ..., x_n^j) = \{x_i^j\}$, $i = 1,...,n$, $j = 1,...,m$. Tikslas yra surasti tokią tiesinę transformaciją A, kad $Y = A^T X$, $y \in R^d$, $d<n$.

Pagrindinių komponenčių analizė (*angl. principal component analysis, PCA*) yra klasikinis statistikinis tiesinės projekcijos metodas. Šis metodas dar žinomas kaip *Karhunen-Loeve transformacija* arba *Hotelling transformacija*. Tai ortogonali tiesinė duomenų transformacija plačiai naudojama duomenų analizėje (pavyzdžiui, klasifikavimui arba atpažinimui) kaip daugiamačių duomenų dimensijos mažinimo metodas.

Naudojant pagrindinių komponenčių analizės metodą ieškoma geriausio poerdvio daugiamačių duomenų projekcijai, t.y. poerdvio, kuriame išlaikyta kiek įmanoma daugiau originalios erdvės duomenų savybių bei informacijos. Metodo tikslas – rasti kryptį, kuria dispersija yra didžiausia (Jollife, 1986), (Haykin, 1999), (Taylor, 2003). Didžiausią dispersiją turinti kryptis vadinama pirmąja pagrindine komponente (PC_1). Ji eina per duomenų centrinį tašką. Taškų visumos kvadratinis vidutinis atstumas iki šios tiesės yra minimalus, t.y. ši tiesė yra kiek galima arčiau visų duomenų taškų (2.1 pav.). Antrosios pagrindinės komponentės (PC_2) ašis taip pat turi eiti per duomenų centrinį tašką ir ji turi būti nekoreliuota (ortogonali) pirmosios pagrindinės komponentės ašiai.

Esminė PCA idėja – sumažinti duomenų dimensiją kiek galima tiksliau išlaikant duomenų dispersijas. Tai padeda geriau vizualizuoti duomenis, o tuo pačiu ir palengvina duomenų suvokimą.

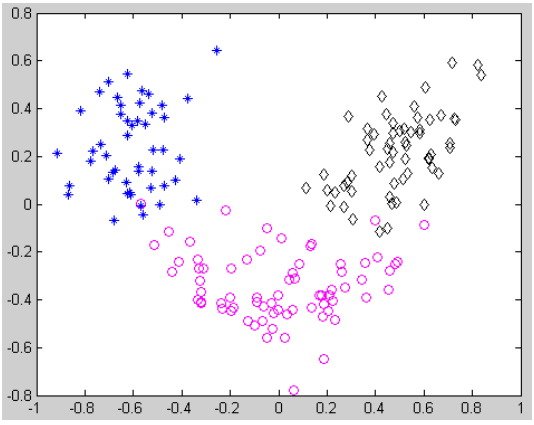


2.1 pav. *Pirmoji (PC1) ir antroji (PC2) pagrindinės komponentės*

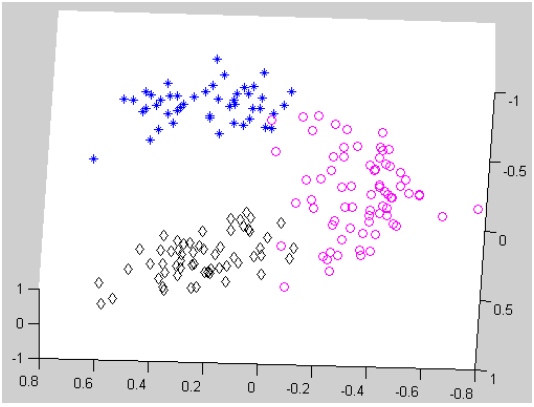
Pagrindinių komponenčių analizės metodą sudaro duomenų kovariacinės matricos nuosavų reikšmių λ (*angl. eigenvalue*) ir nuosavų vektorių e (*angl. eigenvector*) skaičiavimai. Kiekvienas nuosavas vektorius yra vadinamas pagrindine komponente. Pagrindinės komponentės yra lygties $Ce = \lambda e$ sprendinys *e*. Šioje lygtyje e yra vektorius-stulpelis, *C* yra duomenų kovariacinė matrica, ir λ – nuosava reikšmė, randama iš charakteringos lygties $|C - \lambda I| = 0$. Čia *I* yra vienetinė matrica, kurios matmenys tokie pat, kaip matricos *C*. Ženklu |.| apibrėžtas diskriminantas. Komponenčių kryptys yra ortogonalios, ir komponentės su didžiausiomis nuosavomis

reikšmėmis turi didžiausią dispersiją. Nuosavų vektorių skaičius yra lygus kintamųjų skaičiui, t.y. *n*. Nuosavas vektorius, susijęs su didžiausia nuosava reikšme, turi tokią pat kryptį kaip pirmoji pagrindinė komponentė. Antroji pagrindinė komponentė atitinka antrąjį nuosavą vektorių ir t.t. Pagrindinių komponenčių skaičius parenkamas atsižvelgiant į projektinės erdvės dimensiją. Yra sukurti efektyvūs algoritmai pagrindinėms komponentėms rasti. Taip pat, gali būti taikomi ir dirbtiniai neuroniniai tinklai (3 skyrius).

Sudarykime pagrindinių komponenčių matricą $A=\{e_i\}$, $i=1,...,n$, iš nuosavų vektorių e kaip vektorių-eilučių. Kiekvienas šios matricos vektorius-eilutė yra ortogonalus bet kuriam kitam. Transformuokime bet kurį duomenų vektorių X pagal formulę $Y=A(X-\overline{X})$, čia $X=(x_1^j, x_2^j, ..., x_n^j)^T$, $\overline{X}=(\overline{x}_1, \overline{x}_2, ..., \overline{x}_n)^T$, $\overline{x}_i=\frac{1}{m}\sum_{j=1}^{m} x_i^j, i=1,...,m$, $A=(e_1, e_2, ..., e_n)^T$.



2.2 pav. *PCA pavyzdys (vyno duomenų aibės projekcija dvimatėje erdvėje)*



2.3 pav. *PCA pavyzdys (vyno duomenų aibės projekcija trimatėje erdvėje)*

Daugiamačių duomenų transformacijai galima naudoti ne visus nuosavus vektorius, o tik pirmuosius. Tegu matrica A_d sudaryta iš *d* pirmųjų nuosavų vektorių. Tai reiškia, kad pradinį duomenų tašką projektuojame į koordinačių sistemą, turinčią *d* dimensijų. Vizualizavime naudojama $d = 2$ arba $d = 3$.

2.2 ir 2.3 paveiksluose pateiktas PCA metodo pavyzdys vyno duomenims, kai duomenys projektuojami į dvimatę ir trimatę erdves (atitinkamai dvi ir trys pagrindinių komponenčių ašys).

Naudojant pagrindinių komponenčių metodą, negalima gauti išsamios duomenų charakteristikos, nes nagrinėjamos tik tiesinės priklausomybės. PCA yra netinkamas metodas daugiamačių duomenų dimensijos mažinimui, jeigu tarp tų duomenų egzistuoja stiprūs netiesiniai sąryšiai. Didelę įtaką PCA metodo rezultatui daro taškai-atsiskyrėliai (*angl. outliers*), nes jie yra izoliuoti ir dirbtinai padidina dispersijos reikšmę.

Standartiniai PCA metodai netinka didelės dimensijos duomenims arba didelėms duomenų aibėms apdoroti. Daugiamatės erdvės duomenims kovariacinė matrica yra labai didelė. *d*-matės erdvės duomenims, tenka skaičiuoti $d \times d$ dydžio matricą, o algoritmo sudėtingumas gaunamas $O(d^3)$. Kai kurie algoritmai, realizuojantys PCA projekciją naudoja efektyvesnius matricų pertvarkymus (Wilkinson, 1965). Jų sudėtingumas yra $O(d^2)$. Dėl didelės skaičiavimų apimties PCA metodas netinka dirbant su didelės dimensijos daugiamačiais duomenimis.

Projekcijos paieška. Analizuojant *n*-mačius taškus, norima pamatyti jų išsidėstymą *n*-matėje erdvėje. Kai $n > 3$, tiesiogiai pamatyti šiuos taškus neįmanoma, tačiau yra galimybė projektuoti juos į dvimatę ar trimatę erdvę. Skirtingos projekcijos duoda skirtingus rezultatus, be to, ne visada pavyksta išlaikyti duomenų struktūrą. Rasti tinkamą *n*-mačių duomenų projekciją padeda projekcijos paieškos metodas (Huber, 1985), (Jones, 1987). Projekcijos paieškos metodas (*angl. projection pursuit, PP*) buvo pasiūlytas ir tyrinėtas eksperimentiškai J. B. Kruskalo (Kruskal, 1972). Šis metodas leidžia surasti „įdomių“ galimų daugiamačių duomenų projekcijų. Projekcija yra laikoma „įdomia“, jeigu ji vaizduoja neatsitiktinius debesis.

Metodo tikslas – rasti tokias tiesines komponenčių kombinacijas (dažniausiai dvimates ar trimates), kad transformuoti duomenys išlaikytų pradinių duomenų struktūrą. Tarkime, kad yra duomenų matrica $\mathbb{X}$, susidedanti iš *n*-mačių taškų. Erdvės R^n projekciją į R^2 galima išreikšti taip: $(z_1, z_2) = (\mathbb{X}a^T, \mathbb{X}b^T)$. Čia *a* ir *b* yra *n*-mačiai vektoriai-eilutės; a^T, b^T – transponuoti *a* ir *b* vektoriai, t.y. vektoriai-stulpeliai; z_1, z_2 – gauti po projekcijos *d*-mačiai vektoriai-stulpeliai. Gauta projekcija priklauso nuo vektorių *a* ir *b* parinkimo. Parenkant vektorius *a* ir *b* reikia nustatyti matą, išreiškiantį savybę, kurią norime stebėti, vektoriais (z_1, z_2). Pavadinkime šį matą $I(z_1, z_2)$. Kartais jis vadinamas *projekcijos indeksu* arba *indekso funkcija* (*angl. projection, index function*). Dažniausiai naudojamas statistikinis klasterizavimo matas,

kuris leidžia nustatyti duomenų klasterius. Šis matas yra vidutinis atstumas iki artimiausio kaimyno (*angl. mean nearest neighbour distance*). Kuo mažesnė šio mato reikšmė viename klasteryje, tuo duomenys yra geriau suklasterizuoti. Taigi, tegu $I(z_1, z_2)$ yra vidutinis atstumas iki artimiausio kaimyno. Užrašas $I(z_1, z_2)$ gali būti pakeistas į $I(\mathbb{X}a', \mathbb{X}b')$. Projekcijos parinkimo uždavinys susiveda į optimizavimo uždavinį: reikia parinkti tokius vektorius *a* ir *b*, kad funkcijos $I(\mathbb{X}a', \mathbb{X}b')$ reikšmė būtų mažiausia. Iš esmės tai ir yra projekcijos paieškos procesas.

Projekcijos paieškos metodas leidžia apdoroti didelės dimensijos duomenis. Naudojant šį metodą galima atvaizduoti naujus daugiamačius taškus neperskaičiuojant viso algoritmo. Šio metodo skaičiavimo apimtis yra proporcinga daugiamačių taškų kiekiui, t.y. *O(N*log(N))*, kur *N* – taškų kiekis.

Faktorinė analizė. Faktorinėje analizėje (*angl. factor analysis*) (Harman, 1967), (Mardia, 1995) tiriamos tiesinės priklausomybės tarp parametrų (kintamųjų). Ji taikoma kintamųjų skaičiaus mažinimui, daugiamačių duomenų struktūros nustatymui, kintamųjų klasifikavimui ir grupavimui. Faktorinės analizės metodo prielaida yra ta, kad nagrinėjami parametrai priklauso nuo tam tikrų paslėptų faktorių. Metodo tikslas atskleisti tokius ryšius ir daugiamačių duomenų dimensijos mažinimui panaudoti tam tikrą faktoriaus modelį.

Spėjama, kad kiekvienas faktorius įtakoja daug stebimų parametrų. Jis vadinamas *bendru faktoriumi* (*angl. common factor*). Stebimi parametrai yra tiesinė kombinacija bendrų paslėptų faktorių; kiekvieno faktoriaus koeficientas vadinamas *svoriu* (*angl. loading*). Kiekvienas parametras turi dar vieną komponentę, *specifinį faktorių* (*angl. specific factor*), kuri aprašo to parametro nepriklausomą kintamumą (*angl. independent variability*). Specifiniai faktoriai gali būti interpretuojami kaip vektoriaus triukšmas arba paklaida. Faktorių skaičius *k* nustatomas pagal tam tikrus kriterijus.

Daugiamatis taškas $X = (x_1, ..., x_n) \in R^n$ atitinka *k*-faktoriaus modelį, jeigu jis gali būti išreikštas tokia forma (Mardia, 1995): $X = \Delta f + U$. $\Delta = \{\lambda_{ij}, i = 1, ..., n, j = 1, ..., k\}$ yra konstantų matrica (svoriai), $f = (f_1, ..., f_k)$ yra bendrų paslėptų faktorių vektorius ir $U = (u_1, ..., u_n)$ yra specifinių faktorių vektorius. Daugiamatis taškas $X \in R^n$ gali būti užrašytas taip $x_i = \sum_{j=1}^{k} \lambda_{ij} f_i + u_i$, $i = 1, \ldots, n$.

Tiesinė diskriminantinė analizė (*angl. Linear discriminant analysis, LDA*) arba **Fišerio diskriminantinė analizė** (*angl. Fisher discriminant analysis*). Tiesinės diskriminantinės analizės metodas (Fukunaga, 1990), (Duda, 1973), (Duda, 2000) transformuoja daugiamatės erdvės duomenis į mažesnės dimensijos erdvę taip, kad tam tikros klasės atskiriamumo kriterijus būtų optimalus. LDA metodas ieško tokių krypčių, kuriomis klasės yra geriausiai atskiriamos.

Kartais daugiamačių duomenų dimensijai mažinti naudojamas **nepriklausomų komponenčių analizės** (*angl. independent component analysis, ICA*) metodas. Nepriklausomų komponenčių analizė, tai metodas, kuris transformuoja pagrindines komponentes į statistiškai nepriklausomas komponentes. ICA metodas gali būti taikomas įvairių problemų sprendimui: vaizdų analizėje, duomenų analizėje, signalų apdorojimui, požymių išskyrimui. Plačiai taikomas statistikoje, neuroniniuose skaičiavimuose. Detalesnė informacija apie ICA yra pateikta literatūroje (Comon, 1994), (Hyvarinen, 2001), (Roberts, 2000).

2.2. Netiesinės projekcijos metodai

Tarkime, kad kiekvieną *n*-matį tašką $X^i \in R^n$ atitinka mažesnės dimensijos taškas $Y^i \in R^d$, ($d < n$). Pažymėkime atstumą tarp taškų X^i ir X^j – d_{ij}^*, o atstumą tarp taškų Y^i ir Y^j – d_{ij}, $i, j = 1,...,m$. Norint surasti daugiamatės erdvės taškų X^i projekcijas žemesnio matavimo erdvėje Y^i yra skaičiuojami atstumai tarp visų taškų atitinkamose erdvėse.

Ieškoma tokia *d*-matės erdvės taškų $Y^i \in R^d$ projekcija, kad atstumai d_{ij} tarp projektinės erdvės taškų būtų kiek galima artimesni atstumams d_{ij}^* tarp taškų pradinėje erdvėje. Dažniausiai atstumams d_{ij} ir d_{ij}^* skaičiuoti naudojami Euklido atstumai. Tačiau d_{ij} nebūtinai turi tenkinti trikampio nelygybės sąlygą $d_{ik} \le d_{ij} + d_{jk}$ ir simetriškumo sąlygą $d_{ij} = d_{ji}$; todėl bendru atveju (kai šios sąlygos nėra tenkinamos) naudojama nepanašumo (*angl. dissimilarity*) sąvoka, o ne atstumas.

Kadangi dažnai neįmanoma surasti tokios projekcijos, kurioje $d_{ij} = d_{ij}^*$ visiems *i* ir *j*, yra apibrėžiamas kokybės kriterijus *E* skirtingų projekcijų įvertinimui. Priklausomai nuo kriterijaus parinkimo, galutinėje projekcijoje tam tikri atstumai tarp taškų (maži ar dideli) bus išlaikyti geriau negu kiti. Yra pasiūlytos trys funkcijos daugiamačių duomenų projekcijos paklaidos įvertinimui (kartu nurodyti ir funkcijų gradientai) (Duda, 2000):

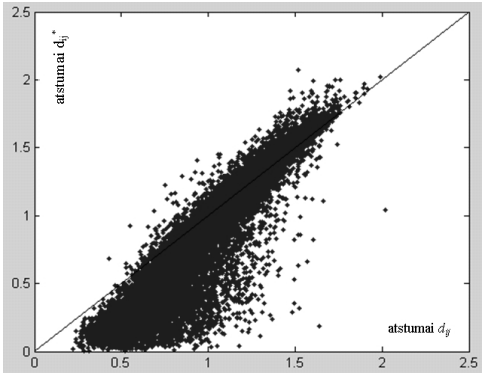
$$E_1 = \frac{1}{\sum_{i<j} d_{ij}^{*2}} \sum_{i<j} (d_{ij} - d_{ij}^*)^2 \,,\ \nabla_{yk} E_1 = \frac{2}{\sum_{i<j} d_{ij}^{*2}} \sum_{j \ne k} \left((d_{kj} - d_{kj}^*) \frac{y_k - y_j}{d_{kj}} \right), \tag{2.1}$$

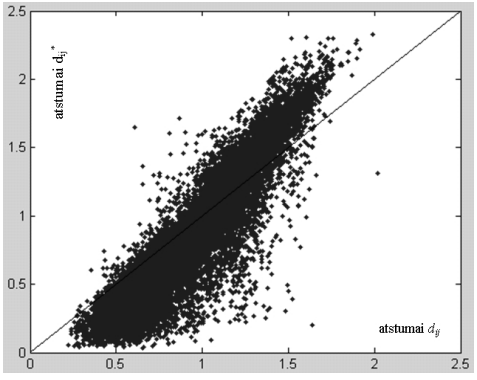
$$E_2 = \sum_{i<j} \left(\frac{d_{ij} - d_{ij}^*}{d_{ij}^*} \right)^2 \,,\ \nabla_{yk} E_2 = 2 \sum_{j \ne k} \left(\frac{d_{kj} - d_{kj}^*}{d_{kj}^{*2}} \cdot \frac{y_k - y_j}{d_{kj}} \right), \tag{2.2}$$

$$E_3 = \frac{1}{\sum_{i<j} d_{ij}^*} \sum_{i<j} \frac{(d_{ij} - d_{ij}^*)^2}{d_{ij}^*}, \quad \nabla_{yk} E_3 = \frac{2}{\sum_{i<j} d_{ij}^*} \sum_{j \neq k} \left(\frac{d_{kj} - d_{kj}^*}{d_{kj}^*} \cdot \frac{y_k - y_j}{d_{kj}} \right). \qquad (2.3)$$

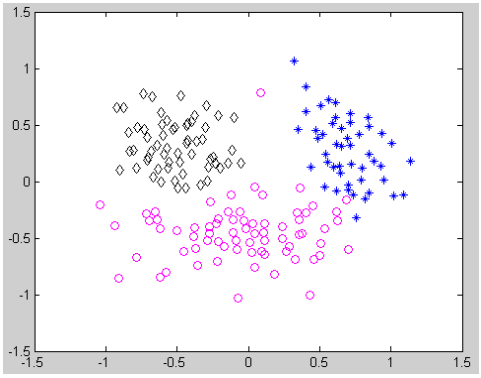
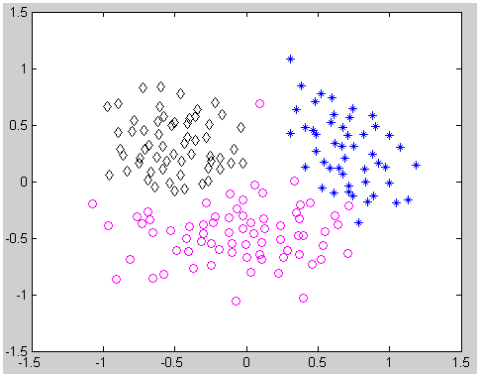
Kriterijus E_1 (2.1) formulėje akcentuoja didžiausias paklaidas, nepriklausomai nuo to ar atstumai d_{ij}^* dideli ar maži. Kriterijus E_2 (2.2) formulėje akcentuoja didžiausias santykines paklaidas, nepriklausomai ar paklaidos reikšmė maža ar didelė. Kriterijus E_3 (2.3) formulėje numato kompromisą tarp pirmų dviejų kriterijų, pabrėždamas paklaidų sandaugas ir santykines paklaidas.

Atstumų d_{ij}^* ir d_{ij} taškinė diagrama, vadinama Šepardo diagrama (*angl. Shepard diagram*) rodo, kokie atstumai geriau atspindi duotus nepanašumus. Šepardo diagrama, gauta minimizuojant kriterijaus funkcijas E_1 ir E_3 vyno duomenims pavaizduota 2.4 paveiksle. Kuo arčiau tiesės $y=x$ yra sukoncentruoti taškai, tuo geriau išlaikomi atstumai. Minimizuojant kriterijaus funkciją E_1, geriausiai išlaikomi didesni atstumai, negu mažesni. Minimizuojant funkciją E_3, didesni atstumai išlaikomi blogiau, o mažesni geriau. Iš 2.4(a) paveikslo matyti, kad, minimizuojant kriterijaus funkciją E_1, diagramos taškai labiau sukoncentruoti žemiau tiesės $y=x$ (ypač kairiajame apatiniame kampe). Minimizuojant kriterijaus funkciją E_3 (2.14(b) paveikslas) taškai pasiskirsto arčiau tiesės $y=x$. Kriterijaus funkcijos parinkimas gali įtakoti duomenų atvaizdavimo rezultatą. Tai priklauso nuo pradinių duomenų specifikos ir atstumų tarp taškų. Projekcijos rezultatai, minimizuojant E_1 ir E_3 funkcijas (projekcijos gautos, naudojant daugiamačių skalių metodą) pateikti 2.5 paveiksle.

a) 

b) 

2.4 pav. *Šepardo diagrama vyno duomenims, kai naudojama: a) kriterijaus funkcija E_1, b) kriterijaus funkcija E_3*

a)  b) 

2.5 pav. *Vyno duomenų atvaizdavimas, kai naudojama: a) kriterijaus funkcija E_1 (E_1=0,2138); b) kriterijaus funkcija E_3 (E_3=0,0574)*

Daugiamačių skalių metodas

Daugiamačių skalių metodas (*angl. multidimensional scaling*, *MDS*) – tai grupė metodų, plačiai naudojamų daugiamačių duomenų analizėje įvairiose šakose, ypač ekonomikoje, socialiniuose moksluose ir kt. *n*-mačiai taškai projektuojami į mažesnės dimensijos erdvę (dažniausiai į R^2 arba R^3) siekiant išlaikyti atstumus (nepanašumus) tarp analizuojamos aibės objektų (Kaski, 1997), (Borg, 2005), (Kruskal, 1984). Analizės rezultate gautuose dvimačiuose grafikuose tie objektai, kurie yra panašūs, yra pavaizduojami arčiau vieni kitų, o mažiau panašūs – toliau vieni nuo kitų. Minimizavimo problema yra sprendžiama taip, kad atstumai tarp taškų mažesnės dimensijos erdvėje atitiktų duotus nepanašumus kaip įmanoma geriau. Atvaizdavimas paprastai yra netiesinis, ir padeda atskleisti bendrą analizuojamų duomenų struktūrą.

Pradiniai duomenys, kurie analizuojami šiuo metodu, turi būti kvadratinė, simetrinė matrica, susidedanti iš ryšių tarp analizuojamų duomenų aibės elementų. Tai gali būti atstumų arba nepanašumų matrica. Ryšiais tarp aibės elementų gali būti ir Euklido atstumai. Tačiau, bendru atveju, tai nebūtinai turi būti atstumai griežtai matematine prasme.

Vienas MDS pavyzdys galėtų būti toks: tarkime turime matricą, sudarytą iš atstumų tarp pagrindinių šalies miestų; MDS analizės rezultate gautume miestų išdėstymą žemėlapyje, t.y. dvimatėje plokštumoje (Leeuw, 2003). Kitas MDS matricos pavyzdys yra koreliacijų tarp duomenų parametrų matrica. Jei tie duomenys traktuojami kaip panašumai, MDS algoritmu stipriai koreliuoti parametrai atvaizduojami arti vieni kitų, silpnai koreliuoti – toliau vieni nuo kitų.

Vienas MDS tikslų yra rasti optimalią daugiamačių duomenų konfigūraciją dvimatėje erdvėje. Yra daugybė skirtingų MDS variantų su skirtingomis paklaidų funkcijomis (STRESS) ir jas optimizuojančiais algoritmais (Borg, 2005). Pagal analizuojamus duomenis MDS algoritmai gali būti skirstomi į *metrinius* (*angl. metric*) ir *nemetrinius*

(*angl. non-metric*). Pirmasis MDS algoritmas metriniams duomenims buvo pasiūlytas W.S. Torgensono 1952 metais (Torgenson, 1952), vėliau MDS algoritmai buvo taikyti ir nemetriniams duomenims (Shepard, 1962a), (Shepard, 1962b).

Metriniai MDS algoritmai (Taylor, 2003) naudojami tada, kai įmanoma rasti Euklido atstumus tarp analizuojamų duomenų elementų, t.y. analizuojami metriniai duomenys. Pagrindinis šių algoritmų tikslas – pavaizduoti daugiamačius taškus dvimatėje erdvėje taip, kad atstumai tarp dvimačių taškų būtų kiek galima artimesni atstumams tarp atitinkamų daugiamačių taškų. Tam minimizuojama tam tikra paklaidos funkcija.

Tarkime kiekvieną *n*-matį tašką $X^i \in R^n$ atitinka mažesnės dimensijos taškas $Y^i \in R^d$ ($d < n$). Pažymėkime atstumą tarp taškų X^i ir X^j – d_{ij}^*, o atstumą tarp taškų Y^i ir Y^j – d_{ij}, $i, j = 1,...,m$. Metrinis MDS algoritmas bando priartinti atstumus d_{ij} prie atstumų d_{ij}^*. Jei naudojama kvadratinė paklaidos funkcija, tai minimizuojama tikslo funkcija E_{MDS} gali būti užrašyta taip:

$$E_{MDS} = \sum_{\substack{i,j=1 \\ i<j}}^{m} w_{ij}(d_{ij}^* - d_{ij})^2 . \qquad (2.4)$$

Paklaidos funkcija E_{MDS} dar vadinama STRESS funkcija. Dažnai naudojami tokie svoriai:

$$w_{ij} = \frac{1}{\sum_{\substack{k,l=1 \\ k<l}}^{m} (d_{kl}^*)^2}; \quad w_{ij} = \frac{1}{d_{ij}^* \sum_{\substack{k,l=1 \\ k<l}}^{m} d_{kl}^*}; \quad w_{ij} = \frac{1}{m d_{ij}^*} .$$

Taip pat gali būti naudojama kiek kitokios išraiškos funkcija, vadinama SSTRESS:

$$E_{MDS} = \sum_{\substack{i,j=1 \\ i<j}}^{m} w_{ij}((d_{ij}^*)^2 - d_{ij}^2)^2 .$$

Vienas iš paprasčiausių šių funkcijų minimizavimo būdų – gradientinis nusileidimas. Pradėjus nuo atsitiktinės pradinių dvimačių taškų konfigūracijos, iteraciniame procese dvimačių taškų $Y^i \in R^2$ koordinatės y_k^i, $i = 1,...,m$, $k = 1, 2$, keičiamos pagal formulę $y_k^i(m'+1) = y_k^i(m') - \alpha \frac{\partial E_{MDS}(m')}{\partial y_k^i(m')}$. Čia m' yra iteracijos numeris, o α – parametras, įtakojantis optimizavimo žingsnį.

Tačiau galimi ir kiti optimizavimo būdai, tokie kaip SMACOF (*angl. scaling by majorization a complicated function*) algoritmas, pagrįstas tikslo funkcijos mažorizavimu (Borg, 2005), jungtinių gradientų metodas, kvazi-Niutono metodas, deterministinis atkaitinimo modeliavimo algoritmas (*angl. simulated annealing*) (Klock, 1999), genetinio algoritmo ir lokalaus nusileidimo metodų kombinacijos (Mathar, 1993), (Podlipskytė, 2003). Funkcijų STRESS ir SSTRESS minimizavimo uždaviniai yra sudėtingi dėl kriterijų daugiaekstremalumo (Žilinskas, 2003). Darbe (Podlipskytė, 2003) yra pateikta kriterijų STRESS ir SSTRESS galimų optimizavimo strategijų apžvalga bei jų tyrimai.

Kartais, analizuojant objektus, yra prasmingos ne atstumų skaitinės reikšmės, o atstumų tarp objektų eilės numeriai. Tada tikslinga naudoti *nemetrinius* MDS algoritmus, kuriuose objektų nepanašumai nėra atstumai. Nepanašumą tarp *i*-tojo ir *j*-tojo objektų apibrėžkime realiu skaičiumi δ_{ij}. Kartais šis matas nėra tinkamas Euklido erdvei, todėl jis transformuojamas funkcija *f*: $f(\delta_{ij})$. Nemetriniuose MDS algoritmuose dažniausiai minimizuojama (2.5) paklaidos (STRESS) funkcija:

$$E_{MDS} = \sum_{\substack{i,j=1 \\ i<j}}^{m} w_{ik}\left(f(\delta_{kl}) - d_{kl}\right)^2 . \qquad (2.5)$$

MDS algoritmo trūkumai: MDS algoritmo tikslo funkcijos optimizavimas reikalauja palyginimų tarp visų taškų porų; MDS algoritmai yra neefektyvūs, dirbant su didelės apimties duomenų aibėmis; MDS negali vizualizuoti naujų taškų tol, kol nebus perskaičiuoti visi analizuojami taškai.

Sammono algoritmas

Sammono projekcija, kartais dar vadinama **Sammono netiesinė projekcija** (Sammon, 1969), yra netiesinis daugelio kintamųjų objektų atvaizdavimo žemesnio matavimo erdvėje metodas. Tai vienas iš gana plačiai naudojamų daugiamačių skalių (MDS) grupės metodų (Bezdek, 1995). Dažniausiai nagrinėjami atvejai, kai projektinės erdvės, į kurią atvaizduojami *n*-mačiai taškai, dimensija yra 2 arba 3. Sammono projekcijos tikslas yra minimizuoti skirtumus tarp atstumų tarp taškų pradinėje erdvėje ir atstumų tarp atitinkamų taškų projektinėje erdvėje. Sammono projekcija negali greitai apdoroti naujų taškų be papildomų skaičiavimų. Visi atstumai tarp taškų turi būti skaičiuojami iš naujo, naudojant visą turimą duomenų aibę. Todėl tai sudaro sunkumų, kai duomenys pastoviai atnaujinami arba papildomi naujais taškais, ir kai duomenų aibės yra labai didelės.

Sakykime, kad yra daugiamačiai taškai $X^i = (x_1^i, x_2^i, ..., x_n^i)$, $i = 1,...,m$, priklausantys erdvei R^n. Sprendžiamas uždavinys – šiuos *n*-mačius taškus $X^i = (x_1^i, x_2^i, ..., x_n^i)$, $i = 1,...,m$, atvaizduoti (gauti projekciją) plokštumoje R^2. Juos

atitiks dvimačiai taškai $Y^1, Y^2, ..., Y^m \in R^2$. Čia $Y^i = (y_1^i, y_2^i)$, $i = 1,...,m$. Pažymėkime d_{ij}^* atstumą tarp daugiamačių taškų X^i ir X^j, d_{ij} – atstumą tarp taškus X^i ir X^j atitinkančių dvimačių taškų Y^i ir Y^j ($i, j = 1,..,m$). d_{ij}^* ir d_{ij} dažniausiai yra Euklido atstumai, bet gali būti naudojami ir kiti. Sammono algoritmas minimizuoja projekcijos paklaidą:

$$E_S = \frac{1}{\sum\limits_{\substack{i,j=1 \\ i<j}}^{m} d_{ij}^*} \sum_{\substack{i,j=1 \\ i<j}}^{m} \frac{(d_{ij}^* - d_{ij})^2}{d_{ij}^*}.$$

Funkcija E_S sutampa su funkcija E_{MDS} (2.4 formulė), kai $w_{ij} = \frac{1}{d_{ij}^* \sum\limits_{\substack{k,l=1 \\ k<l}}^{m} d_{kl}^*}$.

Sammono paklaida (*angl. Sammon's stress, Sammon's error*) E_S – tai matas, kuris parodo, kaip tiksliai išlaikomi atstumai tarp taškų pereinant iš didesnio matavimo erdvės į mažesnio matavimo erdvę. Sammono paklaidą labiau įtakoja atstumai tarp artimiausių taškų, negu tarp tolimesnių. Pagrindinis uždavinys minimizuoti šią paklaidos funkciją E_S. Tam gali būti naudojami įvairūs optimizavimo metodai, nurodyti aprašant daugiamačių skalių metodą. J. W. Sammon darbe (Sammon, 1969) pasiūlė vieną funkcijos E_S minimizavimo strategiją, kurią kai kurie tyrėjai vadina kvazi-Niutono, kiti – greičiausio nusileidimo (*angl. steepest-descent*) minimizavimu (Kohonen, 2001). Jis konverguoja greičiau, negu paprastas gradientinis metodas. Bet gali būti naudojamos ir kitos paprastesnės strategijos. Darbe (Press, 1992) nustatyta, kad žymiai efektyvesnis metodas yra jungtinių gradientų metodas (*angl. conjugate-gradient*).

Kvazi-Niutono (greičiausio nusileidimo) metode dvimačių taškų $Y^i \in R^2$ koordinatės y_k^i, $i = 1,...,m$, $k = 1, 2$, randamos naudojantis (2.6) iteracine formule:

$$y_k^i(m'+1) = y_k^i(m') - \alpha \frac{\frac{\partial E_S(m')}{\partial y_k^i(m')}}{\left| \frac{\partial^2 E_S(m')}{\partial (y_k^i)^2(m')} \right|} \qquad (2.6)$$

Dalinės išvestinės randamos pagal (2.7)–(2.9) formules:

$$\frac{\partial E_S}{\partial y_k^i} = -\frac{2}{c} \sum_{\substack{j=1 \\ i \neq j}}^{m} \left(\frac{d_{ij}^* - d_{ij}}{d_{ij}^* d_{ij}} \right) \left(y_k^i - y_k^j \right) \qquad (2.7)$$

$$\frac{\partial^2 E_S}{\partial (y_k^i)^2} = -\frac{2}{c} \sum_{\substack{j=1 \\ i \neq j}}^{m} \frac{1}{d_{ij}^* d_{ij}} \left[\left(d_{ij}^* - d_{ij} \right) - \frac{\left(y_k^i - y_k^j \right)^2}{d_{ij}} \left(1 + \frac{d_{ij}^* - d_{ij}}{d_{ij}} \right) \right] \quad (2.8)$$

$$c = \sum_{\substack{i,j=1 \\ i<j}}^{m} d_{ij}^* \quad (2.9)$$

Čia m' yra iteracijos numeris, o α yra parametras, įtakojantis optimizavimo žingsnį. J. W. Sammon jį vadino „magiškuoju faktoriumi“. Šis pavadinimas tradiciškai naudojamas ir kituose darbuose (Kohonen, 2001), (Konig, 2000). Vienoje iteracijoje perskaičiuojami m taškų $Y^i \in R^2$, $i = 1,...,m$, koordinatės. Gauta projekcijos E_S paklaida priklauso ir nuo parametro α ir nuo taškų $Y^1, Y^2, ..., Y^m \in R^2$ pradinių reikšmių parinkimo. Eksperimentiškai nustatyta, kad mažiausia paklaida gaunama, kai $\alpha \in [0,3; 0,4]$ (Sammon, 1969), (Kohonen, 2001). Tačiau negalima teigti, kad tas diapazonas yra optimalus visais atvejais.

Apibendrinta Sammono algoritmo schema yra tokia:

1. Skaičiuojami atstumai tarp analizuojamų taškų pradinėje erdvėje;
2. Atsitiktinai parenkami projektinės erdvės taškai;
3. Skaičiuojama projekcijos (Sammono) paklaida E_S;
4. Atnaujinamos projektinės erdvės koordinatės pagal (2.6);
5. Jeigu Sammono paklaidos reikšmė mažesnė už pasirinktą slenkstį arba iteracijų skaičius viršija nustatytąjį, tuomet algoritmas sustabdomas, priešingu atveju vėl pradedam nuo 3 žingsnio.

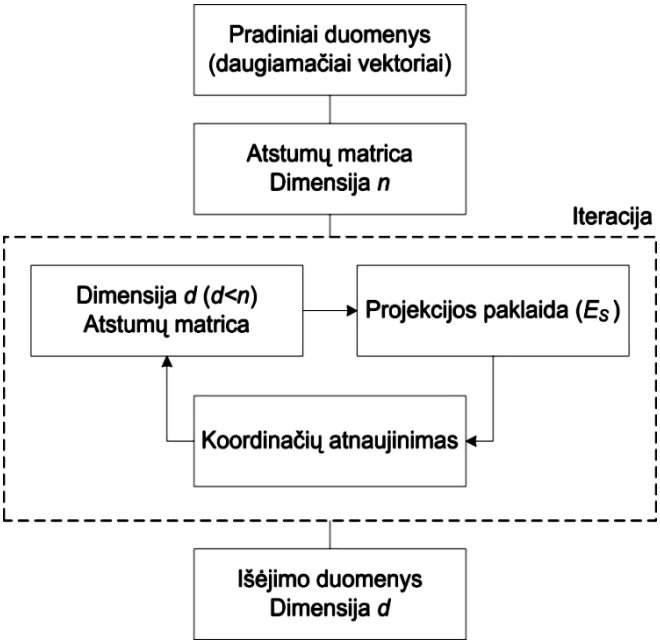
Sammono projekcijos struktūrinė schema pavaizduota 2.6 paveiksle.

Nors šiuo metu yra ir kitų paklaidos E_S optimizavimo metodų, J. W. Sammon pasiūlytas variantas sėkmingai naudojamas darbuose (Dzemyda, 2005), (Kohonen, 2001), (Kaski, 1997), (Konig, 2000).

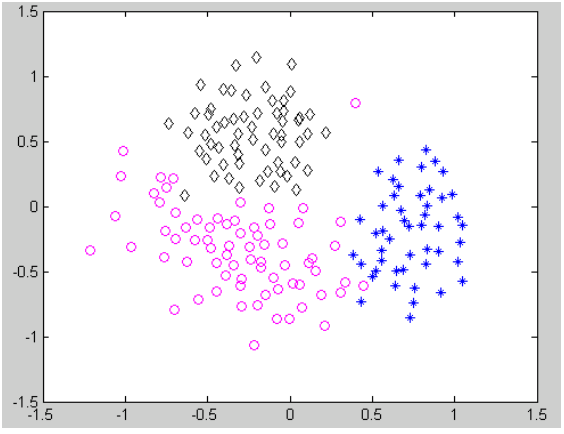
Darbe (Montvilas, 2003a) pasiūlytas nuoseklaus atvaizdavimo metodas daugiamačiams duomenims stebėti realiu laiku. Tokiu atvaizdavimu siekiamas tikslas – išlaikyti vidinę atstumų tarp duomenų taškų struktūrą po jų atvaizdavimo. Metodas skiriasi nuo Sammono vienalaikio daugiamačių duomenų atvaizdavimo į plokštumą metodo tuo, kad po pradiniame etape vienu metu atvaizduotų keleto duomenų taškų vėliau galima dirbti nuosekliai realiu laiku. Metodas taip pat tirtas ir darbuose (Montvilas, 2003b), (Montvilas, 2006). Tačiau metodas gali ne visada teisingai atvaizduoti naujus taškus. Pradžioje dalis analizuojamų duomenų aibės taškų yra vaizduojama klasikiniu Sammono metodu, o likę arba atsiradę nauji taškai dėstomi

naudojant trianguliacijos metodą. Tokiu būdu pakankamai teisingai gali būti atvaizduojami tik tie taškai, kurie yra panašūs į pradinės aibės taškus. Panašus metodas pasiūlytas darbe (Pekalska, 1999).

Vietoj Euklido atstumų, skaičiuojant Sammono paklaidą, gali būti naudojami ir kiti atstumo matai (Yang, 2004).



2.6 pav. *Sammono projekcijos struktūrinė schema*



2.7 pav. *Sammono projekcijos pavyzdys (vizualizuoti vyno duomenys)*

Trianguliacijos metodas

Darbe (Lee, 1977) pateiktas nuoseklaus daugiamačių taškų atvaizdavimo į plokštumą metodas naudojant trianguliacijos (*angl. triangulation*) metodą. Atvaizduojant daugiamačius duomenis plokštumoje tiksliai išlaikomi atstumai nuo kiekvieno taško iki kažkurių kitų dviejų taškų; atstumai tarp tų dviejų taškų taip pat yra tiksliai išlaikyti. Tuo būdu gali būti tiksliai išlaikomi $(2m-3)$ atstumų (čia m – analizuojamų taškų skaičius).

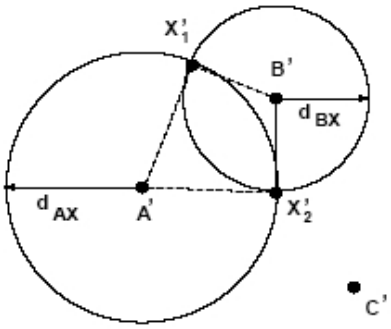
Iš turimų n-mačių taškų sudaromas minimalaus jungimo medis (*angl. minimum spanning tree, MST*) (Graham, 1985), vėliau daugiamačiai taškai atvaizduojami plokštumoje. Minimalaus jungimo medžio idėja paremta grafų teorija. Grafo jungimo medis (*angl. spanning tree*) yra grafas, kuriame visos viršūnės sujungtos į medį. Grafo šakos turi svorius arba ilgius. Medis, kurio svoris yra mažiausias, vadinamas minimalaus jungimo medžiu. Yra daug skirtingų algoritmų minimalaus jungimo medžiui rasti (Kruskal, 1956), (Prim, 1957). Galimi du metodai daugiamačius taškus atvaizduoti plokštumoje naudojantis trianguliacijos metodu: antro arčiausio kaimyno metodas (*angl. the second nearest neighbor approach*) ir atramos taško metodas (*angl. the reference point method*).

Antro arčiausio kaimyno metodas. Tarkime taškas X^j jau atvaizduotas plokštumoje, o taškas X^k tiesiogiai sujungtas su X^j minimalaus jungimo medyje. Reikia atvaizduoti plokštumoje tašką X^k. Taškas X^j turi būti arčiausias taškui X^k iš visų jau atvaizduotų taškų. Sakykime, kad tarp visų atvaizduotų taškų, kurių atstumai iki X^j yra tiksliai išlaikyti, X^i bus taškas, arčiausias iki X^k. Įprastai X^i yra tiesiogiai sujungtas su X^j minimalaus jungimo medyje. Tada naudojant trianguliacijos metodą, X^k plokštumoje atidedamas taip, kad jo atstumai iki X^i ir X^j būtų tiksliai išlaikyti.

Trianguliacijos metodo schema:

- imame du n-mačius taškus A ir B;
- plokštumoje atidedame jų projekcijas A' ir B' tiksliai išlaikydami atstumus tarp jų $d^*_{AB} = d_{A'B'}$;
- ieškome taško X projekcijos X' plokštumoje, siekiant, kad atstumai d^*_{AX} ir d^*_{BX} būtų tiksliai išlaikyti ir plokštumoje, t.y. $d^*_{AX} = d_{A'X'}$ ir $d^*_{BX} = d_{B'X'}$. Tam brėžiami apskritimai su centrais A' ir B' ir spinduliais d^*_{AX} ir d^*_{BX}. Taško X' vieta bus ten, kur apskritimai susikerta arba liečiasi. Jei apskritimai tik liečiasi, tai bus vienintelis sprendinys X'. Tačiau apskritimai gali susikirsti dviejuose

taškuose X_1', X_2' (2.8 pav.). Artimiausias kaimynas (2.8 pav. tai taškas C') nulemia taško X' vietą.



2.8 pav. *Trianguliacijos metodo veikimo schema*

Atramos taško metode parenkamas vienas atramos taškas, iki kurio atstumai nuo visų kitų taškų bus visada išlaikomi. Tuo būdu kiekvienam taškui X^j išlaikomi atstumai iki dviejų taškų: iki taško, kuris tiesiogiai sujungtas su tašku X^j minimalaus jungimo medyje ir iki atramos taško.

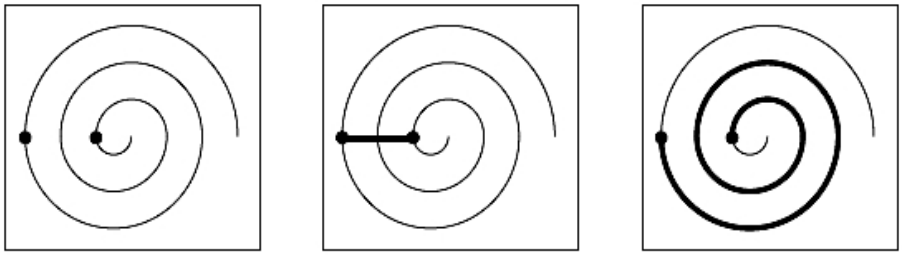
Darbe (Lee, 1977) pasiūlytos taisyklės, pagal kurias taškai surūšiuojami tam tikra tvarka, ir tada jie vienas po kito atvaizduojami plokštumoje.

ISOMAP metodas

ISOMAP (Tenenbaum, 2000) yra daugiamačių skalių (MDS) grupės metodas, kuris skirtas daugiamačių duomenų dimensijos mažinimui. Šio metodo skirtumas nuo klasikinio MDS metodo yra tas, kad atstumų matas yra apibrėžiamas kitaip. ISOMAP metode daroma prielaida, kad pradinėje erdvėje analizuojami duomenys išsidėstę ant žemesnio matavimo daugdaros[1]. Prieš tai analizuotuose metoduose, atstumams tarp taškų skaičiuoti paprastai buvo naudojami Euklido atstumai. Skaičiuojant atstumus neatsižvelgiama į daugdaros formą. Naudojant MDS metodą susiduriama su sunkumais, atvaizduojant netiesines duomenų struktūras, tokias kaip, pavyzdžiui, spiralę. Projektuojant spiralę iš dvimatės erdvės į vienmatę, ISOMAP metodas naudoja geodezinius (kreivinius) atstumus. Geodezinis (arba kreivinis) atstumas – tai trumpiausio kelio ilgis, einant daugdaros kreivu paviršiumi. 2.9 paveiksle pavaizduota, kaip skaičiuojamas Euklido atstumas tarp dviejų taškų (2.9(b) pav.) ir geodezinis atstumas tarp tų pačių taškų (2.9(c) pav.).

[1] Daugdara (*angl. manifold*), tai abstrakti topologinė matematinė erdvė, kurioje kiekvieno taško aplinka yra labai panaši į Euklido erdvę, tačiau jos globali struktūra yra sudėtinga. Pavyzdžiui, Žemės paviršius yra daugdara.

Pagrindinė ISOMAP metodo idėja yra ta, kad daugiamačių skalių metodas atliekamas ne pradinėje (Euklido) erdvėje, bet netiesinės duomenų daugdaros geodezinėje erdvėje. Taigi taikant ISOMAP metodą ieškoma tokio poerdvio, kuriame geriausiai išlaikomi geodeziniai atstumai tarp taškų.



2.9 pav. *(a) du taškai ant spiralės, (b) Euklido atstumas tarp dviejų taškų, (c) Geodezinis atstumas tarp taškų*

ISOMAP algoritmą sudaro trys etapai:

1) Daugiamatėje erdvėje surandami kiekvieno taško taškai „kaimynai“;
2) Skaičiuojami geodeziniai atstumai tarp visų įmanomų taškų porų;
3) Surandamos analizuojamų daugiamačių taškų projekcijos mažesnio matavimo erdvėje, naudojant daugiamačių skalių metodą (MDS) ir išlaikant atstumus tarp atitinkamų taškų.

Taškų „kaimynų“ paieškai gali būti naudojamas artimiausių kaimynų metodas (*angl. nearest neighbors algorithm*) arba taškai gali būti grupuojami pasirinkus tam tikro fiksuoto dydžio spindulį. Kaimynystės ryšių vaizdavimui naudojamas grafas, kuriame kiekvienas duomenų taškas (viršūnė) sujungtas su artimiausiu „kaimynu“.

Antrame žingsnyje apskaičiuojami geodeziniai atstumai tarp visų daugdaros taškų porų. Paskutiniame ISOMAP metodo etape pritaikomas klasikinis MDS metodas atstumų matricai, ir surandamos analizuojamų daugiamačių taškų projekcijos.

Yang (Yang, 2004) pasiūlė vietoj klasikinio MDS metodo naudoti Sammono algoritmą. De Silva ir Tenenbaum (Silva, 2002) sukūrė metodą, kuriame koreguojami grafo atstumai, priklausomai nuo duomenų struktūros ir tankumo.

Lokaliai tiesinis atvaizdavimas

2000-ais metais S. Roweis ir L. Saul pasiūlė (Roweis, 2000) dar vieną daugiamačių duomenų dimensijos mažinimo metodą, **lokaliai tiesinį atvaizdavimą** (*angl. locally linear embedding, LLE*). LLE metodo savybės: surandant analizuojamų daugiamačių duomenų projekcijas mažesnio matavimo erdvėje, išlaikomi kaimynystės

ryšiai tik tarp artimiausių taškų; atskleidžiama netiesinės daugdaros globali struktūra; duomenys vienareikšmiškai atvaizduojami į mažesnės dimensijos erdvę.

LLE metodas suranda tokį poerdvį, kuriame geriausiai išlaikoma analizuojamų duomenų lokalioji tiesinė struktūra.

LLE algoritmą sudaro trys etapai:

1) Randami kiekvieno analizuojamojo duomenų taško *k*-artimiausi kaimynai. Tai daroma, konstruojant orientuotąjį grafą, kurio kiekviena viršūnė sujungta briaunomis su taškais „kaimynais“.
2) Kiekvienas taškas išreiškiamas, kaip jo kaimynų tiesinė kombinacija. Tam tikslui yra apskaičiuojami svoriai, kurie minimizuoja paklaidas tarp originalių ir transformuotų duomenų.
3) Fiksavus svorius, apskaičiuojami projekcijos taškai.

Mažinant dimensiją LLE algoritmu, pasiseka identifikuoti daugdaros neaiškią struktūrą. Tuo tarpu, MDS ir PCA metodais tolimi daugiamačiai taškai ant daugdaros atvaizduojami į artimus taškus plokštumoje, tokiu būdu suardoma daugdaros struktūra. ISOMAP ir LLE algoritmai duoda gerus vizualizavimo rezultatus, kai analizuojami duomenys sudaro vieną klasterį (grupę). Kai duomenų aibė sudaryta iš kelių, visiškai atskirtų klasterių, projekcijos rezultatai būna žymiai blogesni.

MDS grupės metodai stengiasi išlaikyti santykinius atstumus tarp visų duomenų aibės taškų. LLE metodas nereikalauja išlaikyti atstumų tarp labiausiai nutolusių duomenų taškų, o tik tarp artimiausių kaimynų.

Kitas metodas, turintis daug bendro su LLE metodu, yra **Laplaso tikriniai atvaizdavimai** (*angl. Laplacian eigenmaps*). Laplaso tikriniai atvaizdavimai suranda daugiamačių duomenų projekcijas mažesnio matavimo erdvėje, išlaikant daugdaros pagrindines lokalias savybes (Belkin, 2001), (Belkin, 2003). Lokalios savybės yra pagrįstos atstumais tarp artimiausių taškų porų.

2.3. Išvados

Pagrindinis daugiamačių duomenų vizualizavimo tikslas – tai duomenų pavaizdavimas žmogui suprantama ir suvokiama forma. Šiame skyriuje analizuoti daugiamačių duomenų projekcijos metodai. Projekcijos metodų tikslas – pateikti daugiamačius duomenis mažesnės dimensijos erdvėje taip, kad būtų kiek galima tiksliau išlaikyta tam tikra duomenų struktūra, bei palengvinti didelės dimensijos duomenų interpretavimą bei apdorojimą. Yra tiesiniai ir netiesiniai projekcijos metodai. Tiesiniuose metoduose naudojamos tiesinės transformacijos, o netiesiniuose – netiesinės. Populiariausi tiesiniai projekcijos metodai yra: pagrindinių komponenčių analizė, nepriklausomų komponenčių analizė, tiesinė diskriminantinė analizė.

Dažniausiai naudojami netiesiniai projekcijos metodai yra daugiamačių skalių metodas ir jo atskiras atvejis – Sammono projekcija.

Daugiamačių duomenų analizės ir vizualizavimo metodai padeda atvaizduoti žmogui suprantamesne forma sudėtingais ir dažnai nežinomais tarpusavio ryšiais susietus daugiamačius duomenis. Metodų įvairovė yra pakankamai plati, todėl būtina įvairių metodų analizė, norint išsirinkti tinkamiausius konkrečiam atvejui.

3

Dirbtinių neuroninių tinklų taikymas daugiamačiams duomenims vizualizuoti

Tiesioginio sklidimo neuroniniai tinklai plačiai taikomi klasifikavime, klasterizavime, funkcijų aproksimavime, prognozavime, optimizavime. Kita pastaruoju metu plačiai vystoma taikymo sritis – tai daugiamačių duomenų vizualizavimas.

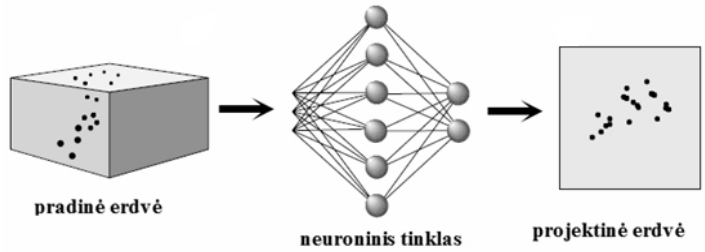
Pastaruoju metu buvo pasiūlytas didelis skaičius dirbtinių neuroninių tinklų ir mokymo algoritmų daugiamačiams duomenims vizualizuoti (Mao, 1995), (Kohonen, 2002), (Kraaijveld, 1995), (Sanger, 1989). Šie tiriamieji darbai gali būti suskirstyti į dvi grupes (nors šis klasifikavimas nėra labai tikslus). Pirmoji grupė, tai nauji arba egzistuojantys neuroninio tinklo modeliai daugiamačių duomenų projekcijai. Šio tipo pavyzdžiai apima saviorganizuojančius neuroninius tinklus (Kohonen, 2001), (Kohonen, 2002), netiesinės projekcijos metodus, netiesinę diskriminantinę analizę, pagrįstą paslėptų neuronų funkcionalumu tiesioginio sklidimo tinklo klasifikatoriuose (Mao, 1995). Šie tinklai atskleidžia įdomias duomenų savybes, kurias „nemato“ klasikiniai metodai. Todėl jie gali būti naudojami kaip nauji įrankiai ar papildomi būdai daugiamačių duomenų projekcijoms rasti. Antroji grupė tyrinėtojų ištyrė neuroninių tinklų ir mokymo taisyklių savybes ir nustatė ryšius tarp klasikinių metodų ir neuroninių tinklų. Buvo pastebėta, kad kai kurie neuroniniai tinklai faktiškai realizuoja duomenų projekcijos ir požymių išskyrimo algoritmus. Keletas klasikinių

duomenų projekcijos metodų buvo realizuoti naudojant neuroninių tinklų struktūras (Baldi, 1989), (Mao, 1995), (Oja, 1991), (Sanger, 1989). Nors tokių bandymų rezultate nebūtinai gaunami nauji duomenų vizualizavimo metodai, tokie tinklai iš tikrųjų yra pranašesni už tradicinius metodus:

a) dauguma mokymo algoritmų ir neuroninių tinklų adaptuojasi prie besikeičiančių sąlygų, vadinasi jie gali būti pritaikyti realioms aplinkoms, kur reikalingos prisitaikymo sistemos; tokie algoritmai lengvai realizuojami;

b) naudojant neuroninius tinklus, galima išvengti trūkumų, būdingų klasikiniams algoritmams.

3.1 paveiksle pavaizduota neuroninių tinklų, naudojamų daugiamačių duomenų dimensijos mažinimui, bendra schema.



3.1 pav. *Neuroninis tinklas daugiamačių duomenų vizualizavimui*

Pastaraisiais metais daug dėmesio yra skiriama neuroninių tinklų taikymo galimybėms daugiamačiams duomenims vizualizuoti. 1991 metais Kramer (Kramer, 1991) pasiūlė netiesinį pagrindinių komponenčių analizės metodą (*angl. autoassociator model*) tiesioginio sklidimo neuroninio tinklo mokymui ir pritaikė metodą daugiamačių duomenų projekcijos radimui. Jin ir kt. (Jin, 2000) naudojo neuroninius tinklus kinų kalbos simbolių atpažinimui. Vieni pirmųjų, nagrinėję neuroninių tinklų taikymą daugiamačiams duomenims vizualizuoti, buvo Oja (Oja, 1991), Usui (Usui, 1991). Taip pat dirbtinių neuroninių tinklų plataus spektro taikymams vizualizavime skirtas ir tyrimas (Dzemyda, Kurasova, Medvedev, 2007).

Yra pasiūlyta daug algoritmų tiesioginio sklidimo neuroninio tinklo, skirto dimensijos mažinimui, mokymui. Saund (Saund, 1989) naudojo trijų sluoksnių neuroninį tinklą su vienu paslėptu sluoksniu. Idrissi ir kt. (Idrissi, 2004) nagrinėjo daugiasluoksnio neuroninio tinklo taikymą daugiamačiams duomenims vizualizuoti, tinklo mokyme naudojant jungtinių gradientų metodą.

Pastaruoju metu stebima tendencija, kad MDS (daugiamačių skalių) tyrimus vykdantys mokslininkai dažnai atsiriboja nuo kitų metodų tyrimų, ar net ignoruoja tuos metodus. Antra vertus, kitų vizualizavimo metodų tyrimuose nėra lyginimų ar sąsajų su MDS tipo metodais. Šiame darbe siekiama praplėsti MDS tipo metodų realizacijas dirbtinių neuroninių tinklų taikymu, tuo būdu stiprinant ryšį tarp skirtingų vizualios duomenų analizės krypčių.

3.1. Dirbtinių neuroninių tinklų koncepcija

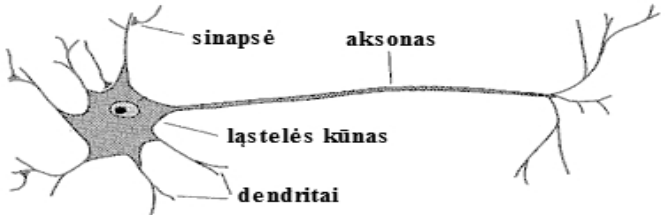
Dirbtiniai neuroniniai tinklai (DNT) (*angl. artificial neural networks*) (Haykin, 1999), (Verikas, 2003) pradėti tyrinėti, kaip biologinių neuroninių sistemų modelis. Pagrindinis dirbtinių neuroninių tinklų teorijos tikslas nėra kuo detaliau modeliuoti biologinius neuronus, o išsiaiškinti ir pritaikyti biologinių neuronų sąveikos mechanizmus efektyvesnėms informacijos apdorojimo sistemoms kurti. Dažnai dirbtiniai neuroniniai tinklai vadinami tiesiog neuroniniais tinklais. Šiuo metu jie vis plačiau naudojami dėl kelių pagrindinių priežasčių. Visų pirma, neuroninis tinklas yra galingas modeliavimo aparatas. Jo pagalba galima modeliuoti ypač sudėtingas funkcijas. Antra, neuroniniai tinklai apmokomi iš pavyzdžių. Turint duomenų pavyzdžius ir naudojant mokymo algoritmus, neuroninis tinklas pritaikomas prie duomenų struktūros. DNT yra taikomi klasifikavimui, klasterizavimui, funkcijų aproksimavimui, prognozavimui, optimizavimui, medicininei diagnozei, finansinėms prognozėms, intelektinei paieškai, ir kt. Be daugelio kitų sprendžiamų uždavinių, dirbtiniai neuroniniai tinklai sėkmingai taikomi daugiamačiams duomenims vizualizuoti.

Kiekvienas neuroninis tinklas turi įėjimus ir išėjimus. Įėjimuose neuroniniam tinklui perduodamos kintamųjų reikšmes iš išorės. Išėjimuose formuojama prognozė, valdymo signalai ir pan. Be įėjimo ir išėjimo neuronų, dažnai egzistuoja ir tarpiniai (taip vadinami paslėpti) neuronai, atliekantys vidinį vaidmenį tinkle. Įėjimo, paslėpti ir išėjimo neuronai jungiami vieni su kitais. Paprastas neuroninis tinklas turi tiesioginio sklidimo struktūrą: signalai sklinda iš įėjimų pirmyn per visus paslėptus elementus ir pasiekia išėjimo neuronus. Tokia struktūra yra stabili. Tačiau, jei neuroninis tinklas yra rekurentinis (turintis jungtis atgal iš tolimesnių į ankstesnius neuronus), jis gali būti nestabilus ir dažniausiai turi be galo sudėtingą dinamiką. Dirbama ir su rekurentiniais neuroniniais tinklais, tačiau realių problemų sprendimui dažniausiai naudojami tiesioginio sklidimo.

Biologinis neuronas

Žmogaus nervų sistema – labai sudėtingas neuronų tinklas. Pagrindinis šios sistemos elementas – smegenys, susideda iš daugelio neuronų, sujungtų vieni su kitais. Kiekvienas *neuronas* yra speciali ląstelė, galinti generuoti elektrocheminį signalą. Neuronas turi išsišakojusią įėjimo struktūrą, vadinamą *dendritais*, ląstelės kūną, vadinamą *soma* ir besišakojančią išėjimo struktūrą, vadinamą *aksonu*. (3.2 pav.). Vienos ląstelės aksonas su kitos ląstelės dendritais jungiasi per *sinapses*. Kai sužadinama pakankamai neuronų, prijungtų prie neurono dendritų, tas neuronas irgi sužadinamas ir jis generuoja elektrocheminį impulsą. Signalas per sinapses perduodamas kitiems neuronams, kurie vėlgi gali būti sužadinami. Neuronas sužadinamas tik tuo atveju, jei bendras dendritais gautas signalas viršija tam tikrą lygį, vadinamą *sužadinimo slenksčiu*. Turint didžiulį skaičių visiškai paprastų elementų,

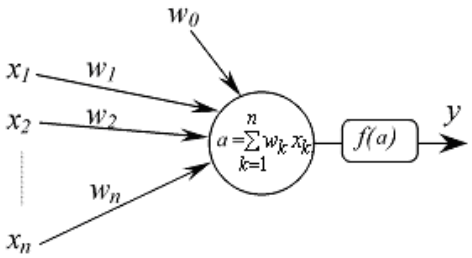
kurių kiekvienas skaičiuoja svorinę įėjimo signalų sumą ir generuoja binarinį signalą, jei suminis signalas viršija tam tikrą lygį, įmanoma atlikti palyginus sudėtingas užduotis (Verikas, 2003).



3.2 pav. *Biologinis neuronas*

Dirbtinio neurono modelis

Dirbtinis neuronas (3.3 pav.) yra svarbiausias neuroninio tinklo elementas. Praėjusio amžiaus ketvirtame dešimtmetyje buvo pasiūlytas dirbtinio neurono modelis. Tai elementarus procesorius, skaičiuojantis įėjimo kintamojo netiesinę funkciją.



3.3 pav. *Dirbtinis neuronas*

Pirmasis formalus dirbtinio neurono apibrėžimas buvo pasiūlytas darbe (McCulloch, 1943):

- Neuronas gauna keletą įėjimo reikšmių. Kiekviena įėjimo jungtis $x_1, x_2, ..., x_n$ (3.3 pav.) turi savo perdavimo koeficientą (svorį) $w_1, w_2, ..., w_n$ ir slenksčio reikšmę w_0.
- Skaičiuojama įėjimo ir svorių sandaugų suma:

$$a = w_1 x_1 + w_2 x_2 + ... + w_n x_n = \sum_{k=1}^{n} w_k x_k \ .$$

Pagal sužadinimo signalą naudojant aktyvacijos funkciją (neurono perdavimo funkciją) $f(a)$ (3.1) skaičiuojama neurono išėjimo reikšmė *y*. Ši reikšmė tampa lygi vienam, jeigu suma viršija slenkstinę reikšmę w_0, nuliui – jeigu neviršija.

$$f(a)=\begin{cases}1, \text{ jeigu } a \geq w_0\\ 0, \text{ jeigu } a < w_0\end{cases}. \qquad (3.1)$$

Neurono išėjimo reikšmę *y* galima užrašyti naudojantis (3.2) formule:

$$y = f\left(\sum_{k=1}^{n} w_k x_k\right). \qquad (3.2)$$

Funkcijos $f(a)$ reikšmės formulėje (3.1) yra susikoncentravusios aplink tašką w_0, bet ta pati išėjimo reikšmė gali būti gauta, jeigu funkcijos $f(a)$ reikšmės būtų sukoncentruotos aplink tašką 0, o svoris w_0, priskirtas įėjimui x_0 ($x_0 = 1$) ir įtrauktas į (3.2) sumą. Tada (3.2) virsta (3.3) formule:

$$y = f\left(\sum_{k=0}^{n} w_k x_k\right). \qquad (3.3)$$

Dirbtinio neurono modelyje gali būti naudojamos ne tik (3.1) funkcija, bet taip pat ir kitokios aktyvacijos funkcijos, pvz., sigmoidinė funkcija $f(a)=\frac{1}{1+e^{-a}}$, hiperbolinis tangentas $f(a)=\tanh(a)=\frac{e^{a}-e^{-a}}{e^{a}+e^{-a}}$ ir kt.

Kaip matyti iš 3.2 ir 3.3 paveikslų, biologinio neurono ir dirbtinio neurono modelio sandara yra gana panaši.

Dirbtiniai neuronai gali būti jungiami į *dirbtinius neuroninius tinklus*. Neuroninis tinklas gali būti pavaizduotas kaip grafas, kurio viršūnės yra neuronai, o šakos (su svoriais) jungia neuronų išėjimus su įėjimais (Jain, 1996). Pagal jungimo konstrukciją neuroniniai tinklai gali būti išskiriami į dvi pagrindines grupes:

- tiesioginio sklidimo (*angl. feedforward*) tinklai, kuriuose nėra grafo kilpų;
- atbulinio sklidimo (*angl. feedback*) (arba rekurentiniai) tinklai, kuriuose yra grafo kilpos.

Dirbtinių neuroninių tinklų mokymas

Galimybė mokytis yra esminė intelekto savybė. DNT mokymo procesas gali būti apibrėžtas kaip tinklo struktūros ir jungčių svorių keitimo uždavinys, siekiant, kad tinklas galėtų atlikti jam skirtą užduotį. Neurono mokymo procese iteratyviai

keičiamos svorių reikšmės, atsižvelgiant į įėjimo ir išėjimo reikšmes, siekiant gauti reikiamą rezultatą.

Skirtingos tinklų architektūros reikalauja skirtingų jų mokymo algoritmų. Yra trys pagrindinės neuronų mokymo paradigmos:

- Mokymo su mokytoju algoritmai (*angl. supervised learning*);
- Mokymo be mokytojo algoritmai (*angl. unsupervised learning*);
- Hibridinis mokymas (*angl. hybrid learning*).

Jeigu žinomos trokštamos išėjimų reikšmės *t*, gali būti taikomi taip vadinami *mokymo su mokytoju algoritmai*. Tokiame mokyme tinklo išėjimų reikšmės *y* kiekvienam įėjimui yra tiesiogiai susijusios su žinomomis trokštamomis to išėjimo reikšmėmis *t*. Tinklas koreguojamas, keičiant svorių vektorių reikšmes, siekiant gauti kiek galima mažesnę paklaidą, t.y. ieškoma tokių svorių, kad skirtumas tarp trokštamų išėjimo reikšmių *t* ir reikšmių *y*, gautų apmokius neuroninį tinklą, būtų kiek galima mažesnis.

Kartais trokštamos tinklo išėjimo reikšmės nėra žinomos. Tada naudojami *mokymo be mokytojo algoritmai.* Juose tinklas mokomas pačiam ieškoti koreliacijų ar panašumų tarp mokymo aibės įėjimų. Čia nėra grįžtamojo ryšio, pasakančio, kuris atsakymas bus arba yra teisingas. Mokymo be mokytojo algoritmuose nėra mokytojo signalo. Čia turima tik mokymo aibė $\{X^j,\ j=1,...,m\}$, susidedanti iš taškų, priklausančių erdvei R^n. Metodų tikslas yra kategorizuoti mokymo duomenis arba rasti juose kokius nors reguliarumus ar ypatumus. Gali būti sprendžiamas ir toks uždavinys: taškus $X^j,\ j=1,...,m,$ reikia atvaizduoti mažesnio matavimo erdvės taškų rinkiniu taip, kad ryšiai, esantys tarp X^j, būtų išlaikyti ir tarp naujos aibės taškų (Hassoun, 1995). Mokymo be mokytojo sėkmė glūdi tame, kad parenkamas nuo mokytojo nepriklausomas kriterijus, mokymo metu tas kriterijus optimizuojamas parenkant tinkamas svorių reikšmes.

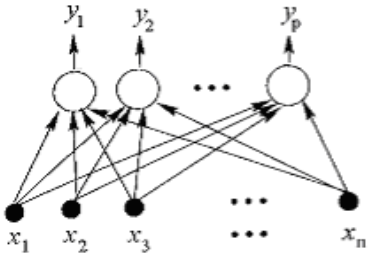
Galimos trys mokymo be mokytojo strategijos, kurios leidžia spręsti atitinkamus trijų tipų duomenų analizės uždavinius:

1. Hebbo tipo mokymas (*angl. Hebbian learning*),
2. varžytinių tipo mokymas (*angl. competitive learning*),
3. saviorganizuojančio žemėlapio (neuroninio tinklo) mokymas (*angl. self-organizing maps*).

Hibridinis mokymas jungia mokymo su mokytoju ir be mokytojo tipus: dalis tinklo svorių nustatomi pagal mokymą su mokytoju, kita dalis gaunama iš mokymo be mokytojo.

Perceptronas, jo mokymas

Paprasčiausios architektūros yra taip vadinami *tiesioginio sklidimo* (*angl. feedforward*) neuroniniai tinklai, kuriuose galimos tik vienkryptės į priekį (*angl. unidirectional forward*) jungtys. Paprasčiausias tokio tipo neuroninis tinklas – *perceptronas*, susidedantis iš vieno sluoksnio *p* neuronų, sujungtų su *n* įėjimais (3.4 pav.) (Silipo, 2003). Neuronų skaičius *p* lygus išėjimų skaičiui. Kai kurie autoriai perceptrone naudoja tik vieną neuroną (Raudys, 2001), tada $p = 1$.



3.4 pav. *Perceptronas*

Perceptrone kiekvienas išėjimas y_i yra įėjimo vektoriaus $X = (x_1, x_2, ..., x_n)$ funkcija, kuri skaičiuojama pagal (3.4) formulę.

$$y_i = f(a_i) = f\left(\sum_{k=0}^{n} w_{ik} x_k \right), \quad i = 1, ..., d \tag{3.4}$$

čia w_{ik} jungties iš *k*-tosios įėjimo vektoriaus komponentės į *i*-tąjį išėjimą svoris.

Tarkime, kad yra *m* mokymo aibės elementų. Įėjimo vektoriai X^j, $j = 1, ..., m$, susieti su trokštamomis reikšmėmis $T^j = (t_1^j, t_2^j, ..., t_d^j)$. Perceptrono mokymo procese svoriai w_{ik} keičiami taip, kad tinklo išėjimo reikšmės $Y^j = (y_1^j, y_2^j, ..., y_d^j)$ kiekvienam įėjimui X^j būtų kiek galima artimesnės trokštamoms reikšmėms T^j, t.y. gautųsi kiek galima mažesnė paklaida. Paklaidos matas $E(W)$ apibrėžiamas kaip svorių matricos $W = \{w_{ik}\}$ funkcija. Jeigu paklaidos funkcija $E(W)$ yra diferencijuojama pagal svorius w_{ik}, jos minimumas gali būti randamas gradientiniais optimizavimo metodais. Geriausiai žinomas yra gradientinio nusileidimo algoritmas. Dažniausiai naudojama paklaidos funkcija yra kvadratinių paklaidų (3.5) suma.

$$E(W) = \frac{1}{2} \sum_{j=1}^{m} \sum_{i=1}^{p} (y_i^j - t_i^j)^2 = \sum_{j=1}^{m} E^j(W), \tag{3.5}$$

$$E^j(W) = \frac{1}{2} \sum_{i=1}^{p} (y_i^j - t_i^j)^2 . \tag{3.6}$$

Pradžioje generuojamos atsitiktinės svorių w_{ik} reikšmės. Tada gradientinio nusileidimo algoritmu judama antigradiento kryptimi, svorių reikšmes keičiant pagal (3.7)–(3.9) iteracines formules.

$$w_{ik}(m'+1) = w_{ik}(m') + \Delta w_{ik}(m'), \tag{3.7}$$

$$\Delta w_{ik}(m') = -h\frac{\partial E(m')}{\partial w_{ik}} = -h\sum_{j=1}^{m}\frac{\partial E^j(m')}{\partial w_{ik}} = \sum_{j=1}^{m}\Delta w_{ik}^j(m'), \tag{3.8}$$

$$\Delta w_{ik}^j(m') = -h\frac{\partial E^j(m')}{\partial w_{ik}}. \tag{3.9}$$

Čia h yra mažas teigiamas skaičius, vadinamas *mokymosi greičiu* (*angl. learning rate*), m' – iteracijos numeris. Galimos dvi svorių keitimo strategijos. Vienu atveju svoriai w_{ik} pakeičiami pagal (3.8) formulę pateikus į tinklą visus mokymo aibės taškus. Kitoje strategijoje svoriai w_{ik} pakeičiami pagal (3.9) formulę po kiekvieno mokymo aibės taško pateikimo į tinklą.

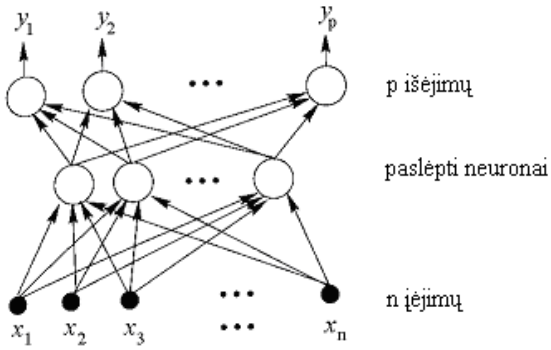
Daugiasluoksniai tiesioginio sklidimo neuroniniai tinklai

Tinklai, turintys daugiau nei vieną neuronų sluoksnį, kuriuose galimi tik ryšiai į priekį iš įėjimų į išėjimus, yra vadinami *daugiasluoksniais perceptronais* (*angl. multilayer perceptrons*) arba *daugiasluoksniais tiesioginio sklidimo neuroniniais tinklais* (*angl. multilayer feedforward neural networks*). Kiekvienas toks tinklas sudarytas iš įėjimų aibės, išėjimų neuronų sluoksnio ir paslėptų neuronų sluoksnių tarp įėjimų ir išėjimų (3.5 pav.).

Tarkime, kad yra daugiasluoksnis neuroninis tinklas, kuriame yra L sluoksnių, $l = 0,1,...,L$, čia sluoksnis $l = 0$ žymi įėjimus, o $l = L$ žymi išėjimus. Kiekviename sluoksnyje l yra n_l neuronų. Kiekvieno i-tojo neurono išėjimų reikšmė y_i l-tajame sluoksnyje gali būti apskaičiuojama pagal (3.4) formulę. Įėjimai x_k į i-tąjį neuroną l-tajame sluoksnyje atitinka išėjimus y_k $(l-1)$-ajame sluoksnyje. Tada (3.4) formulė tampa (3.10) formule.

$$y_i^l = f_i(a_i) = f_i\left(\sum_{k=0}^{n_{l-1}} w_{ik}y_k^{l-1}\right), \quad i = 1,...,n_l \tag{3.10}$$

čia a_i atitinka įėjimus į i-tąjį neuroną, $f_i()$ – jų aktyvacijos funkcija, n_{l-1} – neuronų skaičius $(l-1)$-ajame sluoksnyje.



3.5 pav. *Tiesioginio sklidimo neuroninis tinklas*

„Klaidos sklidimo atgal“ mokymo algoritmas

Taikant vienasluoksnio perceptrono mokymo idėją daugiasluoksniam neuroniniam tinklui, būtina žinoti paslėptų neuronų išėjimų reikšmes. Jei paklaidos ir aktyvacijos funkcijos yra diferencijuojamos, ieškant minimalios paklaidos gali būti naudojama gradientinio nusileidimo strategija. Algoritmas, kuris realizuoja gradientinio nusileidimo mokymo strategiją daugiasluoksniam tiesioginio sklidimo neuroniniam tinklui, vadinamas *„klaidos sklidimo atgal“ algoritmu* (*angl. back-propagation learning algorithm*). Jį sudaro du žingsniai:

- įėjimų reikšmių „sklidimas pirmyn“ iš įėjimų į išėjimų sluoksnį;
- paklaidos „sklidimas atgal“ iš išėjimų į įėjimų sluoksnį.

Tiek perceptrono mokyme, tiek „klaidos sklidimo atgal“ algoritme naudojama mokymo su mokytoju strategija.

Naudosime (3.6) dalinę kvadratinių sumų paklaidą $E^j(W)$, svoriai bus keičiami pagal (3.9) formulę. Pirmame algoritmo žingsnyje įėjimų reikšmėms X^j apskaičiuojamos išėjimų reikšmės Y^j. Įvertinama paklaidos funkcija $E^j(W)$ išėjimų sluoksnyje L. Tuo baigiama „sklidimo pirmyn“ fazė. Jei paklaidos funkcija $E^j(W)$ nelygi nuliui, reikia keisti svorių matricą ΔW. Panašiai, kaip ir vienasluoksniame perceptrone pagal (3.11) formulę keičiami visi svoriai w_{ik} jungčių, kurios jungia k-tąjį neuroną $(l-1)$-ajame sluoksnyje su i-tuoju neuronu (l)-ajame sluoksnyje.

$$\Delta w_{ik}^j = -h\frac{\partial E^j}{\partial w_{ik}}. \qquad (3.11)$$

Dalinės išvestinės išreiškiamos taip:

$$\frac{\partial E^j}{\partial w_{ik}} = \frac{\partial E^j}{\partial a_i^j}\frac{\partial a_i^j}{\partial w_{ik}}. \tag{3.12}$$

Iš (3.10) formulės gauname:

$$\frac{\partial a_i^j}{\partial w_{ik}} = y_k^j. \tag{3.13}$$

Naudojant pažymėjimą

$$\delta_i^j = \frac{\partial E^j}{\partial a_i^j}, \tag{3.14}$$

(3.13) ir (3.14) išraiškas įstačius į (3.12) ir (3.11) gauname:

$$\frac{\partial E^j}{\partial w_{ik}} = \delta_i^j y_k^j \tag{3.15}$$

$$\Delta w_{ik}^j = -h\delta_i^j y_k^j \qquad \begin{array}{l} i \in l-\text{ajam sluoksniui} \\ k \in (l-1)-\text{ajam sluoksniui} \end{array} \tag{3.16}$$

Išėjimų sluoksniui, t.y. kai neuronas $i \in L$-ajam sluoksniui, turime:

$$\delta_i^j = \frac{\partial E^j}{\partial a_i^j} = f'(a_i^j)(y_i^j - t_i^j) \quad i \in \text{išėjimų sluoksniui } L \tag{3.17}$$

Lieka problema, kaip rasti $\frac{\partial E^j}{\partial a_i^j}$ paslėptiems neuronams, t.y. kai $i \in$ l-ajam sluoksniui, kai $l < L$. Naudojant dalines išvestines bendru atveju galima parašyti:

$$\delta_i^j = \frac{\partial E^j}{\partial a_i^j} = \sum_{s=1}^{n_{l+1}} \frac{\partial E^j}{\partial a_s^j}\frac{\partial a_s^j}{\partial a_i^j}, \tag{3.18}$$

čia n_{l+1} žymi neuronų $(l+1)$-ajame sluoksnyje skaičių. Išraiška $\frac{\partial E^j}{\partial a_s^j}$ yra lygi dydžiui δ_s^j, apibrėžtam s-tajam neuronui $(l+1)$-ajame sluoksnyje. Atsižvelgiant į (3.10) formulę gauname $\frac{\partial a_s^j}{\partial a_i^j} = f'(a_i^j)w_{si}$. Tada paslėptiems i-tiesiems neuronams:

$$\delta_i^j = f'(a_i^j)\sum_{s=1}^{n_{l+1}} w_{si}\delta_s^j \quad \begin{matrix} i \in \text{sluoksniui } l < L \\ s \in \text{sluoksniui } l+1 \end{matrix}. \qquad (3.19)$$

Iš pradžių reikia apskaičiuoti δ reikšmes išėjimų sluoksnyje L pagal (3.17) formulę. Tada palaipsniui skaičiuoti δ reikšmes paslėptiems neuronams tarpiniuose sluoksniuose $l < L$ naudojant $(l+1)$-mo sluoksnio δ reikšmes (3.19).

Kai visi svoriai pakeičiami pagal (3.16) formulę, į tinklą pateikiamas sekantis mokymo taškas X^j ir procedūra kartojama vėl. Sustojimo kriterijus gali būti arba iš anksto nustatyta paklaidos funkcijos slenksčio reikšmė, arba atitinkamas atliktų iteracijų skaičius.

Norint pagreitinti mokymo procesą, yra koreguojama *delta* taisyklė (3.16), pridedant tam tikrą konstantą (angl. *momentum term*) (Haykin, 1999):

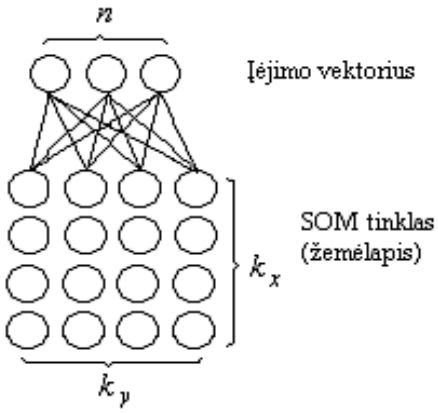
$$\Delta w_{ik}^j(m') = -h\delta_i^j(m')y_k^j(m') + \alpha\Delta w_{ik}^j(m'-1),$$

čia α dažniausiai yra teigiama konstanta $0 \le \alpha \le 1$, ir yra vadinama *momentum* konstanta (arba reikšmė) (*angl. momentum constant*). Kai $\alpha = 0$, gaunama (3.16) formulė. *Momentum* reikšmė naudojama „klaidos sklidimo atgal“ mokymo algoritme, norint išvengti „užstrigimo“ lokaliame minimume ir pagreitinti tinklo konvergavimo procesą. Mokymo procesas tampa labiau stabilus.

3.2. Saviorganizuojantys neuroniniai tinklai

Saviorganizuojantys neuroniniai tinklai (žemėlapiai) (*angl. self-organizing maps* (SOM)) (Kohonen, 2001) dar vadinami Kohoneno neuroniniais tinklais arba Kohoneno saviorganizuojančiais žemėlapiais. Šio tipo neuroninių tinklų pavadinimas kilo iš to, kad saviorganizuojantis žemėlapis, naudodamas mokymo aibę, pats save sukuria (organizuoja). SOM tinklo esmė – duomenų topografijos išlaikymas. Taškai, esantys arti vieni kitų nagrinėjamoje erdvėje, yra atvaizduojami arti vieni kitų ir SOM žemėlapyje. SOM žemėlapiai gali būti naudojami siekiant vizualiai pateikti duomenų klasterius, bei ieškant daugiamačių duomenų projekcijas į mažesnės dimensijos erdvę, įprastai į plokštumą.

Saviorganizuojantis neuroninis tinklas (SOM) yra neuronų $M = \{m_{ij}, i = 1,...,k_x,\ j = 1,...,k_y\}$, išdėstytų dvimačio tinklelio (lentelės) mazguose, masyvas. Dvimačio neuroninio tinklo schema parodyta 3.6 paveiksle. Kiekvienas žemėlapio neuronas sujungtas su kiekvienu įėjimo vektoriumi (3.6 pav., kad jo neperkrauti, pavaizduotos tik pirmos žemėlapio eilutės jungtys su įėjimo vektoriais). Galima stačiakampė arba šešiakampė tinklo struktūra.



3.6 pav. *Dvimačio SOM tinklo schema*

Keturkampės tinklo struktūros atveju k_x yra lentelės eilučių skaičius, k_y – stulpelių skaičius. Kiekvienas *n*-matis mokymo aibės taškas $X \in \{X^1, X^2, ..., X^m\}$ mokymo metu yra susiejamas su vienu tinklo neuronu, kuris yra *n*-matis vektorius. Mokymo pradžioje vektorių M_{ij} komponentės generuojamos atsitiktinai. Kiekviename mokymo žingsnyje vienas iš mokymo aibės taškų $X \in \{X^1, X^2, ..., X^m\}$ pateikiamas į tinklą. Randama, iki kurio neurono M_c taško X^j Euklido atstumas yra mažiausias. Vektorius m_c pavadinamas neuronu-nugalėtoju. Neuronų komponentės keičiamos pagal formulę:

$$m_{ij} \leftarrow m_{ij} + h_{ij}^c (X - m_{ij}),$$

čia, $h_{ij}^c(t)$ yra kaimynystės funkcija (Dzemyda, 2008). Viena iš galimų kaimynystės funkcijos išraiškų yra tokia: $h_{ij}^c = \frac{\alpha}{\alpha \eta_{ij}^c + 1}$, $\alpha = \max\left(\frac{e+1-\hat{e}}{e}; 0{,}01\right)$ (e – mokymo epochų skaičius, $\hat{e}$ – vykdomos epochos numeris, viena mokymo epocha – tai mokymo proceso dalis, kai visus daugiamačius taškus pateikiame tinklui po vieną kartą, ją sudaro *m* mokymo žingsnių) (Dzemyda, 2001). Dydis η_{ij}^c yra kaimynystės tarp neuronų m_c ir m_{ij} eilė. Greta neurono-nugalėtojo esantys neuronai vadinami pirmos eilės kaimynai, greta pirmos eilės kaimynų esantys neuronai, išskyrus jau paminėtus – antros eilės kaimynai ir t.t. Kiekvienos epochos metu perskaičiuojami tie neuronai m_{ij}, kuriems $\eta_{ij}^c \le \max[\alpha \max(k_x, k_y), 1]$.

Vienas iš tinklo mokymosi kokybės įvertinimo kriterijų yra kvantavimo paklaida (*angl. quantization error*) $E_{SOM} = \frac{1}{m}\sum_{j=1}^{m} \| X_j - m_{c(j)} \|$. Tai vidutinis atstumas tarp kiekvieno *n*-mačio duomenų taško $X^j = (x_1^j, x_2^j, ..., x_n^j)$ ir jo neurono-nugalėtojo $m_{c(j)}$, *m* – analizuojamų daugiamačių taškų skaičius.

3.3. Radialinių bazinių funkcijų neuroniniai tinklai

Radialinių bazinių funkcijų (*angl. radial basis function, RBF*) neuroniniai tinklai (Powell, 1987), (Broomhead, 1988), (Moody, 1989), (Haykin, 1999) sprendžia panašius uždavinius kaip ir daugiasluoksnis perceptronas. Radialinių bazinių funkcijų neuroniniai tinklai paslėptajame sluoksnyje naudoja radialines bazines funkcijas.

Jei $X^j = (x_1^j, ..., x_n^j)$, $j = 1, ..., m$ yra įėjimo taškai, $Y^j = (y_1^j, ..., y_d^j)$, $j = 1, ..., m$ yra tinklo išėjimo taškai (Y^j yra tinklo reakcija į X^j), $t^j = (t_1^j, ..., t_d^j)$ yra trokštama tinklo reakcija į duomenų taškus X^j, $j = 1, ..., m$. Nebūtinai $t^j = Y^j$ baigus tinklo mokymą. Tikslas – apmokyti tinklą kuo geriau reaguoti į įėjimo taškus.

Radialinė bazinė funkcija ϕ, tai funkcija, kurios reikšmė priklauso tik nuo atstumų nuo tam tikro pradžios taško, todėl $\phi(x) = \phi(\|x\|)$, arba $\phi(x, c) = \phi(\|x - c\|)$. Naudojant radialines bazines funkcijas paprastai sprendžiamas tam tikros *n* kintamųjų funkcijos $Y = (y_1(X), ..., y_d(X))$, $X \in R^n$, $Y \in R^d$, aproksimavimo pagal jos stebėjimo *m* argumentų taškuose X^j, $j = 1, ..., m$ rezultatus $t^j = (t_1^j, ..., t_d^j)$, $j = 1, ..., m$, uždavinys. Funkcija aproksimuojama svorine, taip vadinamų radialinių bazinių funkcijų, suma.

Radialinių bazinių funkcijų metodo taikymo pradžia siejama su tikslia duomenų imties interpoliacija daugiamatėje erdvėje. Panaudojant ir išvystant tikslaus interpoliavimo idėjas, gaunamas RBF neuroninis tinklas. Tinklo pagalba sudaroma interpoliuojanti funkcija, kurioje bazinių funkcijų skaičių apsprendžia ne mokymo imties dydis, bet atvaizduojamos funkcijos sudėtingumas.

Realizacijos ypatumai:

1) Radialinių bazinių funkcijų skaičius *M* parenkamas daug mažesnis už duomenų taškų skaičių *m*.
2) Bazinių funkcijų centrų radimas – mokymo proceso dalis.
3) Kiekviena bazinė funkcija gali turėti savus specifinius parametrus (Gausinės bazinės funkcijos atveju, tai plotis σ), kurie taip pat nustatomi mokymo metu.

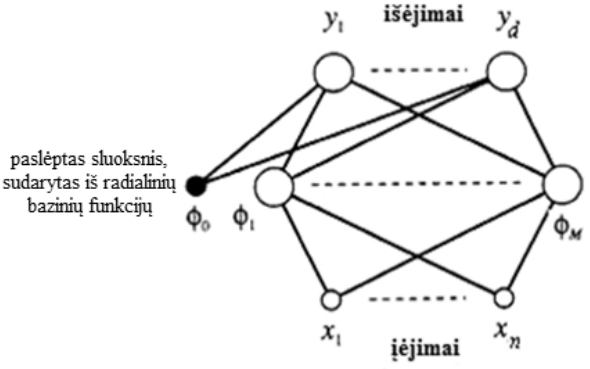
4) Į svorinę bazinių funkcijų sumą įtraukiami laisvieji nariai. Jie kompensuoja bazinių funkcijų skaičiaus sumažinimą.

Tuo būdu gaunama tokia RBF neuroninio tinklo perdavimo funkcija $y_k(X) = \sum_{j=1}^{M} w_{kj}\phi_j(X) + w_{k0}$.

Laisvasis narys w_{k0} gali būti įtrauktas į sumą pridedant papildomą bazinę funkciją $\phi_0 = 1$:

$$y_k(X) = \sum_{j=0}^{M} w_{kj}\phi_j(X). \qquad (3.20)$$

RBF neuroninis tinklas, atitinkantis (3.20) funkciją parodytas 3.7 paveiskle.



3.7 pav. *RBF neuroninis tinklas*

Gausinės funkcijos atveju:

$$\phi_j(X) = \exp\left(-\frac{\| X - \mu_j \|^2}{2\sigma_j^2}\right),$$

čia μ_j yra radialinės bazinės funkcijos ϕ_j centro taškas ($\mu_j \in R^n$); $\| X - \mu_j \|$ – atstumas tarp taškų X ir μ_j.

Jeigu, pavyzdžiui, yra žinomas analizuojamų duomenų klasterių skaičius arba jį galima įvertinti, tai radialinių bazinių funkcijų skaičius gali būti lygus klasterių skaičiui, o μ_j – atitinkamo klasterio svorio centras.

RBF neuroninių tinklų skirtumai nuo daugiasluoksnio perceptrono:

- Visi daugiasluoksnio perceptrono parametrai paprastai nustatomi vienu metu, naudojant globalinį mokymą su mokytoju. Radialinių bazinių funkcijų

neuroninis tinklas mokomas dviem etapais, pirmiausia mokymo be mokytojo metodais nustatant bazinių funkcijų parametrus, tada antrojo sluoksnio svoriai nustatomi naudojant greitas tiesines mokymo su mokytoju procedūras.

- Daugiasluoksnis perceptronas gali turėti daugelį svorių sluoksnių ir norimą sujungimų tarp sluoksnių būdą, nebūtinai naudojant visus įmanomus svorius. Tame pačiame tinkle gali būti naudojamos įvairios perdavimo funkcijos. Radialinių bazinių funkcijų neuroniniai tinklai dažniausiai turi paprastą architektūrą ir susideda paprastai iš dviejų sluoksnių. Pirmajame talpinami bazinių funkcijų parametrai, antrajame formuojamos tiesinės bazinių funkcijų kombinacijos.

3.4. SAMANN metodas

Sammono projekcija (Sammon, 1969) (2.2 skyrius) yra netiesinis daugelio kintamųjų objektų atvaizdavimo žemesnio matavimo erdvėje metodas. Sammono metodas minimizuoja projekcijos (Sammono) paklaidą E_S:

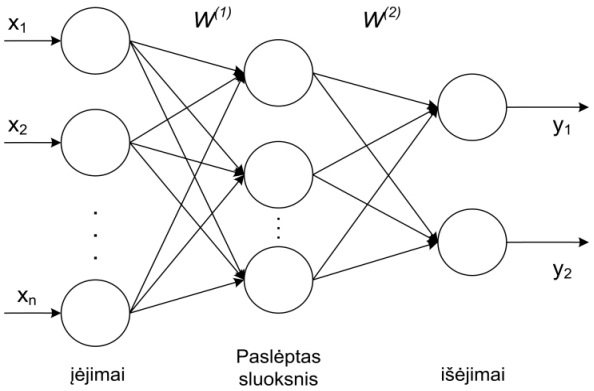
$$E_S = \frac{1}{\sum_{\mu=1}^{m-1}\sum_{\nu=\mu+1}^{m} d^*_{\mu\nu}} \sum_{\mu=1}^{m-1}\sum_{\nu=\mu+1}^{m} \frac{[d^*_{\mu\nu} - d_{\mu\nu}]^2}{d^*_{\mu\nu}}, \qquad (3.21)$$

čia $d^*_{\mu\nu}$ – atstumas tarp *n*-mačių taškų X^μ ir X^ν, $d_{\mu\nu}$ – atstumas tarp *d*-mačių taškų Y^μ ir Y^ν, į kuriuos projektuojami taškai X^μ ir X^ν ($d < n$). Analizuojamų taškų skaičius lygus *m*.

Sammono algoritmo vienas iš trūkumų yra tas, kad atsiradus analizuojamų taškų aibėje naujam taškui, norint ir jį atvaizduoti projektinėje erdvėje, reikia perskaičiuoti visų jau atvaizduotų taškų projekcijas. Mao ir Jain (Mao, 1995) pasiūlė Sammono paklaidą minimizuoti naudojant tiesioginio sklidimo neuroninius tinklus. Pasiūlyta specifinė „klaidos sklidimo atgal“ mokymo taisyklė, pavadinta SAMANN (Sammono algoritmas + dirbtiniai neuroniniai tinklai (*angl. Artificial neural networks, ANN*)), kuri leidžia įprastam tiesioginio sklidimo neuroniniam tinklui (3.8 pav.) realizuoti Sammono projekciją mokymo be mokytojo būdu. 3.8 paveiksle pavaizduotas neuroninio tinklo atskiras atvejis, turintis du išėjimus, ir skirtas analizuojamų duomenų projekcijos radimui plokštumoje. Detalesnė informacija apie SAMANN algoritmą pateikta 4 skyriuje.

Atvaizduojant daugiamačius duomenis į mažesnio matavimo erdvę, labiausiai domina keli šio netiesinio atvaizdavimo aspektai: atstumai tarp atitinkamų taškų turi būti maksimaliai išlaikyti, pereinant į mažesnio matavimo erdvę (t.y. turi būti kuo

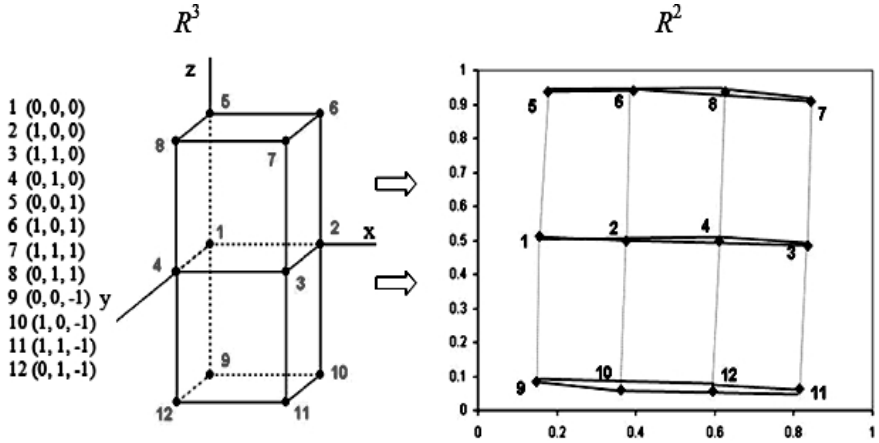
tikslesnė daugiamačių duomenų projekcija plokštumoje); turi būti išlaikyta pradinės duomenų aibės struktūra (t.y. visi taškai turi būti atvaizduoti atitinkamai į „savo“ vietas); galimybė vizualizuoti naujus, „nematytus“ taškus.



3.8 pav. *Tiesioginio sklidimo dviejų sluoksnių dirbtinis neuroninis tinklas Sammono projekcijai*

Kaip alternatyva SAMANN mokymo be mokytojo taisyklei, gali būti naudojamas tiesioginio sklidimo neuroninio tinklo mokymas su mokytoju standartiniu „klaidos sklidimo atgal“ algoritmu. Tarkime, kad yra *n*-mačiai taškai X^j, $j=1,...,m$. Šie taškai Sammono algoritmu atvaizduojami plokštumoje, t.y. gaunamos taškų X^j projekcijos Y^j. Vėliau taškai X^j, $j=1,...,m$, pateikiami į tiesioginio sklidimo neuroninį tinklą kaip įėjimai, trokštamos išėjimų reikšmės imamos taškus X^j atitinkančių taškų Y^j komponentės; tinklas apmokomas standartiniu „klaidos sklidimo atgal“ algoritmu. Darbe (Ridder, 1997) SAMANN algoritmas palygintas su dviem daugiamačių duomenų projekcijos metodais paremtais Sammono algoritmu: trianguliacija ir neuroniniu tinklu, apmokomu standartiniu „klaidos sklidimo atgal“ algoritmu. Darbe (Ridder, 1997) kaip tiesioginio sklidimo neuroninio tinklo mokymo su mokytoju trūkumas atžymėta tai, kad procesas reikalauja didesnių programinių resursų, nes susideda iš dviejų etapų: duomenų mokymui suformavimas Sammono algoritmu ir po to sekantis tinklo mokymas tais duomenimis.

3.9 paveiksle pavaizduota trimačio kubo projekcija plokštumoje. Iš paveikslo matome, kad SAMANN metodas gerai išlaiko atstumus tarp atitinkamų taškų, projektuojant daugiamačius duomenis į plokštumą, ir išlaiko pradinės duomenų aibės struktūrą. Kitos SAMANN tinklo galimybės bei metodo analizė aprašomi 4 ir 5 skyriuose.



3.9 pav. *SAMANN algoritmu gauta dviejų kubų projekcija plokštumoje*

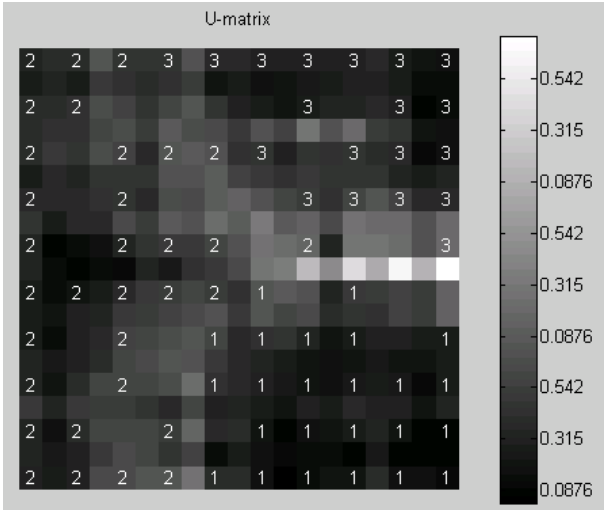
3.5. SOM tinklo taikymas daugiamačiams duomenims vizualizuoti

Saviorganizuojantis neuroninis tinklas yra tinkamas daugiamačių duomenų vizualizavimo įrankis, kuris ne tik gali daugiamačius duomenis atvaizduoti plokštumoje, bet prieš tai juos klasterizuoja. Tokiu būdu tiksliau atskleidžiama duomenų struktūra. Darbe (Flexer, 2001) parodyta, kad SOM tinklas sėkmingai naudojamas ir klasterizavime, ir vizualizavime.

Po SOM tinklo mokymo, analizuojami taškai (mokymo aibės ar kitų duomenų) pateikiami į tinklą. Kiekvienam taškui randamas neuronas-nugalėtojas. Taškų pavadinimai (numeriai ar klasės pavadinimai) užrašomi tuose žemėlapio (lentelės) langeliuose, kuriuos atitinka jų neuronai-nugalėtojai, t.y. taškai išsidėsto tarp žemėlapio elementų. Tai galima laikyti kaip *n*-mačių taškų išsidėstymą plokštumoje, t.y. SOM tinkle daugiamačiai duomenys transformuojami į tam tikrą diskrečią struktūrą. Taškų vietą plokštumoje nusako tinklelio mazgai, t.y. eilučių ir stulpelių numeriai. Paprasčiausiu atveju gaunama lentelė (stačiakampės tinklo topologijos atveju), kurios langeliuose surašyti analizuojamų taškų pavadinimai. Tačiau tokia lentelė nėra labai informatyvi, sunku pasakyti, kaip toli yra taškai, esantys gretimuose lentelės langeliuose. Todėl būtina ieškoti metodų, kaip pagerinti tokio vaizdo kokybę.

Unifikuota atstumų matrica (*U-matrica*) (*angl. unified distance matrix*) yra vienas iš populiaresnių SOM tinklo vizualizavimo metodų. U-matricą sudaro atstumai tarp kaimyninių SOM neuronų. Pvz. turint [1x5] tinklą $[m_1, m_2, ..., m_5]$ (1 eilutė, 5 stulpeliai), U-matrica bus vienos eilutės ir devynių stulpelių vektorius

$[u_1, u_{12}, u_2, u_{23}, u_3, u_{34}, u_4, u_{45}, u_5]$. Čia $u_{ij} = \| m_i - m_j \|$ yra atstumas tarp dviejų kaimyninių neuronų, o u_i yra specialiai apibrėžta reikšmė, pvz. vidutinis atstumas tarp kaimyninių neuronų: $u_3 = \frac{u_{23} + u_{34}}{2}$. Radus U-matricą, jos reikšmes reikia pateikti SOM tinkle. Darbuose (Ultsch, 1990), (Kraaijveld, 1995) pasiūlytas metodas, pagal kurį vidutiniai atstumai tarp kaimyninių neuronų yra pateikiami pilkos skalės atspalviais (vėliau imta naudoti ir kitų spalvų skales). Jei vidutiniai atstumai tarp kaimyninių neuronų yra maži, tuos neuronus atitinkantys tinklo langeliai spalvinami šviesia spalva; tamsi spalva reiškia didelius atstumus. Taigi klasteriai yra nustatomi pagal šviesius atspalvius, o ribos – pagal tamsesnius (Kohonen, 2001), (Kohonen, 2002). 3.10 paveiksle, iliustruojančiame U-matricą, matyti, kaip atsiskiria vynų klasės.



3.10 pav. *U-matrica (analizuoti vyno duomenys)*

Dažniausiai yra naudojama dvimatė U-matrica, tačiau darbe (Takatsuka, 2001) siūloma naudoti trimatį SOM tinklo vizualizavimą, kuris padeda geriau atskirti susidariusius klasterius.

Kiekviena neurono komponentė atitinka skirtingas įėjimo taško komponentes. Naudojant U-matricos vizualizavimo ideologiją galima atvaizduoti atskiras komponentes. Tokiu būdu gaunamos taip vadinamos *komponenčių plokštumos* (*angl. component planes*). Iš jų galima matyti, kokią įtaką daro viena ar kita komponentė.

SOM tinklo vizualizavimui gali būti naudojamos *histogramos*. Histograma parodo, kiek taškų priklauso klasteriui, apibrėžtam vienu neuronu.

Paminėtųjų SOM tinklo vizualizavimo metodų trūkumas tas, kad ne visada aiškiai atskleidžiama analizuojamų duomenų struktūra. Kartais gautų rezultatų interpretavimas yra sudėtingas uždavinys. Todėl kyla poreikis SOM tinklo vizualizavimui ieškoti geresnio būdo. Kadangi SOM neuronai yra daugiamačiai taškai, juos galima projektuoti į plokštumą naudojant kurį nors daugiamačių taškų projekcijos metodą, pvz., Sammono projekciją (Sammon, 1969). Kaip parodyta darbe (Kurasova, 2005), SOM tinklo jungimas su Sammono projekcija yra vienas iš efektyvesnių daugiamačių duomenų vizualizavimo būdų.

3.6. Hebbo mokymas ir jo taikymas pagrindinių komponenčių radimui

Kaip jau buvo parodyta, pagrindinės komponentės gali būti naudojamos daugiamačiams duomenims vizualizuoti. Darbe (Kvedaras, 1974) pateikti pagrindinių komponenčių radimo skaitiniai metodai. Dirbtiniai neuroniniai tinklai, apmokomi Hebbo taisykle, taip pat gali būti naudojami ieškant pagrindinių komponenčių.

Viena iš dirbtinių neuroninių tinklų mokymo be mokytojo taisyklių, paremta Hebbo teorija, yra vadinama *Hebbo mokymo taisykle* (*angl. Hebbian learning*). Kiekvienas įėjimo vektorius siejamas su vienu išėjimo vektoriumi. Šiuo mokymu siekiama, kad kuo dažniau įėjimo aibė pateikiama duotajam neuronui, tuo atitinkamas atsakymas yra stipresnis. Sinapsių koeficientai (svoriai *W*) kinta proporcingai koreliacijai tarp prieš-sinapsinio *X* ir po-sinapsinio *Y* signalų.

Hebbo mokymo idėja: tarkime vektorius $X^j = (x_1, x_2, ..., x_n)$ yra neuroninio tinklo įėjime, tada išėjimo vektorius yra $Y^j = (y_1, y_2, ..., y_d)$; kai į tinklą pateikiamas vektorius $X^j + \varepsilon$ artimas vektoriui X^j, išėjimo vektorius turėtų būti $Y^j + \delta$ artimas vektoriui Y^j. Vektoriai X^j, $j = 1,...,m$, priklauso erdvei R^n, o vektoriai Y^j, $j = 1,...,m$, priklausys erdvei R^d ($d < n$). Būtent dėl šios priežasties Hebbo mokymo taisyklė gali būti taikoma daugiamačių duomenų dimensijai sumažinti, o kai $d = 2$ ir jų atvaizdavimui plokštumoje.

Pradžioje analizuokime vieno neurono atvejį. Neurono išėjimo reikšmė apskaičiuojama taip: $y^j = \sum_{k=1}^{n} x_k^j w_k = (X^j)^T W$, čia X^j – įėjimų vektorius-stulpelis, $(X^j)^T$ – vektorius-eilutė, tai transponuotas įėjimo vektorius-stulpelis X^j, W – svorių vektorius-stulpelis, y^j – išėjimas. Tada mokymo (svorių keitimo) taisyklė:

$$W(m'+1) = W(m') + hy^j X^j \text{ arba}$$

$$\Delta W = hy^j X^j . \tag{3.22}$$

čia h – mokymo greitis, m' – iteracijos numeris. Apskaičiuojame (3.22) formulės vidurkį pagal visus įėjimų vektorius, gauname:

$$\langle \Delta W \rangle = h\langle yX \rangle = h\langle XX^T W \rangle . \tag{3.23}$$

(3.23) formulėje ženklu $\langle \cdot \rangle$ žymimas vidurkis, $\langle yX \rangle = \frac{1}{m}\sum_{j=1}^{m} y^j X^j = \frac{1}{m}\sum_{j=1}^{m} (X^j)(X^j)^T (W^j) = \langle XX^T W \rangle$.

Kadangi X ir W nepriklausomi dydžiai, (4.3) formulė gali būti parašyta taip:

$$\langle \Delta W \rangle = h\langle yX \rangle = h\langle XX^T \rangle \langle W \rangle , \tag{3.24}$$

$C = \langle XX^T \rangle$ yra autokoreliacinė matrica.

Matrica C yra simetrinė matrica, pagrindinėje diagonalėje yra įėjimų komponenčių kvadratų vidurkiai. Jos nuosavų ortogonalių vektorių e_k nuosavos reikšmės λ_k, $k = 1,...,n$ yra teigiamos arba lygios nuliui ir yra lygties $Ce_k = \lambda_k e_k$ sprendiniai (Hassoun, 1995).

$$C = \begin{pmatrix} \langle x_1^2 \rangle & \langle x_1 x_2 \rangle & \dots & \langle x_1 x_n \rangle \\ \langle x_2 x_1 \rangle & \langle x_2^2 \rangle & \dots & \langle x_2 x_n \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle x_n x_1 \rangle & \langle x_n x_2 \rangle & \dots & \langle x_n^2 \rangle \end{pmatrix} \tag{3.25}$$

Normuota Hebbo mokymo taisyklė:

$$\begin{cases} W(1) \ \text{ pasirenkame laisvai} \\ W(m'+1) = \dfrac{W(m') + hy^j X^j}{\| W(m') + hy^j X^j \|} \end{cases} . \tag{3.26}$$

Naudojant vidutines svorių keitimo reikšmes, (3.26) formulę galima pakeisti taip: $\langle W(m'+1) \rangle = \dfrac{\langle W(m') \rangle + hC\langle W(m') \rangle}{\| \langle W(m') \rangle + hC\langle W(m') \rangle \|}$. Ši formulė, esant didelei dydžio h reikšmei, gali būti užrašyta: $\langle W(m'+1) \rangle = \dfrac{C\langle W(m') \rangle}{\| C\langle W(m') \rangle \|}$. O tai yra klasikinis metodas (*angl.*

power method) simetrinės matricos B maksimalią nuosavą reikšmę atitinkančiam nuosavam vektoriui rasti: $e_1(m'+1) = \frac{Be_1(m')}{\| Be_1(m') \|}$. Šiuo klasikiniu metodu randama ir didžiausia nuosava reikšmė λ_1. Norint rasti kitas nuosavas reikšmes naudojamas taip vadinamas „išsiurbimo“ metodas (*angl. deflation method*). Jame išanalizuojamos matricos atimama matrica $e_1 e_1^T$, padauginta iš λ_1: $B' = B - \lambda_1 e_1 e_1^T$. Tuo pačiu metodu randama matricos B' didžiausia nuosava reikšmė, o tai bus antra pagal dydį matricos B nuosava reikšmė. Tokiu pat būdu randamos ir kitos nuosavos reikšmės (Hassoun, 1995).

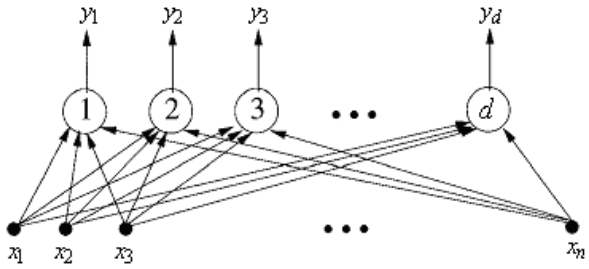
Modifikuota Hebbo taisyklė yra vadinama Oja taisykle:

$$\begin{cases} W(1) \quad \text{pasirenkame laisvai} \\ W(m'+1) = W(m') + hy^j X^j - h(y^j)^2 W(m'). \\ \qquad\qquad = W(m') + h(X^j - y^j W(m'))y^j \end{cases} \tag{3.27}$$

Taigi, naudojant vieną neuroną, apmokomą Hebbo taisykle, galime rasti pirmąją pagrindinę komponentę. Norint rasti d pirmųjų pagrindinių komponenčių, galimi du būdai, kuriuos ir aptarsime.

Sakykime, kad yra taškai $X^1, X^2, ..., X^m \in R^n$; sprendžiamas uždavinys – gauti jų projekciją erdvėje R^d, t.y. taškus $Y^1, Y^2, ..., Y^m \in R^d$, $d < n$. Pagrindinių komponenčių analizės (PCA) idėja – rasti sistemą, sudarytą iš d ortogonalių vektorių ir transformuoti vektorių $X = (x_1, x_2, ..., x_n)$ į vektorių $Y = (y_1, y_2, ..., y_d)$ siekiant išlaikyti duomenų struktūrą (detaliau apie PCA skaitykite 2.1 skyriuje).

Norint rasti d pirmųjų pagrindinių komponenčių galima naudoti vieno sluoksnio d neuronų tinklą (3.11 pav.) ir atitinkamai modifikuoti Hebbo mokymo taisyklę.



3.11 pav. *Įėjimo vektoriaus projekcija į d pirmąsias pagrindines komponentes*

Sakykime, kad yra tinklas, sudarytas iš d neuronų. $W_i = (w_{i1}, w_{i2}, ..., w_{in})$ bus jungčių į i-tąjį neuroną svorių vektorius. Y. H. Oja išplėtė savo (3.27) mokymo taisyklę neuroniniam tinklui, sudarytam iš d neuronų (3.11 pav.). Svorių keitimo taisyklė:

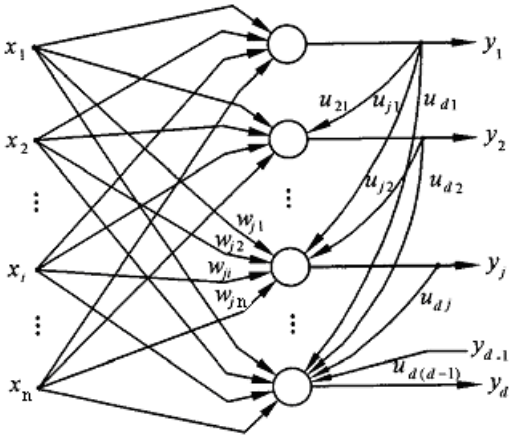
$$\Delta w_{ij} = \eta \left(x_j - \sum_{k=1}^{d} w_{kj} y_k \right) y_i . \qquad (3.28)$$

Kitą panašią taisyklę pasiūlė T. D. Sanger (Sanger, 1989):

$$\Delta w_{ij} = \eta \left(x_j - \sum_{k=1}^{i} w_{kj} y_k \right) y_i . \qquad (3.29)$$

Abi formulės (3.28) ir (3.29) yra identiškos Oja (3.27) formulei, kai $d = 1$. Kai $d > 1$, jos tarpusavyje skiriasi tik viršutine sumavimo riba. Mokymo pabaigoje gaunami svorių vektoriai W_i yra ortogonalūs. Kartais Oja taisyklė gali nerasti matricos C nuosavų vektorių krypčių. Tačiau tuo atveju, d svorių vektoriai konverguoja į tokį pat poerdvį kaip ir matricos C d pirmųjų nuosavų vektorių. Čia svorių vektoriai priklauso nuo pradinių sąlygų ir nuo duomenų pateikimo į tinklą eilės. Sanger taisyklė yra nejautri pradinėms sąlygoms ir duomenų pateikimo į tinklą tvarkai. Svoriai W_i konverguoja į nuosavus vektorius $\pm e_i$, pirmo neurono ($i = 1$) svorių rinkinys lygus didžiausią nuosavą reikšmę atitinkančiam nuosavam vektoriui, $W_1 = \pm e_1$.

Kitame pagrindinių komponenčių radimo metode naudojamas vienasluoksnis neuroninis tinklas, sudarytas iš d neuronų, su pagalbinėmis jungtimis tarp neuronų (3.12 pav.).



3.12 pav. *PCA neuroninis tinklas su pagalbinėmis jungtimis*

3.7. Kreivinių komponenčių analizė

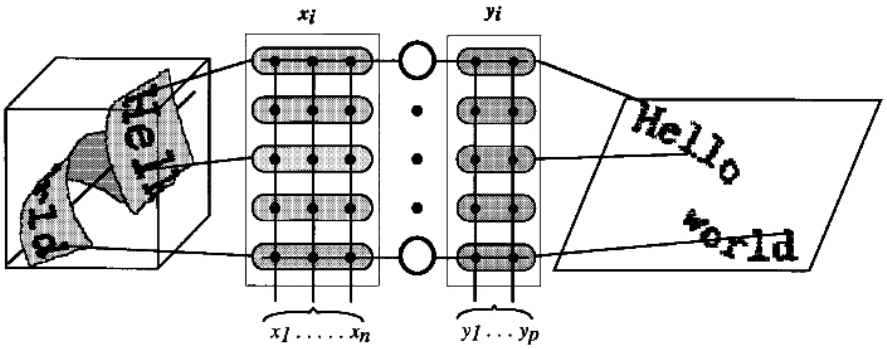
Saviorganizuojantys neuroniniai tinklai yra daugiamačių duomenų vizualizavimo įrankis, kuris daugiamačius duomenis atvaizduoja plokštumoje ir atskleidžia duomenų struktūrą. SOM tinklas transformuoja turimus duomenis į tam tikrą fiksuotą topologinę struktūrą. Tačiau, jeigu ši struktūra nesutampa su vidine turimų duomenų struktūra, tai gautas atvaizdavimas nebus tikslus. Kreivinių komponenčių analizės algoritmas (*angl. curvilinear component analysis*, *CCA*) buvo pasiūlytas tam, kad pradinių duomenų atvaizdavimas būtų patikimesnis (Demartines, 1997). CCA naudoja visiškai naują paklaidos funkciją *E* (3.30). Algoritmas išskaido duomenų daugdarą ir suranda tų duomenų projekciją mažesnio matavimo erdvėje.

CCA algoritmas yra Kohoneno saviorganizuojančių neuroninių tinklų (SOM) patobulinimas. Galutinė atvaizduojamųjų duomenų struktūra nėra iš anksto fiksuota, bet yra tolydi erdvė, kurios forma priklauso nuo pradinių duomenų daugdaros. Algoritmą sudaro du atskiri žingsniai:

1) pradinių duomenų taškų kvantavimas į *K* kvantuotų pradinės erdvės poerdvių;

2) gautų kvantuotų taškų netiesinė projekcija.

3.13 paveiksle pavaizduota šį algoritmą realizuojančio tinklo struktūra ir atvaizdavimo iš trimatės erdvės į dvimatę pavyzdys. Iš pradžių analizuojamos atvaizduojamųjų duomenų struktūra ir pradinė projekcija, o vėliau duomenys yra projektuojami, atsižvelgiant į sukonstruotą atvaizdavimą tarp pradinės erdvės ir projektinės erdvės. Po mokymo tinklas turi galimybę projektuoti bet kurį kitą naują daugiamatį tašką.



3.13 pav. *CCA algoritmo tinklo struktūra*

Tinklo pirmajame sluoksnyje atliekamas pradinių duomenų taškų kvantavimas, naudojant bet kurį taškų kvantavimo metodą (Ahalt, 1990). Išėjimo sluoksnis konstruoja netiesinį atvaizdavimą, siekiant minimizuoti struktūros skirtumus tarp

kvantavimo erdvės ir projektinės erdvės. Tam yra naudojama tikslo funkcija (Demartines, 1997):

$$E = \frac{1}{2}\sum_{i}^{K}\sum_{j\neq i}^{K}(d_{ij}^{*} - d_{ij})^2 F(d_{ij}, \lambda_y), \qquad (3.30)$$

čia d_{ij}^{*} atstumas tarp daugiamačių taškų X^i ir X^j, d_{ij} – atstumas tarp taškus X^i ir X^j atitinkančių dvimačių taškų Y^i ir Y^j $(i, j = 1,..,m)$. $F(d_{ij}, \lambda_y)$ yra svorio funkcija, nuo kurios priklauso lokalios topologijos išlaikymas. Jos reikšmės yra intervale (0; 1). Svorio funkcija yra aprėžta, monotoniškai mažėjanti ir priklauso tik nuo atitinkamų atstumų tarp taškų projektinėje erdvėje. Būtent tuo CCA metodas skiriasi nuo tradicinių MDS metodų.

CCA metodo idėja yra ta, kad dėmesys koncentruojamas į taškus, kurie mažiausiai nutolę vienas nuo kito. Paklaidos funkcija, atsižvelgiant į projektinės erdvės taškus, minimizuojama naudojant tokią procedūrą:

$$\Delta y_j = \alpha(t) F(d_{ij}, \lambda_y)(d_{ij}^{*} - d_{ij})\frac{y_j - y_i}{d_{ij}}, \forall j \neq i,$$

$\alpha(t) = \frac{\alpha_0}{1+t}$ yra mokymosi parametras, priklausantis nuo laiko. Mokymo proceso metu $\alpha(t)$ mažėja.

Lyginant su klasikiniu PCA metodu, CCA metodo privalumas yra tas, kad duomenis, kurie yra pasiskirstę ne pagal normalųjį skirstinį, gali būti atvaizduoti labai tiksliai, naudojant netiesinį atvaizdavimą.

3.8. NeuroScale metodas

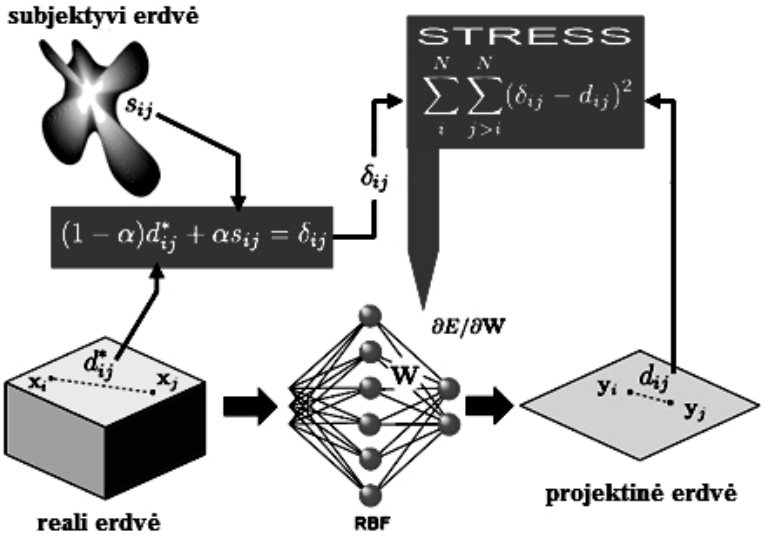
NeuroScale algoritmas yra Sammono projekcijos (2.2 skyrius) modifikacija, kurioje panaudojami radialinių bazinių funkcijų (3.3 skyrius) tiesioginio sklidimo neuroniniai tinklai (Lowe, 1996), (Lowe, 1997). NeuroScale naudoja papildomą informaciją apie duomenis ir tos informacijos pagrindu gali koreguoti analizuojamų duomenų projekcijas. Vizualizuojant duomenis, algoritmas bando išlaikyti analizuojamų duomenų geometrinę struktūrą ir atstumus tarp atitinkamų taškų pradinėje erdvėje ir projektinėje erdvėje.

Sammono algoritmo vienas iš trūkumų tas, kad atsiradus analizuojamų taškų aibėje naujam taškui, norint ir jį atvaizduoti plokštumoje, reikia perskaičiuoti visų jau atvaizduotų taškų projekcijas. NeuroScale algoritmas, kaip ir SAMANN algoritmas (4.1 skyrius), tokio trūkumo neturi.

NeuroScale metodas transformuoja daugiamatės erdvės taškus į mažesnio matavimo erdvę, naudojant tiesioginio sklidimo radialinių bazinių funkcijų neuroninį tinklą. Tinklo mokymo algoritmas sutampa su Sammono algoritmu (2.2 skyrius), tik yra minimizuojama kita projekcijos paklaida:

$$E = \sum_{i}^{m} \sum_{j<i}^{m} (\delta_{ij} - \|y_i - y_j\|)^2 ,$$

čia $\delta_{ij} = (1-\alpha) d_{ij}^* + \alpha s_{ij}$, $0 \le \alpha \le 1$. s_{ij} – tai papildomos subjektyvios žinios (subjektyvūs nepanašumai) apie kiekvienos taškų poros nepanašumus. Kiekvienai taškų porai surandamas ne tik d_{ij}^*, bet ir subjektyvios žinios s_{ij}. Parametras α leidžia kontroliuoti subjektyvios metrikos įtaką tinklo išėjimams. Kitaip sakant, α padeda surasti kompromisą tarp projekcijų su mokytoju ir be mokytojo. Nenaudojant subjektyvios metrikos gali būti naudojami tiesiog atstumai tarp vektorių X_i ir X_j, tai yra $\delta_{ij} = d_{ij}^*$. Detalesnis algoritmo aprašymas yra pateiktas darbuose (Lowe, 1997), (Tipping, 1996). 3.14 paveiksle pavaizduotas NeuroScale metodo schemos pavyzdys (Tipping, 1996).



3.14 pav. *NeuroScale metodo struktūrinė schema*

NeuroScale metodo privalumas yra tas, kad jeigu analizuojamų taškų aibėje atsiranda naujas *n*-matis taškas *X*, jis pateikiamas jau apmokytam tinklui, tinklo išėjime gaunamos taško *Y*, kuris yra *X* projekcija, komponentės. Tinklo „nematyti“ taškai plokštumoje „randa“ vietą tarp savo klasių taškų.

NeuroScale metodo vizualizavimo rezultatai yra žymiai geresni, lyginant su SOM (Noel, 1998). Tačiau šis metodas yra neefektyvus, analizuojant didelės apimties duomenų aibes (kai taškų skaičius didesnis nei 1000).

3.9. Išvados

Norint atskleisti daugiamačių duomenų struktūrą, vien tik klasikinių vizualizavimo metodų nepakanka. Šiam tikslui sėkmingai gali būti naudojami dirbtiniai neuroniniai tinklai (Mao, 1995). Neuroniniai tinklai yra galingas įrankis, naudojamas tais atvejais, kai formali analizė yra sudėtinga arba neįmanoma, tokiais atvejais, kaip atpažinimas, netiesinių sistemų identifikavimas ir valdymas. Neuroninių tinklų ypatumas yra tai, kad jie gali pamėgdžioti žmogaus smegenų veiklą, mokytis bei reaguoti. Prisitaikymas arba mokymasis – pagrindinis neuroninių tinklų tyrimų objektas.

Šiame skyriuje susistemintos esminės dirbtinių neuroninių tinklų koncepcijos. Nagrinėjami tiesioginio sklidimo neuroniniai tinklai, saviorganizuojantys neuroniniai tinklai, radialinių bazinių funkcijų neuroniniai tinklai, neuroninių tinklų mokymo būdai.

Klasikiniai vizualizavimo metodai kartais yra nepajėgūs susidoroti su savo užduotimis. Tačiau yra ir neuroninių tinklų, kurie realizuoja duomenų projekcijos algoritmus – pagrindinių komponenčių metodą ir Sammono projekciją. Šiame skyriuje taip pat nagrinėti kiti metodai, kuriuose dirbtiniai neuroniniai tinklai naudojami daugiamačiams duomenims vizualizuoti: kreivinių komponenčių analizė, NeuroScale metodas.

Ne visi iš nagrinėtų neuroninių tinklų leidžia atlikti naujų taškų vizualizavimą be tinklo permokymo. Tačiau tokią galimybę turi, pavyzdžiui, NeuroScale ir SAMANN tinklai.

Naudojant specifinį mokymo be mokytojo „klaidos sklidimo atgal“ algoritmą, pavadintą SAMANN, ieškoma tokio daugiamačių taškų išsidėstymo plokštumoje, kad Sammono paklaida būtų minimali. Metodo privalumas yra tas, kad jis gali būti naudojamas naujų analizuojamos aibės taškų atvaizdavimui plokštumoje neperskaičiavus visų jau atvaizduotų taškų projekcijų. SAMANN tinklo mokymo proceso konvergavimas yra labai lėtas, todėl toliau šiame darbe ieškoma būdų, kaip jį pagreitinti.

4

SAMANN neuroninio tinklo mokymo spartinimas

Nustatant ryšius tarp klasikinių metodų ir neuroninių tinklų, buvo pastebėta, kad kai kurie neuroniniai tinklai faktiškai realizuoja duomenų projekcijos algoritmus. Atliekant tyrimus, keletas klasikinių duomenų projekcijos metodų buvo realizuoti naudojant *neuroninių tinklų struktūras*. Nors tokių bandymų rezultate nebūtinai gaunami nauji projekcijos metodai, tačiau tokie tinklai yra pranašesni už tradicinius metodus. Pavyzdžiui, *SAMANN neuroninis tinklas* (Mao, 1995), sukurtas Sammono netiesinės projekcijos algoritmui, turi naujų duomenų vizualizavimo galimybę, t.y. leidžia surasti naujų taškų projekcijas be papildomų skaičiavimų. O tai yra savybė, kurios neturi originalus Sammono algoritmas.

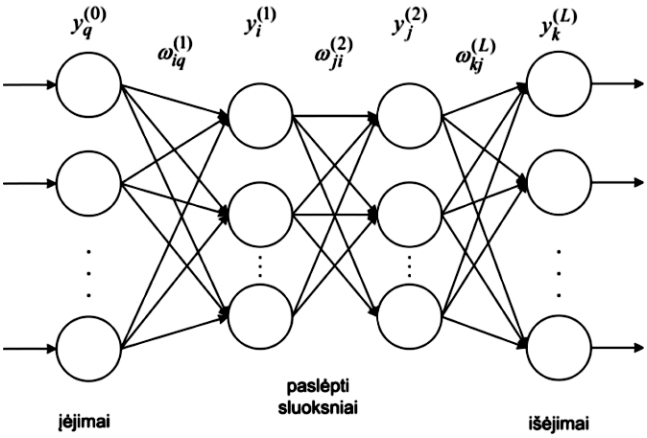
4.1. SAMANN algoritmas

Sammono projekcija (Sammon, 1969) yra netiesinis daugelio kintamųjų objektų atvaizdavimo žemesnio matavimo erdvėje metodas. Jo idėja: atvaizduoti daugiamačius taškus mažesnio matavimo erdvėje išlaikant santykinai panašius atstumus tarp jų. Sammono metodas minimizuoja projekcijos paklaidą (Sammono paklaidą) E_S :

$$E_S = \frac{1}{\sum_{\mu=1}^{m-1}\sum_{\nu=\mu+1}^{m} d^*_{\mu\nu}} \sum_{\mu=1}^{m-1}\sum_{\nu=\mu+1}^{m} \frac{[d^*_{\mu\nu} - d_{\mu\nu}]^2}{d^*_{\mu\nu}}, \qquad (4.1)$$

čia $d^*_{\mu\nu}$ – atstumas tarp *n*-mačių taškų X^μ ir X^ν, $d_{\mu\nu}$ – atstumas tarp *d*-mačių taškų Y^μ ir Y^ν, į kuriuos projektuojami taškai X^μ ir X^ν ($d < n$). Analizuojamų taškų skaičius lygus *m*.

Sammono algoritmo vienas iš trūkumų yra tas, kad atsiradus analizuojamų taškų aibėje naujam taškui, norint ir jį atvaizduoti mažesnio matavimo erdvėje, reikia perskaičiuoti visų jau atvaizduotų taškų projekcijas. J. Mao ir A. K. Jain darbe (Mao, 1995) pasiūlė Sammono paklaidą minimizuoti naudojant tiesioginio sklidimo neuroninius tinklus. Pasiūlyta specifinė „klaidos sklidimo atgal“ mokymo taisyklė, pavadinta SAMANN, kuri leidžia įprastam tiesioginio sklidimo neuroniniam tinklui (4.1 pav.) realizuoti Sammono projekciją mokymo be mokytojo būdu.



4.1 pav. *Tiesioginio sklidimo trijų sluoksnių DNT Sammono projekcijai*

Tarkime, kad $X = (x_1, x_2, ..., x_n)$ yra *n*-mačiai taškai. Jie bus naudojami tinklo mokymui, kaip įėjimų reikšmės. Kiekviename mokymo žingsnyje neuroniniam tinklui pateikiami du *n*-mačiai taškai, o išėjimuose siekiama gauti jų projekcijas *d*-matėje erdvėje, t.y. taškus $Y = (y_1, y_2, ..., y_d)$, ($d < n$). Apskaičiuojamai atstumi tarp neuroninio tinklo išėjimų, t.y. *d*-mačių taškų ir nustatoma Sammono paklaidos reikšmė E_S (4.1). Šios paklaidos pagrindu keičiami neuronų svoriai. Norint gauti analizuojamų duomenų projekcijas plokštumoje, naudojamas SAMANN neuroninis tinklas, turintis du išėjimus.

Pažymėkime *j*-otojo neurono išėjimą *l*-tajame sluoksnyje $y_j^{(l)}$, $j=1,...,n_l$, $l=0,...,L$; čia n_l yra *l*-tojo sluoksnio neuronų skaičius, L – neuronų sluoksnių skaičius. Neuroninio tinklo įėjimai – tai įėjimo taško komponentės, t.y. $y_j^{(0)}=x_j$, $j=1,...,n$. Jungties tarp *i*-tojo neurono $(l-1)$-ajame sluoksnyje ir *j*-tojo neurono *l*-tajame sluoksnyje svorį žymėkime $w_{ji}^{(l)}$. *j*-tojo neurono *l*-tajame sluoksnyje slenksčio reikšmę (*angl. bias*) pažymėkime $w_{j0}^{(l)}$, o $y_0^{(l)}=1$. Kiekvieno neurono išėjimo reikšmei skaičiuoti naudojama sigmoidinė funkcija $f(a)=\dfrac{1}{1+e^{-a}}$, kurios reikšmių intervalas yra (0; 1); čia *a* – visų neuronų įėjimų ir jų svorių sandaugų suma. Taigi, *j*-tojo neurono *l*-tajame sluoksnyje išėjimas išreiškiamas taip $y_j^{(l)}=f\left(\sum\limits_{i=0}^{n_{l-1}} w_{ji}^{(l)} y_i^{(l-1)}\right)$, $l=1,...,L$.

Pakeiskime naudojamus žymėjimus. Anksčiau taško Y^μ komponentės buvo žymimos taip: $Y^\mu=(y_1^\mu, y_2^\mu,..., y_d^\mu)$. Dabar viršutinis indeksas nurodo sluoksnio numerį neuroniniame tinkle, todėl taško komponentes išreiškime tokiu būdu $Y^\mu=(y_1(\mu), y_2(\mu),..., y_d(\mu))$.

Euklido atstumas $d_{\mu\nu}$ tarp dviejų *d*-mačių taškų Y^μ ir Y^ν apskaičiuojamas pagal formulę:

$$d_{\mu\nu}=\left\{\sum_{k=1}^{d}\left[y_k^{(L)}(\mu)-y_k^{(L)}(\nu)\right]^2\right\}^{\frac{1}{2}}.$$

Atkreipkime dėmesį į tai, kad dėl sigmoidinės funkcijos reikšmių intervalo, kiekvieno išėjimo intervalas yra (0; 1). Jei įėjimo reikšmių intervalas yra platus, tai naudojant projekcijos algoritmą, neįmanoma išlaikyti atstumų tarp daugiamačių taškų. Todėl, visi įėjimo taškai normuojami, norint suvienodinti atstumus tarp taškų pradinėje erdvėje ir projektinėje erdvėje.

Pažymėkime $\lambda=\dfrac{1}{\sum\limits_{\mu=1}^{m-1}\sum\limits_{\nu=\mu+1}^{m} d_{\mu\nu}^*}$. Dydžio λ reikšmė nepriklauso nuo tinklo, taigi ji gali būti apskaičiuota iš anksto. Tuomet Sammono paklaida $E_{\mu\nu}$ dviem taškams μ ir ν skaičiuojama pagal (4.2) formulę, o visiems taškams pagal (4.3) formulę.

$$E_{\mu\nu}=\lambda\frac{\left[d_{\mu\nu}^*-d_{\mu\nu}\right]^2}{d_{\mu\nu}^*}, \qquad (4.2)$$

$$E_S = \sum_{\mu=1}^{m-1} \sum_{\nu=\mu+1}^{m} E_{\mu\nu} \,. \tag{4.3}$$

Dydis $E_{\mu\nu}$ yra proporcingas atstumų tarp μ-tojo ir ν-tojo taškų pokyčiams, todėl jis labiau tinkamas svorių keitimo taisyklėms nustatyti. J. Mao ir A. K. Jain daugiasluoksniam tiesioginio sklidimo tinklui pasiūlė svorių keitimo taisyklę, kuri minimizuoja Sammono paklaidą ir yra pagrįsta gradientinio nusileidimo metodu.

Išėjimo sluoksniui $(l = L)$ gauname:

$$\begin{aligned}\frac{\partial E_{\mu\nu}}{\partial w_{kj}^{(L)}} &= \left(\frac{\partial E_{\mu\nu}}{\partial d_{\mu\nu}}\right)\left(\frac{\partial d_{\mu\nu}}{\partial\left[y_k^{(L)}(\mu) - y_k^{(L)}(\nu)\right]}\right)\left(\frac{\partial\left[y_k^{(L)}(\mu) - y_k^{(L)}(\nu)\right]}{\partial w_{kj}^{(L)}}\right) = \\ &= \left(-2\lambda\frac{d_{\mu\nu}^* - d_{\mu\nu}}{d_{\mu\nu}^*}\right)\left(\frac{y_k^{(L)}(\mu) - y_k^{(L)}(\nu)}{d_{\mu\nu}}\right)\times \\ &\times\left(f'(a_{k,\mu}^{(L)})y_j^{(L-1)}(\mu) - f'(a_{k,\nu}^{(L)})y_j^{(L-1)}(\nu)\right)\end{aligned} \tag{4.4}$$

Čia $f'(a_{k,\nu}^{(L)})$ yra *k*-tojo elemento *L*-tajame sluoksnyje (išėjimo sluoksnyje) sigmoidinės funkcijos išvestinė pagal šio neurono įėjimą $(a_{k,\nu})$:

$$f'(a_{k,\nu}) = (1 - y_k^{(L)}(\nu))y_k^{(L)}(\nu) \,. \tag{4.5}$$

Tegu

$$\delta_k^{(L)}(\mu,\nu) = -2\lambda\frac{d_{\mu\nu}^* - d_{\mu\nu}}{d_{\mu\nu}^* d_{\mu\nu}}[y_k^{(L)}(\mu) - y_k^{(L)}(\nu)], \tag{4.6}$$

$$\Delta_{kj}^{(L)}(\mu) = \delta_k^{(L)}(\mu,\nu)[1 - y_k^{(L)}(\mu)]y_k^{(L)}(\mu), \tag{4.7}$$

$$\Delta_{kj}^{(L)}(\nu) = \delta_k^{(L)}(\mu,\nu)\left[1 - y_k^{(L)}(\nu)\right]y_k^{(L)}(\nu) \,. \tag{4.8}$$

Įstačius (4.5)–(4.8) formules į (4.4), gauname:

$$\frac{\partial E_{\mu\nu}}{\partial w_{kj}^{(L)}} = \Delta_{kj}^{(L)}(\mu)y_j^{(L-1)}(\mu) - \Delta_{kj}^{(L)}(\nu)y_j^{(L-1)}(\nu) \,. \tag{4.9}$$

Taigi išėjimo sluoksnio svorių atnaujinimo taisyklė:

$$\Delta w_{kj}^{(L)} = -h\frac{\partial E_{\mu\nu}}{\partial w_{kj}^{(L)}} = -h(\Delta_{kj}^{(L)}(\mu)y_j^{L-1}(\mu) - \Delta_{kj}^{(L)}(\nu)y_j^{L-1}(\nu)) \,. \tag{4.10}$$

čia h – mokymosi greičio parametras.

Panašiai gaunamos bendros svorių atnaujinimo taisyklės kiekvienam paslėptam sluoksniui, $l = 1,...,L-1$:

$$\Delta w_{ji}^{(l)} = -h\frac{\partial E_{\mu\nu}}{\partial w_{ji}^{(l)}} = -h(\Delta_{ji}^{(l)}(\mu)y_i^{l-1}(\mu) - \Delta_{ji}^{(l)}(\nu)y_i^{l-1}(\nu)), \qquad (4.11)$$

čia

$$\Delta_{ji}^{(l)}(\mu) = \delta_j^{(l)}(\mu)\left[1 - y_j^{(l)}(\mu)\right]y_j^{(l)}(\mu), \qquad (4.12)$$

$$\Delta_{ji}^{(l)}(\nu) = \delta_j^{(l)}(\nu)\left[1 - y_j^{(l)}(\nu)\right]y_j^{(l)}(\nu), \qquad (4.13)$$

ir

$$\delta_j^{(l)}(\mu) = \sum_{k=1}^{d}\Delta_{kj}^{(l+1)}(\mu)w_{kj}^{(l+1)}, \qquad (4.14)$$

$$\delta_j^{(l)}(\nu) = \sum_{k=1}^{d}\Delta_{kj}^{(l+1)}(\nu)w_{kj}^{(l+1)}. \qquad (4.15)$$

Kaip ir standartiniame „klaidos sklidimo atgal“ mokymo algoritme, dydžiai $\delta_j^{(l)}(\mu)$ ir $\delta_j^{(l)}(\nu)$ keičiasi grįžtant iš $(l+1)$-ojo sluoksnio į l-tąjį sluoksnį, atitinkamai μ-tajam ir ν-tajam taškams.

Iš (4.10) ir (4.11) formulių seka, kad norint atnaujinti neuroninio tinklo svorius, į tinklą tuo pat metu turi būti paduota taškų pora. Tai galima padaryti sukonstruojant du identiškus tinklus arba tiesiog laikyti atmintyje visus pirmojo taško išėjimus prieš pateikiant tinklui antrąjį tašką.

Apibendrinta SAMMAN algoritmo schema yra tokia:

1) Generuojamos atsitiktinės pradinės SAMANN tinklo svorių reikšmės;
2) Atsitiktinai parenkama *n*-mačių taškų pora, kuri yra pateikiama tinklui; apskaičiuojami tinklo parametrai;
3) Atnaujinami svoriai pagal (4.10) ir (4.11) formules „sklidimo atgal“ būdu, pradedant nuo išėjimo sluoksnio;
4) Kelis kartus kartojami 2–3 žingsniai – tokiu būdu tinklas apmokomas;
5) Tinklui pateikiami visi taškai ir apskaičiuojami tinklo išėjimai; skaičiuojama Sammono paklaida (4.1): jeigu jos reikšmė mažesnė už pasirinktą slenkstį arba iteracijų skaičius viršija nustatytąjį, tuomet algoritmas sustabdomas, priešingu atveju, vėl pradedam nuo 2 žingsnio.

Norint pagreitinti tinklo mokymo procesą, kartais svorių atnaujinimo formulės (4.10) ir (4.11) yra keičiamos, pridedant dar *momentum* reikšmę.

SAMANN algoritmo privalumas yra tas, kad jeigu analizuojamų taškų aibėje atsiranda naujas *n*-matis taškas *X*, jis pateikiamas jau apmokytam SAMANN tinklui, tinklo išėjime gaunamos taško *Y*, kuris yra *X* projekcija, komponentės. Žinoma, jeigu tų naujų taškų yra daug, po kažkurio laiko tinklą reikia permokyti, rasti naujas svorių reikšmes.

Pastebėsime, kad Sammono projekcijos algoritmui reikalinga informacija iš karto apie visus analizuojamos aibės taškus, t. y. algoritmui turime pateikti visą duomenų aibę. SAMANN tinklui pateikiamos taškų poros. Tinklas įvertina atstumus tarp tų dviejų taškų, tačiau saugoja ir koreguoja turimas žinias apie visos aibės taškus. Mokymo metu tinklui padavus naują taškų porą, koreguojama informacija apie visų mokymo aibės taškų tarpusavio išsidėstymą.

Eksperimentams ir tyrimams buvo sukurta programinė SAMANN tinklo realizacija.

4.2. Neurono aktyvacijos funkcijos įtaka tinklo mokymui

Atvaizduojant daugiamačius duomenis mažesnio matavimo erdvėje, labai svarbu pasiekti gerų vizualizavimo rezultatų per trumpą laiką, tinklo mokymas turi būti efektyvus ir mokymo algoritmas turi greitai konverguoti. SAMANN neuroninio tinklo mokymui reikia daug skaičiuojamųjų sąnaudų, todėl naujus svorius ir tikslią duomenų projekciją siekiama gauti per trumpą laiką. Analizuojant SAMANN tinklą, pastebėta, kad projekcijos paklaida priklauso nuo skirtingų parametrų. Tyrimai parodė, kad viena iš lėto tinklo mokymo priežasčių yra neurono aktyvacijos funkcijos savybės.

Kiekviename neurone naudojama sigmoidinė aktyvacijos funkcija $g(x)$, kuri kinta intervale (0.0, 1.0) :

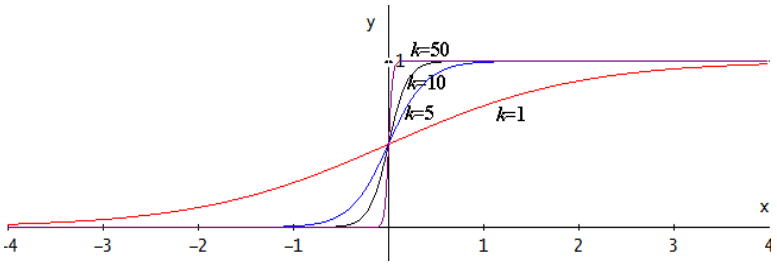
$$g(x) = \frac{1}{\left(1 + e^{-kx}\right)},$$

k yra sigmoidinės funkcijos nuolydžio parametras. Keičiant parametro *k* reikšmes, mes gauname įvairaus statumo sigmoidines funkcijas (4.2 pav.). Ribiniu atveju, kai *k* artėja prie begalybės, sigmoidinė funkcija pavirsta į slenkstinę funkciją. Sigmoidinė funkcija yra diferencijuojama, o slenkstinė – ne. Sigmoidinė funkcija yra griežtai didėjanti funkcija, kuri apjungia savyje tiek tiesines, tiek netiesines savybes (Haykin, 1999).

Sigmoidinė funkcija dažnai naudojama neuroniniuose tinkluose, norint įvesti netiesiškumą. Nors yra ir kitų neurono aktyvacijos funkcijų, tačiau sigmoidinė funkcija yra viena populiariausių dirbant su neuroniniais tinklais.

Viena iš pagrindinių sigmoidinės funkcijos savybių yra labai paprasta šios funkcijos išvestinė:

$$\frac{\partial g(x)}{x} = \frac{\partial}{\partial x}\left(\frac{1}{1 + e^{-kx}}\right), \quad \frac{\partial g(x)}{x} = kg(x)(1 - g(x)).$$



4.2 Pav. Sigmoidinė funkcija esant įvairiom nuolydžio parametro *k* reikšmėm

Bendrąją neuroninio tinklo svorių atnaujinimo formulę visiems paslėptiems sluoksniams, $l = 1, \ldots, L-1$ ir išėjimo sluoksniui ($l = L$) galima užrašyti taip (Mao, 1995):

$$\Delta w_{jt}^{(l)} = -h \frac{\partial E_S(\mu, \nu)}{\partial w_{jt}^{(l)}} = -hk(\Delta_{jt}^{(l)}(\mu) y_j^{(l-1)}(\mu) - \Delta_{jt}^{(l)}(\nu) y_j^{(l-1)}(\nu)), \qquad (4.16)$$

čia $w_{jt}^{(l)}$ yra svoris tarp *j* neurono $l-1$ sluoksnyje ir *t* neurono *l* sluoksnyje, *h* yra mokymosi parametras, *k* yra sigmoidinės aktyvacijos funkcijos nuolydžio parametras, $y_j^{(l)}$ yra *l* sluoksnio *j* neurono išvestis, μ ir ν yra du analizuojamos duomenų aibės taškai, $\Delta_{jt}^{(l)}$ yra paklaidos, sukauptos kiekviename sluoksnyje ir skleidžiamos ankstesniam sluoksniui (Mao, 1995).

Kaip matome, sigmoidinės neurono aktyvacijos funkcijos nuolydžio parametras *k* yra daugiklis svorių atnaujinimo formulėje ir jis įtakoja neuroninio tinklo mokymą.

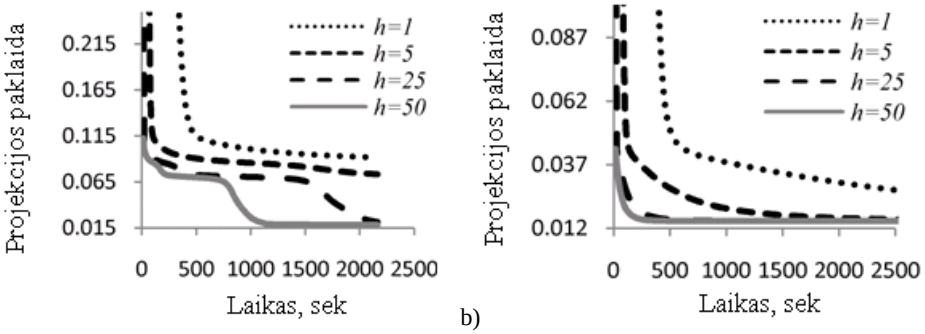
Vertinant aktyvacijos funkcijos parametrų įtaką neuroninio tinklo mokymuisi, buvo naudojamos dvi realios duomenų aibės:

- *1 duomenų aibė*. Pima Indians Diabetes Database. Ši duomenų aibė buvo paimta iš UCI mašininio mokymo saugyklos (Asuncion, 2007). Duomenų aibę sudaro 768 8-mačiai taškai, priklausantys dviem klasėm.
- *2 duomenų aibė*. Statlog duomenų aibė (Asuncion, 2007). Tai duomenų apie kredito korteles aibė. Šią aibę sudaro 690 14-mačių taškų iš 2 klasių.

Visuose eksperimentuose buvo naudojamas tiesioginio sklidimo neuroninis tinklas su vienu paslėptu sluoksniu ir dviem išėjimo neuronais (*d* = 2). Atliekant eksperimentus su skirtingomis aktyvacijos funkcijos nuolydžio parametro ir tinklo mokymosi parametro reikšmėmis, buvo skaičiuojama projekcijos paklaida ir matuojamas programos vykdymo laikas.

Pirmieji eksperimentai buvo atliekami vizualizuojant nagrinėjamas duomenų aibes, naudojant skirtingas mokymosi parametro reikšmes. Darbe (Medvedev, 2006) buvo parodyta, kad tinkamai parinkus mokymosi parametro reikšmę, galima

pagreitinti neuroninio tinklo mokymosi procesą. Atliekant šiuos eksperimentus, neurono aktyvacijos funkcijos nuolydžio parametro *k* reikšmė buvo 1. Eksperimentai buvo atlikti dirbant su keliomis skirtingomis mokymosi parametro reikšmėmis (*h* = 1, 5, 25, 50).



4.3 pav. Projekcijos paklaidos priklausomybė nuo skaičiavimo laiko, naudojant skirtingas mokymosi parametro reikšmes. (a) *1 duomenų aibė* (b) *2 duomenų aibė*

4.1 lentelė. Projekcijos paklaidos, gaunamos naudojant skirtingas parametrų *h* ir *k* kombinacijas, dirbant su *1 duomenų aibe*

Mokymosi iteracijos numeris	***k*=1**				***h*=5**		
	***h*=1**	***h*=5**	***h*=25**	***h*=50**	***k*=5**	***k*=10**	***k*=30**
10	0,96513	0,11211	0,09041	0,08590	0,02225	0,01910	0,01648
20	0,76135	0,09971	0,08549	0,07354	0,02010	0,01858	0,01568
30	0,28915	0,09454	0,07971	0,07154	0,01932	0,01820	0,01540
40	0,12834	0,09175	0,07356	0,07069	0,01897	0,01775	0,01546
50	0,11203	0,08994	0,07221	0,06997	0,01879	0,01722	0,01548
60	0,10818	0,08860	0,07164	0,06886	0,01870	0,01668	0,01553
70	0,10545	0,08755	0,07120	0,06505	0,01863	0,01627	0,01567
80	0,10318	0,08669	0,07084	0,04550	0,01857	0,01603	0,01582
90	0,10128	0,08594	0,07050	0,02945	0,01850	0,01589	0,01577
100	0,09969	0,08524	0,07016	0,02191	0,01843	0,01579	0,01567
110	0,09834	0,08451	0,06973	0,01929	0,01834	0,01572	0,01557
120	0,09718	0,08368	0,06909	0,01904	0,01826	0,01566	0,01548
130	0,09617	0,08264	0,06796	0,01893	0,01816	0,01560	0,01540
140	0,09530	0,08127	0,06543	0,01885	0,01807	0,01555	0,01534
150	0,09454	0,07957	0,05858	0,01880	0,01797	0,01550	0,01530
160	0,09386	0,07786	0,04605	0,01876	0,01787	0,01545	0,01526
170	0,09326	0,07633	0,03712	0,01873	0,01777	0,01540	0,01524
180	0,09271	0,07510	0,02953	0,01870	0,01766	0,01535	0,01522
190	0,09222	0,07416	0,02438	0,01867	0,01756	0,01531	0,01521
200	0,09177	0,07349	0,02155	0,01865	0,01745	0,01526	0,01520

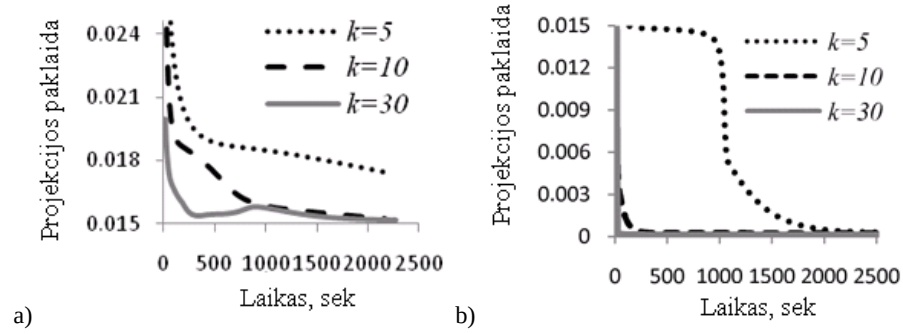
Eksperimentų rezultatai yra pateikiami 4.3 paveiksle. Projekcijos paklaidos reikšmės yra pateikiamos 4.1 ir 4.2 lentelėse. Kaip ir buvo tikėtasi, eksperimentai su abiem duomenų aibėm parodė, kad tinkamas mokymosi parametro parinkimas leidžia ženkliai pagreitinti tinklo mokymosi procesą. Kai h = 50, per tą patį laiką mes gauname beveik 5 kartus mažesnę projekcijos paklaidą dirbant su *1 duomenų aibe* (4.1 lentelė) ir beveik du kartus mažesnę projekcijos paklaidą dirbant su *2 duomenų aibe* (4.2 lentelė). Tačiau abiem atvejais projekcijos paklaida lieka pakankamai didelė. Analizuojant neuroninio tinklo mokymosi grafiką, kuris gaunamas dirbant su *1 duomenų aibe* (4.3a paveikslas), mes matome kelias staigaus paklaidos mažėjimo sritis (h = 25, 50). Už šių sričių ribų paklaida mažėja nežymiai.

4.2 lentelė. Projekcijos paklaidos, gaunamos naudojant skirtingas parametrų h ir k kombinacijas, dirbant su *2 duomenų aibe*

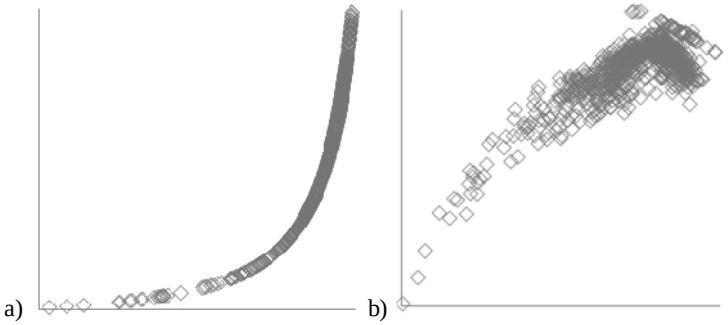
Mokymosi iteracijos numeris	***k*=1**				***h*=5**		
	***h*=1**	***h*=5**	***h*=25**	***h*=50**	***k*=5**	***k*=10**	***k*=30**
10	0,94582	0,04245	0,02435	0,01821	0,01499	0,00086	0,00018
20	0,57042	0,03548	0,01795	0,01546	0,01482	0,00031	0,00016
30	0,10166	0,03051	0,01610	0,01504	0,01478	0,00026	0,00016
40	0,04770	0,02659	0,01541	0,01496	0,01473	0,00026	0,00016
50	0,04232	0,02371	0,01514	0,01494	0,01465	0,00026	0,00016
60	0,04025	0,02164	0,01502	0,01493	0,01450	0,00026	0,00016
70	0,03879	0,02016	0,01497	0,01493	0,01403	0,00025	0,00015
80	0,03753	0,01910	0,01495	0,01493	0,00537	0,00025	0,00015
90	0,03635	0,01831	0,01494	0,01492	0,00383	0,00025	0,00015
100	0,03522	0,01770	0,01493	0,01492	0,00265	0,00025	0,00015
110	0,03414	0,01722	0,01493	0,01492	0,00182	0,00025	0,00015
120	0,03310	0,01682	0,01493	0,01492	0,00126	0,00025	0,00015
130	0,03211	0,01650	0,01492	0,01492	0,00089	0,00025	0,00015
140	0,03116	0,01624	0,01492	0,01492	0,00066	0,00024	0,00015
150	0,03025	0,01602	0,01492	0,01491	0,00051	0,00024	0,00015
160	0,02938	0,01584	0,01492	0,01491	0,00041	0,00024	0,00015
170	0,02857	0,01569	0,01492	0,01491	0,00036	0,00024	0,00015
180	0,02780	0,01556	0,01492	0,01491	0,00032	0,00024	0,00015
190	0,02708	0,01546	0,01492	0,01491	0,00030	0,00024	0,00015
200	0,02640	0,01537	0,01492	0,01491	0,00029	0,00024	0,00015

Kiti eksperimentai buvo atliekami su abiem duomenų aibėm naudojant skirtingas neurono aktyvacijos funkcijos nuolydžio parametro reikšmes. Keičiant neurono aktyvacijos funkcijos nuolydžio parametro reikšmę, pasikeičia ir sigmoidinės funkcijos pavidalas. Taipogi pasikeičia ir mokymosi parametras, kadangi tiek h, tiek k parametrai yra daugikliai 4.16 formulėje. Atliekant eksperimentus buvo nustatyta mokymosi parametro h reikšmė 5. Kadangi tiek h, tiek k parametrai yra daugikliai 4.16 formulėje, imti didesnę h reikšmę nėra tikslinga, nes labai padidėja sandaugos

reikšmė. Buvo atlikti testai su skirtingomis aktyvacijos funkcijos nuolydžio parametro reikšmėmis (k = 5, 10, 30). Taip pat buvo atlikti bandymai ir su didesnėmis parametro k reikšmėmis, tačiau gauti rezultatai buvo blogesni už gautus, dirbant su mažesnėm k reikšmėm.



4.4 Pav. Projekcijos paklaidos priklausomybė nuo skaičiavimo laiko, naudojant skirtingas neurono aktyvacijos funkcijos nuolydžio parametro reikšmes.
(a) *1 duomenų aibė* (b) *2 duomenų aibė*



4.5 pav. *1 duomenų aibės* projekcijos vaizdas, kai aktyvacijos funkcijos nuolydžio parametro reikšmės yra: k = 1 (a) ir k = 30 (b), naudojamas toks pat mokymosi parametras

Eksperimentų rezultatai pateikiami 4.4 paveiksle. Projekcijos paklaidos reikšmės yra pateikiamos 4.1 ir 4.2 lentelėse. Dirbant su abiem duomenų aibėm tinkamas aktyvacijos funkcijos nuolydžio parametro parinkimas leidžia žymiai pagreitinti neuroninio tinklo mokymąsi. Kai k = 30, dirbant su *1 duomenų aibe*, gauta projekcijos paklaida yra 27% mažesnė, negu geriausia paklaida, gauta pirmuose eksperimentuose, kur mokymosi parametro reikšmė h = 50 (4.1 lentelė). Kai k = 30, dirbant su *2 duomenų aibe*, mes gauname projekcijos paklaidą apie 100 kartų mažesnę, negu geriausia projekcijos paklaida rasta pirmuose eksperimentuose (4.2 lentelė). Analizuojant neuroninio tinklo mokymosi grafikus (4.4 paveikslas), galime pastebėti,

kad mokymosi procesas neturi staigių paklaidos mažėjimo sričių. Taip pat iš grafikų matosi, kad jau su $k = 30$ tinklo mokymo metu projekcijos paklaida didėja tam tikrame etape. Šis efektas yra žymiai ryškesnis dirbant su didesniais *k. 1 duomenų aibės* vizualizavimo rezultatas, kai naudojamos aktyvacijos funkcijos nuolydžio parametro reikšmės $k = 1$ ir $k = 30$, esant tam pačiam mokymosi parametrui, yra pateikiamas 4.5 paveiksle.

4.3. Neuroninio tinklo mokymosi aibės parinkimas

Darbe (Ivanikovas, 2007) buvo pastebėta, kad SAMANN neuroninio tinklo mokymas priklauso nuo įvairių parametrų. Eksperimentų rezultatai parodė, kad galima rasti tokį analizuojamos duomenų aibės poaibį, kad, mokant neuroninį tinklą šiuo poaibiu, gaunamos mažesnės projekcijos paklaidos, ir tinklas mokosi greičiau, negu mokant visos analizuojamos duomenų aibės taškais.

Norint nustatyti tinkamiausią neuroninio tinklo mokymo aibės konstravimo būdą, buvo atlikti eksperimentai mokant neuroninį tinklą skirtingai parenkant mokymo aibės taškus. SAMANN tinklas buvo mokomas naudojant tik dalį analizuojamos duomenų aibės taškų. Pirmuose eksperimentuose mokymosi duomenų aibė buvo konstruojama atsitiktinai paimant iš analizuojamos duomenų aibės norimą taškų skaičių.

Testuojant šią strategiją buvo naudojamos dvi duomenų aibės:

- *1 duomenų aibė*. Dirbtinai sukurta atsitiktinai sugeneruotų 10-mačių taškų aibė. Duomenų aibę sudaro 966 taškai iš 4 klasių.
- *2 duomenų aibė*. Puslapių klasifikavimo duomenų aibė (Blake, 1998). Tai reali duomenų aibė, sudaryta iš 5473 10-mačių taškų, sudarančių penkias klases.

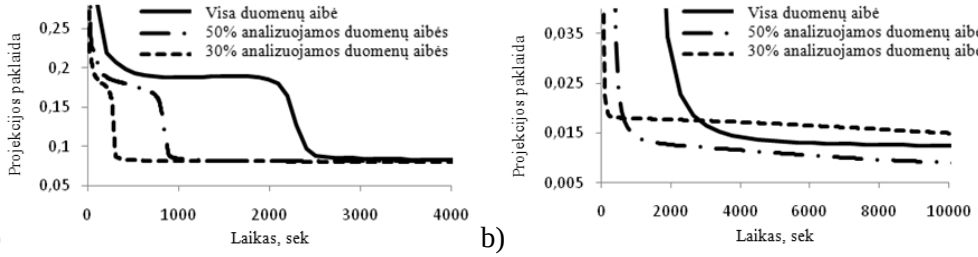
Atliekant eksperimentus buvo dirbama su tiesioginio sklidimo dirbtiniu neuroniniu tinklu su vienu paslėptu sluoksniu ir dviem išėjimo neuronais (d=2). Visuose eksperimentuose buvo naudojama tokia pati neuroninio tinklo konfigūracija ir fiksuotas pradinių svorių rinkinys.

Dirbant su skirtingo dydžio mokymo aibėmis (50% ir 30% visos analizuojamos duomenų aibės dydžio) ir su visa nagrinėjama duomenų aibe, buvo skaičiuojama projekcijos paklaida (skaičiuojant projekcijos paklaidą naudojami visi duomenų aibės taškai) ir matuojamas SAMANN neuroninio tinklo mokymosi laikas. Eksperimentų rezultatai yra grafiškai pateikiami 4.6 paveiksle. Naudojant mokymo aibę, sudarytą iš 50% visos analizuojamos duomenų aibės taškų, neuroninis tinklas mokėsi greičiau, ir buvo gaunamos mažesnės projekcijos paklaidos. Toks rezultatas buvo gautas abiem duomenų aibėm. Neuroninis tinklas mokosi dar greičiau, kai dirbama su mokymo aibe, sudaryta iš 30% visos analizuojamos duomenų aibės taškų, tačiau ne visada gaunamos pakankamai geros projekcijos paklaidos.

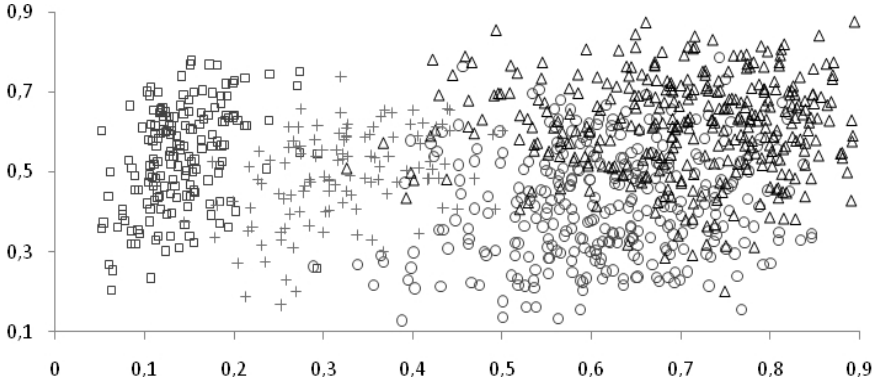
1 duomenų aibės vizualizavimo rezultatas pateikiamas 4.7 paveiksle. Šis rezultatas buvo gautas mokant neuroninį tinklą 50% visos duomenų aibės taškų.

Gauti rezultatai (4.6 paveikslas) rodo, kad galima gauti geresnę projekcijos paklaidą ir paspartinti SAMANN neuroninio tinklo mokymą naudojant mokymo aibę, sudarytą tik iš dalies analizuojamos duomenų aibės taškų.

Kiekvienas eksperimentas trunka daugiau negu valandą, dirbant su *1 duomenų aibe* ir beveik tris valandas, dirbant su *2 duomenų aibe*. Skaičiavimai buvo atliekami dvibranduoliniame kompiuteryje. Dviejų lygiagrečių srautų naudojimas leido dvigubai sutrumpinti skaičiavimų laiką.



4.6 pav. Projekcijos priklausomybė nuo skaičiavimo laiko naudojant skirtingo dydžio mokymosi aibes. (a) 1 duomenų aibė (b) 2 duomenų aibė



4.7 pav. *1 duomenų aibės* vizualizavimo rezultatas

4.3.1. Mokymo aibės sudarymas naudojant klasterizavimo metodus

Kitas galimas būdas sudaryti mokymo aibę analizuojamos duomenų aibės pagrindu yra klasterizavimo metodų naudojimas. Klasterizavimas dažnai naudojamas dirbant su didelės apimties duomenų aibėmis. Bendruoju atveju klasterizavimo metodai leidžia *m* turimų *n*-mačių taškų suskirstyti į *k* klasterių taip, kad kiekviename klasteryje atsidurs labiausiai tarpusavyje panašūs taškai.

Naudojant klasterizavimo metodus mokymo aibės sudarymui, daugiamačių taškų vizualizavimas SAMANN algoritmu yra suskirstomas į tris dalis: pradinių duomenų klasterizavimas, tinklo mokymas, naudojant tiktai klasterizavimo metu gautus duomenis, visų analizuojamos duomenų aibės taškų vizualizavimas. Dažnai prieš pradedant apdoroti duomenis naudinga juos normalizuoti. Sudarant mokymosi aibę buvo išbandyti du klasterizavimo metodai: *k*-vidurkių metodas ir SOM.

K-vidurkių metodas (angl *k*-means) leidžia klasterizuoti nagrinėjamą duomenų aibę į tam tikrą klasterių kiekį (MacQueen, 1967). Klasterių kiekis *k* yra parenkamas pradedant vykdyti algoritmą. Šio metodo veikimo principas yra toks:

1. Inicijuojami *k* klasterių centrai;
2. Kiekvienas analizuojamų duomenų aibės objektas priskiriamas tam klasteriui, iki kurio centro atstumas yra mažiausias;
3. Perskaičiuojami kiekvieno klasterio centrai;
4. Skaičiuojama kvadratinė paklaida pagal (4.17) formulę;
5. 2 – 4 punktai kartojami, kol kvadratinės paklaidos reikšmė tampa mažesnė už pasirinktą slenkstinę reikšmę arba objektai nebepersiskirsto kitiems klasteriams.

K klasterių centrai keičia savo pozicijas tol, kol nepasiekiamas tam tikras lokalus optimumas. Metodas minimizuoja paklaidos funkciją:

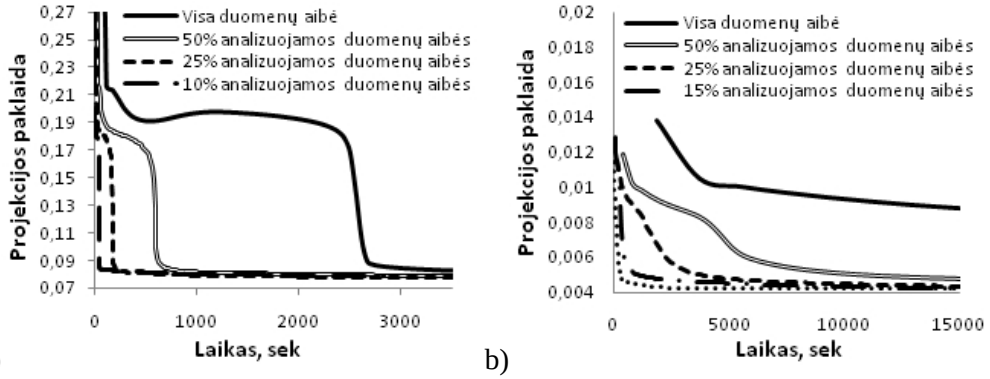
$$E = \sum_{j=1}^{k} \sum_{i=1}^{m} \left\| X_i^{(j)} - C_j \right\|^2, \qquad (4.17)$$

čia $\left\| X_i^{(j)} - C_j \right\|^2$ yra pasirinktas atstumo matas tarp taško $X_i^{(j)}$ ir klasterio centro C_j. Ši paklaidos funkcija rodo kaip toli duomenų aibės taškai yra nuo atitinkamų klasterių centrų.

Mokymo aibė buvo sudaroma iš naujų taškų. Dirbant su k-vidurkių metodu buvo imami klasterių centrai. Klasterių kiekis buvo parenkamas atsižvelgiant į planuojamą mokymo aibės dydį. Projekcijos paklaidos skaičiavimui buvo naudojama originali duomenų aibė.

Dirbant su *1 duomenų aibe* buvo atlikti 4 eksperimentai, parenkant skirtingo dydžio neuroninio tinklo mokymo aibę. Neuroninis tinklas buvo mokomas visa duomenų aibe (966 daugiamačiai taškai) ir sumažintomis mokymo aibėmis, sudarytomis iš naujų taškų – klasterių centrų. Buvo sudarytos 3 sumažintos mokymo aibės: 450 taškų (apie 50% originalios duomenų aibės dydžio), 250 taškų (apie 25% originalios duomenų aibės dydžio), 100 taškų (apie 10% originalios duomenų aibės dydžio).

Dirbant su *2 duomenų aibe* buvo atlikti 5 eksperimentai su skirtingo dydžio neuroninio tinklo mokymo aibėmis. Neuroninis tinklas buvo mokomas visa duomenų aibe (5473 daugiamačiai taškai), ir sumažintomis mokymo aibėmis, sudarytomis iš naujų taškų- klasterių centrų. Buvo sudarytos 4 sumažintos mokymo aibės: 2500 taškų (apie 50% originalios duomenų aibės dydžio), 1400 taškų (apie 25% originalios duomenų aibės dydžio), 900 taškų (apie 15% originalios duomenų aibės dydžio), 400 taškų (apie 7% originalios duomenų aibės dydžio).



4.8 pav. Projekcijos paklaidos priklausomybė nuo skaičiavimo laiko, naudojant skirtingo dydžio mokymo aibę (*k*-vidurkių metodas).
(a) *1 duomenų aibė,* (b) *2 duomenų aibė*

Eksperimentų rezultatai yra pateikiami 4.8 paveiksle. Geriausi rezultatai dirbant su *1 duomenų aibe* buvo pasiekti naudojant 25% originalios duomenų aibės dydžio mokymosi aibę, dirbant su *2 duomenų aibe* geriausi rezultatai buvo pasiekti naudojant 15% originalios duomenų aibės dydžio mokymosi aibę.

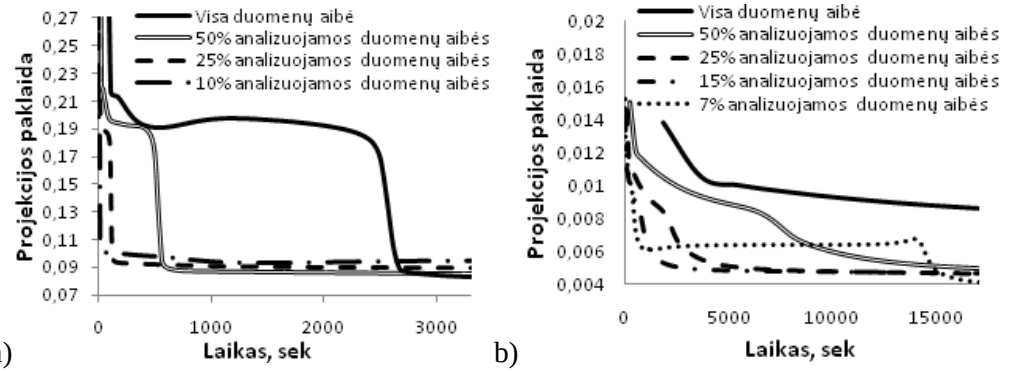
Naudojant SOM pradinės duomenų aibės klasterizavimui, SOM tinklo dydis buvo parenkamas atsižvelgiant į planuojamą mokymo aibės dydį. SOM tinklas buvo apmokomas naudojant originalios duomenų aibės taškus. Konstruojant sumažintą mokymo aibę, imami SOM neuronai-nugalėtojai. Projekcijos paklaida skaičiuojama naudojant originalios duomenų aibės taškus.

Naudojant SOM tinklą mokymo aibės sudarymui, neuroninis tinklas buvo mokomas visa duomenų aibe ir sumažintomis mokymo aibėmis.

Dirbant su *1 duomenų aibe* buvo naudojamos tokio dydžio sumažintos mokymo aibės: 481 taškas (apie 50% originalios duomenų aibės dydžio, naudojamas 25x25 dydžio SOM tinklas), 236 taškai (apie 25% originalios duomenų aibės dydžio, naudojamas 16x16 dydžio SOM tinklas), 95 taškai (apie 10% originalios duomenų aibės dydžio, naudojamas 10x10 dydžio SOM tinklas).

Dirbant su *2 duomenų aibe* buvo naudojamos tokio dydžio sumažintos mokymo aibės: 2140 taškų (apie 50% originalios duomenų aibės dydžio, naudojamas 57x57 dydžio SOM tinklas), 1313 taškų (apie 25% originalios duomenų aibės dydžio,

naudojamas 40x40 dydžio SOM tinklas), 961 taškas (apie 15% originalios duomenų aibės dydžio, naudojamas 33x33 dydžio SOM tinklas), 390 taškų (apie 7% originalios duomenų aibės dydžio, naudojamas 20x20 dydžio SOM tinklas).



4.9 pav. Projekcijos paklaidos priklausomybė nuo skaičiavimo laiko naudojant skirtingo dydžio mokymo aibes (SOM metodas).
(a) *1 duomenų aibė*, (b) *2 duomenų aibė*

Eksperimentų rezultatai yra pateikiami 4.9 paveiksle. Naudojant SOM tinklą mokymo aibės sudarymui dirbant su *1 duomenų aibe* gaunami rezultatai yra šiek tiek blogesni, negu naudojant *k*-vidurkių metodą mokymo aibei sudaryti. Dirbant su *2 duomenų aibe* geri rezultatai yra gaunami naudojant 25% ir 15% originalios duomenų aibės dydžio mokymo aibes. Naudojant 50% ir 7% originalios duomenų aibės dydžio mokymo aibes, gaunami rezultatai yra blogesni už atitinkamus rezultatus, gautus naudojant *k*-vidurkių metodą. Geriausias rezultatas buvo pasiektas naudojant 15% originalios duomenų aibės dydžio mokymo aibę.

Lyginant rezultatus, gaunamus dirbant su *k*-vidurkių metodu ir su SOM, galima teigti, kad k-vidurkių metodas leidžia gauti geresnius rezultatus dirbant su mažesnėmis duomenų aibėmis. Dirbant su didelės apimties duomenų aibėmis abu metodai duoda panašius rezultatus.

Kiekvienas eksperimentas trunka daugiau negu valandą, dirbant su *1 duomenų aibe* ir daugiau negi 5 valandas, dirbant su *2 duomenų aibe*. Skaičiavimai buvo atliekami dvibranduoliniame kompiuteryje. Dviejų lygiagrečių srautų naudojimas leido dvigubai sutrumpinti skaičiavimų laiką.

4.4. Didelių duomenų aibių vizualizavimas naudojant SAMANN neuroninį tinklą

Didelės apimties duomenų aibių analizė, naudojant visą nagrinėjamą duomenų aibę SAMANN tinklo mokymui, yra sudėtingas dėl itin lėto neuroninio tinklo mokymosi proceso. Tačiau tam tikru būdu parenkant tik dalį nagrinėjamos duomenų aibės neuroninio tinklo mokymui, galima ženkliai paspartinti tinklo mokymą. Naudojant klasterizavimo metodus neuroninio tinklo mokymo aibei sudaryti, galima SAMANN tinklu vizualizuoti ir didelės apimties duomenų aibes. Trys realios duomenų aibės buvo naudojamos tiriant galimybes vizualizuoti didelės apimties duomenų aibes su SAMANN tinklu:

1 duomenų aibė. Puslapių klasifikavimo duomenų aibė (Asuncion, 2007). Duomenų aibę sudaro 5473 10-mačiai taškai, sudarantys penkias klases.

2 duomenų aibė. Ranka rašytų skaitmenų atpažinimo duomenų aibė (Asuncion, 2007). Duomenų aibę sudaro 10992 16-mačių taškų, sudarančių 10 klasių (skaičiai nuo 0 iki 9).

3 duomenų aibė. MAGIC teleskopo duomenų aibė (Asuncion, 2007). Duomenų aibę sudaro 19020 10-mačių taškų, sudarančių dvi klases.

Visuose eksperimentuose buvo naudojamas tiesioginio sklidimo neuroninis tinklas su vienu paslėptu sluoksniu ir dviem išėjimo neuronais (d=2). Buvo naudojamas fiksuotas paslėpto sluoksnio neuronų kiekis. Eksperimentų metu buvo skaičiuojamos projekcijos paklaidos ir matuojamas neuroninio tinklo mokymo laikas.

Dirbant su kiekviena duomenų aibe buvo atliekami 3 eksperimentai. Neuroninis tinklas buvo mokomas visa analizuojama duomenų aibe ir naudojant sumažintas mokymo aibes, gautas klasterizuojant analizuojamą duomenų aibę *k*-vidurkių ir SOM metodais. Sumažintos mokymo aibės buvo sudaromos iš klasterių centrų, dirbant su *k*-vidurkių metodu, ir iš neuronų-nugalėtojų, dirbant su SOM metodu. Abiems atvejais mokymo aibės buvo sudaromos iš *naujų* taškų, kurių nebuvo analizuojamoje duomenų aibėje. Sumažintų mokymo aibių apimtys buvo 10 kartų mažesnės už analizuojamos duomenų aibės dydį. Visuose eksperimentuose projekcijos paklaidos skaičiavimui buvo naudojami visi analizuojamos duomenų aibės taškai. Visuose eksperimentuose neuroninio tinklo mokymuisi buvo skiriama apie 20 valandų.

Atliekant eksperimentus su *1 duomenų aibe*, mokant neuroninį tinklą visa duomenų aibe, mokymosi procesas trūko 20,04 valandos. Per tą laiką buvo atlikta tik 116 mokymo iteracijų ir gauta 0,1183 projekcijos paklaida. Naudojant *k*-vidurkių metodą sumažintos mokymo aibės sudarymui, per 20 valandų buvo atliktos 7437 mokymo iteracijos ir gauta 0,0365 projekcijos paklaida. Naudojant SOM metodą sumažintos mokymo aibės sudarymui, buvo apibrėžtas 24x24 SOM tinklas. Mokymo aibė buvo sudaryta iš 559 neuronų-nugalėtojų. Per 20 valandų buvo atlikta 7860 mokymo iteracijų ir gauta 0,036 projekcijos paklaida. Eksperimentų rezultatai

grafiškai pateikiami 4.10 paveiksle. Dirbant su *1 duomenų aibe*, naudojant abu klasterizavimo metodus sumažintos mokymo aibės sudarymui, gaunami žymiai geresni rezultatai, negu naudojant visą analizuojamą duomenų aibę neuroninio tinklo mokymui.

Dirbant su 2 *duomenų aibe*, mokant neuroninį tinklą visa duomenų aibe, mokymosi procesas trūko 20,18 valandos. Per tą laiką buvo atlikta tik 20 mokymo iteracijų ir gauta 0,466 projekcijos paklaida. Naudojant *k*-vidurkių metodą sumažintos mokymo aibės sudarymui, per 20 valandų buvo atlikta 1418 mokymo iteracijų ir gauta 0,303 projekcijos paklaida. Naudojant SOM metodą sumažintos mokymo aibės sudarymui, buvo apibrėžtas 34x34 SOM tinklas. Mokymo aibė buvo sudaryta iš 1068 neuronų-nugalėtojų. Per 20 valandų buvo atliktos 1482 mokymo iteracijos ir gauta 0,308 projekcijos paklaida. Eksperimentų rezultatai grafiškai pateikiami 4.11 paveiksle.

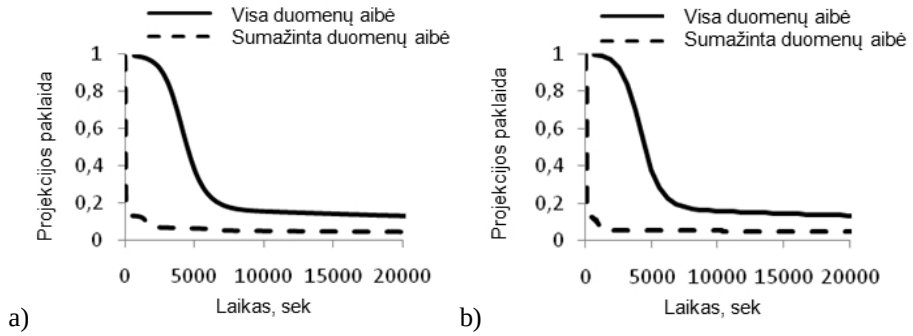
Atliekant eksperimentus su 3 *duomenų aibe*, mokant neuroninį tinklą visa duomenų aibe, mokymosi procesas trūko 20,65 valandos. Per tą laiką buvo atlikta tik 10 mokymo iteracijų ir gauta 0,2758 projekcijos paklaida. Naudojant *k*-vidurkių metodą sumažintos mokymo aibės sudarymui, per 20 valandų buvo atliktos 623 mokymo iteracijos ir gauta 0,086 projekcijos paklaida. Naudojant SOM metodą sumažintos mokymo aibės sudarymui, buvo apibrėžtas 45x45 SOM tinklas. Mokymo aibė buvo sudaryta iš 2015 neuronų-nugalėtojų. Per 20 valandų buvo atlikta 618 mokymo iteracijų ir gauta 0,063 projekcijos paklaida. Eksperimentų rezultatai grafiškai pateikiami 4.12 paveiksle.

Mokant neuroninį tinklą sumažintomis mokymo aibėmis, sudarytomis naudojant *k*-vidurkių ir SOM metodus, gaunami panašūs rezultatai. SOM metodo naudojimo privalumas yra tas, kad šis metodas yra greitesnis, negu *k*-vidurkių metodas. Tačiau, naudojant *k*-vidurkių metodą sumažintos mokymo aibės sudarymui, kartais gaunamos šiek tiek geresnės projekcijos paklaidos reikšmės.

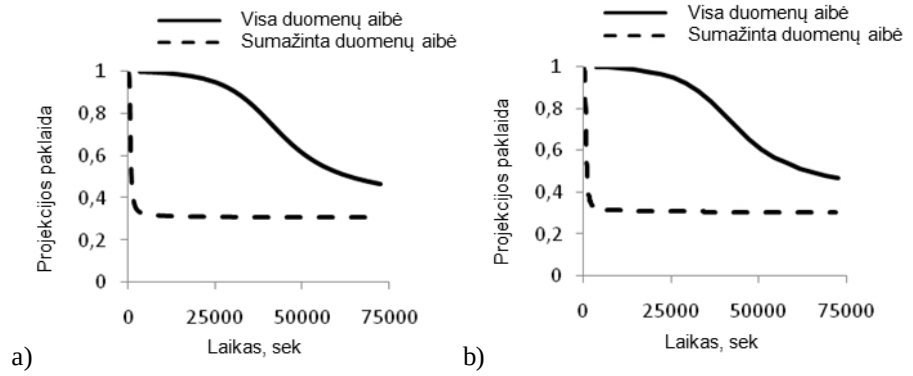
Gauti rezultatai palyginimui yra pateikiami 4.3 lentelėje. Projekcijos paklaidos reikšmės pateikiamos lentelėje buvo gautos po maždaug 3 valandų mokymo. *1 duomenų aibės* vizualizavimo rezultatas yra pateikiamas 4.13 paveiksle.

4.3 lentelė. Eksperimentų rezultatų palyginimas

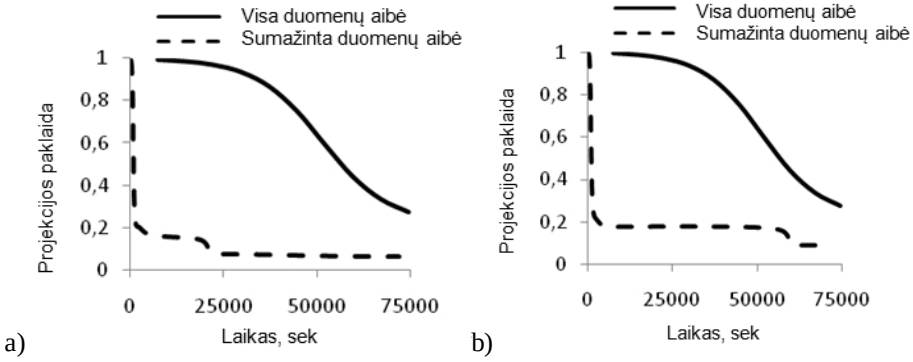
Duomenų aibė	**SAMANN (visa duomenų aibė)**	**SAMANN (sumažinta mokymo aibė)**	
	projekcijos paklaida	projekcijos paklaida (k-vidurkių metodas)	projekcijos paklaida (SOM metodas)
1 duomenų aibė	0,1559	0,0525	0,0476
1 duomenų aibė	0,9898	0,3111	0,3141
1 duomenų aibė	0,9862	0,1755	0,1570



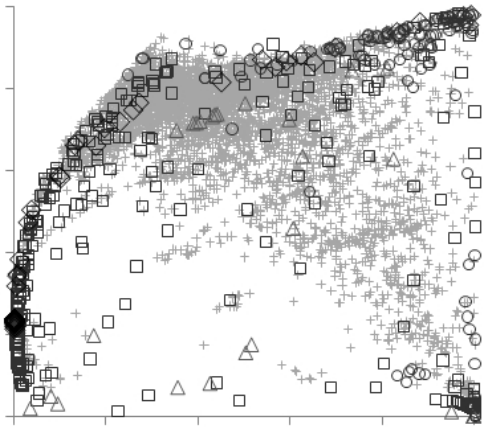
4.10 pav. Projekcijos paklaidos priklausomybė nuo skaičiavimo laiko dirbant su *1 duomenų aibe*, naudojant (a) SOM metodą, (b) *k*-vidurkių metodą



4.11 pav. Projekcijos paklaidos priklausomybė nuo skaičiavimo laiko dirbant su *2 duomenų aibe*, naudojant (a) SOM metodą, (b) *k*-vidurkių metodą



4.12 pav. Projekcijos paklaidos priklausomybė nuo skaičiavimo laiko dirbant su *3 duomenų aibe*, naudojant (a) SOM metodą, (b) k-vidurkių metodą



4.13 pav. *1 duomenų aibės* vizualizavimo rezultatas naudojant SOM klasterizavimo metodą mokymo aibei sudaryti. Penkių klasių taškai žymimi skirtingai

4.5. Išvados

Šiame skyriuje yra pateikiamas specifinis neuroninio tinklo mokymo algoritmas vadinamas SAMANN. Jis leidžia įprastam tiesioginio sklidimo neuroniniam tinklui realizuoti Sammono projekciją. Yra nagrinėjami SAMANN tinklo mokymo ypatumai bei nurodomi parametrai, kurie gali įtakoti mokymo procesą. Pateikiama neuroninio tinklo mokymo priklausomybė nuo neuronų aktyvacijos funkcijos nuolydžio parametro. Parodoma, kad pasirinkus tinkamą nuolydžio parametro reikšmę, galima paspartinti tinklo mokymo procesą ir gauti mažesnę projekcijos paklaidą. Eksperimentų rezultatai parodė, kad didinant aktyvacijos funkcijos nuolydžio parametro reikšmę gaunamas geresnis efektas, negu didinant mokymosi parametro reikšmę.

Dažniausiai neuronų aktyvacijos funkcijos nuolydžio parametro reikšmė prilyginama 1. Tačiau didesnės neuronų aktyvacijos funkcijos nuolydžio parametro reikšmės naudojimas leidžia pagreitinti tinklo mokymo procesą ir tuo pačiu metu gauti geresnę projekcijos paklaidą o tuo pačiu ir geresnį duomenų atvaizdavimą. Eksperimentiškai nustatyta, kad aktyvacijos funkcijos nuolydžio parametro reikšmė turi būti nedidesnė už 50.

Skyriuje pristatomas naujas neuroninio tinklo mokymo aibės sudarymo metodas, kuris leidžia tinklo mokymui naudoti žymiai mažesnę duomenų aibę, negu visa analizuojama duomenų aibė. Atsitiktinai parenkant analizuojamos duomenų aibės taškus į mokymo aibę, gali būti gaunami labai skirtingi rezultatai. Tuo tarpu, sudarant mokymo aibę, klasterizuojant pradinę duomenų aibę, gaunamos mažesnės projekcijos paklaidos, o mokymosi procesas pagreitėja iki 5 kartų, dirbant su mažesnės dimensijos uždaviniais ir daugiau negu 10 kartų, dirbant su didelės dimensijos uždaviniais.

Naudojant pradinės duomenų aibės klasterizavimą sudarant mokymo aibę, labai svarbu tinkamai pasirinkti mokymo aibės dydį. Eksperimentai parodė, kad naudojant 10%–25% pradinės duomenų aibės dydžio mokymo aibę, tinklas mokosi greitai ir gaunama maža projekcijos paklaida. Mažesnių mokymo aibių naudojimas duoda skirtingus rezultatus priklausomai nuo duomenų aibės.

Lyginant rezultatus, gaunamus naudojant mokymo aibę, sudarytą iš atsitiktinai pasirenkamų pradinės duomenų aibės taškų su rezultatais, gaunamais naudojant mokymo aibę, sudarytą klasterizuojant pradinę duomenų aibę, galime padaryti tokias išvadas: mokymo aibės konstravimas, atsitiktinai pasirenkant taškus iš pradinės duomenų aibės, tinka dirbant su pakankamai mažom duomenų aibėm; dirbant su didelėm duomenų aibėm, neuroninio tinklo mokymo aibę geriau sudaryti klasterizuojant pradinę duomenų aibę. Eksperimentai parodė, kad naudojant iš esmės skirtingus klasterizavimo metodus (*k*-vidurkių metodą ir SOM) neuroninio tinklo mokymo aibei sudaryti, gaunami panašūs rezultatai. Naudojant tik pradinės duomenų aibės poaibį SAMANN neuroninio tinklo mokymui, galima analizuoti didelės apimties duomenų aibes.

5

Lygiagrečiosios technologijos ir jų taikymas SAMANN tinklo mokymui

Sprendžiant daugelį šiuolaikinių uždavinių susiduriame su problema, kad dirbant su vienu kompiuteriu negalime laiku gauti atsakymo. Tai dažniausiai atsitinka dėl nepakankamų skaičiuojamųjų resursų arba dėl atminties stokos. Tokiais atvejais gali padėti lygiagretieji skaičiavimai.

Lygiagrečiųjų procesų naudojimas leidžia ženkliai padidinti skaičiavimų efektyvumą, naudojant jau turimas technologijas ir priemones. Pilnaverčiai lygiagretūs procesai realizuojami daugiaprocesorinėse sistemose. Egzistuoja du pagrindiniai daugiaprocesorinių sistemų tipai:

- paskirstytos atminties sistemos;
- bendros atminties sistemos.

2003 metais Intel korporacija išleido naują procesorių su Hyper-Threading technologija, skirtą eiliniam vartotojui. Technologija Hyper-Threading leidžia programinei įrangai vieną fizinį procesorių matyti kaip du virtualius. Vėliau atsirado ir daugiabranduoliniai procesoriai. Nuo to laiko daugiaprocesoriniai kompiuteriai pasidarė prieinami paprastam vartotojui. Lygiagretieji skaičiavimai pasidarė aktualūs ne tik dirbant su superkompiuteriais, bet ir atliekant skaičiavimus, naudojant paprastus personalinius kompiuterius. Šiame skyriuje apžvelgiami lygiagrečiųjų skaičiavimų ypatumai ir galimybės juos taikyti SAMANN neuroninio tinklo mokymui, dirbant su paprastais kompiuteriais.

5.1. Lygiagrečiųjų kompiuterių vystymasis ir jų tipai

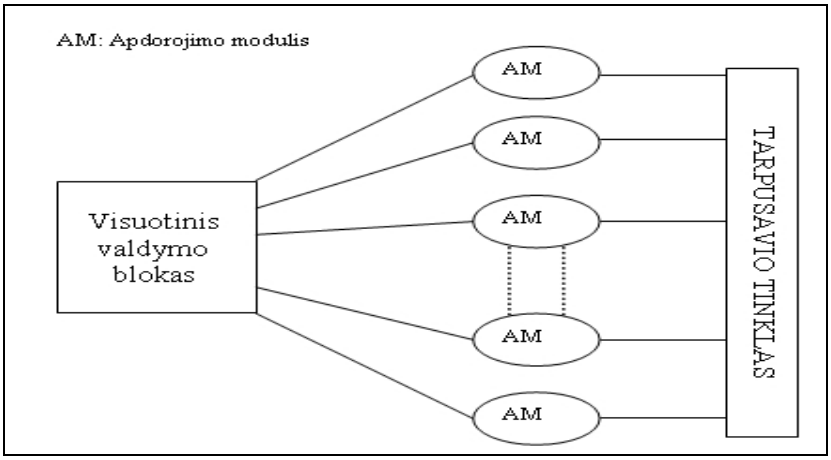
Pirmieji šiuolaikinių kompiuterių analogai atsirado 1950 metais. Jie buvo specializuoti tam tikriems uždaviniams spręsti ir buvo nedidelio galingumo. Didėjant pramonės, o ypač karo pramonės, poreikiams šaltojo karo metu skaičiavimo technika buvo sparčiai vystoma. 1958 metais sukurta pirmoji integralinė schema, o 1971 metais pradėti naudoti mikroprocesoriai. 1976 metais CRAY kompanija sukūrė pirmąjį komercinį vektorinį superkompiuterį CRAY – I. Lyginant su įprastinės architektūros kompiuteriais vektoriniai procesoriai leidžia skaičiavimus pagreitinti nuo 4 iki 20 kartų. Įvairių technikos sričių, pramonės, mokslo šakų kompiuterių spartos poreikis didėjo daug greičiau nei procesorių (net ir vektorinių) sparta. Todėl buvo ieškoma naujų principinių susidariusios problemos sprendimo būdų. Buvo pasinaudota nuo senovės žinomu principu, kad sunkų darbą lengviau įveikti vienu metu dirbant daug darbininkų. Tuo remiasi ir lygiagrečiųjų kompiuterių idėja, kai viename kompiuteryje skaičiavimus atlieka keli procesoriai.

Pirmasis lygiagretusis kompiuteris Illiac IV buvo sukurtas 1974 metais. Jis vienu metu tokią pačią operaciją galėjo atlikti skirtinguose procesoriuose su skirtingais duomenimis. Dabar jau naudojami įvairių tipų lygiagretieji kompiuteriai, juose realizuoti skirtingi konstrukciniai sprendimai. Vienuose kompiuteriuose yra tik keli, tačiau labai galingi procesoriai, kituose kompiuteriuose sujungiami keli šimtai ar net tūkstančiai vidutinio galingumo procesorių. Tokie pat procesoriai yra naudojami ir įprastose darbo stotyse. Tokia technologija iš esmės mažina lygiagrečiųjų kompiuterių kainą (Čiegis, 2005).

Nuo 1990 metų labai paplito virtualieji lygiagretieji kompiuteriai, kurie sukuriami sujungus lokaliuoju tinklu vienos organizacijos darbo stotis ir asmeninius kompiuterius. Tokie kompiuteriai ypač patrauklūs, nereikia jokių papildomų finansinių investicijų, o naudoti galime visus turimus resursus didelių skaičiavimų reikalaujantiems uždaviniams spręsti. Maždaug tais pačiais metais superkompiuterius ir virtualius lygiagrečiuosius kompiuterius bei kompiuterių klasterius pradėjo jungti į tinklynus (angl. grid).

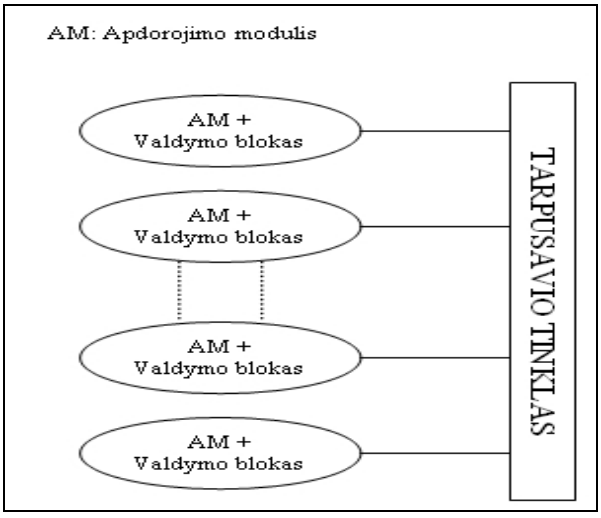
Bendrosios atminties lygiagretieji kompiuteriai

Kai kompiuteryje yra tik vienas valdantis įrenginys ir keli skaičiavimo įrenginiai (žr. 5.1 pav.), tai vykdant programą visi šie skaičiuokliai arba atlieka tokią pačią operaciją su skirtingais duomenimis, arba nevykdo jokių veiksmų. Toks skaičiavimo modelis vadinamas SIMD tipo (*Single Instruction stream – Multiple Data stream*). Tokie lygiagretieji kompiuteriai yra ypač efektyvūs, kai atliekame veiksmus su matricomis, todėl SMID tipo kompiuteriai dar vadinami matriciniais kompiuteriais (Čiegis, 2005).



5.1 pav. SIMD tipo lygiagrečiojo kompiuterio schema

Jeigu kompiuteryje yra keli procesoriai, kurie gali vykdyti skirtingas operacijas, tai turime MIMD tipo (*Multiple Instruction stream – Multiple Data stream*) skaičiavimo modelį (žr. 5.2 pav.). Tokiu kompiuteriu galime realizuoti daug bendresnius algoritmus, tačiau sunkiau sinchronizuoti procesorių darbą (Čiegis, 2005).



5.2 pav. MIMD tipo lygiagretieji kompiuteriai

Visi procesoriai atlieka veiksmus su tam tikrais duomenimis, kuriuos perskaito ir užrašo į jiems skirtas atminties ląsteles. Bendrosios atminties lygiagretieji kompiuteriai turi tik vieną atminties bloką, ir visi procesoriai gali tiesiogiai pasiekti visas atminties ląsteles. Kai duomenų persiuntimo greitis yra vienodas visiems

procesoriams, tai turime kompiuterį su tolygiai pasiekiama bendrąja atmintimi. Tačiau dideliam procesorių kiekiui techniškai sunku realizuoti tokią sąlygą, todėl dažnai atmintis skaidoma į dalis, kurios priklauso skirtingiems procesoriams. Ir šiuo atveju išlieka bendras atminties adresavimas, bet procesoriai greičiau pasiekia duomenis, esančius lokalioje atminties dalyje, nei duomenis, esančius kituose procesoriuose, – tai kompiuteriai, turintys netolygiai pasiekiamą bendrąją atmintį (pasidalintos atminties lygiagretieji kompiuteriai).

Paskirstytos atminties lygiagretieji kompiuteriai

Paskirstytos atminties lygiagretieji kompiuteriai irgi priklauso MIMD tipui. Tačiau čia kiekvienas procesorius gali tiesiogiai perskaityti ir įrašyti tik duomenis, esančius lokalioje atmintyje. Jeigu vykdant algoritmą reikia duomenų, kuriuos saugo kitas procesorius, tai antrasis procesorius turi nusiųsti pirmajam pranešimą su reikalinga informacija. Pranešimo perdavimo mechanizmas yra toks: vienas procesorius siunčia pranešimą, o kitas procesorius, kuriam reikalingi duomenys, laukia, kol ateis pranešimas.

Taigi duomenų mainai paskirstytosios atminties lygiagrečiuosiuose kompiuteriuose yra sudėtingesni nei bendrosios atminties lygiagrečiuose kompiuteriuose. Tačiau šiuo atveju galima išvengti problemos, kuri egzistuoja bendrosios atminties kompiuteriuose, kai keli procesoriai vienu metu bando skaityti ir/arba rašyti į tą pačią atminties vietą. Kadangi visus pakeitimus gali atlikti tik tas procesorius, kurio lokalioje atmintyje saugomi duomenys, tai nesunku kontroliuoti duomenų mainus ir jų eiliškumą (Grama, 2003).

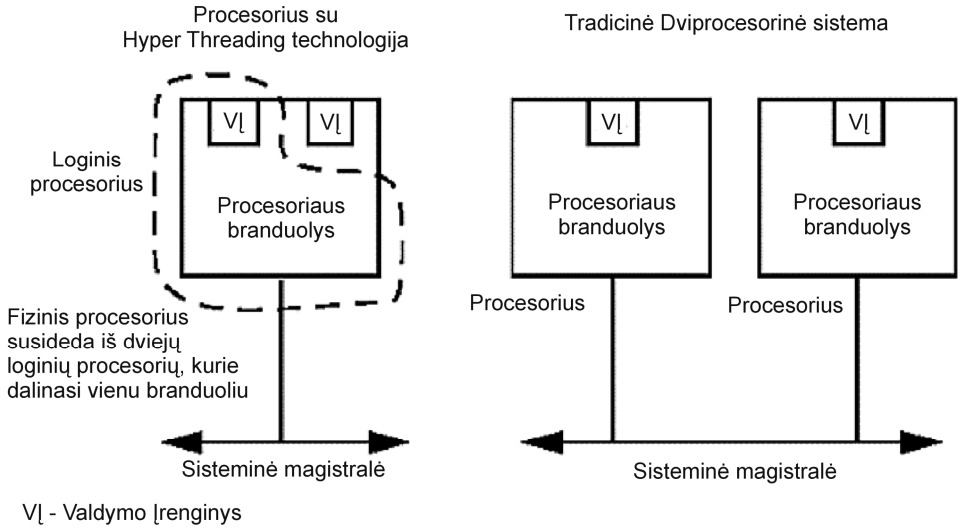
5.2. Lygiagretieji kompiuteriai, skirti plačiam vartotojų ratui

Atsiradus Hyper-Threading technologijai bei daugiabranduoliniams procesoriams, galima teigti, jog lygiagretieji algoritmai tapo neatsiejama kiekvieno kompiuterio dalimi.

Hyper-Threading technologija (HT) atsirado apie 2002 metus naujuose Intel Xeon procesoriuose (skirtuose serveriams ir darbo stotims). Buvo nustatyta, kad paprastas procesorius darbo metu vidutiniškai apkraunamas ne daugiau kaip 60 procentų. Šiuo tikslu Intel korporacijos inžinieriai naujos kartos procesoriuose įdiegė naują technologiją, pavadintą Hyper-Threading. Šios technologijos esmė ta, kad į įprastą procesorių buvo integruotas papildomas valdymo įrenginys. Toks papildymas leidžia operacinei sistemai vieną fizinį procesorių matyti kaip du atskirus loginius procesorius.

Dviprocesorinėje (SMP – Symmetric Multi-Processor) sistemoje abu procesoriai dirba su nuosava spartinančiąja atmintimi (cache) bei registrais, tačiau turi bendrą

darbinę (operatyvinę) atmintį. Kompiuteris, turintis procesorių su Hyper-Threading technologija bei su ta technologija galinčią dirbti sisteminę plokštę, irgi yra dviprocesorinė sistema (virtuali). Procesorius su HT technologija fiziškai yra vienas, bet OS mato jį kaip du. Situaciją paaiškinta 5.3 paveiksle.



5.3 pav. HT procesorius ir dviprocesorinė sistema

Į klasikinio procesoriaus branduolį buvo įdėtas dar vienas papildomas AS-IA32 Architectural State blokas (Valdymo Įrenginys). Šis blokas saugo registrų būsenas (bendro naudojimo, valdančių, APIC bei tarnybinių). Faktiškai pirmas VĮ plius procesoriaus branduolys (šakojimo prognozavimo blokai, ALU – sveikaskaitinio skaičiavimo įrenginys, FPU – slankaus kablelio skaičiavimo įrenginys ir kiti blokai) sudaro pirmą loginį procesorių (LP1), o antras VĮ su tuo pačiu branduoliu sudaro antrą loginį procesorių (LP2). Kiekvienas loginis procesorius (LP) turi nuosavą pertraukimų kontrolerį (APIC – Advanced Programmable Interrupt Controller), tačiau abu loginiai procesoriai dalinasi bendrais registrais ir spartinančiąja atmintimi (cache).

Dirbant su HT technologija galime gauti darbo paspartėjimą, kai procesai naudoja skirtingus procesoriaus blokus, arba kai jie skirtingai dirba su duomenimis, esančiais operatyvioje atmintyje. Jei viena programa kažką skaičiuoja, naudodama lokalius duomenis, o kita nuolat siunčia duomenis iš atminties, tai bendras šių programų vykdymo laikas, naudojant HT, bus mažesnis, net jei šios programos naudoja tuos pačius procesoriaus blokus. Antros programos duomenų siuntimo iš operatyvios atminties komandos gali būti vykdomos, kol pirma programa kažką skaičiuos. Testai parodo, kad HT technologija leidžia iki 30% paspartinti kompiuterio darbą, efektyviau naudodama procesoriaus pajėgumus.

Nuo 2006 metų mikroprocesorių gamintojai siūlo daugiabranduolinius procesorius, prieinamus paprastam vartotojui. Šiuo metu tiek Intel, tiek AMD korporacijos paprastam vartotojui siūlo vienbranduolinius, dvibranduolinius ir keturbranduolinius procesorius. Kai kurie daugiabranduoliniai Intel procesoriai turi ir HT technologiją, tuomet virtualių procesorių skiaičius padvigubėja.

Kompiuteriai su dugiabranduoliniais procesoriais, kaip ir daugiaprocesoriniai kompiuteriai, priklauso SMP (Symmetric Multi Processing) tipui. SMP kompiuteriai tai tokie kompiuteriai, kurie turi du arba daugiau identiškų procesorių, sujungtų bendra sistemine magistrale ir naudojančių bendrą operatyviąją atmintį.

Kadangi daugiabranduoliniai procesoriai tapo prieinami paprastam vartotojui, lygiagretieji skaičiavimai ir lygiagretieji algoritmai pasidarė aktualūs ne tik dirbant su superkompiuteriais, bet ir atliekant skaičiavimus, naudojant paprastus personalinius kompiuterius.

5.3. Lygiagretieji algoritmai

Lygiagretusis algoritmas sudaromas visą užduotį išskaidant į smulkesnes dalis ir paskirstant jas spręsti keliems procesoriams. Optimizuodami lygiagrečiuosius algoritmus, siekiame darbą tolygiai paskirstyti skirtingiems procesoriams, minimizuoti duomenų perdavimo tarp procesorių kaštus bei pritaikyti skaičiavimus kompiuterio tipui bei architektūrai.

Esminiai klausimai, į kuriuos reikia atsakyti kuriant lygiagretųjį algoritmą:

1. Kaip išskaidyti uždavinį į nepriklausomas užduotis?
2. Kokio dydžio užduotis pasirinkti?
3. Kiek reikia procesorių, kad uždavinį išspręstume greičiausiai?
4. Kaip paskirstyti užduotis tarp procesorių?

Sudarydami bet kokį lygiagretųjį algoritmą, pirmiausia turime išskirti nepriklausomas užduotis, kurias skirtingi procesoriai gali spręsti tuo pačiu metu. Kuo didesnis yra tokių užduočių skaičius, tuo daugiau procesorių galime panaudoti uždaviniui spręsti ir tolygiau paskirstyti darbą tarp visų procesorių.

Užduočių paskirstymas procesoriams gali būti atliekamas dviem būdais:

1. Pirmasis būdas – statiškas paskirstymas skaičiavimo pradžioje, pasinaudojant informacija apie užduočių sudėtingumą ir procesorių greitaeigiškumą.

2. Antrasis būdas – dinamiškas darbo paskirstymas skaičiavimo eigoje naudojamas tada, kai užduočių sudėtingumas yra kintamas ar neprognozuojamas ir procesorių greitaeigiškumas gali keistis skaičiavimo metu. Tokiuose algoritmuose realizuojamas atskiras darbo perskirstymo mechanizmas (Čiegis, 2005).

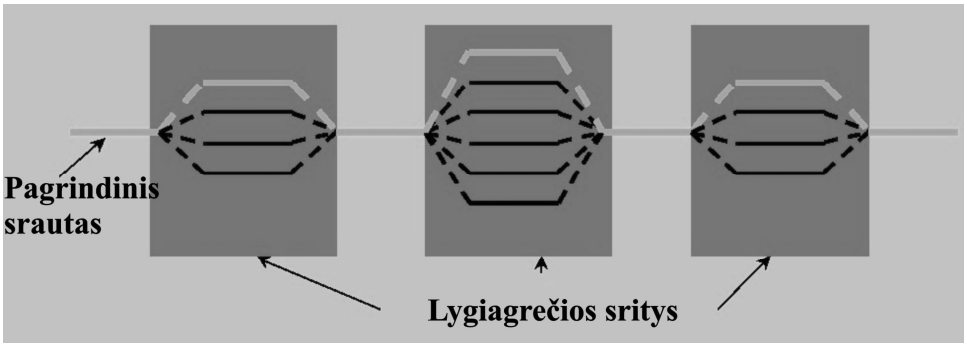
Lygiagrečiųjų algoritmų realizavimo būdai

Lygiagrečiasias programas galima kurti įvairiais būdais, tačiau yra keli populiariausi lygiagrečiųjų programų kūrimo standartai:

1. MPI (Message Passing Interface) biblioteka leidžia sukurti lygiagrečiąją programą, papildant nuoseklų algoritmą darbo paskirstymo ir duomenų valdymo komandomis. MPI biblioteka organizuoja bendravimą tarp procesorių žinučių (message) pagalba. Šis lygiagrečiųjų programų kūrimo būdas yra daugiau tinkamas dirbant su paskirstytos atminties kompiuteriais arba su kompiuterių klasteriais.

2. OpenMP standartas šiuo metu yra vienas populiariausių lygiagrečiųjų kompiuterių su bendrąja atmintimi programavimo būdų. Kuriant lygiagrečiąją programą naudojantis šia biblioteka, kaip pagrindas imama nuoseklioji programa, kuri palaipsniui yra lygiagretinama skirtingose programos kodo vietose, įterpiant atitinkamas direktyvas (Quinn, 2004).

Tiek MPI tiek OpenMP standartai yra naudojami profesionaliose programavimo kalbose (Fortran (77, 90 ir 95), C bei C++).



5.4 pav. OpenMP programos vykdymas. Srautai

Kuriant lygiagrečiąją programą, naudojantis OpenMP standartu, visas programos tekstas suskaldomas į nuosekliąsias ir lygiagrečiąsias sritis. Programos vykdymo pradžioje sukuriamas pagrindinis srautas (srautas-šeimininkas), kuris pradeda programos vykdymą. Tik pagrindinis srautas vykdo visas nuosekliąsias programos sritis. Pradedant vykdyti lygiagrečiąją sritį, sukuriami papildomi srautai. Po sukūrimo kiekvienas srautas gauna unikalų numerį (pagrindinis srautas visada turi numerį 0). Visi srautai vykdo tam tikrą lygiagrečiosios srities kodą. Pasibaigus lygiagrečiajai sričiai, pagrindinis srautas perima valdymą ir toliau tik jis tęsia darbą (5.4 pav.).

Lygiagrečiojoje srityje visi programos kintamieji skirstomi į dvi klases: globalūs (*shared*) ir lokalūs (*private*). Globalus kintamasis yra tiktai vienas visoje programoje, ir visi srautai gali pasiekti jį tuo pačiu vardu. Lokalus kintamasis turi savo atskirą

egzempliorių kiekviename sraute. Jei vienas srautas pakeičia lokalaus kintamojo reikšmę, tai kitame sraute ji nepakinta.

Kuriant lygiagrečiąją programą, naudojantis MPI standartu, turi būti aprašomi veiksmai, kuriuos turi atlikti kompiuteris–šeimininkas (šis kompiuteris atlieka valdymo funkciją; dažniausiai jis paskirsto užduotis tarp kompiuterių–darbininkų, kontroliuoja užduočių atlikimą, o pabaigoje surenka rezultatus), ir veiksmai, kuriuos atlieka kompiuteriai–darbininkai. Taipogi lygiagrečioje programoje turi būti numatytas bendravimas tarp kompiuterių. Bendravimas yra organizuojamas žinučių principu, kai vienas kompiuteris persiunčia kitam reikiamus duomenis.

Lygiagrečiųjų algoritmų naudojimas leidžia ženkliai paspartinti skaičiavimus. OpenMP standartas leidžia kurti lygiagrečias programas, kurias galima vykdyti plačiai paplitusiuose daugiabranduoliniuose ir daugiaprocesoriniuose kompiuteriuose. Programos, sukurtos naudojantis MPI standartu leidžia atlikti skaičiavimus kompiuterių klasteriuose, naudojant daug kompiuterių, sujungtų į tinklą.

Lygiagretūs skaičiavimai buvo plačiai taikomi atliekant inžinerinio uždavinio optimizavimą (Ivanikovas, 2009).

5.4. Hyper-Threading technologijos naudojimo ypatumai

Pradedant darbus su daugiaprocesoriniais kompiuteriais buvo atliktas Hyper-Threading technologijos tyrimas. Buvo nagrinėjama technologijos įtaka dirbant su uždaviniais, intensyviai naudojančiais operatyviąją atmintį. Dirbant su tokiais uždaviniais pagrindinis faktorius, įtakojantis darbo greitį, yra operatyvios atminties sparta. Skaičiavimams naudojant SMP sistemas, atminties greičio įtaka užduoties sprendimo greičiui tampa dar didesnė. Hyper-Threading technologija leidžia operacinei sistemai dirbti su vienu fiziniu procesoriumi kaip su dviem loginiais procesoriais (Ivanikovas, 2005). Tai leidžia padidinti lygiagrečiųjų srautų skaičių ir pagreitinti programos veikimą, optimizuojant techninės įrangos darbą. Norint patikrinti Hyper-Threading technologijos galimybes, dirbant su uždaviniais, intensyviai naudojančiais operatyviąją atmintį, buvo pasirinktas šilumos laidumo uždavinys. Šiame skyriuje pateikiami atlikto tyrimo rezultatai ir parodoma, kaip Hyper-Threading technologija leidžia efektyviau išnaudoti SMP sistemų galimybes, dirbant su dideliais duomenų masyvais.

5.4.1. Darbo su dideliais duomenų masyvais, naudojant SMP sistemas, ypatumai

Operatyvios atminties darbo greitis yra žymiai mažesnis, negu procesoriaus darbo greitis. Todėl netgi vienbranduoliniuose kompiuteriuose procesoriui kartais tenka laukti, kol reikalingi duomenys bus paimti iš operatyvios atminties. Dirbant su SMP sistemomis situacija gali dar pablogėti. Kadangi, kai vienas procesorius dirba su

operatyviąja atmintimi, kiti turi laukti eilėje. Galima situacija, kai keli procesoriai liks be darbo, kol lauks savo eilės pasiekti operatyviąją atmintį. Pasidalintos atminties (NUMA – Non-Uniform Memory Architecture) lygiagretieji kompiuteriai tokių problemų neturi, tačiau šios sistemos yra žymiai brangesnės ir ne taip paplitusios, kaip bendros atminties (UMA – Uniform Memory Architecture) SMP sistemos.

Straipsnyje (Ivanikovas, 2005) buvo parodyta, kad papildomų srautų naudojimas, dirbant su HT technologija ir atliekant paprastus skaičiavimus (slankaus kablelio operacijos, sveikaskaitiniai skaičiavimai), leidžia tik nežymiai pagreitinti skaičiavimus. Tačiau, kai atliekami skirtingo pobūdžio darbai (pvz. skaičiavimas ir darbas su operatyviąja atmintimi), HT technologijos naudojimas leidžia pasiekti geresnių rezultatų.

Lygiagrečiosios programos kūrimui buvo naudojamas OpenMP standartas. Programos buvo testuojamos naudojant kompiuterį su Pentium 4 HT procesoriumi (Pentium 4 HT 3.2GHz, 512 KB L2 cache, 512 MB RAM) ir serverį su dviem Xeon procesoriais (Dual Xeon 3.2 GHz, 1 MB L2 cache, 2GB RAM). Intel Xeon procesoriai irgi turi HT technologiją, todėl šiuo atveju buvo galima dirbti su 4 lygiagrečiais srautais.

Vertinant algoritmo pagreitėjimą ir efektyvumą buvo naudojami tokie kriterijai:

Programos pagreitėjimas: $S_p = \frac{T_1 - T_p}{T_1}$; Algoritmo efektyvumas: $E_p = \frac{T_1}{p \cdot T_p}$,

čia p yra naudojamų srautų kiekis, T_1 – nuoseklios programos vykdymo laikas ir T_p – lygiagrečiosios programos vykdymo laikas dirbant su p srautų (Scott, 2005).

5.4.2. Testinio uždavinio formulavimas

Tiriant Hyper-Threading technologijos galimybes, dirbant su uždaviniais, intensyviai naudojančiais operatyviąją atmintį, buvo pasirinktas šilumos laidumo uždavinys. Tai plačiai naudojamas uždavinys, kurio ypatumas yra didelis apdorojamų duomenų kiekis. Tyrimui buvo pasirinktas uždavinio variantas, kai skaičiuojama plokščios stačiakampės plokštelės temperatūra, kai plokštelė yra šildoma iš kraštų. Uždavinio sprendimui buvo naudojamas baigtinių skirtumų metodas. Plokštelėje buvo apibrėžiamas diskretus tinklelis:

$$w_h = \{(x_i, y_j): x_i = ih, y_j = jh, 0 \le i, j \le N\},$$

kur N yra stulpelių ir eilučių skaičius matricoje w_h , $h = 1/N$ yra tinklelio žingsnis. Temperatūros reikšmės $U_{ij} = U(x_i , y_j)$ yra skaičiuojamos tik tinklelio mazguose. Matricos w_h dydis yra $(N + 1)^2$.

Tinklelio mazgo reikšmės apskaičiavimui naudojami keturi šalia esantys tinklelio mazgai (iš viršaus, iš apačios, iš kairės ir iš dešinės). Gauname tiesinių lygčių sistemą:

$$U_{ij} = (U_{i-1,j} + U_{i+1,j} + U_{i,j-1} + U_{i,j+1}) /4, \ 1 \le i,j \le N\text{-}1.$$

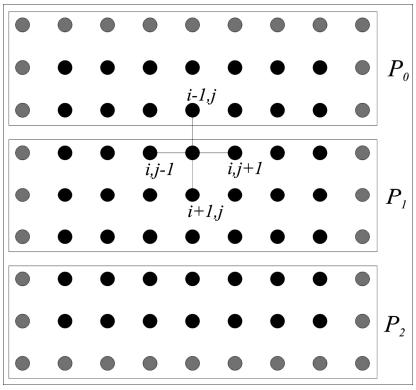
Kraštinių mazgų reikšmės yra paimamos iš sąlygos.

Pateikiama $(N - 1)^2$ tiesinių lygčių sistema yra sprendžiama naudojant Jacobi iteracinį metodą. Pirmiausia pasirenkamos startinės reikšmės U_{ij}^0. Tuomet iteracinis procesas kartojamas tol, kol dviejų iteracijų reikšmės gaunamos pakankamai panašios. Iteracijų kiekis proporcingas tinklelio mazgų kiekiui $O(N^2)$ (Gloub, 1996). Šiam uždaviniui atlikti reikalingi gana paprasti skaičiavimai, tačiau reikia apdoroti didelį duomenų kiekį, todėl šis uždavinys intensyviai naudoja operatyviąją atmintį.

Vykdant programą tie patys skaičiavimai atliekami su visais duomenimis. Todėl mes galime išlygiagretinti algoritmą padalinant tinklelio mazgus tarp procesorių (5.5 paveikslas). Kiekvienas procesorius gauna savo matricos dalį ir atlieka skaičiavimus tik su jam priklausančiais duomenimis. Po kiekvienos iteracijos kaimyniniai procesoriai apsikeičia kraštinių tinklo mazgų reikšmėmis ir tęsia skaičiavimus. Lygiagrečiojo algoritmo schema yra pateikiama 5.1 lentelėje.

5.1 lentelė. Lygiagrečiojo algoritmo schema

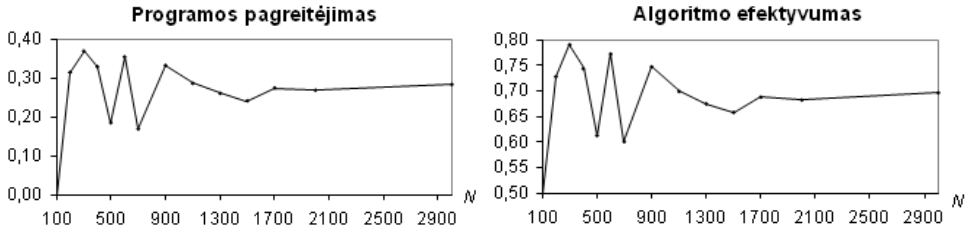
1. Nustatomi pradiniai parametrai: srautų kiekis, matricos dydis (N), tikslumas, ir kraštinės sąlygos.
2. Inicializuojamas OpenMP:
   ```
   #pragma omp parallel
   ```
3. Kiekvienas srautas apibrėžia savo pradinės matricos w_h dalį U_{ij}.
4. paklaida = 2*tikslumas; e = N / srautų kiekis + 2.
5. Kiekvienas srautas apskaičiuoja savo matricos dalį:
   ```
   while(maxpaklaida > tikslumas)
      for(i=1;i<(e-1);i++)
         for(j=1;j<(N+1);j++)
            Uij = (Ui-1,j + Ui+1,j + Ui,j-1 + Ui,j+1 ) /4;
   ```
6. Kiekvienas srautas randa paklaidą, apskaičiuodamas didžiausią skirtumą tarp U_{ij} reikšmių prieš tai buvusioje ir esamoje iteracijoje.
7. Apskaičiuojama maksimali visų srautų paklaida. Šis skaičiavimas yra atliekamas tiktai vieno srauto:
   ```
   maxpaklaida=0.0;
      for(i=0;i<sratų kiekis;i++)
         if (my_ID = = i) if (maxpaklaida<paklaida)
            {#pragma omp critical
            maxpaklaida=paklaida;}
   ```
8. Srautai apsikeičia kraštinių mazgų reikšmėmis U_{ij}.
9. Jei norimas tikslumas nėra pasiektas, kartojami iteraciniai žingsniai nuo 4 iki 7.
10. Matricos w_h dalys, apskaičiuotos skirtingose procesoriuose, yra surenkamos į vieną masyvą.



5.5 pav. Tinklelio mazgų paskirstymas tarp procesorių

5.4.3. Hyper-Threading technologijos testavimo rezultatai

Dirbant kompiuteriu su Pentium 4 HT procesoriumi, buvo atlikti bandymai su nuoseklia programa ir su lygiagrečiąja programa, naudojant du srautus. Testai parodė, kad, naudojant du srautus, mes galime pagreitinti skaičiavimus iki 37%, dirbant su mažesniu duomenų kiekiu ir iki 30%, dirbant su dideliu duomenų kiekiu (5.6 paveikslas).



5.6 pav. Programos pagreitėjimas ir algoritmo efektyvumas
(*N* yra matricos w_h eilučių ir stulpelių skaičius)

Tokiu būdu, dirbant su vienprocesorine sistema su HT technologija, mes galime pagreitinti skaičiavimus, sukuriant papildomus lygiagrečius srautus ir efektyviau naudojant kompiuterio aparatinius resursus. Pateikiamuose grafikuose (5.6 paveikslas) matomi keli maksimumai. Šių maksimumų atsiradimo priežastis yra spartinančiosios atminties naudojimo ypatumai ir šios atminties dydžio ribojimai. Toks pat efektas atsiranda dirbant su dviprocesorine SMP sistema.

Dirbant su dviprocesoriniu serveriu, buvo atlikti bandymai su nuoseklia programa ir su lygiagrečiąja programa, naudojant du, tris ir keturis srautus. Bandymų metų buvo

matuojami programos vykdymo laikai. Testų rezultatai abiems kompiuteriams yra pateikiami 5.2 lentelėje

5.2 lentelė. Programų vykdymo laikai (sek)

N	Pentium 4 HT		Dual Xeon Server			
	Nuosekliai	Lygiagrečiai (2 srautai)	Nuosekliai	Lygiagrečiai (2 srautai)	Lygiagrečiai (3 srautai)	Lygiagrečiai (4 srautai)
100	0,593	0,594	0,688	0,5	0,313	0,204
200	1,891	1,297	2,266	1,062	0,891	0,735
300	4,578	2,89	5,203	2,469	2,344	1,687
400	7,563	5,078	13,813	4,391	3,891	2,234
500	11,437	9,328	25,844	8,563	8,171	5,484
600	17,484	11,297	36,031	16,015	13,203	9,906
700	22,579	18,782	53,281	23,312	19,203	14,531
900	39,156	26,157	80,204	36,907	31,172	25,5
1100	62,047	44,313	138,25	58,469	46,437	33,204
1300	91,391	67,641	198,515	84,25	65,625	54,579
1500	129,422	98,234	258,047	115,063	90,609	74,063
1700	176,188	127,907	331,203	143,891	120,391	100,063
2000	264,078	193,281	424,843	192,719	167,922	158,562
3000	756,266	542,281	1028,484	514,813	419,312	358,125

N yra matricos w_h eilučių ir stulpelių skaičius.

Testai parodė, kad, dirbant su SMP sistema, nuosekli programa yra neefektyvi. Ta pati programa su tais pačiais duomenimis veikė greičiau, dirbant su vienprocesoriniu kompiuteriu. Dirbant su dviem srautais, skaičiavimai pagreitėja iki 68% (5.7 paveikslas). Tokiu būdu didindami srautų skaičių mes padidiname SMP sistemos darbo efektyvumą. Tačiau ir dirbant su dviem srautais, programos vykdymo laikas, naudojant dviprocesorinę sistemą yra palyginamas su programos vykdymo laiku, naudojant vienpocesorinį kompiuterį.

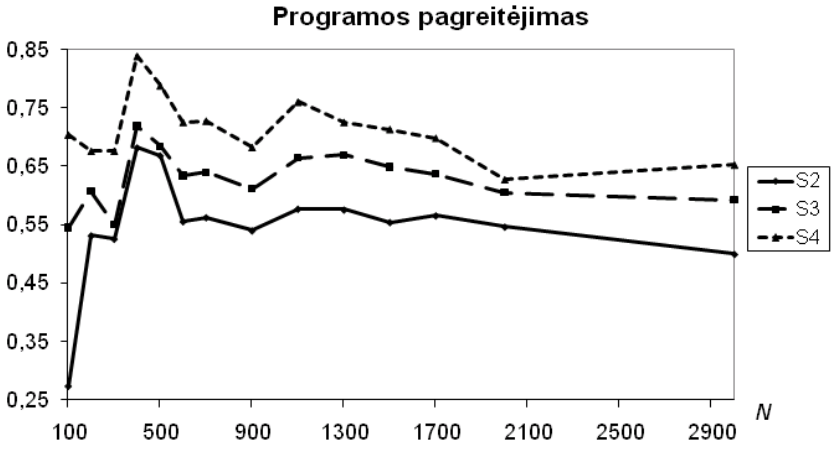
Naudojant Hyper-Threading technologiją mes galime didinti lygiagrečių srautų kiekį iki 4. Taigi dirbant su trim srautais, programos pagreitėjimas siekia 72%, o dirbant su 4 srautais, programos pagreitėjimas siekia 84% (5.7 paveikslas).

Dirbant su dviem srautais, gaunamas algoritmo efektyvumas didesnis už 1 (5.8 paveikslas). Tai reiškia, kad, naudojant du srautus, pagreitėjimą lemia ne tik skaičiavimo resursai, bet ir operatyvios atminties darbo optimizavimas. Panašūs rezultatai buvo pateikiami (Grant, 2007). Dirbant su mažesniu duomenų kiekiu operatyvios atminties darbo optimizavimo efektas yra pakankamai didelis.

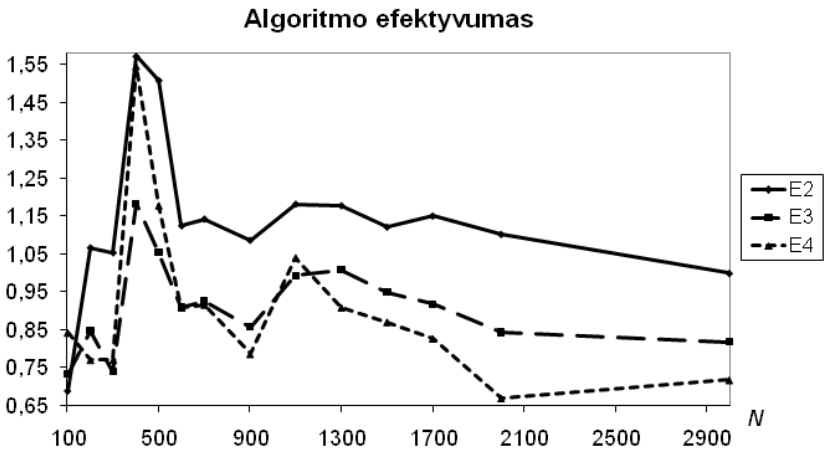
Naudojant daugiau negu 2 lygiagrečius srautus, mes jau nebeturime atskirų procesorių, kurie galėtų šiuos srautus apdoroti (sistema turi tik du fizinius procesorius), tačiau mes gauname programos pagreitėjimą. Testai parodė, kad trijų

srautų naudojimas yra mažiau efektyvus, negu darbas su 4 srautais. Norint gauti geresnius rezultatus, srautų kiekis, dirbant su SMP sistema, turi būti lyginis. Toks srautų kiekis sąlygojamas HT technologijos ypatumais. Kadangi kiekvienas fizinis procesorius yra padalinamas į du loginius procesorius, tai, norint optimizuoti visų procesorių darbą, tenka naudoti lyginį srautų skaičių. Dirbant su 4 srautais yra apkraunami visi 4 loginiai procesoriai, todėl ir programos pagreitėjimas yra didesnis.

Dirbant su mažesniu duomenų kiekiu, algoritmo efektyvumas, naudojant 4 srautus kartais yra didesnis, negu naudojant 3 srautus (N = 500). Dirbant su 3 arba 4 srautais, algoritmo efektyvumas daugumoje atvejų yra mažesnis už 1 (5.8 paveikslas), todėl galima padaryti išvadą, kad šiuo atveju operatyvioji atmintis jau naudojama pakankamai efektyviai.



5.7 pav. Programos pagreitėjimas



5.8 pav. Algoritmo efektyvumas

Galima pastebėti, jog programos pagreitėjimas sumažėja didėjant *N*. Šis efektas atsiranda dėl operatyvios atminties įtakos. Didėjant *N*, operatyvios atminties įtaka skaičiavimams yra mažesnė. Didelėms *N* (*N*>5000) reikšmėms programos pagreitėjimo reikšmė nustoja mažėti ir stabilizuojasi.

Tiek programos pagreitėjimas, tiek algoritmo efektyvumas auga iki *N*=500, o po to pradeda mažėti. Šis efektas atsiranda dėl spartinančiosios atminties įtakos. Kiekvienas sistemos procesorius turi 1MB spartinančiosios atminties. Bendras spartinančiosios atminties dydis dviem procesoriams yra 2MB. Dirbant su realiaisiais skaičiais (double formatas C kalboje naudoja 8 baitus) 500x500 elementų matrica užims maždaug 2MB atminties. Tai yra didžiausia matrica, kuri telpa į 2MB spartinančiąją atmintinę. Dirbant su tokio dydžio arba mažesniais duomenų masyvais, operatyvios atminties naudojimas turi būti minimalus. Spartinančiosios atminties naudojimo ypatumai sąlygoja ir lokalius maksimumus grafikuose 5.7 ir 5.8 paveiksluose. Maksimumai matomi, kai *N* yra nuo 1000 iki 1200. 1000 yra 500 kartotinis, todėl galima šio maksimumo atsiradimo priežastis yra spartinančiosios atminties įtaka.

Panašiai, ir dirbant su vienprocesorine sistema, mes turėjome maksimumus, kai *N* buvo lygus 300, 600 ir 900. Dirbant su didesniais duomenų masyvais, padidėja operatyvios atminties įtaka ir spartinančiosios atmintinės poveikis tampa santykinai nežymus. Todėl mes neturime maksimumų esant didesnėms *N* reikšmėms.

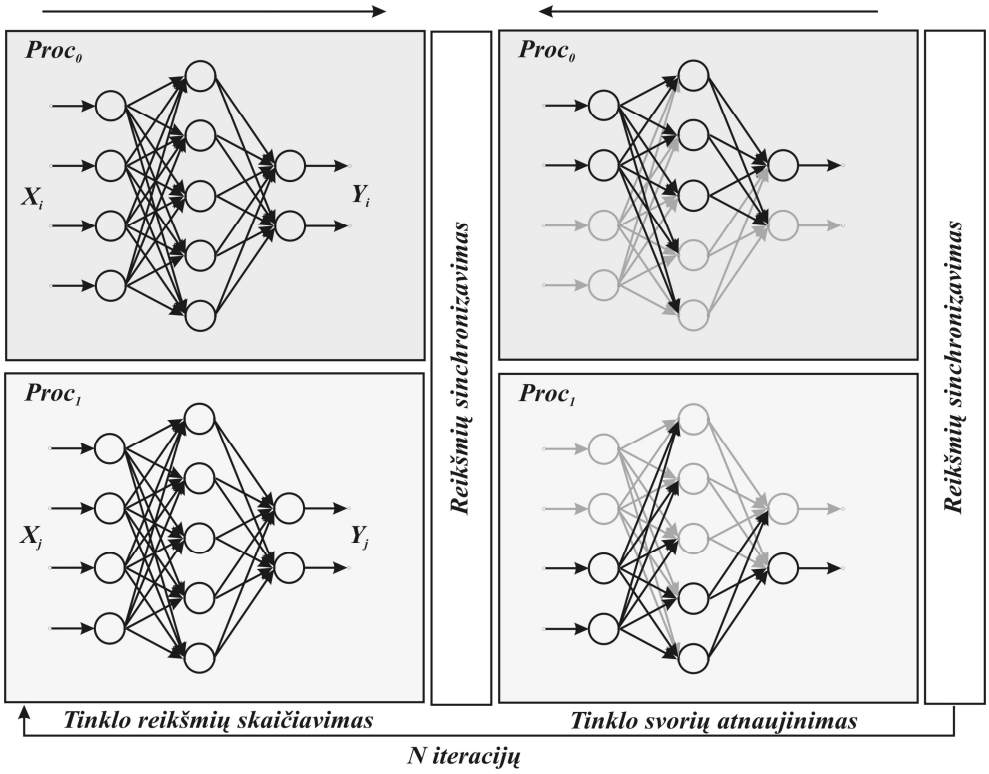
5.5. Lygiagrečiojo SAMANN algoritmo realizacija

Naudojant įprastus algoritmus, SAMANN tinklo mokymas reikalauja didelių skaičiuojamųjų resursų ir užima daug laiko (de Ridder, 1997). Vienas iš būdų pagreitinti neuroninio tinklo mokymą yra lygiagrečiųjų skaičiavimų taikymas.

Vizualizuojant daugiamačius duomenis į plokštumą, SAMANN neuroninis tinklas turi du išėjimo neuronus. Tuomet, mokant neuroninį tinklą, jį galima padalinti į dvi dalis (bendruoju atveju neuroninį tinklą galima dalinti ir į daugiau dalių). Kiekvienas tinklo sluoksnis yra padalinamas į lygias arba beveik lygias dalis. Tuomet neuroninio tinklo mokymo procesą galima išlygiagretinti pritaikant jį bendros atminties kompiuteriams. Kadangi daugiabranduoliniai procesoriai šiuo metu yra labai paplitę, siūlomas lygiagretusis tinklo mokymo algoritmas leis pagreitinti SAMANN tinklo mokymą, dirbant su paprastu kompiuteriu, turinčiu daugiabranduolinį procesorių.

Lygiagretinant SAMANN mokymo algoritmą, buvo naudojamas OpenMP programavimo standartas. Lygiagretusis algoritmas naudoja du srautus, atliekant neuroninio tinklo mokymą. Tinklo mokymo procesas padalinamas į dvi dalis: pradžioje vienas srautas atlieka duomenų įvedimą, taškų normavimą, pradinį tinklo svorių nustatymą; tolimesni veiksmai (tinklo reikšmių skaičiavimas, svorių atnaujinimas) yra atliekami lygiagrečiai dviem srautais. Tokiu būdu lygiagretinamas yra pats tinklo mokymo procesas. Kiekvienas iš dviejų naudojamų procesorių dirba tik

su savo neuroninio tinklo dalimi. Šis algoritmas, skirtingai nuo darbe (Medvedev, 2004) siūlomo algoritmo, lygiagrečiai atlieka neuroninio tinklo mokymą. Darbe (Medvedev, 2004) lygiagrečiai yra mokomi keli skirtingi neuroniniai tinklai.



5.9 pav. SAMANN lygiagrečiojo algoritmo schema

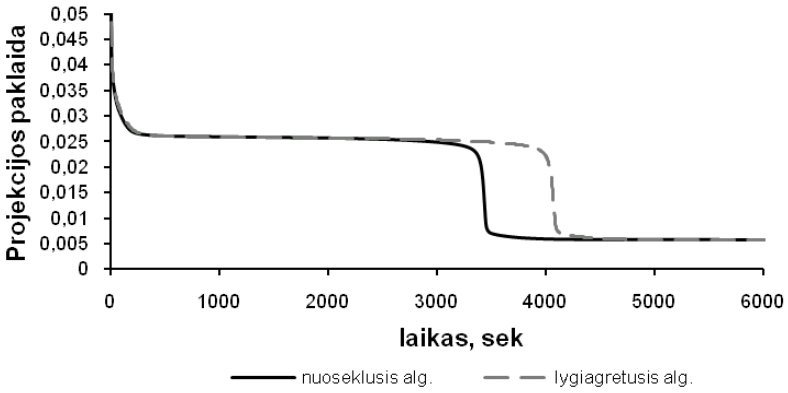
Mokant neuroninį tinklą, *d*-mačiai taškai pateikiami poromis (X_i, X_j), $i, j = 1, 2,\ldots m, i \neq j$, kur m yra bendras mokymo imties taškų skaičius. Lygiagrečiojo algoritmo atveju kiekvienas taškas pateikiamas atskiram procesoriui, ir tinklo reikšmės yra skaičiuojamos lygiagrečiai. Toliau tinklo svoriai yra atnaujinami, naudojant klaidos sklidimo atgal algoritmą. Svorių atnaujinimas atliekamas irgi lygiagrečiai. Kiekvienas procesorius perskaičiuoja tik savo tinklo dalies svorius. Skaičiavimų duomenys yra sinchronizuojami po tinklo reikšmių skaičiavimo ir po kiekvieno tinklo sluoksnio svorių atnaujinimo. Neuroninis tinklas yra apmokomas naudojant tam tikrą mokymo iteracijų skaičių.

Dėl duomenų sinchronizavimo pasiūlytas algoritmas yra neefektyvus, dirbant su mažu neuroniniu tinklu arba su nedidelės dimensijos duomenų aibe. Taipogi šis algoritmas nėra tinkamas paskirstytos atminties kompiuteriams dėl didelių duomenų persiuntimo kaštų.

Eksperimentams su naujuoju algoritmu buvo pasirinktos dvi duomenų aibės:

1. Irisų duomenų aibė (4-mačiai taškai) (Fisher, 1936).
2. Sonaro duomenų aibė (Gorman, 1988). Tai yra reali duomenų aibė, sudaryta iš 208 60-mačių taškų. Šioje aibėje yra 2 klasės: pirmoje klasėje yra 111 taškų (šiai klasei priklauso taškai, gaunami sonaro signalui atsispindint nuo metalinio cilindro), antroje klasėje yra 97 taškai (šiai klasei priklauso taškai, gaunami sonaro signalui atsispindint nuo uolų).

Algoritmo tyrimui buvo naudojamas tiesioginio sklidimo dirbtinis neuroninis tinklas su vienu paslėptu sluoksniu ir dviem išėjimo neuronais (d = 2). Visuose eksperimentuose buvo naudojami tie patys neuroninio tinklo pradiniai svoriai. Dirbant su irisų duomenų aibe buvo naudojami tokie tinklo parametrai: paslėpto sluoksnio neuronų skaičius n_2=20, mokymosi parametras $h = 0,5$, momentum reikšmė 0,05. Dirbant su sonaro duomenų aibe, paslėpto sluoksnio neuronų skaičius buvo keičiamas (n_2=500), o visi kiti parametrai buvo naudojami tie patys.



5.10 pav. Duomenų projekcijos paklaidos priklausomybė nuo skaičiavimo laiko irisų duomenų aibei.

Atliekant eksperimentus, lygiagrečiojo algoritmo rezultatai buvo lyginami su nuoseklaus algoritmo rezultatais. Tiriant algoritmų veikimą, buvo matuojami analizuojamų duomenų projekcijos paklaidos ir programų vykdymo laikai. Skaičiavimams buvo naudojamas kompiuteris su Dual Xeon dvibranduoliniu procesoriumi. Rezultatai, gauti dirbant su irisų duomenų aibe, yra pateikiami 5.10 paveiksle ir 5.3 lentelėje. Lentelės paskutiniame stulpelyje pateikiamas lygiagretaus algoritmo vykdymo laiko ir nuoseklaus algoritmo vykdymo laiko santykis. Dirbant su irisų duomenų aibe, lygiagretusis algoritmas dirba lėčiau už nuoseklųjį, tai yra mažesnė projekcijos paklaida gaunama greičiau naudojant nuoseklųjį algoritmą. Taip atsitinka dėl to, kad apmokant neuroninį tinklą 4-mačiais irisų duomenimis, duomenų sinchronizavimo kaštai yra pakankamai dideli. Dirbant su mažos dimensijos

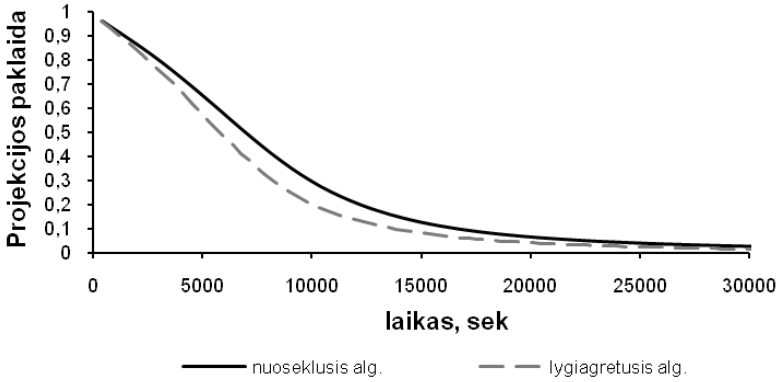
duomenimis ir naudojant nedidelį neuroninį tinklą, skaičiavimų kiekis kiekviename neuroninio tinklo mokymo žingsnyje yra pakankamai mažas, todėl laiko sąnaudos duomenų sinchronizavimui yra santykinai didelės.

5.3 lentelė. Programų vykdymo laikai ir jų santykis, dirbant su *1 duomenų aibe*

Mokymosi iteracijos numeris	Paklaida	Nuoseklaus algoritmo vykdymo laikas (sek)	Lygiagretaus algoritmo vykdymo laikas (sek)	Santykis
2000	0,026057	450	531	1,178
4000	0,025867	901	1062	1,179
6000	0,025770	1351	1598	1,183
8000	0,025661	1805	2133	1,182
10000	0,025497	2255	2670	1,184
12000	0,025192	2705	3204	1,184
14000	0,024429	3157	3738	1,184
16000	0,006443	3609	4274	1,184
18000	0,005801	4059	4814	1,186
20000	0,005712	4512	5349	1,185
22000	0,005696	4963	5882	1,185
24000	0,005688	5415	6413	1,184
26000	0,005682	5865	6947	1,184
28000	0,005676	6316	7481	1,185
30000	0,005671	6768	8017	1,185
32000	0,005666	7218	8569	1,187
34000	0,005661	7670	9109	1,188
36000	0,005657	8121	9648	1,188
38000	0,005653	8572	10187	1,188
40000	0,005649	9024	10734	1,190
42000	0,005646	9475	11280	1,191
44000	0,005642	9926	11825	1,191
46000	0,005639	10377	12366	1,192
48000	0,005635	10828	12904	1,192

Atlikus eksperimentus su sonaro duomenų aibe, gauname kitus rezultatus. Šiuo atveju lygiagretusis algoritmas dirba greičiau už nuoseklųjį. Dirbant su 60-mačiais duomenų taškais ir turint 500 neuronų paslėptame sluoksnyje, skaičiavimų apimtis kiekviename neuroninio tinklo mokymo žingsnyje yra pakankamai didelė. Todėl lygiagrečiojo algoritmo efektyvumas padidėja. Rezultatai, gauti dirbant su sonaro duomenų aibe, yra pateikiami 5.11 paveiksle ir 5.4 lentelėje. Lentelės paskutiniame stulpelyje pateikiamas lygiagretaus algoritmo vykdymo laiko ir nuoseklaus algoritmo vykdymo laiko santykis. Matome, kad lygiagretusis algoritmas veikia apie 20% greičiau, negu nuoseklusis. Dirbant su didelės dimensijos duomenų aibe arba turint daug neuronų paslėptame neuroninio tinklo sluoksnyje, pasiūlytas lygiagretusis algoritmas tampa efektyvus.

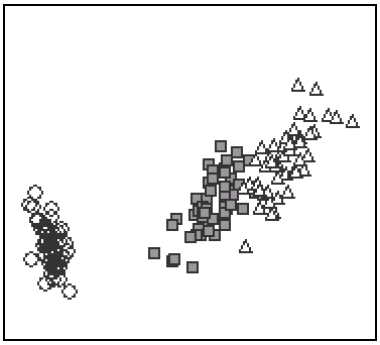
Irisų duomenų aibės vizualizavimo rezultatas pateikiamas 5.12 paveiksle.



5.11 pav. Duomenų projekcijos paklaidos priklausomybė nuo skaičiavimo laiko sonaro duomenų aibei.

5.4 lentelė. Programų vykdymo laikai ir jų santykis, dirbant su *2 duomenų aibe*

Mokymosi iteracijos numeris	Paklaida	Nuoseklaus algoritmo vykdymo laikas (sek)	Lygiagretaus algoritmo vykdymo laikas (sek)	Santykis
4	0,8494	2274	1883	0,828
8	0,7241	4081	3402	0,834
12	0,5841	5900	4898	0,830
16	0,4439	7717	6388	0,828
20	0,3234	9519	7874	0,827
24	0,2326	11337	9351	0,825
28	0,1692	13170	10846	0,824
32	0,1262	14973	12390	0,827
36	0,0968	16775	13852	0,826
40	0,0763	18579	15373	0,827
44	0,0615	20396	16853	0,826
48	0,0506	22211	18318	0,825
52	0,0423	24018	19770	0,823
56	0,0358	25823	21177	0,820
60	0,0307	27633	22577	0,817
64	0,0265	29433	23955	0,814
68	0,0230	31234	25360	0,812
72	0,0201	33038	26727	0,809
76	0,0176	34833	28099	0,807
80	0,0155	36640	29475	0,804
84	0,0137	38433	30879	0,803
88	0,0121	40226	32250	0,802
92	0,0107	42016	33668	0,801
96	0,0095	43809	35079	0,801



5.12 pav. Irisų duomenų aibės projekcija į plokštumą, naudojant SAMANN tinklą.

5.6. Išvados

Šiame skyriuje yra pateikiamos pagrindinės lygiagrečiojo skaičiavimo idėjos, apžvelgiami lygiagrečiųjų kompiuterių tipai. Apžvelgiamos šiuolaikinės daugiaprocesorinės ir daugiabranduolinės sistemos, prieinamos paprastam vartotojui. Parodoma, kad lygiagretieji skaičiavimai tapo neatsiejama kompiuterinės technikos panaudojimo sritimi.

Šiame skyriuje pateikiama Hyper-Threading technologijos apžvalga ir šios technologijos tyrimas, dirbant su užduotimis, intensyviai naudojančiomis operatyviąją atmintį. Eksperimentams buvo naudojamas personalinis kompiuteris su vienbranduoliniu Pentium 4 HT procesoriumi ir dviprocesorinis Xeon serveris. Hyper-Threading technologija leido dirbti su dviem srautais, naudojant vienbranduolinį procesorių ir su keturiais srautais, naudojant dviprocesorinį serverį. Testai parodė, kad Hyper-Threading technologija leidžia pagreitinti vienprocesorinės sistemos darbą iki 30%, optimizuojant operatyviosios atminties veikimą. Dirbant su daugiaprocesorine SMP sistema, net naudojant du srautus, programos vykdymo laikas yra palyginamas su vienprocesorinės sistemos rezultatu (naudojamų procesorių parametrai yra panašūs). HT technologija leidžia pagreitinti programos vykdymą iki 85% ir efektyviau išnaudoti procesoriaus resursus.

Dirbant su dviem loginiais procesoriais vietoj vieno galima padidinti sistemos darbo efektyvumą ir dalinai išspręsti bendros atminties naudojimo problemą. Kol vienas loginis procesorius atlieka skaičiavimus, kitas gali dirbti su operatyviąja atmintimi.

Pasiūlyta lygiagrečioji SAMANN neuroninio tinklo mokymo algoritmo realizacija, kuri atlieka neuroninio tinklo mokymą, padalinus jį į kelias dalis. Ši realizacija leidžia greičiau apmokyti neuroninį tinklą, dirbant su didelės dimensijos duomenų aibėmis. Pasiūlyta realizacija yra neefektyvi, dirbant su Hyper-Threading

technologija, tačiau ji parodo gerus rezultatus dirbant SMP sistemomis. Lygiagrečiojo algoritmo naudojimas, dirbant su mažesnės dimensijos duomenimis, yra neefektyvus. Tačiau dirbant su mažesnės dimensijos duomenimis, mes galime apmokyti neuroninį tinklą ir naudojant įprastą nuoseklų algoritmą. Tuo tarpu, dirbant su didelės dimensijos duomenų aibėm, mokymosi procesas yra labai lėtas ir gali trūkti dešimtys valandų. Šiuo atveju tikslinga naudoti pasiūlytą lygiagretųjį algoritmą, kuris leidžia pagreitinti tinklo mokymą.

6

Bendrosios išvados

1. Analizuojant SAMANN neuroninio tinklo mokymo be mokytojo „klaidos sklidimo atgal“ algoritmą, nustatyta, kaip projekcijos paklaida ir tinklo mokymo konvergavimas priklauso nuo pasirinktų parametrų reikšmių. Vienas iš parametrų, įtakojančių neuroninio tinklo mokymąsi, yra neuronų sigmoidinės aktyvacijos funkcijos nuolydžio parametras. Dažniausiai šio parametro reikšmė imama lygi 1. Tačiau eksperimentai parodė, kad, naudojant didesnę neuronų aktyvacijos funkcijos nuolydžio parametro reikšmę, pavyksta greičiau pasiekti gerus vizualizavimo rezultatus.

2. Nustatyta, kad optimali SAMANN tinklo neuronų aktyvacijos funkcijos nuolydžio parametro reikšmė yra intervale (10;30). Pasirenkant tokias aktyvacijos funkcijos nuolydžio parametro reikšmes, galima žymiai sumažinti skaičiavimų trukmę (iki 3–5 ir net daugiau kartų) ir gauti gerus vizualizavimo rezultatus per trumpesnį laiką, esant fiksuotam mokymo iteracijų skaičiui.

3. Eksperimentai parodė, kad galima rasti tokį analizuojamos duomenų aibės poaibį, kuriuo mokant SAMANN tinklą, mažesnės projekcijos paklaidos gaunamos greičiau, negu tinklo mokymui naudojant visus aibės taškus.

4. Buvo pasiūlytas SAMANN tinklo mokymosi aibės sudarymo metodas, grindžiamas analizuojamos duomenų aibės klasterizavimu. Naudojantis šiuo metodu galima sukonstruoti tokią neuroninio tinklo mokymo aibę, kad tinklas mokysis žymiai greičiau (daugiau negu 5 kartus greičiau), negu naudojant visą analizuojamą duomenų

aibę, o projekcijos paklaida bus neblogesnė už projekcijos paklaidą, gaunamą mokant neuroninį tinklą visa analizuojama duomenų aibe.

5. Atlikti eksperimentai parodė, kad, naudojant pasiūlytą SAMANN tinklo mokymo aibės sudarymo strategiją, galima analizuoti žymiai didesnės (daugiau negu 10 kartų didesnės) apimties duomenų aibes, negu mokant tinklą visa analizuojama duomenų aibe.

6. Ištirtos galimybės tinklo mokymui naudoti daugiaprocesorines sistemas ir Hyper-Threading technologiją. Atliktas HT technologijos tyrimas parodė, kad ji leidžia efektyviau išnaudoti kompiuterio resursus, ypač dirbant su daugiaprocesorinėmis sistemomis. Tačiau šios technologijos ypatumai neleido efektyviai panaudoti jos, dirbant su lygiagrečiuoju SAMANN algoritmu.

7. Pasiūlyta lygiagrečioji SAMANN neuroninio tinklo mokymo algoritmo modifikacija, kuri atlieka tinklo mokymą, padalinant jį į kelias atskiras dalis. Tyrimai parodė, kad skaičiuojant lygiagrečiuoju algoritmu, galima pasiekti geresnius vizualizavimo rezultatus per trumpesnį laiką (lyginant su nuosekliuoju algoritmu), vizualizuojant didelės dimensijos duomenų aibes.

Literatūros sąrašas

1. Ahalt, A.; Krishnamurthy, A. K.; Chen, P.; Melton, D. E. (1990). Competitive Learning Algorithm for Vector Quantization, Neural Networks, vol. 3: 277–290.
2. Asuncion, A.; Newman, D. J. (2007). UCI Machine Learning Repository [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, School of Information and Computer Science.
3. Baldi, P.; Hornik, K. (1989). "Neural networks and principal component analysis: Learning from examples without local minima". in *IEEE Trans. Neural Networks*, vol. 2: 53–58.
4. Belkin, M.; Niyogi, P. (2001). "Laplacian eigenmaps and spectral techniques for embedding and clustering". *Advances in Neural Informatikon Processing Systems*, Vol. 14. Ed. T. G. Dietterich, S. Becker, and Z. Ghahramani. Cambridge, MA: MIT Press, 585–591.
5. Belkin, M.; Niyogi, P. (2003). "Laplacian eigenmaps for dimensionality reduction and data representation", *Neural Comput.*, vol.15(6): 1373–1396.
6. Bezdek, J. C.; Pal, N. R. (1995). "Index of Topological Preservation for Feature Extraction", *Pattern Recognition* 28(3): 381–391.
7. Blake, C. L.; Hettich, S.; Merz, C. J. (1998). *UCI repository of machine learning databases*. Irvine, CA. University of California, Department of Information and Computer Science. [žiūrėta 2007-07-04]. Prieiga per internetą: <http://www.ics.uci.edu/~mlearn/MLRepository.html>

8. Broomhead, D. S.; Lowe, D. (1988). “Multivariable functional interpolation and adaptive networks”, *Complex Systems* 2: 321–355.

9. Borg, I.; Groenen, P. (2005). *Modern Multidimensional Scaling: Theory and Applications. Second Edition.* New York: Springer.

10. Čiegis, R. (2005). *Lygiagretieji algoritmai ir tinklinės technologijos.* Vilnius: Technika.

11. Comon, P. (1994). “Independent component analysis – a new concept?”, in *Signal Processing*, Vol. 36: 287–314.

12. Demartines, P.; Herault, J. (1997). “Curvilinear Component Analysis: A Self-Organizing Neural Network for Nonlinear Mapping of Data Sets”, *EEE Trans. Neural Networks*, Vol. 8(1).

13. Duda, R. O.; Hart, P. E. (1973). *Pattern Recognition and Scene Analysis.* John Wiley.

14. Duda, R. O.; Hart, P. E.; Stork, D. G. (2000). *Pattern Classification.* 2nd Edition. John-Wiley.

15. Dzemyda, G. (2001). “Visualization of a set of parameters characterized by their correlation matrix”, *Computational Statistics and Data Analysis*, 36(1): 15–30.

16. Dzemyda, G. (2005). “Multidimensional data visualization in the statistical analysis of curricula”, *Computational Statistics and Data Analysis*, Vol. 49: 265–281.

17. Dzemyda, G.; Kurasova, O.; Medvedev, V. (2007). “Dimension Reduction and Data Visualization Using Neural Networks”. *Emerging Artificial Intelligence Applications in Computer Engineering. Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies.* Vol. 160, Frontiers in Artificial Intelligence and Applications. Ed. I. Maglogiannis, K. Karpouzis, M. Wallace and J. Soldatos. IOS Press, 25–49.

18. Dzemyda, G.; Kurasova, O.; Žilinskas, J. (2008). *Daugiamačių duomenų vizualizavimo metodai*. Mokslo aidai, Vilnius.

19. Fisher, R. A. (1936). “The Use of Multiple Measurements in Taxonomic Problems”, *Annals of Eugenics*, Vol. 7: 179–188.

20. Flexer, A. (2001). “On the use of self-organizing maps for clustering and visualization”, Intelligent-Data-Analysis, Vol. 5(5): 373–384.

21. Fukunaga, K. (1990). *Introduction to Statistical Pattern Recognition.* 2 ed. Academic Press.

22. Gloub, G. H.; van Loan, C. F. (1996) “Matrix computations”. *The Johns Hopkins University Press*, New York.

23. Gorman, R. P.; Sejnowski, T. J. (1988) “Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets”, in Neural Networks, Vol. 1, (1988) 75-89

24. Graham, R. L.; Hell, P. (1985). “On the history of the minimum spanning tree problem”, *Annals of the History of Computing* 7: 43–57.

25. Grama, A.; Gupta, A.; Karypis, G.; Kumar, V. (2003) “Introduction to Parallel Computing, 2nd Ed” Addison Wesley.

26. Grant, R. E.; Afsahi A. (2007) “A Comprehensive Analysis of Multithreaded OpenMP Applications on Dual-Core Intel Xeon SMPs, Workshop on Multithreaded Architectures and Applications”, *(MTAAP'07), In Proceedings of the 21st International Parallel and Distributed Processing Symposium (IPDPS 2007)*, Long Beach, California, USA.

27. Harman, H. H. (1967). *Modern Factor Analysis*. 2nd edition. University of Chicago Press.

28. Hassoun, M. H. (1995). *Fundamentals of Artificial Neural Networks*. A Bradford Book. London: the MIT Press.

29. Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation*. 2nd ed. Upper Saddle River, New Jersey: Prentice Hall.

30. Hyvarinen, A.; Karhunen, J.; Oja, E. (2001). *Independent Component Analysis*. John Wiley & Sons.

31. Huber, P. J. (1985). “Projection Pursuit”, *The Annals of Statistics*, Vol. 13(2): 435–474.

32. Idrissi, M. J.; Sbihi, A.; Touahni, R. (2004). “An improved neural network technique for data dimensionality reduction in remotely sensed imagery”, *International Journal of Remote Sensing*, 25(10): 1981–1986.

33. Ivanikovas, S., Dzemyda G. (2005) “Lygiagrečiojo programavimo asmeniniais kompiuteriais ypatumai”, *Informacijos mokslai*, T. 34, p. 257-262.

34. Ivanikovas, S., Medvedev, V.; Dzemyda, G. (2007). “Parallel Realizations of the SAMANN Algorithm”, *Lecture Notes in Computer Science, Adaptive and Natural Computing Algorithms*, Vol. 4432: 179–188.

35. Ivanikovas S., Filatovas E., Žilinskas J., (2009) “Experimental Investigation of Local Searches for Optimization of Grillage-Type Foundations” *Springer optimization and its applications. Vol. 27, Parallel scientific computing and optimization : advances and applications*, New York, Springer, p. 103-112

36. Jain, A. K.; Mao, J.; Mohiuddin K. (1996). “Artificial Neural networks: a Tutorial”, *IEEE Computer*, Vol. 29(3): 31–44.

37. Jin, Y.; Ma M. (2000). “A neural-network dimension reduction method for large-set pattern classification”. in *ICMI ’00: Proceedings of the Third International Conference on Advances in Multimodal Interfaces*. Springer-Verlag, 426–433.

38. Jollife, I. T. (1986). *Principal Component Analysis*. Springer-Verlag.

39. Jones, M. C.; Sibson, R. (1987). “What is projection pursuit?”, *J. of the Royal Statistical Society*, ser. A, 150: 1–36.

40. Kaski, S., (1997). *Data Exploration Using Self-Organizing Maps PhD thesis*. Helsinki University of Technology, Department of Computer Science and Engineering, [žiūrėta 2007-06-04]. Prieiga per internetą: <http://www.cis.hut.fi/~sami/thesis/>

41. Klock, H.; Buhmann, J. M. (1999). „Data visualization by multidimensional scaling: A deterministic annealing approach", *Pattern Recognition*, Vol. 33(4): 651–669.

42. Kohonen, T. (2001). *Self-Organizing Maps*. 3nd ed. Springer series in information sciences. Springer-Verlag. Vol. 30.

43. Kohonen, T. (2002). "Self-Organizing Neural networks: Recent Advances and Applications", *Studies in Fuzziness and Soft Computing*, Heidelberg, New York: Physica-Verl. Ed U. Seiffert, L.C. Jain, Vol.78: 1–11.

44. Konig, A. (2000). "Interactive visualization and analysis of hierarchical neural projections for data mining", *IEEE transactions on neural networks*, Vol. 11(3).

45. Kraaijveld, M. A.; Mao, J.; Jain, A. K. (1995). "A Nonlinear Projection Method Based on Kohonen's Topology Preserving Maps", *IEEE Transactions on Neural Networks*, Vol. 6(3): 548–559.

46. Kramer, M. A. (1991). "Nonlinear principal component analysis using autoassociative neural networks", *AIChE Journal*, Vol. 37: 233–243.

47. Kruskal, J. B. (1956). "On the shortest spanning subtree of a graph and the travelling salesman problem", in *Proceedings of the American Mathematical Society*, Vol. 7: 48–50.

48. Kruskal, J. B. (1972). "Linear transformations of multivariate data to reveal clustering", *Multidimensional Scaling: Theory and Application in the Behavioural Sciences, I, Theory*. New York and London: Seminar Press.

49. Kruskal, J. B.; Wish, M. (1984). *Multidimensional Scaling*. Beverly Hills and London: Sage Publications.

50. Kvedaras, B.; Sapagovas, M. (1974). *Skaičiavimo metodai*. Mintis.

51. Kurasova, O., (2005). *Daugiamačių duomenų vizuali analizė taikant savireguliuojančius neuroninius tinklus*. Matematikos ir Informatikos Institutas (daktaro disertacija).

52. Lee, R. C. T.; Slagle, J. R.; Blum, H. (1977). "A triangulation method for the sequential mapping of points from N-space to two-space", *IEEE Transactions on Computers*, 288–299.

53. de Leeuw, W.; van Liere, R. (2003). „Visualization of Multi Dimensional Data using Structure Preserving Projection Methods", *Data Visualization: The State of the Art*, 213–223.

54. Lowe, D.; Tipping, M. E. (1996). "Feed-forward neural networks and topographic mappings for exploratory data analysis", *Neural Computing and Applications*, Vol. 4: 83–95.

55. Lowe, D.; Tipping, M. E. (1997). “Neuroscale: novel topographic feature extraction using RBF networks”, In Mozer,M.C., *Advances in Neural Information Processing Systems*, Ed. by M. Jordan, T. Petsche. London: MIT Press. 543–549.

56. MacQueen, J. B. (1967), “Some Methods for classification and Analysis of Multivariate Observations”, *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*, Berkeley, University of California Press, 1, p. 281–297.

57. Mao, J.; Jain, A. K. (1995). “Artificial neural networks for feature extraction and multivariate data projection”, *IEEE Transactions on Neural Networks*, Vol. 6(2): 296–317.

58. Mardia, K. V.; Kent, J. T.; Bibby J. M. (1995). *Multivariate Analysis. Probability and Mathematical Statistics*. Academic Press.

59. Mathar, R.; Žilinskas, A. (1993). “On Global Optimization in Two-Dimensional Scaling”, *Acta Aplicandae Mathematicae*, Vol. 33: 109–118.

60. McCulloch, W. S.; Pitts, W. (1943). “A logical calculus of the ideas immanent in nervous activity”, *Bulletin of Mathematical Biophysics*, Vol. 5: 115–133.

61. Medvedev, V.; Dzemyda, G. (2004). “Lygiagreti SAMANN vizualizavimo algoritmo realizacija”, *Informacinės technologijos 2004*, konferencijos pranešimo medžiaga. Kaunas: Technologija. 344–350.

62. Medvedev, V.; Dzemyda, G. (2005). “SAMANN neuroninio tinklo mokymo problemos”. *Informacinės technologijos 2005*, konferencijos pranešimo medžiaga. Kaunas: Technologija. 394–399, 2005.

63. Medvedev, V.; Dzemyda G. (2006). “Optimization of the local search in the training for SAMANN neural network”, *Journal of Global Optimization*, Springer, Vol. 35(4): 607–623.

64. Montvilas, A. M. (2003a). “Features of Sequential Nonlinear Mapping”. Informatica. 14(3): 337–348.

65. Montvilas, A. M. (2003b). “Investigation of Sequential Mapping of Multidimensional Data”. *Electronics and Electrical Engineering*. Kaunas: Technologija. No.6(48): 7–12.

66. Montvilas, A. M. (2006). “Data Sructure Influence on Mapping Error”, *Electronics and Electrical Engineering*, 1(65): 34–37.

67. Moody, J.; Darken, C. (1989). “Fast learning in networks of locally-tunned processing units”, *Neural computation*, Vol. 1: 281–294.

68. Noel, N. C. (1998). *Dynamical Embedding and Feature Extraction of Electrocephalographic Data*. Ph.D thesis, Aston University, Aston Street, Birmingham B4 7ET, UK.

69. Oja, E. (1991). “Data compression, feature extraction, and autoassociation in feedforward neural networks”, *Artificial Neural Networks*, 737–745.

70. Pekalska, E.; De Ridder, D.; Duin, R. P. W.; Kraaijveld, M. A. (1999). “A new method of generalizing Sammon mapping with application to algorithm speed-up”, in *Proc. of the 5th Annual Conference of the Advanced School for Computing and Imaging ASCI’99Boasson*, Ed. by J. A. Kaandorp, J. F. M. Tonino. Delft, 221–228.

71. Piatetsky-Shapiro, G.; Smyth, P; Uthurusamy, R.; Fayyad, U. M. (1996). “Advances in Knowledge Discovery and Data Mining”, AAAI/MIT Press.

72. Podlipskytė, A. (2003). *Daugiadimensinių duomenų vizualizacija ir jos taikymas biomediciniių duomenų analizei*. Daktaro disertacija. Kauno technologijos universitetas.

73. Powell, M. (1987). “Radial basis functions for multivariable interpolation: A review”, *Algorithms for Approximation*, 143–167.

74. Press, W. H.; Teukolsky, S. A.; Vetterling, W. T.; Flannery, B. P. (1992). *Numerical Recipes in C.* Second ed. Cambridge: Cambridge University Press.

75. Prim, R. C. (1957). “Shortest connection networks and some generalizations”, *Bell System Technical Journal*, Vol. 36: 1389–1401.

76. Quinn, M. J. (2004). *Parallel Programming in C with MPI and OpenMP*. NewYork: McGraw-Hill Inc.

77. Raudys, Š. (2001). *Statistical and neural Classifiers: an Integrated Approach to Design.* Advances in pattern recognition, Springer-Verlag.

78. de Ridder, D.; Duin, R.P.W. (1997). “Sammon's mapping using neural networks: A comparison”, *Pattern Recognition Letters*, Vol. 18: 1307–1316.

79. Roberts, S.; Everson, R. (2000). *Independent Component Analysis: Principles and Practice*. Cambridge: Cambridge University Press.

80. Roweis, S. T.; Saul, L. K. (2000). “Nonlinear dimensionality reduction by locally linear embedding”, *Science* 290(5500) 2323–2326.

81. Sammon, J. W. (1969). “A nonlinear mapping for data structure analysis”. *IEEE Transactions on Computers*, C-18(5): 401–409.

82. Sanger, T. D. (1989). “Optimal Unsupervised Learning in a Single-Layer Linear Feedforward Neural Network”, *Neural Networks*, 2: 459–473.

83. Saund, E. (1989). “Dimensionality-reduction using connectionist networks”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 11: 304–314.

84. Scott R.; Clark T.; Bagheri B. (2005) “Scientific Parallel Computing”, Princeton University Press

85. Shepard, R. N. (1962a). “The analysis of proximities: Multidimensional scaling with an unknown dstance function”, *I. Psychometrika* 27(2): 125–140.

86. Shepard, R. N. (1962b). “The analysis of proximities: Multidimensional scaling with an unknown distance function”, *II. Psychometrika* 27(3): 219–246.

87. Silipo, R. (2003). “Neural networks”, *Intelligent Data Analysis*, Ed. by M. Berthold and D.J. Hand, 270–320.

88. de Silva, V.; Tenenbaum, J. B. (2002). “Unsupervised learning of curved manifolds”. In *Proceedings of the MSRI workshop on nonlinear estimation and classification*. New York: Springer-Verlag. 453–466.

89. Takatsuka, M. (2001). “An application of the Self-Organizing Map and interactive 3-D visualization to geospatial data”, in *Proceedings of the 6th International Conference on GeoComputation.*

90. Taylor, P., (2003). *Statistical Methods. Intelligent Data Analysis: an Introduction.* Ed. by M. Berthold, D. J. Hand. Springer-Verlag. 69–129.

91. Tenenbaum, J. B.; de Silva, V.; Langford, J. C. (2000). “A global geometric framework for nonlinear dimensionality reduction”, *Science*, 290(5500): 2319–2323.

92. Tipping, M. E. (1996). *Topographic mappings and feed-forward neural networks*. Ph.D thesis, Aston University, Aston Street, Birmingham B4 7ET, UK.

93. Ultsch, A.; Siemon, H. P. (1990). “Kohonen's Self-Organizing Feature Maps for Exploratory Data Analysis”, in *Proc. of INNC'90, International Neural Network Conference*, Dordrecht, Netherlands, 305–308.

94. Usui, S.; Nakauchi, S.; Nakano, M. (1991). “Internal color representation acquired by a five-layer neural network”, *Artificial Neural Networks*, 867–872.

95. Verikas, A.; Gelžinis, A. (2003). *Neuroniniai tinklai ir neuroniniai skaičiavimai.* Kaunas: Technologija.

96. Wilkinson, J. H. (1965). *The Algebraic Eigenvalue Problem*. Claredon Press, Oxford, England.

97. Yang, L. (2004). “Sammon’s nonlinear mapping using geodesic distances”, in *ICPR’04: Proceedings of the Pattern Recognition, 17th International Conference on (ICPR’04)*, Vol. 2, Washington, DC, USA, IEEE Computer Society, 303–306.

98. Žilinskas, A.; Podlipskytė, A. (2003). “On multimodality of the SSTRESS criterion for metric multidimensional scaling”, *Informatica*, Vol. 14(1): 121–130.

Autoriaus publikacijų sąrašas disertacijos tema

Straipsniai tarptautiniuose periodiniuose leidiniuose, įtrauktuose į Mokslinės informacijos instituto pagrindinį sąrašą (*ISI Web of Science*)

1A. Ivanikovas S., Dzemyda. G., (2007) Evaluation of the Hyper-Threading Technology for Heat Conduction-Type Problems *Mathematical Modelling and Analysis* Volume 12 Number 3, p. 459–468.
ISSN 1392-6292 print, ISSN 1648-3510 online

Straipsniai tarptautiniuose periodiniuose leidiniuose, įtrauktuose į Mokslinės informacijos instituto konferencijos darbų (*ISI Proceedings*) sąrašą

2A. Ivanikovas S., Dzemyda G., Medvedev V., (2007). Parallel Realizations of the SAMANN Algorithm. *Lecture Notes in Computer Science,* Adaptive and Natural Computing Algorithms, Springer, Vol. 4432, p. 179–188. ISSN 0302-9743.

3A. Ivanikovas S., Dzemyda G., Medvedev V., (2008) Large Datasets Visualization with Neural Network Using Clustered Training Data. *Lecture Notes in Computer Science,* Advances in Databases and Information Systems, Springer, Vol. 5207, p. 143–152. ISSN 0302-9743.

Straipsnis užsienio leidyklos (*Springer*) knygoje

4A. Ivanikovas S., Filatovas E., Žilinskas J., (2009) Experimental Investigation of Local Searches for Optimization of Grillage-Type Foundations *Springer optimization and its applications. Vol. 27, Parallel scientific computing and optimization : advances and applications*, New York, Springer, p. 103-112. ISBN 978-0387-09-706-0.

Konferencijų pranešimų medžiagoje (ISI Proceedings)

5A. Ivanikovas S., Dzemyda G., Medvedev V., (2008) Neural network-based visualization using clustered data. *EURO Mini Conference „Continuous Optimization and Knowledge-Based Technologies" EurOPT'2008*, konferencijos pranešimo medžiaga, Vilnius, Technika, p. 335–341. ISBN 978-9955-28-283-9.

6A. Ivanikovas S., Dzemyda G., Medvedev V., (2009) Influence of the neuron activation function on the multidimensional data visualization quality. *The XIII International Conference Applied Stochastic Models and Data Analysis ASMDA–2009*, konferencijos pranešimo medžiaga, Vilnius, Technika, p. 299–303. ISBN 978-9955-28-463-5.

Parengė spaudai ir išleido Matematikos ir informatikos institutas
Akademijos 4, LT-08663 Vilnius.
Svetainė internete: www.mii.lt
Spausdino leidyklos „Technologija" spaustuvė,
Studentų g. 54, LT-54124, Kaunas