O’ZBEKISTON RESPUBLIKASI ALOQA, AXBOROTLASHTIRISH VA TELEKOMMUNIKASIYA TEXNOLOGIYALARI DAVLAT QO’MITASI

TOSHKENT AXBOROT TEXNOLOGIYALARI UNIVERSITETI SAMARQAND FILIALI

Axborot va pedagogik texnologiyalar fakulteti

“Informatika va axborot texnologiyalari” kafedrasi

Qo`l yozma xuquqida

UDK

JUMANOV VOXID

TASVIRLARNI TANIShDA NEYRON TARMOQLARINING MODELI, ALGORITMI VA DASTURIY VOSITALARINI KO’PYaDROLI PROSESSORLAR MUHITIDA IShLAB ChIQISh

Mutaxassislik: **5A330201** – Kompyuter tizimlari va ularning dasturiy ta’minoti (tarmoqlar bo’yicha) *bo`yicha magistr akademik darajasini olish uchun yozilgan*

D I S S E R T A T S I Y A

Ilmiy rahbar:

_____dots. Temirov O.

Samarqand - 2013

Kirish

Mavzuning dolzabligi. Hozirgi vaqtda sun’iy neyron to’rlari va masalalarni parallel ishlash ustida nazariy izlanishlar va amaliy qo’llanishlar keskin rivojlanmoqda. Neyron to’rlar analitik tavsifi bo’lmagan va faqatgina eksperimental ma’lumotlar bilan berilgan katta ko’lamdagi amaliy masalalarni yechish imkonini beradi[1].

Neyron to’rlarini sintez qilishda algoritmlarning nozik tomoni bu qaror qabul qilishni tushuntirish bo’lib hisoblanadi. Bu muammoni yechish bilan ko’pchilik tadqiqotchilar shug’ullanmoqdalar. Bu maqsadda ishlatadigan usullar evristik bo’lganligi uchun ular asosida korrekt qaror qabul qilish foydalanuvchining subektiv mulohazasiga bog’liq bo’ladi.

Ko’p o’lchovli chiziqsiz optimizasiyaning an’anaviy iterativ gradiyent algoritmlari bilan o’rganadigan neyron to’rlari modellarining eng ko’p tarqalgani - bu ko’p qatlamli sun’iy neyron to’rlari sinfidir. Ma’lumki, ko’p qatlamli sun’iy neyron to’rlari o’rganishda iterativ algoritmlar yaqinlashuvi, o’rganiladigan berilganlarning (tanlovning) hajmiga, vaznlarning boshlang’ich qiymatiga, shuningdek, o’rganishdagi maksimal xatolarga (o’rganishning sifat mezonlariga), o’rganishdagi takrorlanishlar soniga (o’rganish vaqtining uzayishi mezonlariga) bog’liq.

Shuning uchun, qo’yilgan masalani yechish uchun optimal modellarni tanlashda ularni solishtirish va qaror qabul qilishda neyroto’rlarning xususiyatlarini yetarli darajada baholashga imkon beruvchi xususiy va umumiy mezonlar majmuasini ishlab chiqish zarur.

Bilimlarni ajratib olish algoritmlarini va sifat jihatdan yangi bosqichdagi, kognitolog mutaxassislarga mo’ljallangan, neyron to’rlarining programma vositalarini yaratishga asos bo’luvchi yangi g’oyalar zarur.

Hozirda keng tarqalgan xatolarning teskari tarqalish algoritmlarida va Xopfild neyron to’rlarida qaror qabul qilish jarayonini tushuntirishga harakatlar qilindi. Bu modellardagi algoritmlarning evristik xarakterda ekanligi qaror qabul qilishda neyron

to’rlarining shaffoflik muammosini yechishni yetarli darajada matematik formallashtirishga imkon bermaydi. Natijada, tasvirlarni ajratib olish neyron to’rlari bo’yicha mutaxassisga bog’liq va asosan tavsiya xususiyatiga ega bo’ladi.

Ayni paytda, sun’iy neyron to’ri sohasidagi olimlar tomonidan turli xil amaliy masalalarni yechishda neyromodellarni solishtirishga va tanlashga asos bo’ladigan, ko’p qatlamli neyron to’rlarining mantiqiy shaffofligini miqdoriy baholaydigan bir nechta mezonlar va usullar ishlab chiqilgan.

Ishning maqsadi va vazifalari. Umumlashgan ko’rsatkichlarni hisoblash orqali tajriba ma’lumotlar bazasidan tasvirlarni ajratib olish va ularni ifodalash va modellarda parallel ishlov berishni joriy etish tadqiqot maqsadi hisoblanadi.

Standart ravishda qo’yilgan obrazlarni anglash masalasi qaraladi. Ikkita o’zaro kesishmaydigan K_1, K_2 sinflar vakillarini o’z ichiga olgan $E_0 = \{S_1,...,S_m\}$ obektlar to’plami berilgan deb hisoblanadi. Obektlar n ta turli toifadagi (miqdoriy va sifat) alomatlar bilan tavsiflangan bo’lib, ularning ξ tasi intervallarda (I to’plam), $n-\xi$ tasi nominal (J to’plam) o’lchamlarda o’lchanadi. O’ng’aylik uchun, K_1 sinf vakillarini ro’y bergan holatlar (holatlar) va K_2 - ro’y bermagan holatlar (no holatlar) deb hisoblaymiz.

Ikki sinfli masala qaralishiga sabablardan biri – har qanday obektning umumlashgan bahosi nisbiydir, u qarama-qarshi sinf obektlariga qiyoslash natijasida yuzaga keladi. Ikkinchidan, har qanday k (k>2) sinfli masalani ikki sinfli masalalar kaskadi ko’rinishida yechish mumkin.

Har bir miqdoriy alomat uchun, chegaralarida “holat” yoki “no holat” sinfi ustun bo’lgan intervallarni tanlash masalasi tadqiq qilinadi. Ixtiyoriy mumkin bo’lgan obektning miqdoriy alomatining qiymati ustunlik intervallarining birortasiga ham tushmagan holati mazkur tadqiqotda qaralmaydi.

Tadqiqot obyekti va predmeti. Umumlashgan ko’rsatkichlarni hisoblash orqali tajriba ma’lumotlar bazasidan tasvirlarni ajratib olish va ularni modellarda ifodalash jarayoni va uning natijalari medisina, geologiya, sosiyologiya sohalarining

asosiy masalasi hisoblanadi va shuning uchun yaratilgan algoritm hamda dasturiy ta’minotdan ushbu sohalarda foydalanish mumkin[1,2,3,4,5].

Tadqiqot uslubiyati va uslublari. Tadqiqot uslubiyati neyron to’rlaridagi turli toifadagi, miqdoriy va nominal alomatlar fazosida umumlashgan ko’rsatkichlarni hisoblash imkonini beruvchi ustunlik intervallari usuli, Xopfeld usuli qo’llaniladi va olingan bilimlarni noaniq mantiq modelida tavsiflash amalga oshiriladi.

Tadqiqot natijalarining ilmiy jihatdan yangilik darajasi. Maxsus dasturiy vositalarning kamligi, ushbu soha nisbattan yangi soha ekanligi va yaratilgan dasturiy vosita yangiligi jihatidan farqlanadi.

Tadqiqot natijalarining amaliy ahamiyati va tadbiqi. Yaratilgan dasturiy vositalarni tasvirlarni tanishda va ularga parallel ishlov berish natijasida davturiy vositani ish samaradorligi bir necha marta oshirildi. Xopfeld tarmog’i orqali tasvirlarni tanishning parallel ishlovchi dasturiy vositalari kompyutkrlarni ishlash vaqti tejaladi.

Ish tuzilishi va tarkibi. Ushbu ish kirish qismi, 4 ta bob, xulosa, foydalanilgan adabiyotlar ro’yxati hamda ilovadan iborat. Birinchi bobda, sun’iy neyron to’rlari va ularning qo’llanilish sohalari ko’rsatilgan. Ikkinchi bobda, Xopfild va Ximminglarning neyron to’rlari va algoritmlari berilgan. Uchinchi bobda, ma’lumotlarni parallel qayta ishlash va undan foydalanish yo’riqnomalari keltirilgan. Turtinchi bobda, dasturiy vositadan foydalanish ko’rsatilgan. Xulosa qismida, bajarilgan ishning asosiy natijalari va uning qo’llanilish sohalari ko’rsatilgan.

Bajarilgan ishning asosiy natijalari. Bosh masaladan kelib chiqqan holda ishda quyidagi masalalar hal etildi:

1. Tajriba ma’lumotlarini tarkibiy tuzilishini aniqlash (alomatlar, obektlar berilishi) va ularni berilganlar bazasi ko’rinishida shakllantirildi.
2. Miqdoriy va nominal alomatlarni shkalaga akslantirish va barcha alomatlar uchun yaqinlik matrisasini qurildi.

3. Tanlov obektlarini sinflarga ajralganlik darajasining umumlashgan bahosi hisoblanadi.
4. Algoritmlar tuzish va ularni dastur ko’rinishida amalga oshirildi.
5. Olingan natijalarni tahlil qilindi.
6. Xopfeld tarmog’i orqali tasvirlarni tanishning dasturiy vositalari yaratildi.

Xulosa va takliflarning qisqacha umumlashtirilgan ifodasi.

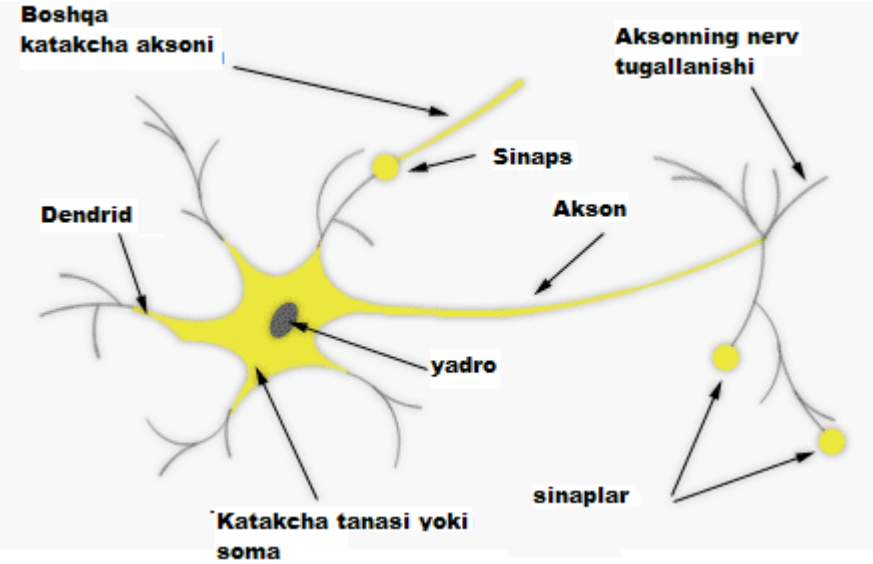
Bajarilgan ishning asosiy natijalari va uning qo’llanilish sohalari ko’rsatilgan.

1-BOB. SUN’IY NEYRON TO’RLARI

1. Sun’iy neyron to’rlarining negizi

Sun’iy neyron to’rlarining rivojlanishida biologiyaning o’rni katta. Izlanuvchilar mavjud tarmoq konfigurasiyasi va algaritmiga mos terminlarni qo’llagan holda aqliy faoliyat tashkilotini tasvirlashadi. Lekin ehtimol shu o’xshashlik bilan tugaydi. Bizni miyaning ishlashi haqidagi bilimlarimiz biroz chegaralangan, oriyentirlab unga taqlid qilganlar kam topilgan. Shuning uchun to’rni ishlab chiquvchilar kerakli funksiyani bajarish qobiliyatiga ega bo’lgan tuzilishni qidirishda zamonaviy biologik bilimlar doirasidan chiqishga majbur bo’lishadi[1,2,3,4,5].

Ishni neyronning prototiplarini ko’rib chiqishdan boshlaymiz. Neyron biologik sistemasining nerv xujayrasi hisoblanadi. U tana va uni tashqi muhit bilan bog’lovchi shohlardan tashkil topgan. (Rasm 1.1).



Rasm 1.1 Biologik neyron

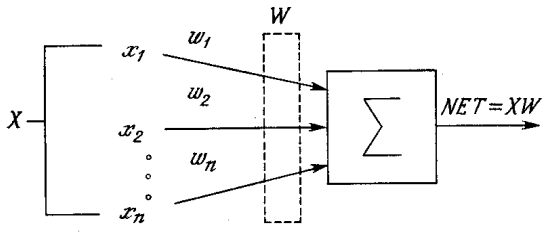
Qo’zg’alishni qabul qiluvchi neyron shoxlari dendrit deb nomlanadi. Qo’zg’alishga javob beruvchi neyrondagi shoxlar akson deb ataladi. Har bir neyronda bitta akson mavjud. Dendrit va aksonlar juda murakkab shoxlangan tuzulishga yega. Neyron aksonlari – qo’zg’olish manbai bilan dendrit orasidagi bog’lanish joyi sinaps deb nomlanadi. Neyronning asosiy funksiyasi qo’zg’alishni Dendritni aksonga

uzatishdan iborat. Lekin turli dendritlardan qabul qilingan signallar, akson signallariga turli xil ta’sir ko’rsatishi mumkin. Agar qo’zg’alishning yig’indisi ba’zi umumiy xolatlar doirasida o’zgaruvchi bo’sag’aviy mohiyatga olib kelsa, neyron signalni uzatadi. Bunga zid xolatlarda aksonga signal uzatilmaydi: neyron qo’zg’alishga javob bermaydi. Bu asosiy sxemada qiyinchilik va cheklanishlar ko’p, shuningdek ko’pchilik sun’iy neyron to’rlarini shu oddiy xossalar modellashtiradi.

2. Bir qatlamli sun’iy neyron to’ri

Sun’iy neyron. Sun’iy neyron birinchi yaqinlashishda biologik neyron xossalarini imitasiya qiladi. Har bir sun’iy neyronga boshqa neyronlar chiqishi bo’lgan qandaydir signallar to’plami kiradi. Har bir kiruvchi signal sinaptik kuchga mos vaznga ko’paytiriladi va ularning yig’indisi neyronning aktivlik darajasini aniqlaydi. Har bir signal o’ziga mos keluvchi w_1, w_2,..., w_n vaznlarga ko’paytiriladi va Σ bilan belgilangan yig’uvchi blokka kelib tushadi. Har bir vazn bitta biologik sinapsis «kuchiga» mos keladi. Vaznlar to’plami W vektori orqali belgilanadi. Biologik yelement tanasiga mos keluvchi yig’uvchi blok, mos vaznlariga ko’paytirilgan kiruvchi qiymatlarni algebraik tarzda yig’adi va neyron chiqishini shakllantiradi. Bu miqdor NET bilan belgilanadi. Yuqoridagi fikrlar vektor ko’rinishda quyidagicha ko’rinishda bo’ladi:

$$NET = XW.$$



Rasm 1.2 Sun’iy neyron

Aktivlash funksiyalari. Keyingi qadamda NET signali, odatda *F* aktivlash funksiyasi orqali hisoblanib, neyronning OUT chiqish signalini hosil qiladi. Aktivlash funksiyasi odddiy chiziqli funksiya bo’lishi mumkin[6,7,8,9]

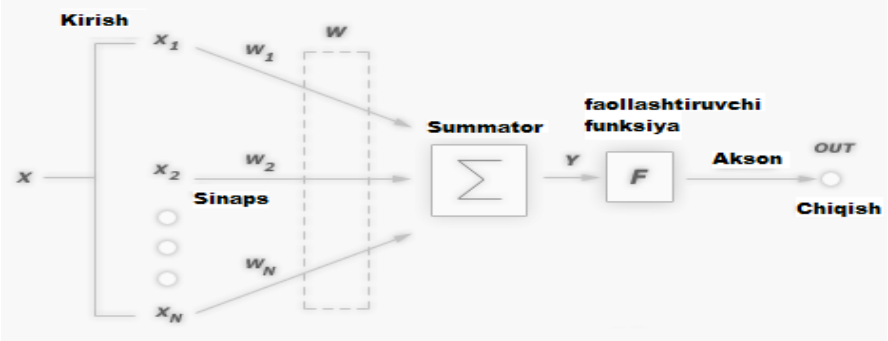
OUT = *K*(NET),

bu yerda *K* – quyidagicha aniqlangan chegara funksiyasi doimiysi

OUT = 1, agar NET > T,

OUT = 0 boshqa holatlar uchun,

bu yerda *T* – qandaydir chegaraviy doimiy qiymat. Aktivlash funksiyasi biologik neyron chiziqsiz o'tkazuvchanlik xususiyatini yanada to'liq ifodalovchi funksiya bo'lishi va neyron to'ri uchun keng imkoniyatlar berishi mumkin.



Rasm 1.3. Aktivlash funksiyali sun'iy neyron

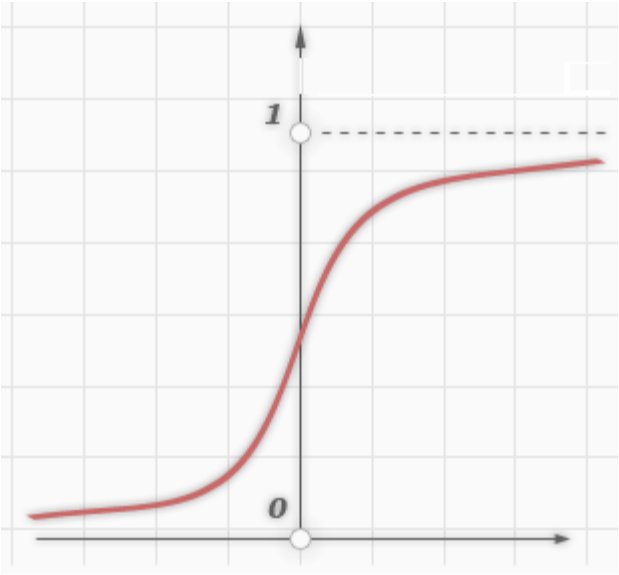
1.3-rasmdagi *F* bilan belgilangan blok NET signallarini qabul qiladi va OUT signalini chiqaradi. Agar *F* blok NET kattaligining o'zgarish diapazonini siqsa, ya'ni NET kattalikning har qanday qiymatida OUT qandaydir chekli oraliqqa tegishli bo'lsa, u holda *F «siquvchi» funksiya* deb nomlanadi. Ko'p hollarda «siquvchi» funksiya sifatida 1.4-rasmda ko'rsatilgan logistik yoki «sigmoidal» (S-shakldagi) funksiya ishlatiladi. Bu funksiya matematik ko'rinishi - *F(x)* q $1/(1 + ye^{-x})$. Shunday qilib[10],

$$\mathrm{OUT} = \frac{1}{1 + e^{-\mathrm{NET}}}.$$

Yelektron sistemalar bilan o'xshashlik nuqtai-nazaridan aktivlash funksiyasini sun'iy neyronning chiziqsiz kuchaytirgich xossasi deb qarash mumkin. Kuchaytirgich koyeffisiyenti OUT kattaligi ortirmasini, uni keltirib chiqargan NET kattaligining nisbatan katta bo'lmagan ortirmasiga nisbati sifatida hisoblanadi. Katta kuchaytirish koyeffisentli logistik funksiyaning markazidagi sohalarda kichik signallarni qayta-ishlash muammosini yechilsa, musbat va manfiy chekkadagi sohalardagi pasayadigan kuchaytirgichlar yesa juda katta ta'sirlarni qayta-ishlashga mos keladi. Shunday qilib,

neyron kiruvchi signalning keng diapazonida katta kuchaytirgich bilan amal qiladi, ya’ni past signallar kuchaytiriladi va aksincha, katta signallar pasaytiriladi.

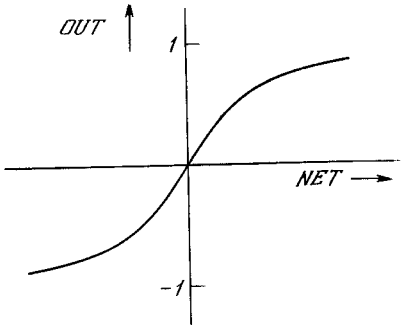
$$\mathrm{OUT} = \frac{1}{1 + e^{-\mathrm{NET}}} = F(\mathrm{NET}) .$$



Rasm 1.4. Sigmoidal logistik funksiyasi

Boshqa keng qo’llaniladigan aktivlash funksiyalardan biri giperbolik tangens. Shakli bo’yicha u logistik funksiyaga o’xshash va biologlar tomonidan nerv katagining aktivlashuvining matematik modeli sifatida ishlatiladi. Sun’iy neyron to’rining aktivlash funksiyasi ko’rinishida u quyidagicha yoziladi:

OUT = th(x).



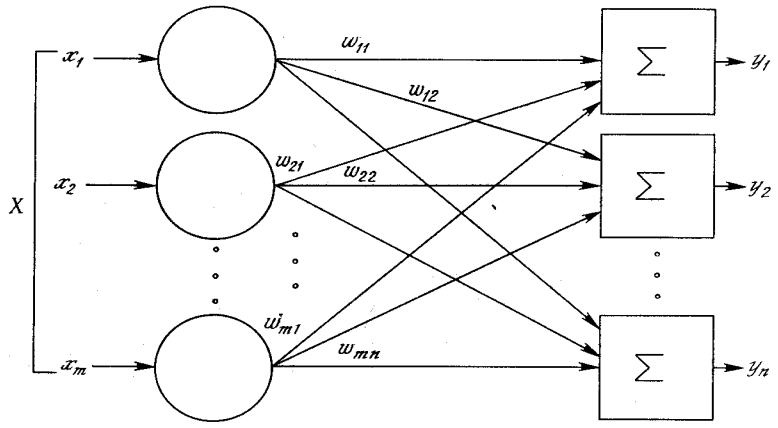
Rasm 1.5. Giperbolik tangens funksiyasi

Giperbolik tangens funksiyasi logistik funksiyalardek S shaklidagi funksiyadir, lekin u koordinata boshiga nisbatan simmetrik va NET q 0 nuqtada OUT chiquvchi signal qiymati nolga teng (1.5-rasm). Logistik funksiyadan farqli ravishda giperbolik

tangens turli ishoradagi qiymatlarni qabul qiladi va bu hol bir qator to’rlar uchun qo’l keladi. Sodda sun’iy neyron modeli biologik neyronning ayrim xossalarini inkor qiladi. Masalan, u sistema dinamikasiga ta’sir qiluvchi vaqt bo’yicha to’xtashlarni inobatga olmaydi. Kiruvchi signallar darhol chiquvchi signallarni yuzaga keltiradi. Va, juda muhim bo’lgan chastotli modulyasiya funksiyasi ta’siri yoki biologik neyronning sinxronlashtiruvchi funksiyasi hisobga olinmaydi, garchi bu xossalarni bir qator tadqiqotchilar hal qiluvchi deb hisoblashadi. Bu cheklanishlarga qaramasdan, bunday neyronlardan hosil bo’lgan neyronlar biologik sistemani yeslatuvchi ko’p xossalarni namoyon qiladi[11,12,13,14].

3. Kup qatlamli neyron to’rlari

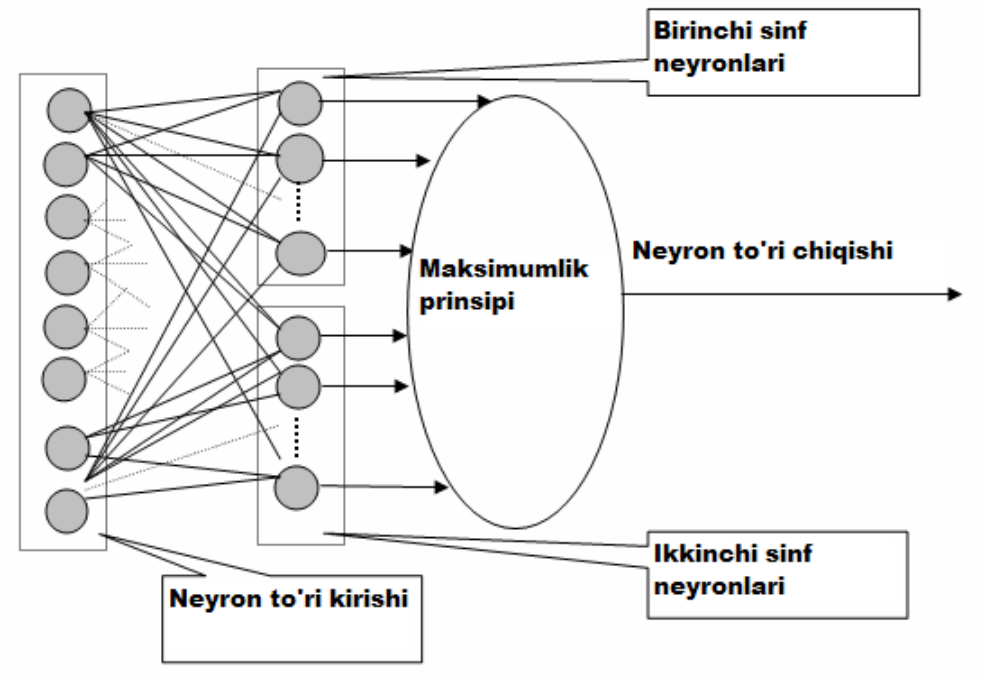
Garchi bitta neyron oddiy anglash prosedurasini ham amalga oshira olmaydi, lekin bir qancha neyronlarni neyron to’riga birlashtirishda neyron hisoblarning kuchi yuzaga keladi. Neyron guruhi qatlam hosil qiluvchi sodda neyron to’ri 1.6-rasmda ko’rsatilgan. Izohlab o’tish kerakki, chap tomondagi qirra-aylanalar faqat kiruvchi signallarni taqsimlash uchun xizmat qiladi. Ular birorta hisoblash amallarini bajarmaydi va shu sababli qatlam hisoblanmaydi. Hisoblash amallarini bajaruvchi neyronlar to’rtburchaklar bilan belgilangan. X kiruvchi to’plamdagi har bir yelement alohida vazn bilan har bir neyron bilan bog’langan. O’z navbatida har bir neyron kiruvchi qiymatlar «sozlangan» yig’indisini chiqaradi.



Rasm 1.6. Bir qatlamli neyron to’ri

Vaznlarni W matrisa yelementlari sifatida qarash o'ng'aydir. Matrisa m satr va n ustunga yega bo'lib, m –kirishlar soni, n-neyronlar soni. Masalan, $w_{i,j}$ – bu uchinchi kirishni ikkinchi neyron bilan bog'lovchi vazndir. Shunday qilib, kompoyentalari neyronlarning OUT bo'lgan chiquvchi N vektorni hisoblashni matrisali ko'paytma N = XW sifatida keltirish mumkin, N va X –satr-vektorlar.

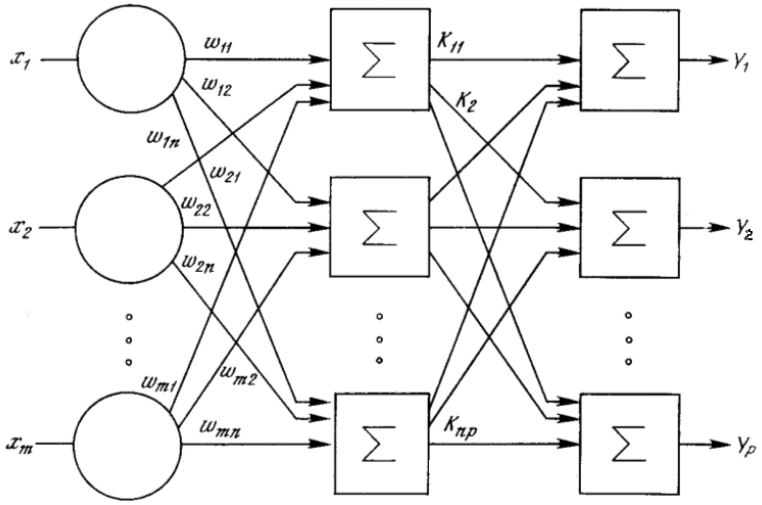
Bir qatlamli neyron to'rlari masala yechimi sifatida «g'olib barchasiga yega» prinsipi keng qo'llaniladi. Bu prinsip mohiyati quyidagicha: kiruvchi X uchun birinchi qatlamdagi qaysi neyron maksimum (minimum) qiymat qabul qilsa, o'sha neyron qayta-ishlanayotgan obektni o'ziga «tortgan» hisoblanadi. Mazkur neyronning barcha xossalari ayni shu obektga ham tegishli bo'ladi, masalan qatlam neyronlari sinflar vakillari sifatida qaralsa, o'ziga tortgan neyron (obekt) qaysi sinfga tegishli bo'lsa, noma'lum (yangi) obekt ham shu sinfga tegishli bo'ladi va hakoza. Maksimumlik prinsipi bo'yicha amal qiladigan bir qatlamli sun'iy neyron to'ri 1.7-rasmda keltirilgan.



Rasm 1.7. Maksimumlik prinsipida amal qiluvchi bir qatlamli sun'iy neyron to'ri

Hajm jihatdan katta va murakkab neyron to'rlari, odatda, mos ravishda katta hisoblash imkoniyatlariga yega. Garchi neyronning juda ko'p tuzilishlari yaratilgan

bo’lsa ham ko’p qatlamli neyron to’rlari miyaning ayrim qatlamli bo’laklarini nusxasidir. Bunday to’rlar bir qatlamli neyronlarga nisbatan o’rganish sig’imi kengroq hisoblanadi va hozirda uo’p qatlamli to’rlarni o’rgatish algoritmlarining bir qancha turlari yaratilgan. Shu o’rinda, qayd yetib o’tish zarurki, hozirda soha olimlari tomonidan bir va ko’p qatlamli neyron to’rlarining o’zaro yekivivalentligi matematik tarzda isbot qilingan[15,16,17,18].



Rasm 1.8. Ikki qatlamli neyron to’ri

Ko’p qatlamli neyron to’rlari qatlamlar kaskadi bilan hosil bo’lishi mumkin. Bir qatlam chiqishi keyingi qatlam uchun kirish bo’ladi. Bunday neyron turi 1.8-rasmda keltirilgan.

Teskari bog’lanishli to’rlar. Yuqorida ko’rilgan to’rlarda teskari bog’lanishlar yo’q yedi, ya’ni qandaydir qatlamning chiqishidan chiqib, xuddi shu qatlam yoki oldingi qatlamlar kirishiga boruvchi bog’lanishlar yo’q yedi. Bunday to’rlar to’g’ri tarqaluvchi to’rlar sinfini tashkil qiladi va ular katta qiziqish uyg’otadi va juda keng ravishda qo’llaniladi. Chiqishlarida kirishlariga bog’lanish bo’lgan to’rlar teskari bog’lanishli to’rlar deyiladi. Teskari bog’lanishlari bo’lmagan to’rlarda xotira yo’q, ularning chiqishi faqat ayni paytgai kirishlar va vaznlar bilan aniqlanadi. Ayrim ko’rinishdagi teskari bog’lanishli neyron to’rlarida chiqish qiymatlari kirishga qaytariladi, oqibatda chiqish ayni paytdagi kirish va oldingi chiqish bilan aniqlanadi.

Shu sababli teskari bog'lanishli to'rlar inson miyasining qisqa muddatli xotirasi xossalariga o'xshash xossalarga yega bo'ladi. To'r chiqishlari qisman oldingi kirishlarga bog'liq bo'ladi.

4 Ma'lumotlarning intellektual tahlilida klassifikasiya masalalari

Klassifikasiya masalasi yeng oddiy va keng yoyilgan masala hisoblanadi. Bu masala bo'yicha bir nechta ta'riflar keltiramiz

Klassifikasiya – o'rganilayotgan predmetlar, holatlar, rod bo'yicha jarayonlar, ko'rinishlar, tiplari, qandaydir mavjud belgilari bo'yicha tadbiq qilishni qulaylashtirish uchun tizimli taqsimlanishidir. Bunday o'xshashliklarni darajasi aks yetadigan chiquvchi tushunchalarni guruhlarga ajratiladi va ularni ma'lum tartibda joylashtiriladi.

Klassifikasiya – obektlar orasidagi o'xshashliklar yoki farqlarni aniqlash uchun tanlangan o'xshash klassifikasiya qilingan belgilarga (bir yoki bir nechta xossalariga) yega bo'lgan obektlar to'plamining qandaydir prinsiplari bo'yicha tartiblanishidir.

Klassifikasiya quyidagi qoidalarga rioya yetishni talab yetadi:

- har bir dalolatnomada bo'linishni faqat bir asosga qo'llash zarur bo'ladi;
- bo'linishlar bir-biriga mos keladigan bo'lishi zarur, ya'ni tur tushunchalarini umumiy hajmi rod tushunchalarining umumiy hajmiga tenglashtirish zarur bo'ladi;
- bo'linish qismlar o'zaro bir-birini istisno qilishi shart, ularning hajmlarining yesa kesishish shart yemas;
- bo'linishlar ketma-ket bo'lishi shart.

farqlaydi:

- tashqi belgilar bo'yicha ishlab chiqilgan va kerakli tartibda predmetlar to'plami(jarayonlar, holatlar)ni zeb berilishi uchun xizmat qiladigan *yordamchi (sun'iy) klassifikasiyani*;
- predmetlar va holatlarning ichki umumiyligida xarakterlanadigan mavjud belgilari bo'yicha ishlab chiqiladigan *tabiiy klassifikasiyani*. U ilmiy tadqiqot ishlarida natijalarga va muhim muhitga yega bo'ladi hamda klassifikasiya qilinadigan

obektlarning qonuniyatlarini o’rganish natijalarini mo’ljalga oladi va mustahkamlaydi.

Ajratib belgilarning tegishliligida ularning bo’linish tushunchalarini birikmalari va proseduralarida *klassifikasiya* quyidagicha bo’lishi mumkin:

oddiy – rod tushunchalari bo’linishi faqat belgilar bo’yicha hamda faqat bir marta barcha ko’rinishlarni ochilguniga qadar bo’ladi. Bu klassifikasiyaga misol: dixotomiya, ya’ni bir-biriga qarama-qarshi bo’lgan ikki tushunchadan (masalan, «A» yoki «A yemas») iborat a’zolarning bo’linishi bo’ladi.

murakkab – har xil asoslar bo’yicha bir tushunchaga bo’linish uchun qo’llaniladi va shunday oddiy bo’linishlarni bir butunlikda sintez qilinadi. Bu klassifikasiyaga misol sifatida kimyoviy yelementlar davriy sistemasini olish mumkin.

Klassifikasiya orqali obektlarning (kuzatishlar, hodisalar) oldindan ma’lum sinflarning biriga obektlarning tegishli bo’lishini tushunamiz.

Klassifikasiya bu aniq guruhlarning xarakteristikalarini aniqlashga muvofiq natijalarini hosil qiladigan qonuniyatlardir.

Klassifikasiya strategiya bo’yicha nazorat qilinadigan va boshqariladigan o’qitish deb ataladigan *o’qituvchi bilan o’qitish*ga (supervised learning) tegishli bo’ladi.

Klassifikasiya masalalari ko’pincha tanlanmalar asosidagi kategorial tegishli o’zgaruvchilarni bashorat qilish deb ataladi (ya’ni tegishli o’zgaruvchilar bu kategoriyalar hisoblanadi). Tanlanmalar orasida uzluksiz va/yoki kategirial o’zgaruvchilar. Masalan, kimdir firmaning mijozlaridan biri bo’lsa, u ma’lum tovarning potensial xaridori bo’ladi, kimningdir xaridor bo’lmasligini, kimdir firmaning xizmatlaridan foydalansa, kimningdir foydalanmasligini va h.k.larni bashorat qilish mumkin. Masalaning bu tipi binar klassifikasiyaga mos keladi, unga tegishli o’zgaruvchilar faqat ikki qiymatni qabul qilishi mumkin (masalan, «ha» yoki «yo’q», 1 yoki 0).

Klassifikasiyaning boshqa variantida agar tegishli o’zgaruvchilar oldindan aniqlangan sinflarning ba’zi to’plamlaridan qiymatlarni qabul qilish mumkin bo’lganda vujudga keladi. Masalan, qachon mijoz qaysi markadagi avtomobilni sotib olishni xohlashini, bashorat qilish zarur bo’ladi. Bu holatlarda tegishli o’zgaruvchilar uchun sinflar to’plami qarab chiqiladi.

Klassifikasiya **bir o’lchovli** (bir belgi bo’yicha) va **ko’p o’lchovli** (ikki yoki undan ortiq belgilar bo’yicha) bo’lishi mumkin.

Ko’p o’lchovli klassifikasiya biologlar tomonidan inson organizmini klassifikasiya qilish uchun diskreminasiya muammolarini yechish uchun ishlab chiqilgan. Ularning bu yo’nalishga bag’ishlangan birinchi ishi R.Fisher tomonidan (1930 y.) ishlab chiqilgan. Biologiya klassifikasiyaning ko’p o’lchovli uslublari qayta ishlash uchun yeng ko’p talab qilingan va qulay muhit bo’lgan va bo’lib qoladi.

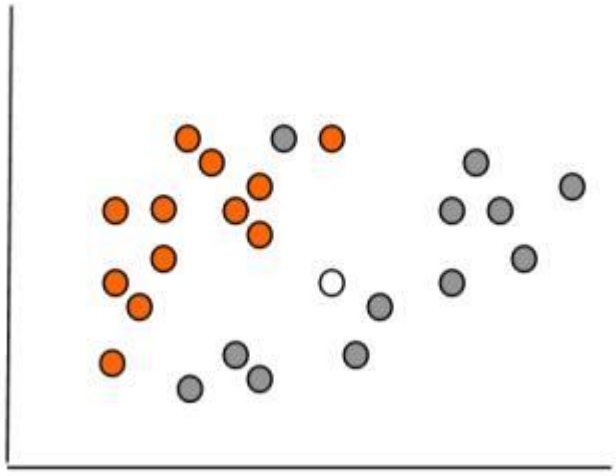
Klassifikasiya masalasini oddiy misolda qaraymiz. Faraz qilaylik, turistlik agentligida mijozlarning yoshi va oylik tushumi haqidagi berilganlar saqlangan bazaga yega bo’lsin. Ikki xil ko’rinishda reklama materiallari ham berilgan: qimmatroq, komfort dam olish joyi va arzonroq, yoshlar dam olish joyi. Ko’rinib turibdiki, mijozlar ikki sinfga bo’linadi: 1-sinf va 2-sinf. MB 5.1-jadvalda keltirilgan:

1.1-jadval. Turistlik agentligi mijozlarining MB

Mijoz kodi	**Yoshi**	**Tushumi**	**Sinfi**
1	18	25	1
2	22	100	1
3	30	70	1
4	32	120	1
5	24	15	2
6	25	22	1
7	32	50	2
8	19	45	2

9	22	75	1
10	40	90	2

Masala. Yangi mijozning qaysi sinfga mos kelishini aniqlash va uni reklamada keltirilgan joylarning qay biriga yuborishga to'ri keladi. Aniqlik uchun bu MBni ikki o'lchovli o'lchamlarga (yoshi va tushumi bo'yicha) 1-sinf (to'q sariq belgiga) va 2-sinfga (qoramtir belgiga) mos keladigan obektlar to'plami ko'rinishida qo'yamiz.



1.9-r**asm.** Ikki o'lchovli o'lchamda MBning obektlar to'plami.

Bizning masalamizning yechilishi rasmda keltirilgan oq belgidagi yangi mijozning qaysi sinfga mos kelishini aniqlash bilan hosil qilinadi.

Klassifikasiya jarayoni. Klassifikasiya jarayonining maqsadi kiritiluvchi parametrlar sifatida prognozlaydigan atributlar foydalanadigan va tegishli atribut qiymatini oladigan model ko'rinishidan iborat bo'ladi. Klassifikasiya jarayonida aniqlangan kriteriyalar bo'yicha sinflardagi obektlar to'plamlarining bo'linishida joylashadi.

Klassifikatorlargabelgilar vektori bo'yicha mos kelgan obekt oldindan ma'lum sinflar orqali aniqlanadigan qandaydir asl mohiyatga aytiladi.

Matematik metodlar yordamida klassifikasiyani o'tkazish uchun uning matematik apparatidan foydalanib qo'llanishi mumkin bo'lgan obektning formal tavsifining mavjud bo'lishi zarur. Bizning holatimizdagi bunday tavsiflar MB yuzaga

keltiradi. Har bir obekt (ya’ni MB yozuvi) shu obektning ba’zi bir xossalari haqidagi axborotlarni keltiradi.

Asos qilib olingan berilganlar to’plami (yoki berilganlar tanlanmasi) ikki to’plamga bo’linadi: o’qitadigan va test o’tkazuvchi.

O’qitadigan to’plam (training set) o’qitish (konstruksiyalash) modeli uchun foydalaniladigan berilganlarni o’z ichiga oladigan to’plamdir. Bu to’plam kiruvchi va chiquvchi (butun) qiymatlar misollaridan tuziladi. Chiquvchi qiymatlar o’qitish modeli uchun mo’ljallangan.

Test o’tkazuvchi (test set) to’plam ham misollarning kiruvchi qv chiquvchi qiymatlardan tuziladi. Bu yerda chiquvchi qiymatlar ish qobiliyati modellarida tekshirish uchun foydalaniladi.

Klassifikasiya jarayoni ikki bosqichdan iborat bo’ladi: konstruksiyalashgan modellar va undan foydalanish.

Konstruksiyalashgan modellar: oldindan aniqlangan sinflar to’plamini tavsiflaydi

- berilganlar to’plamining har bir misoli oldindan ma’lum sinflarning biriga mos keladi;
- bu bosqichda o’qitiladigan to’plamda foydalaniladi, unda konstruksiyalashgan modellar yuzaga keladi;
- olingan model klassifikasion qoidalar, daraxsimon yechimlar va matematik foymulalarda tasvirlanadi.

Foydalanuvchi modellar: yangi yoki noma’lum qiymatlarni klassifikasiya qiladi.

To’g’rilikni (aniqlikni) baholash modellari

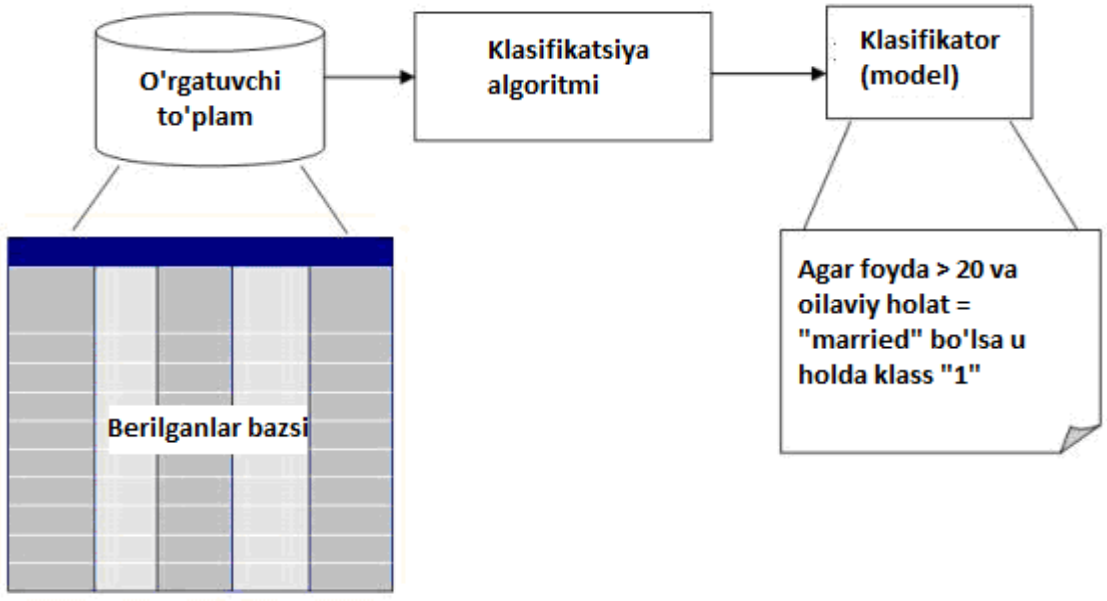
1. Test misoldan olingan ma’lum qiymatlar olingan modellar foydalanib olingan natijalar bilan taqqoslanadi

2. Aniqlik darajasi - bu test to’plamida to’g’ri klassifikasiya qilingan misolning foizidir

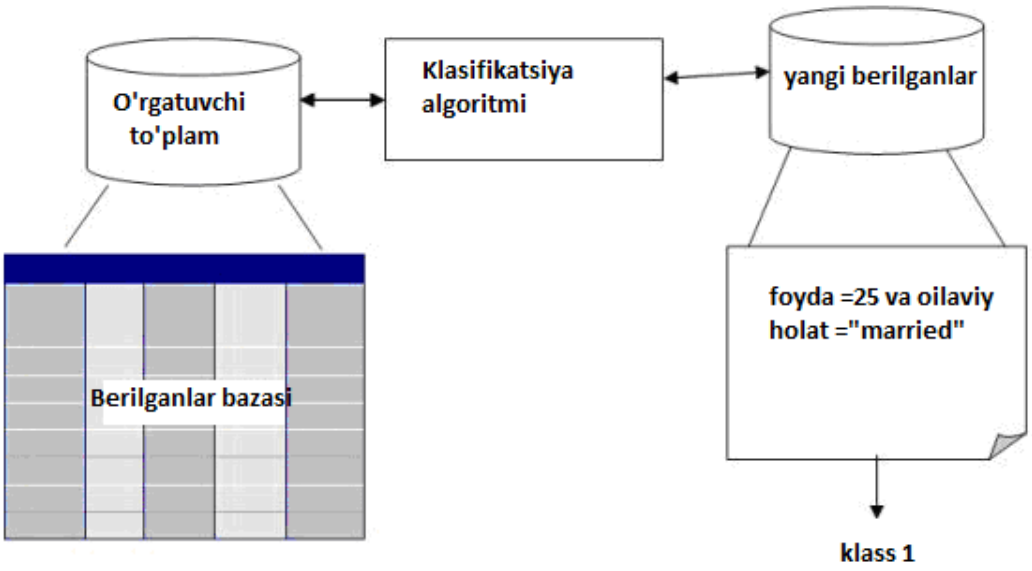
3. Test to’plami ya’ni test o’tkaziladigan to’plamida qurilgan model o’qitadligan to’plamga tegishli bo’lmasligi lozim.

- Agar aniqlik modellari ruxsat yetilgan bo’lsa ya’ni misollarni klassifikasiya qilish uchun noma’lum sinf modellaridan foydalanish mumkin bo’ladi.

Aynan konstruksiyalashgan modellar va ulardan foydalanish bo’yicha klassifikasiya jarayoni 5.2-5.3 rasmlarda keltiriladi



1.10-rasm. Klassifikasiya jarayoni. Konstruksiyalashgan modellar

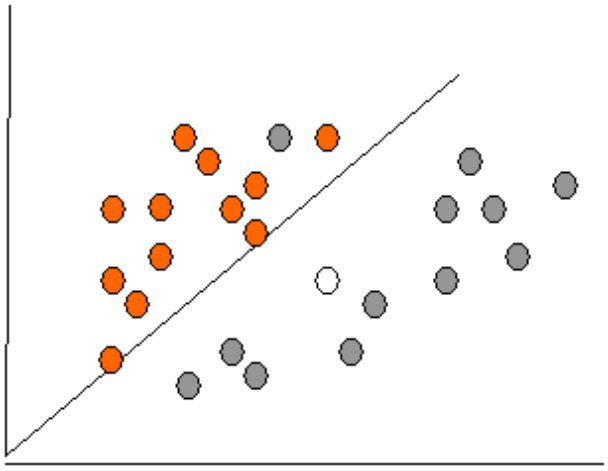


1.11-rasm. Klassifikasiya jarayoni. Foydalanuvchi modellar

Klassifikasiya masalalarini yechish uchun qo'llaniladigan uslublar. Klassifikasiya masalalarini yechishda turli xil uslublardan foydalaniladi va ulardan asosiylari:

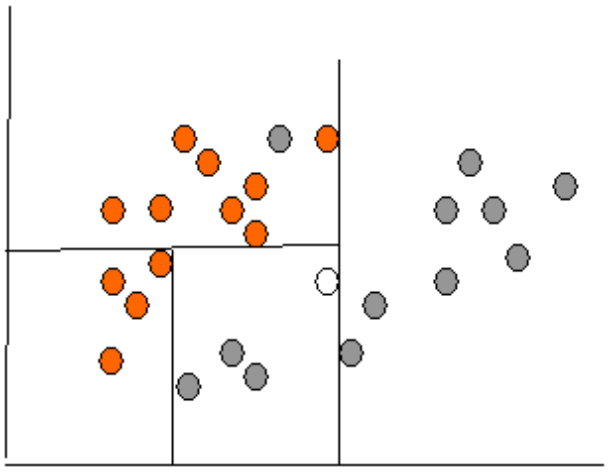
- Daraxsimon yechimlardan foydalanib klassifikasiya qilish;
- bayesov (sodda) klassifikasiyasi;
- sun'iy neyron tarmoqlari yordami bilan klassifikasiya qilish;
- tayanch vektorlar uslublari bilan klassifikasiya qilish;
- statistik uslublarning chiziqli regressiya qismligida klassifikasiya qilish;
- qo'shniga yaqinlashishi metodlari yordamida klassifikasiya qilish;
- CBR-uslublari bilan klassifikasiya qilish;
- genetik algoritmlar yordamida klassifikasiya qilish.

Klassifikasiya masalalarining sxemotexnik yechilishini quyidagi ba’zi metodlar (chiziqli regressiya, daraxsimon yechimlar, va neyron tarmoqlari) bilan 5.4-5.6 rasmlarda tasvirlanadi:



1.12-rasm. Chiziqli regressiya uslubida klassifikasiya masalasining yechilishi

```
if X > 5 then grey
  else if Y > 3 then orange
    else if X > 2 then grey
      else orange
```



1.13-rasm. Daraxtsimon yechimlar uslubida klassifikasiya masalasining yechilishi

1.14-rasm. Neyron tarmoqlari uslubida klassifikasiya masalasining yechilishi

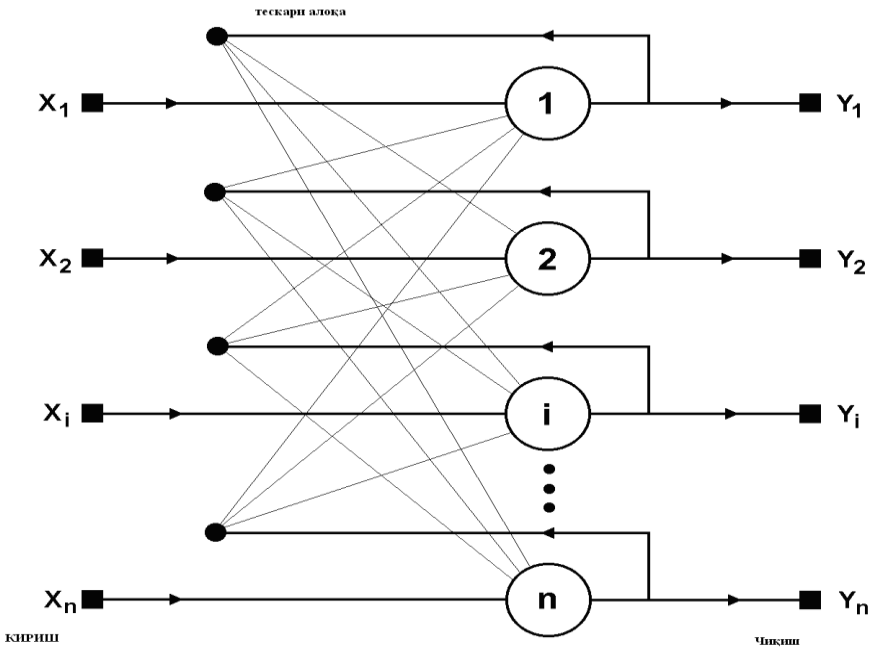
Klassifikasiyaning aniqligi: xatolik darajasini baholash. Klassifikasiya aniqligini baholash kross-tekshiruv yordamida o’tkazilishi mumkin. Kross-tekshiruv (Cross-validation) – bu kross-tekshiruvchi to’plam deb ataladigan test o’tkazuvchi to’plamdagi berilganlarda klassifikasiya aniqligini baholash proseduraSidir. Test o’tkazuvchi to’plamning klassifikasiya aniqligi bilan o’qitadigan to’plamning aniqlik klassifikasiyasi taqqoslanadi. Agar test o’tkazuvchi to’plam klassifikasiyasi aniqlikka yaqinroq natijalarni bersa (xuddi klassifikasiya o’qitadigan to’plami singari), keltirilgan model kross tekshiruvdan o’tgan deb hisoblanadi.

2-bob. Xopfild va Ximmingning neyron to’rlari

1. Xopfildning neyron to’rlari haqida tushincha

Sunniy neyron to’rlari konfigurasiyalari orasida klassifikasiyalashda o’qitish prinsiplari bo’yicha o’qituvchi yordamida o’rgatish va o’qituvchisiz o’rgatish prinsiplariga to’g’ri kelmaydi. Bunday hollarda og’irlik koeffisiyentlari qayta ishlanayotgan axborotlar yordamida izlab topiladi va barcha o’rgatishlar xuddi shu hisoblashga keltiriladi. Bir tomondan aprior axborotlarni o’qituvchining yordami sifatida qabul qilish kerak, boshqa tomondan tarmoq tasvirlarni haqiqiy ma’luotlar kelguncha xotirada saqlab qoladi. Bunday mantiqiy bog’lanishli tarmoqlar sifatida Xopfild va Ximming to’rlarini yaxshi tanilgan.

Quyida qirishi va chiqishi bitta bo’lgan bir qatlamli Xopfildnig neyron tarmog’i keltirilgan.



Rasm.2.1. Xopfild tarmog’ining strukturali sxemasi.

Assosiativ xotira kabi bu tarmoqda yechiladigan masala qo’yidagicha shakllantiriladi. Shakl ko’rinishidagi (tasvir, raqamlashgan ovozlar kabi jaryonlarni

yoki obyektlarni ifodalovchi) qandaydir ikkilik signallardan tashkil topgan bo'lsin. Tarmoq unga kirib kelayotgan ideal bo'magan mos tasvirli signallarni ajratib saqlab qolsin, yoki kirib kelgan ma'lumotlar birorta ham shaklga mos kelmasligi haqida xabar bersin. Umumiy holda, ixtiyoriy signalni $\mathbf{X} = \{ x_i: i=0...n-1\}$ vektor , n – tarmoqdagi neyronlar soni, kiruvchi va chiquvchi vektorlar hajmi. Har bir x_i element + yoki -1 ga teng. k – shaklni ifodalovchi vektorni $\mathbf{X^k}$ vektor bilan ifodalaymiz va uning kompnentalarini mos holda – x_i^k , k=0...m-1, m – shakllar soni, bilan belgilaymiz. Tarmoqqa berilgan ma'lumotlar asosida u shaklni tanisa, u holda unga kiruvchi ma'lumot $\mathbf{Y} = \mathbf{X^k}$ bo'ladi , bu yerda **Y** tarmoqqa kiruvchi $\mathbf{Y} = \{ y_i: i=0,...n-1\}$ vektor qiymati. Aks holda, chiquvchi vektor hyech qanday namunadagi shakl bilan mos kelmaydi.

Agar, masalan, signallar qandaydir shaklni ifodalasa, u holda u tarmoqqa kirishda uni grafik ko'rinishida ifodalaydi va namunadagi birorta shakl bilan mos kelganligini yoki mos kelmaganligini aniqlaydi.

Tarmoq qiymatlarni qabul qilishda og'irlik koeffisiyenti qo'yidagi shaklda ifodalanadi [][]:

$$w_{ij} = \begin{cases} \sum_{k=0}^{m-1} x_i^k x_j^k, i \neq j \\ 0, \qquad i = j \end{cases} \qquad (1)$$

Bu yerda i i j – indekslar bo'lib, mos holda oldsnaptik va orqsnaptik neyronlar; x_i^k, x_j^k – i- va j, k- shakl vektorining elementlari.

Tarmoq bajaradigan ishning algoritmi qo'yidagicha (p – iterasiya nomeri):

1. Tarmoqqa kirish uchun no'malum signal beriladi. Unga aksonlarning qiymatlarini beriladi:

$$y_i(0) = x_i , i = 0...n-1, \qquad (2)$$

shuning uchun tarmoq sxemasida kiruvchi snapslar shartli xarakterda bo'ladi. Nol esa, tarmoqning nolinchi iterasiyasini bildiradi.

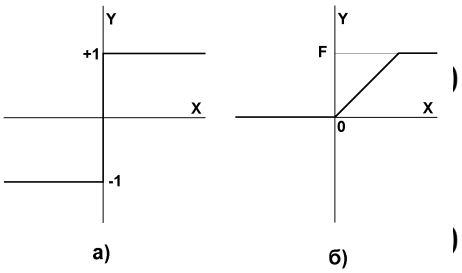
2. Neyronlarnng yangi holati hisoblanadi

$$s_j(p+1) = \sum_{i=0}^{n-1} w_{ij} y_i(p), \; j=0...n-1$$

va aksonlarning yangi qiymatlari

$$y_j(p+1) = f\left[s_j(p+1)\right]$$

bu yerda f – sakrash ko'rinishidagi faollashtiruvchi funksiya, uning rasmi 2-rasmda ko'rsatilgan.

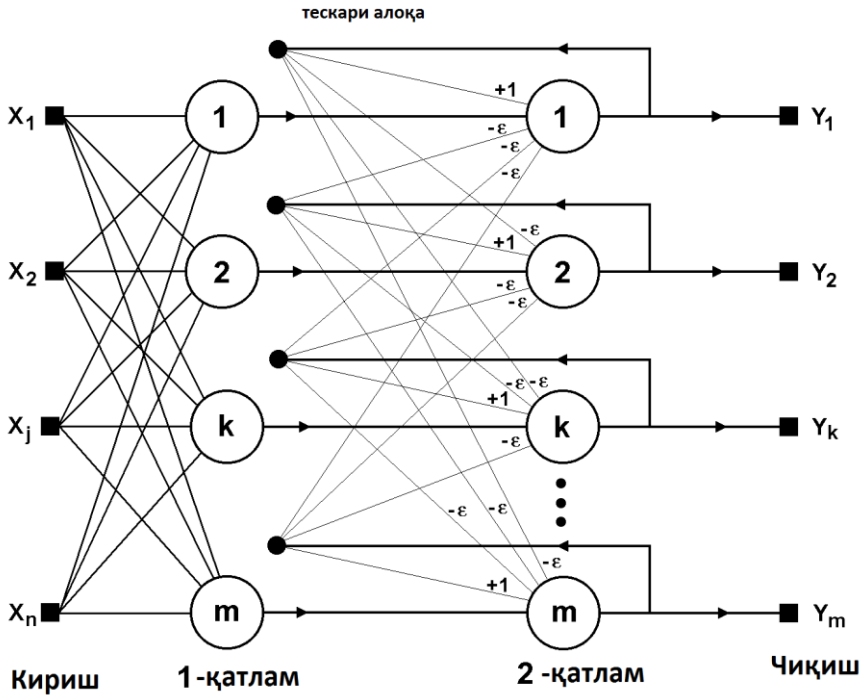


2.2. rasm. Faollashtiruvchi funksiya

3. Oxirgi iterasiyada aksonlarning qiymati tekshiriladi. Agar ha bo'lsa, u holda 2 qadamga o'tiladi, aks holda tamomlanadi. Bu holatda chiquvchi vektor namunadagi shaklga uxshash eng yaxshi shaklga ega bo'ladi.

2. Ximmingning neyron to'rlari haqida tushincha

Yuqorida aytib o'tilgandek, ba'zida tarmoq shaklni aniqlay olmaydi va notug'ri shaklnng qiymatini chiqaradi. Bu tarmoqning chegaralanganligini bildiradi. Xopfild tarmog'ida saqlanib qoluvchi shakllar soni m, 0.15•n tadan oshib ketmasligi kerak. Bundan tashqari, agar ikkita A va B shakllar bir biriga juda uxshash bo'lsa, u holda ular tarmoqda kesishgan assosiasiyalarni tashkil etadi, ya'ni tarmoqqa A ni qiymatlarini berish B ni tarmoqdan chiqaradi, yoki aksincha.



2.3-rasm. Xemming tarmog'ining strukturali sxemasi.

Agar tarmoq shaklning aniq nomerini chiqarsa, u holda assosiativli xotira Ximming tarmog'ini yetarlicha yaxshi qo'llaydi(3-rasm.).

Tarmoq ikkit aqatlamdan iborat. Birinchi va ikkinchi qatlamlar m neyronlardan iborat bo'ladi, bu yerda m shakllar soni. Birinchi qatlamdagi neyronlar n ta tarmoq kirishi bilan bog'langan sinapslardan iborat. Ikkinchi qatlamdagi neyronlar teskari aloqa qiluvchi sinapslar bilan bog'langan. Yagona musbat teskari aloqadagi sinaps o'zining neyronlari bilan bog'langan.

Tarmoq ishining maqsadi, barcha tasvirlar bilan testlanayotgan tasvirlar orasidagi maofani topish sanaladi. Ximming masofasi bu ikkita binar vektorlarning bitlardagi farqi tushiniladi. Tarmoq namuna bilan no'malum tasvirlar orasidagi minimal Ximming masofasini tanlashni amalga oshiradi va natijada bu tasvirga mos bitta neyron faol bo'lib qoladi.

Birinchi qatlamga og'irlik koeffisiyentlarini ta'minlashda va faollashtiruvchi funksiyaga qiymat berish qo'yidagicha amalga oshiriladi:

$$w_{ik} = \frac{x_i^k}{2},\ i=0...n-1,\ k=0...m-1 \quad (5)$$

$$T_k = n / 2,\ k = 0...m-1 \quad (6)$$

Bu yerda x_i^k –k- shaklning i-elementi.

Sinapslarni tuxtauvchi og'irlik koeffisiyentlari ikkinchi qatlamda qandaydir $0 < \varepsilon < 1/m$ qiymatni qabul qiladi. O'z askoni bilan bog'langan neyron sinapsi +1 og'irlikka ega bo'ladi.

Xemming tarmoqining bajaradigan ishni algoritmi qo'yidagicha:

1. Tarmoqa birinchi qatlamdagi hisoblangan va uni qiymatini e'tiborga olgan no'malum **X** = {x_i:i=0...n-1}, vektor kiritiladi(yuqori indeks qatlam nomerini bildiradi):

$$y_j^{(1)} = s_j^{(1)} = \sum_{i=0}^{n-1} w_{ij} x_i + T_j,\ j=0...m-1 \quad (7)$$

Shundan sung olingan qiymatlar asosida ikkinchi qatlamning aksonlari hisoblanadi:

$$y_j^{(2)} = y_j^{(1)},\ j = 0...m-1 \quad (8)$$

2. Ikkinchi qatlamning yangi holati hisoblanadi:

$$s_j^{(2)}(p+1) = y_j(p) - \varepsilon \sum_{k=0}^{m-1} y_k^{(2)}(p), k \neq j, j = 0...m-1 \quad (9)$$

va uning aksonlarining qiymati:

$$y_j^{(2)}(p+1) = f\left[s_j^{(2)}(p+1)\right],\ j = 0...m-1 \quad (10)$$

Faollashtiruvchi f funksiyaning qiymati (2b-rasm) ko'rinishda bo'ladi, bu yerda F yetarlicha katta bo'lib, ixtiyoriy argumentning qiymati undan oshib ketmaydi.

3. Oxirgi iterasiyada ikkinchi qatlamning qiymatlari o'zgargan yoki o'zgarmaganligi tekshiriladi, agar ha bo'lsa, u holda 2 qadamga, aks holda esa sikl to'xtatiladi.

Algoritmni baholashdan ko'rinib turibdiki, birinchi qatlam shartli ahamiyatga ega bo'lib qolmoqda: birinchi qadamda og'irlik koeffisiyentlarining qiymatlarinidan

bir marta foydalaniladi, shuning uchun birinchi qatlam qo’yida ko’rsatilganidek tarmoqdan umuman olib tashlandi.

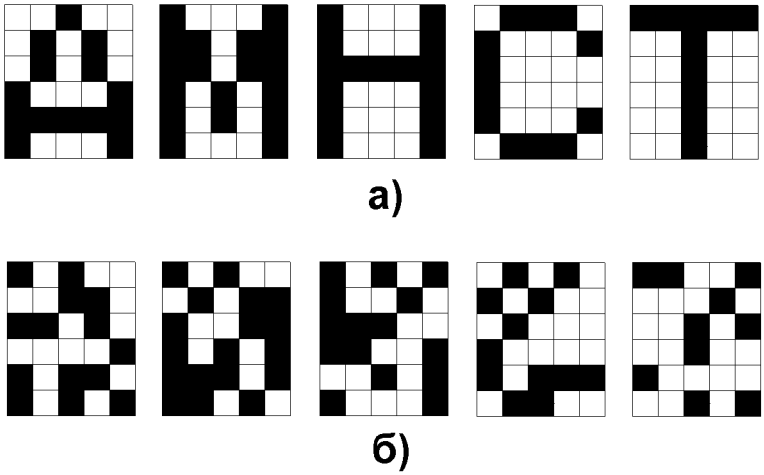
Xemming tarmog’ining dasturiy modeli maxsus sinflarni saralashga asoslangan.

NetHN sinfida qo’yidagi elementlar aniqlangan:

Nin va Nout – ma’lumotlarga mos ravishda kiruvchi vektorlarning o’lchami va shakllar soni;

dx va dy – ikkita kordinatalar orqali kiruvchi shakllarning o’lchamlari, dx*dy = Nin, bu funksiyalar LoadNextPattern fayldan ma’lumotlarni kiritish uchun foydalaniladi;

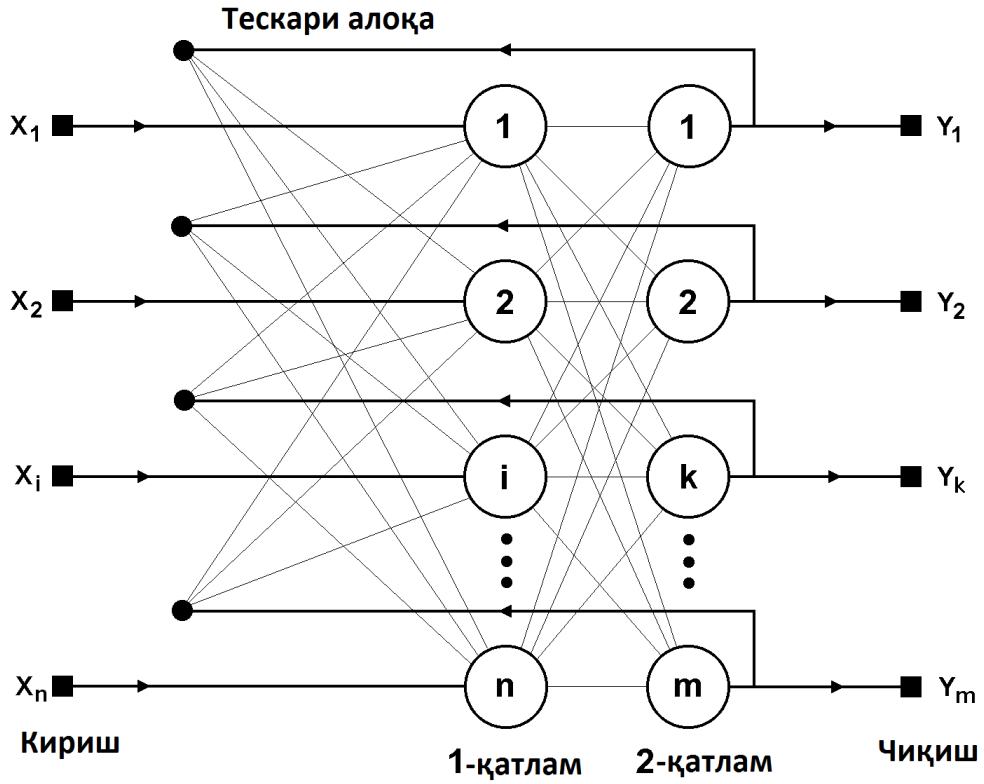
Bu yerda ikkinchi qatlam beshta harfni tanish uchun ishlatilmoqda



2.4-rasm. Namunaviy va test qilinadigan shakllar.

3. Ikki yo’nalishli assosativ xotira

Dastur Borland C++ 6 muhitida testdan o’tkazildi. Taklif etilayotgan sinflar kattaroq Ximming tarmoqlari uchun ham o’rinli bo’ladi. Shu o’rinda IYAX tarmog’i Xopfild tarmog’ining mantiqiy davomidir, buning uchun ikkinchi qatlamni qo’shish kifoya qiladi.



2.5-rasm. Ikki yo'nalishli assosativ xotira (IYAX) strukturali sxemasi.

IYAX strukturasi 5–rasmda ko'rsatilgan. Tarmoq juft assosiativ shakllarni saqlab qolish xususiyatiga ega. Faraz qilaylik, juft shakllar $\mathbf{X^k} = \{x_i^k : i=0...n-1\}$ va $\mathbf{Y^k} = \{y_j^k : j=0...m-1\}$vektor ko'rinishda ifodalangan bo'lsin, k=0...r-1, bu yerda r – juftlarning soni. Qandaydir $\mathbf{P} = \{p_i : i=0...n-1\}$ vektorni tarmoqqa kiritilishi, ikkinchi qatlamda boshqa $\mathbf{Q} = \{q_j : j=0...m-1\}$ vektorni hosil bo'lishini ta'minlaydi va yana birinchi qatlamga kirib keladi.

Har qaysi siklda chiquvchi ikkita qatlamdagi vektorlar ham namunaviy juft shakllarga yaqinlashadi, birinchi **X** esa kirishda berilgan **P** ga, ikkinchisi esa assosialashgan **Y** ga mos keladi. Birinchi qatlamdagi vektorlar orasidagi assosasiya $\mathbf{W^{(1)}}$ og'irlik matirisasi orqali ifodalanadi. Iqqinchi qatlamdagi og'irlik matrisasi $\mathbf{W^{(2)}}$ esa birinchi matrisaning $(\mathbf{W^{(1)}})^T$ trnsponirlanganiga teng bo'ladi. Tarmoqni o'rgatish Xopfild tarmog'idagi kabi qo'yidagi **W** matrisa elementlarini hisoblash orqali amalga oshiriladi:

$$w_{ij} = \sum_k x_i y_j,\ i = 0...n-1,\ j = 0...m-1 \qquad (11)$$

Bu formula qo'yidagi matrisa tenglamasining kengaytirilgan ko'rinishidir

$$\mathbf{W} = \sum_k \mathbf{X}^{\mathbf{T}} \mathbf{Y} \qquad (12)$$

Xususiy holda, shakllar vektor ko'rinishida yozilganda, matrisalarning ko'paytmasining ulchamlari mos ravishda [n*1] va [1*m] bo'ladi va (11) formulaga keltiriladi.

Xulosa qilib shuni aytish mumkinki, Xopfild va Ximmeng va IYAX to'rlari to'lq bo'lmagan va buzilgan tasvirlarni samarali va oddiy holda tiklashni amalga oshiradi. Tarmoqning katta bo'lmagan hajmi, ya'ni nafaqat tasvirlarni saqlab qoladi, balki ularni umumlashtirishni ham amalga oshiradi, masalan Xemming tarmog'i maksimal uxshatish me'zonlari bo'yicha amalga oshiriladi [3].

Neyron to'rlarini qo'llash ikki bosqichda amalga oshiriladi:

1) Neyron arxitekturasini tanlash.

2) Neyron to'rlarini og'irlik koeffisiyentlarini tanlash.

Birinchi bosqichda qo'idagilarni tanlash zarur bo'ladi:

a) qanday neyronlardan foydalanamiz(kirish soni, faollashtiruvchi funksiya);

b) qanday qilib ularni bir-biri bilan bog'lash mumkin;

v) neyron tarmog'iga kirish va chiqish uchun nimani olish zarur.

Ikkinchi bosqichda esa tanlangan neyron tarmog'ini "o'rgatish" zarur bo'ladi, ya'ni u ishlashi uchun shunday og'irlik koeffisiyentini topish kerak.

3-bob. Ma’lumotlarni parallel qayta ishlash

1. Ma’lumotlarni parallel qayta ishlashning tushinchasi

Bir necha amallarni bir vaqtda bajarish g’oyasidan iborat bo’lgan ma’lumotlarni parallel hisoblash ikki xil ko’rinishi mavjud. Bular: parallel va konveyer.

Agar biror qurilma bitta amalni vaqt birligida bajarsa, u holda mingta amalni ming vaqt birligida bajaradi. Agar xuddi shunday bir vaqtda ishlay oladigan va bir–biriga mustaqil beshta qurilma mavjud deb qaralsa, u holda ular yuqoridagi mingta amalni mingta vaqt birligida emas, balki ikki yuzta vaqt birligida bajaradi. Xuddi shunday N ta qurilmadan iborat tizim 1000 ta amalni 1000/N vaqt birligida bajaradi. Unga o’xshash holatlarni hayotdan ham keltirish mumkin. Masalan, agar bitta askar polizga 10 soatda ishlov bersa, u holda 50 askardan iborat rota bir vaqtda ishlab polizga 12 minutda ishlov beradi. Bu parallel amallar prinsipi hisoblanadi.

Konveyerli qayta ishlash. Qo’zg’aluvchan vergulli shaklda tasvirlangan haqiqiy ikkita sonni qo’shish uchun nima qilish kerak? Bunda bir qator mayda amallar bajariladi. Bular: tartibini solishtirish, tartibini tenglash, normallash va boshqa amallar. Dastlabki kompyuterlarning prosessorlari yuqorida keltirilgan barcha “mikro amallarni” har bir argumentlar juftligi uchun natijani hosil qilguncha ketma-ket bajargan va bundan keyin qo’shiluvchilarning keyingi juftligini qayta ishlashga o’tgan. Konveyerli qayta ishlash g’oyasida umumiy amal bir necha bosqichlarga ajratiladi. Har bir bosqich bajarilgandan keyin uning natijasi keyingi bosqichga beriladi va shu bilan birga kiruvchi ma’lumotlarning yangi qismi qabul qilinadi. Bunda oldin bajarilgan amallarni natijalarini qo’llash orqali qayta ishlash tezligi oshiriladi. Faraz qilaylik, amal beshta mikro amaldan iborat bo’lishi va ularni har biri bitta vaqt birligida bajaradi. Agar ajralmas yagona ketma-ket qurilma mavjud bo’lsa, u 100 ta argumentlar juftligini 500 vaqt birligida bajaradi. Agar har bir mikro amal konveyrli qurilmaning alohida bosqichida bajarilsa, u holda bunday qurilmaning har bir qayta ishlash bosqichining beshinchi vaqt birligida birinchi 5 ta argumentlari aniqlanadi. Birinchi natija vaqtning 5-birligidan keyin olinadi. 100 ta juftlikdan iborat

to’plam esa 5+99=104 vaqt birligidan keyin olinadi. Ya’ni parallel qurilmaga nisbatan 5 marta tez bajariladi. Bir qarashda konveyerli qayta ishlashni parallel qurilmalarini o’rniga zarur miqdordagi konveyr qurilmalarini qo’llash mumkindek ko’rinadi. Biroq bunda hosil bo’lgan sistema narxi va murakkabligi oshadi. Unumdorlik esa o’zgarmay qoladi. Parallel dastur tuzish uchun, dasturdagi bir vaqtda va bir-biridan mustaqil prosessorlarda bajariladigan amallar guruhini ajratib olish kerak. Buning imkoniyati mavjudligi dasturda informasion bog’liqliklar mavjudligi yoki yo’qligi bilan aniqlanadi. Agar dasturning biror amali natijasi ikkinchi amal argumenti sifatida qo’llanilsa amallar informasion bog’liq deb ataladi. Agar V amali A amaliga informasion bog’liq bo’lsa, u holda V amali faqt A amali tugagandan keyin bajariladi. Agar A va V amallari informasion bog’liqmas bo’lsa, u holda algoritmda ularni bajarish ketma-ketligiga cheklanish qo’yilmaydi, xususan ular bir vaqtda bajarilishi mumkun. Shunday qilib, dasturni informasion bog’liq amallarni aniqlashdan va ularni hisoblash qurilmalariga taqsimlashdan, sinxronlashdan va zarur kommunikasiyani o’rnatishdan iborat bo’ladi.

2. Ma’lumotlarni almashish va qayta ishlash usullari

Massivli-parallel kompyuterlar paydo bo’lishi bilan parallel jarayonlarni aloqasini qo’lab-quvvatlovchi kutubxona interfeyslar keng tarqaldi. Bu yo’nalishning toifali vakiliga Message Passing Interface (MPI) interfeysi misol bo’ladi. Bu interfeys amalda vektor-konveyerli super - EXM dan tortib shaxsiy kompyutergacha bo’lgan barcha parallel platformalarda mavjud. Dasturning qaysi parallel jarayonlari dasturning qaysi qismida va jarayonlar bilan ma’lumotlar almashishi yoki o’z ishini sinxronlab borishi kerakligini dasturchining o’zi belgilaydi. Odatda parallel jarayonlarning manzil sohasi turlicha bo’ladi. Xususan, bu g’oyaga MPI va PVI da amal qilinadi. Boshqa texnologiyalarda, masalan Shmem da lokal (private) va umumiy (shared) o’zgaruvchilari qo’llaniladi. Bu o’zgaruvchilarga dasturning barcha jarayonlari murojat etishi mumkin va Put/Get toifasidagi operasiyalar yordamida umumiy xotira bilan ishlash usuli tashkil etiladi. Linda sistemasi o’ziga xos

xususiyatga ega bo’lib, unda ixtiyoriy ketma-ketlikda tilga to’rtta: in, out, read va eval funksiyalarini qo’shadi va parallel dasturlar tuzish imkonini beradi. Afsuski, ushbu keltirilgan g’oya oddiy bo’lishiga qaramasdan, uni amalda qo’llash muammolar tug’diradi.

3. Xabarlarni almashish interfeys texnologiyasi

Taqsimlangan xotira parallel kompyuterlardagi keng tarqalgan dasturlash texnologiyasi MPI texnologiyasi hisoblanadi. Bunday sistemalardagi parallel jarayonlarning o’zaro muloqat usuli bir-biri bilan xabarlar almashishdan iborat. Bu usul texnologiyasi nomi - Message Passing Interface (xabarlar almashish interfeysi) deb aks ettirilgan. MPI standartida sistema amal qilishi va dastur yaratishda foydalanuvchilar amal qilishi kerak bo’lgan qoidalar mavjud. MPI Fortran va Si bilan ishlashni qo’llab-quvvatlaydi. Interfeysning to’liq versiyasida 125 tadan ko’proq prosedura va funksiyalar mavjud. MPI MIMD(Multiple Instruction Multiple Data) stilidagi parallel dasturlarni qo’llab quvvatlaydi. Unda turli matnlar bilan berilgan jarayonlar birlashtiriladi. Biroq bunday dasturlarni tuzish va otladka etish murakkab. Shuning uchun amalda dasturchilar SPMD (SINGLE PROGRAM MULTIPLE DATA) parallel dasturlash modelidan foydalanishadi. Unda parallel jarayonlar uchun bitta dastur kodi qo’llaniladi. Xozirgi paytda MPI ko’proq neyron to’rlari bilan ishlashni qo’llab-quvvatlaydi. MPI kutubxonasining parallel jarayonlar bilan mo’ljallangan funksiyalarini qo’llab-quvvatlash uchun dasturni kompilyasiya etishda zarur kutubxona modullarini ulash zarur. Buni buyruq sifatida ulash yoki ko’pgina sistemalarda maxsus mpicc (Si tilidagi dasturlar uchun), mpicc (Si++ tilidagi dasturlar uchun) va mpif77/mpif90 (Fortran77/90 tilidagi dasturlar uchun) buyruq va funksiyalar mavjud. Kompilyatorning “-o name” bo’limi yaratiladigan ishchi fayl a.out deb nomlanadi, Masalan: mpiff77-o program program.f.

Ishchi fayl yaratilgandan keyin uni zarur miqdordagi jarayonlar uchun ishga tushirish kerak. Buning uchun odatda MPI-dasturlarini ishga tushiruvchi mpirun buyrug’i mavjud, Masalan: mpirun – np N< dastur va argumentlari> Bu yerda N –

bitta masaladagi jarayonlar miqdori. Ishga tushirilgandan keyin bitta dastur ishga tushirilgan jarayonlar tomonidan chaqiriladi. Chaqirish natijasi sistemaga bog'liq ravishda terminalga yoki nomi ko'rsatilgan faylga yoziladi. Qolgan barcha obyektlar: proseduralar, konstantalar va MPI da aniqlangan ma'lumotlar toifalari MPI old qo'shimchaga ega bo'ladi. Agar foydalanuvchi dasturda bunday qo'shimchali nomlardan foydalanmasa ham MPI obyektlar bilan muammo tug'ilmaydi. Bundan tashqari Si tilida funksiya nomlaridan bosh va kichik xarflar farqlanadi. Odatda MPI funksiya nomidagi MPI qo'shimchasidan keyingi harf bosh harf bilan, keyingilari kichik harflar bilan yoziladi. MPI konstantalar nomi esa butunligicha bosh harflar bilan yoziladi. MPI interfeysining tavsifi mpif.h (mpi.h) faylida mujassamlashtirgan. Shuning uchun MPI – dastur boshqa include 'mpif.h' direktivasi (ko'rsatmasi) (Si tilida dasturda #include 'mpi.h') joylashishi kerak. MPI dastur – bu o'zaro muloqat qiluvchi parallel dasturlar to'plamidir. Har bir jarayon bir marta yuz beradi va dasturni parallel qismini tashkil etadi. MPI dastur bajarilishi jarayonida qo'shimcha jarayonlarni tashkil etish yoki mavjudlarini yo'qotish mumkin emas. Har jarayon o'zining manzil sohasida joylashadi va umumiy o'zgaruvchilar MPI da yo'q. Jarayonlar o'rtasida muloqatning yagona usuli xabarlar almashishdir. Parallel jarayonlarni muloqatini mustaqil amalga oshirish uchun jarayonlar guruhi tanlab olinib ular uchun alohida muhit kommunikatorlar yaratiladi. Jarayonlar faqat bitta kommunikatorlar ichida muloqat qilinadi va turli kommunikatorlarga yuborilgan xabarlar kesishmaydi. Fotran tilidagi kommunikatorlar INTEGER toifasiga ega bo'ladi. Dastur ishga tushishida yuzaga keladigan jarayonlar to'la qamrovli MPI_COMM_WORLD nomli kommunikator doirasida ishlaydi deb hisoblanadi. Ushbu kommunikator doimo mavjud bo'ladi va ishga tushgan barcha MPI dasturlarni muloqat qilish uchun xizmat qiladi. Bundan tashqari dastur ishga tushishida bitta jarayonga ega bo'lgan MPI_COMM_SELF kommunikatori, hamda bitta ham jarayonga ega bo'lmagan MPI_COMM_NULL kommunikatori mavjud bo'ladi. Jarayonlar o'rtasida muloqatlar ma'lum kommuniqatorlar doirasida amalga oshadi.

Turli kommunikatorlarga berilgan xabarlar kesishmaydi. MPI dasturidagi har bir jarayon o’zi tegishli bo’lgan guruh ichida takrorlanmas atributga – musbat butun sondan iborat jarayon nomeriga ega bo’ladi. Ushbu atribut yordamida jarayonlar o’rtasidagi asosiy muloqatlar olib boriladi. Bitta kommunikatordagi jarayonlar har xil nomerga ega bo’ladi, biroq jarayon bir vaqtda bir necha kommuniqatorga tegishli bo’lishi mumkin. Bu holda jarayonning bitta kommunikator ichida nomeri ikkinchi kommunikator ichidagi nomeridan farq qilishi mumkin. Bunday jarayonning ikkita asosiy atributi tushinarli bo’ladi: kommunikator va kommunikatorda nomer. Agar guruhda n ta jarayon bo’lsa, u holda joriy guruhdagi ixtiyoriy jarayon nomeri 0 dan n – 1 gacha oraliqda yotadi. Jarayonlarni o’zaro muloqat qilishining asosiy usuli xabarlar almashishdan iborat. Xabar–bu biror toifadagi ma’lumotlar to’plamidir. Har bir xabar bir qancha atributlarga ega bo’ladi. Masalan, jo’natuvchi jarayon nomeri, qabul qiluvchi jarayon nomeri, xabar identifikatori va boshqalar. Xabarning asosiy atributlaridan biri uning identifikatori yoki tegi hisoblanadi. Xabarni qabul qilayotgan jarayon xabar identifikatori bo’yicha unga bitta jarayondan kelgan ikkita xabarni farqlab oladi. Xabar identifikatori musbat butun son bo’lib 0 va MPI_TAG_UP diopazonda yotadi va MPI_TAG_UP 32767 dan kichik emas. Xabar atributlari bilan ishlash uchun massiv (Si tilida struktura) kiritilgan. Massiv elementlari yordamida xabar qiymatlariga murojat etiladi. MPI proseduralarini ko’pchiligida oxirgi argumentda muvaffaqiyatli tugaganlik haqidagi ma’lumot qaytariladi. Muvaffaqiyatli bajarilganda MPI_SUCCES qiymati, aks holda xatolik kodi qaytariladi. Prosedura bajarilishida yuz bergan xatolikni uning tavsifidan bilib olishi mumkin. Turli xatolik kodlariga mos tavsiflar mpif.h faylida joylashadi.

4. MPI texnologiyasi

Dsturni parallel qismini o’rnatish MPI_INIT(IERR). Qolgan MPI proseduralar MPI_INIT chaqirilgandan keyin chaqirilishi mumkin. Har bir dasturni parallel qismini o’rnatish faqat bir marta bajariladi Si tilida MPI_INIT funksiyasida ko’rsatgichlar

yordamida dasturning buyruq satridagi argc va argv parametrlari beriladi. Ular yordamida parallel jarayonlarga parametrlar beriladi.

Dasturning parallel qismi ishini tugatish MPI_FINALIZE(IERR). MPI ning ixtiyoriy prosedurasi, shuningdek MPI_INIT ga keyin murojat qilish taqiqlanadi. MPI_FINALIZE chaqirilishigacha xabarlar almashishda uni ishtirok etishini talab qiluvchi dasturning barcha jarayonlar ishi tugatilishi lozim.

MPI ni amalga oshishiga bog'liq ravishda 'Before MPI_INIT' va 'After MPI_FINALIZE' satrlarini dasturning bitta jarayoni yoki ishga tushirilgan barcha jarayonlar chop qilinishi mumkin. 'Parallel section' satrini barcha jarayonlar chop qiladi. Turli jarayonlar tomonidan satrni chiqarish tartibi ixtiyoriy amalga oshadi.

Si tilidagi MPI dasturning umumiy sxemasi quyidagicha tasvirlanadi:

```
#include "mpi.h"
main(int argc,char **argv)
{
  ...
 MPI_INIT(&argc,&argv);
 ...
  MPI_FINALIZE();
  ...
}
```

```
MPI_INITIALIZED(FLAG,IERR)
LOGICAL FLAG
INTEGER IERR
```

Prosedura FLAG argumentida agar prosedura dasturning parallel qismidan chaqirilsa TRUE, aks holda FALSE qiymatini qaytaradi. Navbatdagi prosedura MPI_INIT chaqirilishigacha chaqirilishi mumkin bo'lgan yagona MPI proseduradir:

```
MPI_COMM_SIZE(COMM,SIZE,IERR)
INTEGER COMM,SIZE,IERR
```

Prosedura SIZE argumentida COMM kommunikatoridagi parallel jarayonlar sonini qaytaradi.

MPI_COMM_RANK(COMM,RANK,IERR)

Prosedura RANK argumentida COMM kommunikatoridagi jarayon nomerini qaytaradi. Agar MPI_COMM_SIZE prosedurasi COMM kommunikatori uchun SIZE qiymatini qaytarsa, u holda MPI_COMM_RANK prosedurasining RANK o'zgaruvchi qiymati 0, size–1 diopazonda bo'ladi. Navbatdagi misolda har bir ishga tushirilgan jarayon o'zining MPI_COMM_WORLD kommunikatoridagi nomerini va joriy kommunikatordagi jarayonlar sonini qaytaradi.

```
program example2
include 'mpi.h'
integer   ierr,size,rank
call MPI_INIT(ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, size,ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD, rank,ierr)
print  *, 'process',rank,size,size
call MPI_FINALIZE(ierr)
end
```

Print prosedurasi chaqirilishi natijasida tasvirlanuvchi sart dastur ishga tushishida qancha jarayon yaratilgan bo'lsa shuncha marta tasvirlanadi, catrlarni chiqarish tartibi oldindan ma'lum emas va ixtiyoriy tartibda bo'lishi mumkin. Biroq har bir tasvirlanuvchi satr bir-biri bilan aralashib ketmasligi kafolatlanadi.

DOUBLE PRECISION MPI_WTIME(ierr)

INTYeGYeR IYeRR

Bu funksiya chaqiruvchi jarayonda o'tgan biror vaqt momentidan joriy vaqtgacha o'tgan vaqtni sekundlarda qaytaradi. Agar dasturning biror qismi ushbu funksiya ichiga olinsa, u holda qiymatlar farqi joriy qismni bajarilish vaqtini ko'rsatadi. Bunda o'tgan momenti o'zgarmaydi. Ushbu funksiya o'z ishini natijasini

parametr bilan emas, ochiq holda qaytaradi. Turli jarayonlarning taymerlari sinxronlashmagan bo'lishi mumkin va turlicha qiymat qaytarish mumkin. Bunda MPI_WTIME_IS_GLOBAL (1-sinxronlashgan ,0-yo'q) paramer qiymati orqali aniqlash mumkin.

DOUBLE PRECISION MPI_WTICK(IERR)

INTEGER IERR

Bu funksiya chaqiruvchi jarayonda taymerni qiymatini sekundlarda qaytaradi. Bu funksiya ham o'z ish natijasini parametr orqali emas, balki ochiq holda qaytaradi.

MPI_GET_PROCESSOR_NAME(NAME ,LEN,IERR)

CHARACTER *(*) NAME

INTEGER LEN , IERR

Bu funksiya NAME satrida chaqiruvchi jarayon ishiga tushurilgan tugun nomini qaytaradi. LEN o'zgaruvchida nomdagi simvollar miqdori qaytariladi. Bu qiymat MPI_GET_PROCESSOR_NAME konstanta qiymatidan oshmaydi. Bu prosedura yordamida MPI dasturning jarayonlari qaysi fizik prosessorga mo'ljallanganligi aniqlanadi.

Navbatdagi misolda MPI_GET_PROCESSOR_NAME prosedurasini qo'llash ko'rsatilgan:

```
program example3
include 'mpi.h'
integer ierr,rank,len,i,NTIMES
parametr (NTIMES=100)
character*(MPI_MAX_PROCESSOR_NAME) name
double precision time_start,time_finish,tick
call MPI_INIT(IERR)
call MPI_COMM_RANK(MPI_COMM_WORLD,rank, ierr)
call MPI_GET_PROCESSOR_NAME(name,len,ierr)
tick = MPI_WTICK(ierr)
```

```
time_start = MPI_WTIME(ierr)
do  i=1 , NTIMES
time_finish = MPI_WTIME(ierr)
end do
call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
print *,'processor',name(1:len),
&   'process',rank,': tick =', tick,
&   'time=',(time_finish-time_start)/NTIMES
call MPI_FINALIZE(ierr)
end
```

5.Alohida jarayonlar o’rtasidagi xabarlarni uzatish-qabul qilish

Amalda MPI kommunikasion texnologiyasidan foydalanib yozilgan barcha dasturlar faqat parallel jarayonlarni yaratish va tugatish vositalariga emas, balki jarayonlar o’zaro muloqat qilish vositasiga ham ega bo’lishi kerak. MPI da bunday muloqat xabarlarni yaqqol uzatish orqali amalga oshiriladi. MPI dagi xabarlar almashuvi proseduralar ikkita guruhga bo’linadi.

Birinchi guruhga dasturning faqat ikkita jarayoni muloqat qilish uchun mo’ljallangan proseduralar kiradi. Bu amal individual yoki nuqta-nuqta deb ataladi.

Ikkinchisiga guruh proseduralarida kommunikatordagi barcha jarayonlar xabarlar almashishi inobatga olinadi. Bunday amallar jamoa deb ataladi.

Xabarlar almashish proseduralarini tahlil qilish nuqta-nuqta toifali amal bajaruvchi proseduralardan boshlaymiz, bunday muloqatda ikkita jarayon qatnashadi va ularning biri jo’natuvchisi, ikkinchisi qabul qiluvchi hisoblanadi. Jo’natuvchi jarayon ma’lumotlar almashishini ta’minlovchi proseduralardan birini chiqarishi kerak va biror kommunikatrodagi qabul qiluvchi jarayon nomerini ko’rsatishi kerak, qabul qiluvchi jarayon esa uzatuvchi jarayon nomerni ko’rsatgan holda qabul qilish prosedurasidan birini chiqarishi kerak. Ba’zi hollarda uzatuvchi jarayon nomeri

noma’lum bo’lishi mumkin. Joriy guruh proseduralari ham o’z navbatda ikkita sinfga bo’linadi: blokirovkali prosedura (sinxronli) va blokirovkasiz prosedura (asinxron).

Blokirovkali xabar olish proseduralari almashishi biror shart bajarilishigacha o’zi ishni tugatadi. Asinxron proseduralarini qaytishi esa mos kommunikatsion amal bajarilishi darrov amalga oshadi.

Blokirovkali proseduradan to’g’ri foydalanishdan boshi berk holatlar vujudga kelishi mumkin, shuning uchun qo’shimcha ehtiyot choralarni ko’rish kerak.

Asinxron proseduralaridan foydalanishda boshi berk holatlar yuz bermaydi, biroq ularni qo’llashda ma’lumotlar massividan to’g’ri foydalanish talab etiladi.

mpi_send(buf,count,datatype,dest,msgtag,comm,ierr)

<type> buf(*)

integer count,datatype, dest,msgtag ,comm, ierr

Blokirovkali massiv jo’natish DATATYPE toifali COUNT ta elementlardan iborat MSGTAG identifikatorli BUF xabari COMM kommunikatoridagi DEST nomerli jarayoniga jo’natilgan. Jo’natilgan xabarlarning barcha elementlari ketma-ket holda BUF buferda joylashishi kerak. Bu amal qabul qiluvchi jarayon o’rnatilmagan bo’lsa ham bajariladi. Bunda xabar bevosita qabul buferiga nusxalangani kabi biror sistema buferiga ham joylashtiriladi (agar MPI da belgilangan bo’lsa). COUNT qiymati nol bo’lishi ham mumkin. Jarayon o’z-o’ziga ham xabar berishi mumkin. Bu xavfli emas va boshi berk holatlarga ham, olib kelishi mumkin. DATATYPE parametri Fotran tilida INTEGER toifasiga ega bo’ladi(SI tilida MPI_Datatype). Uzatiladigan elementlar toifasi Fotran tilidan konstantalar bilan ko’rsatiladi. Bu konstantalar quyidagi jadvalda keltirilgan:

MPI dagi ma’lumotlar toifasi	Fotrandagi ma’lumotlar toifasi

MPI_INTEGER	integer
MPI_REAL	REAL
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER(1)
MPI_BYTE.	8 bit, toifalashmagan ma'lumotlar uchun qo'llaniladi.
MPI_PACKYeD	Ma'lumotlar joylashish toifasi.

Agar mpi qo'llaniladigan til ma'lumotlarining qo'shimcha toifasiga ega bo'lsa, u holda bu toifalar mpi da ham qo'llab quvvatlanishi kerak, mavjud toifalarning to'liq ro'yxati mpif.h(mpi.h) faylida keltiriladi. Xabarlar jo'natishda mavjud bo'lmagan jarayonlar uchun maxsus mpi_proc_null qiymatini qo'llash mumkin, bunday jarayonli operasiyalar mpi_success tugatish kodi bilan tez tugaydi. Masalan, bir birlikka ortiq nomerli jarayonga xabar jo'natish uchun quyidagi fragmentdan foydalanish mumkin:

```
call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
next = rank+1
if(next .eq. size) next =MPI_PROC_NULL
call MPI_SEND(buf,1,MPI_REAL,next,
&  5, MPI_COMM_WORLD,ierr)
```

Bu holda jarayon xaqiqatda ma'lumotlar jo'natmaydi va dasturni bajarishda davom etadi.

Blokirovka proseduradan qaytgandan keyin barcha parametrlarni takroriy qo'llanilishini kafolatlaydi, ya'ni mpi_send dan qaytgandan keyin joriy prosedurani chaqirishda qo'llanilgan o'zgaruvchilarning ixtiyoriy birini berilgan xabarni buzilishga xavfsiramasdan ishlatish mumkin. Buni kafolatlash usulini tanlash: oraliq buferga nusxalash yoki bevosita dest jarayoniga berish mpi ni yaratayotgan dasturchilar ixtiyotiga bog'liq.

Shuni ta’kidlab o’tish kerak, mpi_send jarayonidan qaytish xabar dest jarayonning jarayon elementini tark etganini ham bildirmaydi. Faqat joriy prosedurani chaqirishda foydalanilgan o’zgaruvchilarni bexatar o’zgartirish kafolatlanadi. Shunga o’xshash noaniqliklar har doim ham foydalanuvchilarni qanoatlantirmaydi. Xabarlar almashish imkoniyatini kengaytirish uchun mpi da uchta prosedura mavjud. Bu proseduralardagi parametrlar ham xuddi mpi_send dagi kabi, biroq ularni har biri o’ziga xos xususiyatga ega. mpi quyidagi ma’lumotlar almashish proseduralar modifikasiyalarini qo’llab quvvatlaydi:

- mpi_vsend buferatsiya bilan xabar uzatish. Agar uzatilgan xabar qabul qiluvchi tomonidan qabul qilinmasa, u holda xabar maxsus buferga yoziladi va tezda proseduradan qaytariladi. Bu prosedurani bajarilishi mos qabul qiluvchi prosedurani chaqirilishiga bog’liq emas, qolaversa, agar buferda yetarlicha joy bo’lmasa prosedura xatolik kodini qaytaradi. Buferlash uchun massiv belgilashni ishini foydalanuvchi o’z zimmasiga olishi kerak.

- mpi_ssend sinxronlab xabarlar uzatish, ushbu proseduradan chiqish faqat jo’natilgan xabar qabul qiluvchi jarayon tomonidan qabul qilingandan keyin mumkin. Bu bilan xabar almashishni sinxronlab tugatish orqali jo’natma buferidan takroriy foydalanish imkoniyati yaratiladi va dasturdagi qabul qiluvchi jarayon xabarni qabul qilishi kafolatlanadi, sinxronlab xabar almashish dastur bajarilishini sekinlatadi, biroq sistemada xabarlarni ortiqcha buferlardan holi qiladi.

-mpi_rsend tayorgarlik bo’yicha xabar uzatish. bu proseduradan faqat qabul qiluvchi jarayon qabulni amalga oshirgandan keyin foydalanish mumkin, aks holda bu prosedura chaqirilishi noto’g’ri hisoblanadi va uning natijasi mavxum bo’ladi. mpi_rsend prosedurasi chaqirilishidan oldin xabar qabul qilinganligini kafolatlash uchun jarayonlarni yaqqol va noyaqqol sinxronlash opersiyalaridan foydalanish kerak, ko’pincha mpi_rsend prosedurasi uzatuvchi va qabul qiluvchi o’rtasidagi muloqat bayonnomasini qisqartiradi va ma’lumotlar almashishdagi sarf-harajatlarni kamaytiradi.

Foydalanuvchi jo’natuvchi jarayonda mpi_vsend prosedurasini chaqirganda xabarlarni buferlash uchun zarur maxsus massiv belgilash kerak.

mpi_vuffyer_attach(vuf,sizye,iyerr)

<typye> vuf(*)

intyegyer sizye, iyerr

size o’lchami buf massivini buferlab xabar jo’natishda ishlatish uchun yaratish, bunday buferdan har bir jarayonda faqat bittasi bo’lishi kerak, bufer sifatida belgilangan massivdan dasturda maqsadlarda ishlatmaslik kerak. Buferlash uchun yaratilgan massiv o’lchami umumiy o’lchamidan kichik bo’lmasligi kerak va minimum mpi_vsend_overhead konstantasidan aniqlangan qiymatga teng bo’lishi kerak.

mpi_vuffyer_dyetach(vuf, sizye, iyerr)

<type> vuf (*)

intyegyer sizye, iyerr

Ajralgan bufer massivini boshqa maqsadlarda ishlatish uchun bo’shatish, bu prosedura buf va size argumentlarida bo’shatilgan massiv manzili va o’lchami qiymatlarini qaytaradi. Bu prosedurani chaqirgan jarayon joriy buferdagi xabarlar jo’natilguncha uni blokirovkalab turadi.

Odatda mpi da jo’natiladigan xabarlarni buferlash uchun biror o’lchamdagi xotira qo’llaniladi, biroq dastur barcha buferlovchi jo’natmalarga yetarli buferni aniq ko’rsatish tavsiya etiladi.

Navbatdagi misolda buferlab xabarlar almashish qo’llanilgan:

```
program example4
include 'mpi.h'
integer BUFSIZE
parameter (BUFSIZE =4+MPI_BSEND_OVERHEAD)
byte buf (BUFSIZE)
integer rank,ierr,ibufsize, rbuf
```

```
integer status(MPI_STATUS_SIZE)
call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,rank,ierr)
if(rank .eq.  0)  then
call  MPI_BUFFER_ATTACH(buf,BUFSIZE, ierr)
call  MPI_BSEND(rank,1,MPI_INTEGER,1, 5, MPI_COMM_WORLD,ierr)
&  call MPI_BUFFER_DETACH(buf,ibufsize,ierr)
end if
if(rank  .eq. 1) then
call MPI_RECV(rbuf,1,MPI_INTEGER,0,5,MPI_COMM_WORLD,ierr)
print  *,'Process 1 received',rbuf,'from process',status(MPI_SOURCE)
end if
call MPI_FINALIZE(ierr)
end
MPI_RECV(BUF,COUNT,DATATYPE,SOURCE,MSGTAG,COMM,
STATUS,IERR)
<type> BUF(*)
INTEGER COUNT,DATATYPE,SOURCE,MSGTAG,COMM,IERR,
STATUS(MPI_STATUS_SIZE)
```

datatype toifali, elementlari count dan ko’p bo’lmagan msgtag identifkatorli xabar elementlari bilan buf buferiga blokirovkalab qabul qilish, agar real qilingan elementlar soni count miqtoridan kam bo’lsa, u holda buf buferida qabul qilingan xabar elementlarga mos elementlar o’zgaradi. Agar qabul qilinayotgan elementlari count qiymatidan ko’p bo’lsa, u holda xatolik yuz beradi. Buni bartaraf etish uchun mpi_probe(mpi_iprobe) prosedurasi yordamida keladigan xabar strukturasi aniqlanadi. Agar qabul qilingan xabarlarda elementlar sonini aniq bilish kerak bo’lsa, u holda mpi_get_count prosedurasidan foydalanish mumkin. Blokirovkalash amali

mpi_recv proseduraSidan qaytgandan keyin xabarning barcha elementlari qabul qilingan va buf buferiga joylashgan bo'ladi.

Quyidagi keltirilgan dasturda nol jarayoni bir raqamli jarayonga xabar jo'natadi va undan javob kutadi. Agar dastur ko'p sonli jarayon bilan ishga tushirilsa, u holda jo'natmalar bajarish nol bo'ladi. Qolgan jarayonlar MPI_INIT prosedurasi yordamida tashkil etilgandan keyin a va b o'zgaruvchilarning boshlang'ich qiymatlarini chop qiladi, keyin MPI_FINALIZE prosedurasini bajarib ishini tugatadi.

```
program example5
include 'mpi.h'
integer ierr,size,rank
real a,b
integer status(MPI_STATUS_SIZE)
call MPI_INIT(ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD,size,ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,rank,ierr)
a=  0.0
b= 0.0
if(rank  .eq. 0) then
b=1.0
call MPI_SEND(b,1,MPI_REAL,1 ,5,MPI_COMM_WORLD,ierr);
call MPI_RECV(a,1,MPI_REAL, 1,5,MPI_COMM_WORLD,status,ierr);
else
if (rank  .eq. 1) then
a=2.0
call MPI_RECV(b,1,MPI_REAL, 0,5,MPI_COMM_WORLD,status,ierr);
call MPI_SEND(a,1,MPI_  REAL ,0,5,MPI_COMM_WORLD,ierr);
end if
end if
```

```
print  *,'process',rank,'a=',a,'b=',b
call MPI_FINALIZE(ierr)
end
```

Navbatdagi misolda har bir toq nomerni jarayon bir birlikka, ko’p nomerli qo’shni jarayonga xabar jo’natadi. Qo’shimcha ravishda eng katta nomerni jarayonga xabar jo’natmasligi uchun tekshiriladi. b o’zgaruvchining qiymati faqat toq nomerli jarayonlarda o’zgaradi.

```
program yexamplye6
include 'mpif.h'
integer ierr,size,rank,a,b
integer status(MPI_STATUS_SIZE)
call MPI_INIT(ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD,size,ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,rank,ierr)
a=rank
b=-1
if(mod(rank,2) .eq. 0) then
if(rank+1 .lt. size) then
call MPI_Send(a,1,MPI_INTEGER,rank+1,5,MPI_COMM_WORLD,ierr);
end if
else
callMPI_Recv(b,1,MPI_INTEGER,rank-1,5,MPI_COMM_WORLD,status,ierr);
end if
print *,'process',rank,'a=',a,'b=',b
call MPI_FINALIZE(IERR)
end
```

Keltirilgan misolda SOURCE va MSGTAG argumentlari o’rniga quyidagi konstantalarni qo’llash mumkin:

- MPI_ANY_SOURCE itiyoriy jarayondan berilgan xabar mos kelishini bildiruvchi belgi;
- MPI_ANY_TAG ixtiyotiy identifikatorli xabar mos kelishini bildiruvchi belgi;

Ushbu ikkita konstanta baravar ishlatilganida ixtiyoriy jarayonning ixtiyoriy identifikatorli xabar qabul qilinadi. Qabul qilingan xabarning aniq atributlarini status massivining mos elementlari yordamida aniqlash mumkin. Fotran status parametri butun sonli va MPI_STATUS_SIZE o'lchami massiv hisoblanadi. MPI_SOURCE, MPI_TAG va MPI_ERROR konstantalari berilgan massivning mos elementlari qiymatlariga murojat etish qoidasi hisoblanadi:

- status (MPI_SOURCE) xabarni jo'natuvchi jarayon nomeri;
- status (MPI_TAG) xabar identifikatori;
- status (MPI_ERROR) xatolik kodi;

Si tilidagi status parametri MPI_SOURCE, MPI_TAG va MPI_ERROR maydonlaridan iborat MPI_SOURCE toifali struktura hisoblanadi.

Xabarlarni jo'natish va qabul qilish amallarining ba'zi nosimmetrik jihatlariga e'tibor qiling. MPI_ANY_SOURCE konstanta yordamida ixtiyoriy jarayondan xabar qabul qilish mumukin. Biroq ma'lumotlarni jo'natishda qabul qiluvchi jarayonning nomerini aniq ko'rsatish kerak. Standartga ko'ra, agar bir jarayon bitta MPI_RECV chaqiruvga ketma-ket ikkita xabar jo'natsa, ikkinchi jarayon dastlab oldin jo'natilgan xabarni qabul qiladi. Shu bilan birga agar ikkita xabar bir vaqtda turli jarayonlar tomonidan jo'natilsa u holda qabul qiluvchi jarayon tomonidan xabarlarni qabul qilish tartibi oldindan ma'lum emas.

MPI_GET_COUNT(STATUS,DATATYPE,COUNT,IERR)

INTEGER COUNT,DATATYPE,IERR,STATUS(MPI_STATUS_SIZE)

status parametrining qiymatlari orqali prosedura qabul qilingan xabarning (MPI_RECV ga murojat etgandan keyin) yoki DATATYPE toifali xabarning qabul qilingan elementlarining (MPI_PROBE yoki MPI_IPROBE ga murojat etib) COUNT

miqdorini aniqlaydi. Joriy prosedura qabul qilingan xabarlarni saqlash uchun ajratilgan xotira sohasining xajmini o'lchash uchun zarur.

MPI_PROBE(SOURCE,MSGTAG,COMM,STATUS,IERR)

INTEGER SOURCE, MSGTAG, COMM,IERR, STATUS(MPI_STATUS_SIZE)

status massivida COMM kommunikatoridagi SOURCE nomeri jarayondan kutilayotgan MSGTAG identifikatori xabar strukturasi xaqidagi ma'lumotlarni olish. Bu proseduradan qaytish mos identifikatorli xabar va mos jo'natuvchi jarayon nomeri o'qish uchun mumkin bo'lgandagina amalga oshadi. Bu prosedura xabar kelish faktini aniqlaydi (uni qabul qilmaydi). MPI_PROBE chaqirilgandan keyin MPI_RECV ham shu parametrlar bilan chaqirilsa, u holda ham MPI_PROBE prosedurasi yordamida olingan xabarlar qabul qilinadi.

Navbatdagi misolda keluvchi xabar strukturasini aniqlash uchun MPI_PROBE prosedurasini qo'llash namoish etilgan. 0 jarayoni 1 va 2 jarayonlarning ixtiyoriy biridan xabar kutadi. Biroq ushbu jarayonlar jo'natadigan xabarlar xar xil toifaga ega. Keluvchi xabarni qaysi o'zgaruvchiga joylashishni aniqlash uchun jarayon avval MPI_PROBE chaqiruvi yordamida xabar qayerdandan kelganligini aniqlaydi. Bevosita MPI_PROBE dan keyingi MPI_RECV chaqiruvi ishonchli ravishda xabarni qabul qiladi va shundan keyin boshqa jarayondan xabar qabul qilinadi.

```
program example7
include 'mpif.h'
integer rank,ierr,ibuf,status(MPI_STATUS_SIZE)
real rbuf
call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,rank,ierr)
ibuf=rank
rbuf=1.0*rank
if(rank.eq.1)
call MPI_SEND(ibuf,1,MPI_INTEGER,0,5,MPI_COMM_WORLD,ierr)
```

```
if(rank.eq.2)
call MPI_SEND(rbuf,1,MPI_REAL, 0,5,MPI_COMM_WORLD,ierr)
if(rank .eq.0) then
call MPI_PROBE(MPI_ANY_SOURCE,5,MPI_COMM_WORLD,ierr)
if(status(MPI_SOURCE) .EQ.1) then
call MPI_RECV(ibuf,1,MPI_INTEGER,1,5,MPI_COMM_WORLD,status,ierr)
call MPI_RECV(rbuf,1,MPI_REAL,2,5,MPI_COMM_WORLD,status,ierr)
else
if(status(MPI_SOURCE) .EQ.2) then call
MPI_RECV(rbuf,1,MPI_REAL,2,5,MPI_COMM_WORLD,status,ierr)
call MPI_RECV(ibuf,1,MPI_INTEGER,1,5,MPI_COMM _WORLD,status,ierr)
end if
end if
print *,'Process 0 recv' ,ibuf,'from process 1',rbuf,'from process 2'
end if
call MPI_FINALIZE(ierr)
end
```

Navbatdagi misolda ikkita jarayonni galma-gal xabar almashish modeli ko’rsatiladi. Bunda xabar almashish vaqti xabar uzunligiga bog’liq aniqlanadi. Shu tarzda parallel kompyuterdagi kommunikatsion tarmoqning asosiy xarakteristikasi aniqlanadi. Bular: yashiringanlik (nol uzunlikdagi xabarni almashish vaqti) va maksimal o’tkazuvchanlik qobilyati (bir sekundagi Mbayt miqdorida), hamda xabarning uzunligi, NMAX konstanta uzatiladigan xabarning maksimal uzunligi cheklanishini bildiradi, NTIMES konstantasi esa natijani o’rtachasini aniqlashdagi takrorlanishlar miqdorini bildiradi. Yashirinlikni aniqlash uchun avval nol uzunlikdagi xabar jo’natiladi, keyin xabar uzunligi ikki marta ko’payadi. Bunda jo’natish avval real*8 toifali bitta elementni jo’natishdan boshlanadi.

```
program example8
```

```
include 'mpif.h'
integer ierr,rank,size,i,n,1max,NMAX,NTIMES
parameter (NMAX=1 000 000,NTIMES=10)
double precision time_start,time,bandwidth,max
real *8 a(NMAX)
integer status (MPI_STATUS_SIZE)
call MPI_INIT(ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD,size,ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,rank,ierr)
time_start=MPI_WTIME(ierr)
n=0
max= 0.0
lmax=0
do while (n.le.NMAX)
  time_start=MPI_WTIME(ierr)
do i=1,NTIMES
     if(rank .eq. 0) then
callMPI_SEND(a,n,MPI_DOUBLE_PRECISION,1,1, MPI_COMM_WORLD,ierr)
callMPI_RECV(a,n,MPI_DOUBLE_PRECISION,1,1,
MPI_COMM_WORLD,status,ierr)
end if
if(rank .eq. 1) then
callMPI_RECV(a,n,MPI_DOUBLE_PRECISION,0,1,
MPI_COMM_WORLD,status,ierr)
callMPI_SEND(a,n,MPI_DOUBLE_PRECISION,0,1, MPI_COMM_WORLD,ierr)
end if
enddo
time=(MPI_WTIME(ierr)-time_start)/2/NTIMES
```

```
bandwidth=(8*n*1.d0/(2**20))/time
if(max .lt. bandwidth) then
max=bandwidth
lmax=8*n
end if
if(rank .eq.0) then
 if(n.eq.0) then
print *,'latency=',time,'seconds'
else
print *, 8*n,'bytes,bandwidth=',bandwidth,'Mb/s'
end if
end if
if(n .eq.0) then
n=1
else
n=2*n
end if
end do
if(rank .eq.0) then
print *, 'max bandwidth=', max, 'Mb/s,length='
,lmax,'bytes'
end if
call MPI_FINALIZE(ierr)
end
```

6. Blokirovkalangan xabarlarni jo’natish va qabul qilish

MPI da ma’lumotlarni asinxron almashish uchun bir qator proseduralar mavjud. Blokirovkalovchi proseduralardan farqli ravishda bunday proseduralardan qaytish chaqirilgan keyin hyech bir jarayonni ishi to’xtatilmasdan darrov amalga oshadi.

Dastur bajarilishi davomida asinxron ishga tushirilgan amallar yuz beradi va qayta ishlanadi. Amalda, bu imkoniyat effektiv dasturlar tuzishda juda foydali hisoblanadi. Aslida, dasturchi biror holatda unga boshqa jarayonda hisoblanayotgan massiv kerak bo'lishini biladi. Dasturchi oldindan joriy massivni olish uchun dasturda asinxron so'rov joylaydi. Massiv haqiqatda zarur bo'lishiga qadar u boshqa bir foydali maqsadlarda qo'llanilib turadi. Bu yerda ham ko'pchilik holatlarda xabarni jo'natib bo'linishini kutmasdan navbatdagi hisoblashlarni bajarish mumkin. Asinxron xabar almashishni tugatish uchun xabar almashish amali tugaganligini yoki davom etayotganligini tekshiruvchi qo'shimcha prosedura zarur bo'ladi. Faqat shundan keyingina almashinayotgan xabarni buzilishiga xavfsiramasdan jo'natish buferidan foydalanish mumkin. Agar xabarlarni uzatish qabul qilish amallarini hisoblashlar fonida yashirin imkoniyati mavjud bo'lsa, undan foydalanish kerak. Biroq amaliyot doimo nazariya bilan mos kelavermaydi. Bu konkret xaqiqiylikka bog'liq. Afsuski, asinxron amallar apparatura va sistema tomonidan effektiv qo'llab-quvvatlanmaydi. Shuning uchun fonda hisoblashlarni bajarish effekti nolga teng yoki unchalik katta bo'lmasa xayron bo'lmaslik kerak. Keltirilgan eslatmalar effektivlik bilan bog'liq. Shunday bo'lsada asinxron amallar funksionalligi bo'yicha juda foydali hisoblanadi va shuning uchun asinxron amallar har bir xaqiqiy dasturda mavjud.

MPI_ISEND(BUF,COUNT,DATATYPE,DEST,MSGTAG,COMM,
REQUEST,IERR)
<type> BUF(*)
INTEGER COUNT,DATATYPE,DEST,MSGTAG,COMM,REQUEST,IERR

BUF buferidan blokirovkalamasdan COMM kommunikatoridagi DEST jarayoniga MSGTAG identifikatorli DATATYPE toifasi xabarning COUNT elementlarini jo'natish. Bu proseduradan qaytish uzatish jarayoni yaratilgandan keyin, BUF buferidagi to'la xabar qayta ishlanib bo'linmasidan darhol amalga oshadi. Ya'ni bu holat joriy buferdan jo'natish ishlari tugatilganligini tasdiqlovchi qo'shimcha ma'lumotlarni olmasdan foydalanish mumkin emasligini bildiradi. BUF buferidan

undagi uzatilayotgan xabarni buzilish xataridan holi ravishda qayta foydalanish momentini qaytaruvchi REQUEST parametri yordamida MPI_WAIT hamda MPI_TEST toifali proseduralar yordamida aniqlash mumkin. REQUEST parametri Fotran tilida INTEGER toifaga ega va ma'lum blokirovkalanmagan amalni identifikatsiyalash uchun qo'llaniladi. MPI_SEND prosedurasini uchta modifikatsiyasiga mos uchta qo'shimcha MPI_ISEND prosedura mavjud. Bular quyidagilar:

- MPI_IBSEND buferdagi xabarni blokirovkalamasdan jo'natish;
- MPI_ISSEND xabarlarni sinxronli blokirovkalamasdan jo'natish;
- MPI_IRSEND tayyorligi bo'yicha blokirovkalamasdan xabar jo'natish;

MPI_IRECV(BUF,COUNT,DATATYPE,SOURCE,MSGTAG,
COMM,REQUEST,IERR) <type> buf(*)
INTEGER COUNT ,DATATYPE,SOURCE,MSGTAG,COMM,REQUEST,IERR

STATUS massivini to'ldirgan holda COMM kommunikatoridagi SOURCE nomerli jarayondan MSGTAG identifikatorli DATATYPE toifali xabarni COUNT dan ko'p bo'lmagan elementlarini blokirovkalamasdan BUF buferiga qabul qilish. Blokirovkalab qabul qilishdan farqli ravishda proseduradan qaytish qabul jarayoni yuzaga kelish bilan va to'la xabar qabul qilinishi, hamda u BUF buferidan yozilmasdan darrov amalga oshadi. Qabul jarayoni tugaganligi REQUEST parmetri, hamda MPI_WAIT va MPI_TEST oilasidagi proseduralari yordamida aniqlanadi.

MPI_SEND, MPI_ISEND proseduralarini ixtiyoriy biridan, ularning uchta modifikatsiyasini ixtiyoriy biridan jo'natilgan xabar MPI_RESV va MPI_IRESV proseduralarini ixtiyoriy biri tomonidan xabar qabul qilinadi.

Blokirovkalanmagan amallarni qo'llashda bu amal tugamagunicha ishlatilayotgan massivga ma'lumotlar yozmaslik kerak.

MPI_IPROBE(SOURCE,MSGTAG,COMM,FLAG,STATUS,IERR)
LOGICAL FLAG
INTEGER SOURCE,MSGTAG,COMM,IERR,STATUS(MPI_STATUS_SIZE)

COMM kommunikatoridagi SOURCE nomerli jarayondan kutilayotgan MSGTAG identifikatorli xabar strukturasi haqida ma’lumotlarni STATUS massivida joylash. Agar mos atributli xabar qabul qilinishi mumkin bo’lsa, FLAG parametri qiymati TRUE ga teng ko’rsatilgan atributli xabarni xali qabul qilish mumkin bo’lmasa FLAG parametri qiymati FALSE ga teng bo’ladi.

MPI_WAIT(REQUEST,STATUS,IERR)

INTEGER REQUEST,IERR,STATUS(MPI_STATUS_SIZE)

REQUEST identifikatori bilan mos bog’langan COUT asinxron amalini tugashini kutish. Blokirovkalanmagan qabul qilish amallari uchun STATUSES massivida mos parametr aniqlanadi. Agar bitta yoki bir necha ma’lumotlar almashish amalida xatolik yuz bersa, u holda STATUSES massividagi elementlarning xatolik maydonida mos qiymatlar o’rnatiladi. Prosedura bajarilgandan keyin REQUEST parametrinig mos elementlari MPI_REQUEST_NULL qiymatida o’rnatiladi.

Quyidagi keltirilgan misolda barcha jarayonlar blokirovkalanmagan amallar yordamida halqa topologiyasiga mos holda yaqin qo’shnilari bilan xabarlar almashadi. Ushbu maqsadda blokirovkalovchi amallar qo’llanilsa tupik xolatlar yuzaga keladi.

```
program example9
include 'mpif.h'
integer ierr,rank,size,prev,next,reqs(4),buf(2)
integer stats(MPI_STATUS_SIZE,4)
call MPI_INIT(ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD,size,ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,rank,ierr)
prev=rank-1
next=rank+1
if(rank .eq. 0) prev =size-1
if(rank .eq. size-1) next=0
callMPI_IRECV(buf(1),1,MPI_INTEGER,prev,5,
```

```
MPI_COMM_WORLD,reqs(1),ierr)
callMPI_IRECV(buf(2),1,MPI_INTEGER,next,6,MPI_COMM_WORLD,
reqs(2),ierr)
callMPI_ISEND(rank,1,MPI_INTEGER,prev,6,MPI_COMM_WORLD,
reqs(3),ierr)
callMPI_ISEND(rank,1,MPI_INTEGER,next,5,MPI_COMM_WORLD,
reqs(4),ierr)
call MPI_WAITALL(4,reqs,stats,ierr);
print *,'process',rank,'prev=',buf(1),'next=',buf(2)
call MPI_FINALIZE(ierr)
end
```

MPI_WAITANY(COUNT,REQUEST,INDEX,STATUS,IERR)

INTEGER COUNT,REQUEST(*),INDEX,STATUS(MPI_STATUS_SIZE),IERR

REQUEST identifikatori bilan bog'langan asinxron COUNT amallardan birini tugashini kutish. Agar chaqiruv vaqtigacha kutilayotgan amallardan bir necha tugagan bo'lsa, u holda tasodifiy ravishda ixtiyoriy biri tanlanadi. INDEX parametrida REQUEST massividagi tugagan amal identifikator saqlanuvchi element nomeri joylashadi. Blokirovkalamaydigan qabul qilishda STATUS parametri aniqlanadi. Prosedura bajarilgandan keyin REQUEST massivining mos elementida MPI_REQUEST_NULL qiymati o'rnatildi.

MPI_WAITSOME(INCOUNT,REQUEST,OUTCOUNT,INDEXES,

STATUSES,IERR)

INTEGER INCOUNT,REQUEST(*),

OUTCOUNT,INDEXES(*),IERR,STATUSES(MPI_STATUS_SIZE,*)

REQUEST identifikatori bilan bog'langan INCOUNT ta asinxron amallardan birini tugashini kutish. INDEXES massivining dastlabki OUTCOUNT elementida REQUEST massivi elementlarining nomeri va identifikatori saqlanadi. STATUSES massivining dastlabki OUTCOUNT elementida tugagan amallar parametrlari

saqlanadi. Prosedura bajarilishi tugagandan keyin REQUESTS parametrining mos elementlarida MPI_REQUEST_NULL qiymati o’rnatiladi.

Navbatdagi misolda "master-slave" (barcha jarayonlar bitta jarayon bilan muloqat qiladi) kommunikasion sxemani tashkil etish uchun MPI_WAITSOME proseduralidan foydalanish ko’rsatilgan. 0 jarayonidan tashqari barcha jarayonlar siklning xar bir iterasiyasiga slave proseduralini chaqirish orqali massivning o’ziga tegishli lokal qismini aniqlab oladi va shundan keyin uni bosh jarayonga jo’natadi. 0 jarayoni avval qolgan barcha jarayonlardan blokirovkalangan qabulni belgilaydi, real dan keyin aqalli bittaga xabar kelishini kutadi. Keluvchi xabarlar uchun 0 jarayoni qayta ishlovchi master jarayonini chaqiradi. Bundan keyin yana blokirovkalanmagan qabulni o’rnatadi. Shu tarzda 0 jarayoni joriy vaqtda tayyor bo’lgan ma’lumotlar blokini qayta ishlaydi. Bunda dastur to’g’ri ishlashi uchun 0 jarayoni kelgan xabarlarni qayta ishlab ulgurishini ta’minlash kerak. Ya’ni slave proseduralari master proseduralariga nisbatan ko’proq ishlashi kerak. Bundan tashqari misolda cheksiz sikl yozilgan, shuning uchun aniq dastur uchun ishni tugatish sharti bo’lishi kerak.

```
program example10
include 'mpif.h'
integer rank,size,ierr,N,MAXPROC
parametr(N=1000,MAXPROC=128)
integer req(MAXPROC),num,indexes(MAXPROC)
integer statuses(MPI_STATUS_SIZE,MAXPROC)
double precision a(N,MAXPROC)
call MPI_INIT(ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD,size,ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,rank,ierr)
if(rank .ne. 0) then
do while (.TRUE.)
call slave (a,N)
```

```
callMPI_SEND(a,N,MPI_DOUBLE_PRECISION,0,5, MPI_COMM_WORLD,ierr)
end do
else
do i=1,size-1
callMPI_IRECV(a(l,i),N,
MPI_DOUBLE_PRECISION,i,5,MPI_COMM_WORLD,req(i),ierr)
end do
do while (.TRUE.)
call MPI_WAITSOME(size-1,req,num,indexes,statuses,ierr)
do i=1,num
call master(a(1,indexes(i)),N)
callMPI_IRECV(a(1,indexes(i)), N,MPI_DOUBLE_PRECISION,
indexes(i),5,MPI_COMM_WORLD,req(indexes(i)),ierr)
end do
end do
end if
call MPI_FINALIZE(ierr)
end
subroutine slave (a,n)
double precision a
integer n
a massivning lokal qismini qayta ishlash
end
subroutine master(a,n)
double precision a
integer n
a massivni qayta ishlash
end
```

MPI_TEST(REQUEST,FLAG,STATUS,IERR)

LOGICAL FLAG

INTEGER REQUEST,IERR,STATUS(MPI_STATUS_SIZE)

REQUEST identifikatori bilan bog'langan MPI_ISEND yoki MPI_IRECV asinxron amalning tugashini tekshirish. FLAG parametrida qiymat qaytariladi. Agar amal tugagan bo'lsa, bu qiymat TRUE, aks holda FALSE ga teng bo'ladi. Agar qabul prosedurasi tugagan bo'lsa, u holda qabul qilingan xabar atributlari va uzunligini STATUS parametri yordamida oddiy tarzda aniqlash mumkin. Prosedura bajarilganidan keyin REQUEST parametrining mos element MPI_REQUEST_NULL qiymatida o'rnatiladi.

MPI_TESTALL(COUNT,REQUESTS,FLAG,STATUSES,IERR)

LOGICAL FLAG

INTEGER COUNT,REQUESTS(*),STATUSES(MPI_STATUS_SIZE,*),IERR

REQUEST identifikatori bilan bog'langan asinxron COUNT amalini tugashini tekshirish. Agar ko'rsatilgan identifikator bilan bog'langan barcha amallar tugagan bo'lsa, FLAG parametrida prosedura TRUE qiymatini qaytaradi. Bu holda xabar parametri STATUSES massivida ko'rsatiladi. Agar amallardan biri tugamagan bo'lsa, u holda FALSE qiymati qaytariladi va STATUSES massivi elementlarining aniqlanganligi kafolatlanmaydi. Prosedura bajarilgandan keyin REQUEST parametrlarining mos elementlari MPI_REQUEST_NULL qiymatida o'rnatiladi.

MPI_TESTANY(COUNT,REQUESTS,INDEX,FLAG,STATUS,IERR)

LOGICAL FLAG

INTEGER

COUNT,REQUESTS(*),INDEX,FLAG,STATUS(MPI_STATUS_SIZE),IERR

REQUEST massividagi identifikatorlar bilan bog'langan asinxron amallardan kamida bittasi tugaganligini tekshirish. Agar asinxron amallardan aqalli bittasi tugagan bo'lsa, FLAG parametrida TRUE qiymati qaytariladi. Bunda INDEX va REQUESTS massividagi mos elementlar nomeri saqlanadi, STATUS da esa xabar parametrlari

saqlanadi. Agar bitta ham tugamagan bo'lsa, FALSE qiymati qaytariladi. Agar yuqoridagi prosedurasini chaqirish vaqtida bir necha kutilayotgan amallardan bir nechasi tugagan bo'lsa, u holda tasodifiy ravishda ulardan biri tanlanadi. Prosedura bajarilgandan keyin REQUESTS parametrining mos elementlari MPI_REQUEST_NULL qiymatida ko'rsatiladi.

MPI_TESTSOME(INCOUNT,REQUESTS,OUTCOUNT,INDEXES,
STATUSES,IERR)

INTEGER INCOUNT,REQUESTS(*),OUTCOUNT,INDEXES(*),IERR,STATUSES
(MPI_STATUS_SIZE,*)

Keltirilgan proseduraga MPI_WAITSOME prosedurasi o'xshash bo'lib, bu prosedura tez qaytariladi. Agar joriy prosedurani chaqirish vaqtigacha testlanmayotgan amallarning xech biri tugamagan bo'lsa, u holda OUTCOUT qiymatini 0 ga teng bo'ladi.

Navbatdagi misolda jarayonlar o'rtasida satlari bo'yicha taqsimlangan kvadrat matrisani transponirlashni bajaruvchi blokirovkalanmagan amallarni qo'llash namoyish etilgan. Avval xar bar jarayon lokal holda a massivning nl satrini aniqlaydi. Keyin MPI_ISEND va MPI_IRECV blokirovkalamaydigan amallar yotdamida transponirlash uchun zarur bo'lgan ma'lumotlar almashunivlar belgilanadi. Boshlangan almashinuvlar fanida har bir jarayon o'ziga tegishli A massivning lokal qismini transponirlaydi. Bundan keyin jarayon MPI_WAITWAY prosedurasini chaqirish orqali boshqa ixtiyoriy bir jarayondan xabar kelishini kutadi va joriy jarayondan kelgan a massiv qismini transponirlaydi. Qayta ishlash barcha jarayonlardan xabar olib bo'lguncha davom etadi. Oxirida berilgan a massiv va transponirlangan b massiv chop qilinadi.

```
program example11
include 'mpif.h'
integer ierr,rank,size,N,nl,i,j
parametr (N=9)
```

```
double precision a(N,N),b(N,N)
call MPI_INIT(ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD,size,ierr)
call MPI_COMM_RANK(MPI_COMM_RANK,rank,ierr)
n1=(N-1)/size+1
call work(a,b,n,n1,size,rank)
call MPI_FINALIZE(ierr)
end
```

7. Muloqatning kechiktirilgan so’rovlari

Joriy guruh proseduralari bir doirada uzatish qabul qilishdagi ortiqcha xarajatlarni kamaytirishga imkon yaratadi. Shuningdek jarayonlar va tarmoq nazorati o’rtasida zarur ma’lumotlar almashishdagi ortiqcha urinishlarni kamaytiradi. Ko’pincha dasturda bir xil parametrli almashinuvlarni ko’p marta bajarishga to’g’ri keladi. Bunday holda almashish amalini bir marta belgilash va keyin undan ma’lumotlarni ichki strukturasini belgilash va qo’llash uchun ortiqcha vaqt yo’qotmasdan har bir yaqinlanishida ko’p marta foydalanish mumkin. Bundan tashqari shu tarzda bir nechta uzatish qabul qilish so’rovlarini bitta buyruq bilan ishga tushirish uchun birlashtirish mumkin. Xabarni qabul qilish usuli uni jo’natish usuliga bog’liq emas. Yuqorida keltirilgan so’rovlar yoki oddiy usulda jo’natilgan xabarlar oddiy usulda ham yoki kechiktirilgan so’rovlar yordamida ham qabul qilinishi mumkin.

```
MPI_SEND_INIT(BUF,COUNT,DATATYPE,DEST,MSGTAG,COMM,
REQUEST,IERR)
<type>BUF(*)
INTEGER COUNT ,DATATYPE,DEST,MSGTAG,COMM,REQUEST,IERR
```

Xabar jo’natish uchun kechiktirilgan so’rovni rasmiylashtirish. Bunda jo’natish amali boshlanmaydi.

MPI_SEND va MPI_ISEND, proseduralarining uchta modifikatsiyasiga o’xshash MPI_SEND_INIT prosedurasining uchta qo’shimcha varianti mavjud:

- MPI_BSEND_INIT buferlab xabar jo’natish uchun kechiktirilgan so’rov rasmiylashtirish;
- MPI_SSEND_INIT sinxronlab xabar jo’natish uchun kechiktirilgan so’rov rasmiylashtirish;
- MPI_RSEND_INIT tayorligi bo’yicha xabar jo’natish uchun kechiktirilgan so’rov rasmiylashtirish.

MPI_RECV_INIT(BUF,COUNT,DATATYPE,SOURCE,MSGTAG,COMM, REQUEST,IERR)

<type> BUF(*)

INTEGER COUNT,DATATYPE,SOURCE,MSGTAG,COMM,REQUEST,IERR

REQUEST parametri qiymatiga mos ma’lumot almashish amalini bajarish uchun kechiktirilgan so’rov yaratish. Amal blokirovkalanmagan holda ishga tushadi.

MPI_START(REQUEST,IERR)

INTEGER REQUEST,IERR

REQUEST massivining dastlabki COUNT elementlari qiymatiga mos ma’lumotlar almashish amallarini bajarish uchun COUNT ta kechiktirilgan so’rovlarni belgilash. Amal blokirovkalanmagan holda ishga tushadi.

MPI_STARTALL(COUNT,REQUEST,IERR)

INTEGER COUNT,REQUEST,IERR

Blokirovkalamaydigan amallardan farqli ravishda kechiktirilgan so’rov yordamida ishga tushirilgan amal tugagandan keyin REQUEST parametri qiymati saqlanib qoladi va keyinchalik uni ishlatish mumkin.

MPI_REQUEST_FREE(REQUEST,IERR)

INTEGER REQUEST,IERR

Joriy prosedura REQUESTparametri bilan bog’liq ma’lumotlar strukturasini o’chiradi. U bajarilgandan keyin REQUEST parametri MPI_REQUEST_NULL qiymatida o’rnatiladi. Agar ushbu so’rov bilan bog’liq amal bajarilayotgan bo’lsa, u holda uning ishi tugatiladi.

Navbatdagi misol halqa topologiyasida qo’shni jarayonlar o’rtasida ikki yo’nalishdagi amallar uchun kechiktirilgan so’rovlar o’rnatiladi. Amallarning o’zi esa navbatdagi siklning xar bir yaqinlashuvida yuklanadi. Sikl tugaganidan keyin kechiktirilgan so’rovlar o’chiriladi.

```
prev=rank-1
next=rank+1
if(rank.eq.0) prev=size-1
if(rank.eq.size-1) next=0
callMPI_RECV_INIT(rbuf(1),1,MPI_REAL,prev,5,
MPI_COMM_WORLD,reqs(1),ierr)
callMPI_RECV_INIT(rbuf(2),1,MPI_REAL,next,6,
MPI_COMM_WORLD,reqs(2),ierr)
callMPI_SEND_INIT(sbuf(1),1,MPI_REAL,prev,6,
MPI_COMM_WORLD,reqs(3),ierr)
callMPI_SEND_INIT(sbuf(2),1,MPI_REAL,next,5,
MPI_COMM_WORLD,reqs(4),ierr)
do i=...
  sbuf(1)=...
  sbuf(2)=...
call MPI_STARTALL(4,reqs,ierr)
  ...
call MPI_WAITALL(4,reqs,stats,ierr) ;
 ...
end do
call MPI_REQUEST_FREE(reqs(1),ierr)
call MPI_REQUEST_FREE(reqs(2),ierr)
call MPI_REQUEST_FREE(reqs(3),ierr)
call MPI_REQUEST_FREE(reqs(4),ierr)
```

8. Suniy neyron to’rlarini o’rgatishda parallel algorimlardan foydalanish

Ko’plab hisoblash ishlari katta miqdordagi amallarni bajarishni talab etadi va zamonaviy mashinalar ham bu masalalarni hal etishda ko’p vaqt sarflaydi. Bir qancha masalalarni hal etishda qat’iy belgilangan vaqt talab etiladi, aks holda masala o’z dolzarbligini yo’qotadi. Ko’plab masalalarni hal etishda neyron tarmoqlari mexanizmi qo’llaniladi, ko’rsatilgan parametrlarni tanlash me’zoni sifatida qo’yidagini formulani ko’rsatish mumkin[19,20,21,22]:

$$\varepsilon = \|d - y\| = \sqrt{\sum_{i=1}^{N}\sum_{j=1}^{p}\left(d_{i,j} - y_{i,j}\right)^2} = \sqrt{\sum_{i=1}^{N}\sum_{j=1}^{p}\left(d_{i,j} - F(\vec{x}_i, \vec{w})_j\right)^2} \quad (1)$$

bu yerda $d_{i,j}, y_{i,j}$ chiquvchi qiymatlar.

Neyron to’rlarini modellashtirishda parallel algoritmlarning samaradorligini baholash qo’yidagicha amalga oshiriladi:

Neyron to’rlarini o’rgatish algoritmlarini modellashtirish murakkab hisoblash jarayonlarini yuzaga keltiradi. SNT larning ko’plab ilovalarda hisoblash murakkabligi SNTlarda o’rnatilgan aloqalar sonining kavdratiga proporsional bo’lishi ko’zatilgan. SNTlarni o’rgatish bir necha ko’nlab amalga oshirilishi mumkin. Bu jarayonlarni jadallashtirish tadqiqotchiga aniq natijalar olishiga ko’maklashadi. SNTlar orqali yechilayotgan masalalar xozirda ko’payib bormoqda. Xozirda SNTlarning matematik va ularni o’rgatish modellarini yaratishda ko’pyadroli prosesslarda ishlash uchun parallel algoritmlarini yaratish ommaviy tus olib ketdi.

Ushbu ishda har xil mashinalarda dasturni ishlatish orqali parallel algoritmlarni ishlash vaqtining samaradorligi tekshirildi. Masalan Intel Core i7 920 prosessori har biri ikkita mantiqiy prosessordan tashkil topgan 4 fizik yadroli kompyuterda ko’rsatkich 2·4 = 8.

Tajriba natijalari qo’yidagi jadvalda ko’rsatilgan.

Parallel algoritmning ishlash vaqtining jadvali

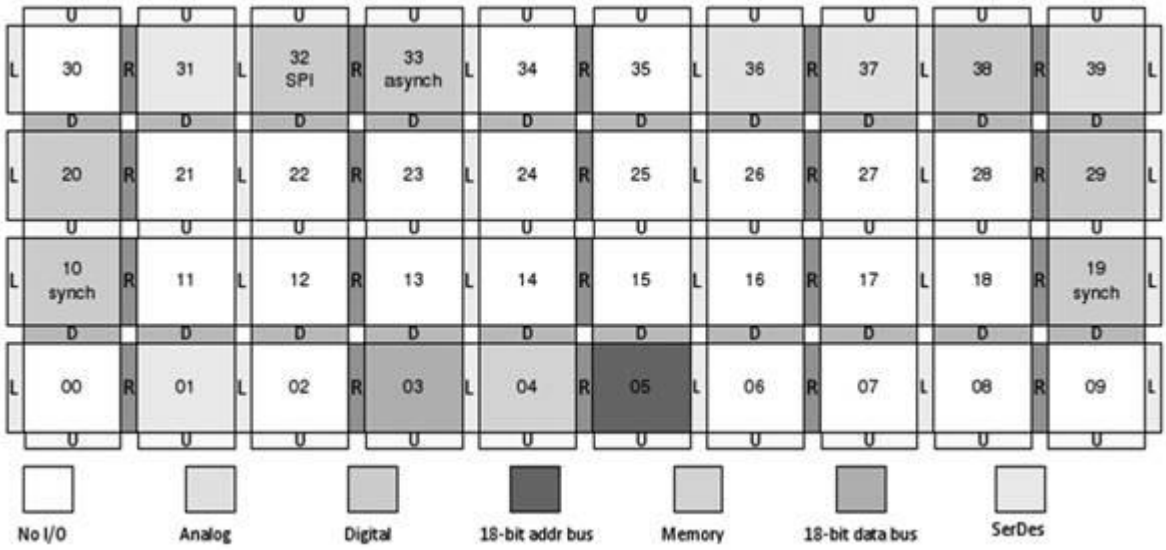
№	Prosessor	Chiqarilgan vaqti	Dasturning ishlash	Dasturning parallel	Samaradorligi

			vaqti	ishlagan vaqti	
1.	Pentium 4 HT@3.0	2004	13.07	11.35	57.6
2.	Pentium D 805@2.66	2006	13.17	7.52	87.5
3.	Core 2 Duo E6400@2.14	2006	6.28	3.58	87.5
4.	Xeon E5345 (x2)@2.33	2006	6.50	1.89	81.4
5.	Core 2 Quad Q6600@2.4	2007	6.34	1.57	100.8
6.	Core 2 Duo E7200@2.53	2008	5.33	3.06	87.1
7.	Core 2 Duo E8400@3.0	2008	4.46	2.48	90.1
8.	Core 2 Quad Q8300@2.53	2008	5.34	1.59	84.1
9.	Core i7 920(w/o HT)@2.66	2008	4.78	1.31	90.9
10.	Core i7 860@2.8	2009	4.56	1.12	51.0

Qo’yida neyron to’rlarini o’rgatishda ko’pyadroli muhitdan foydalanish usulini ko’rsatib utamiz.

SEAForth40 (S40C18) prosessori

SEAforth-S40C18 prosessori IntellaSys firmasi tomonidan ishlab chiqilgan matrisali prosessorlar oilasiga mansub. Bu massiv 40 ta yadrodan tashkil topgan bo’lib, har biri ma’lumot va dasturlarni saqlab turadi..

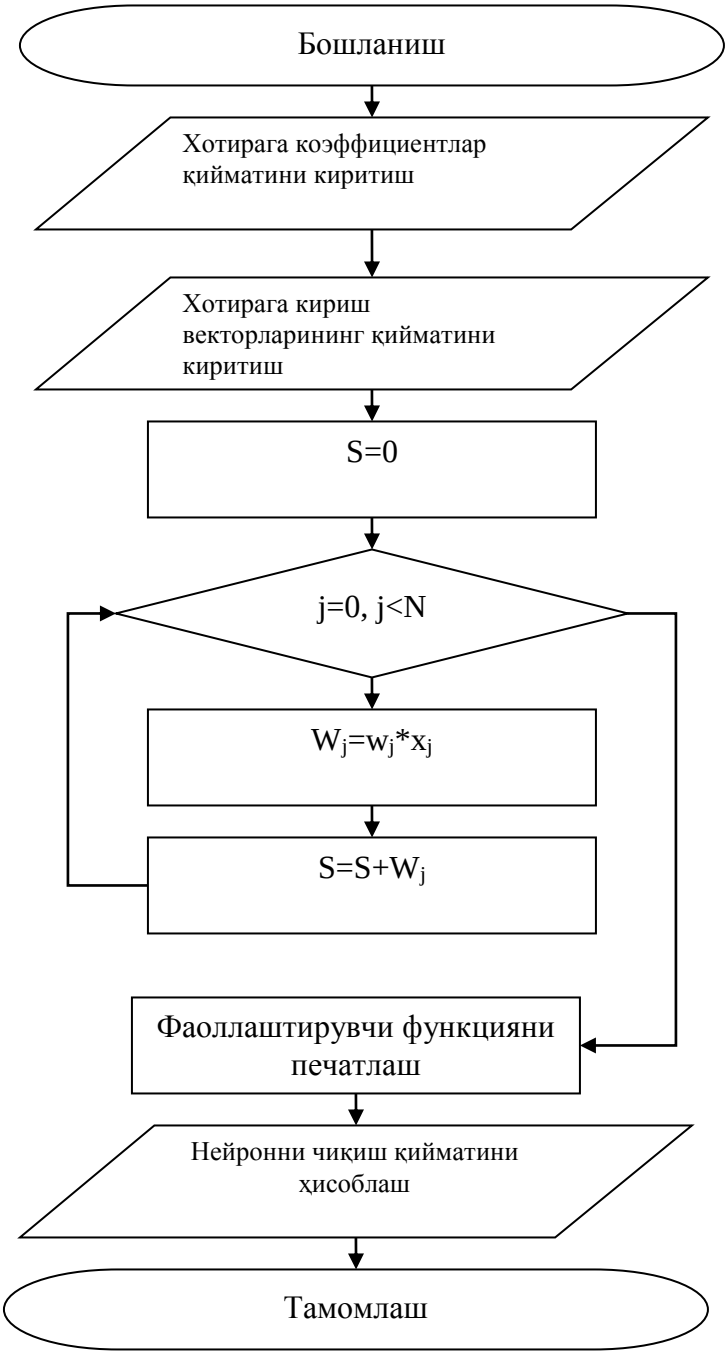


9-rasm. SeaForth-S40C18 prosessor qurilmasi.

Har qaysi tugun o’zining xususiy taktlariga ega, bu esa SEAforth asosida ma’lumotlarni qayta ishlashning asinxronli tizimini yaratadi.

Neyronlarni qiymatlarini hisoblash ma’lumotlar oqimini hosil qiladi. Avval ma’lumotlar neyronga kiradi, keyin esa ular mos ravishda og’irlik koeffisentlariga

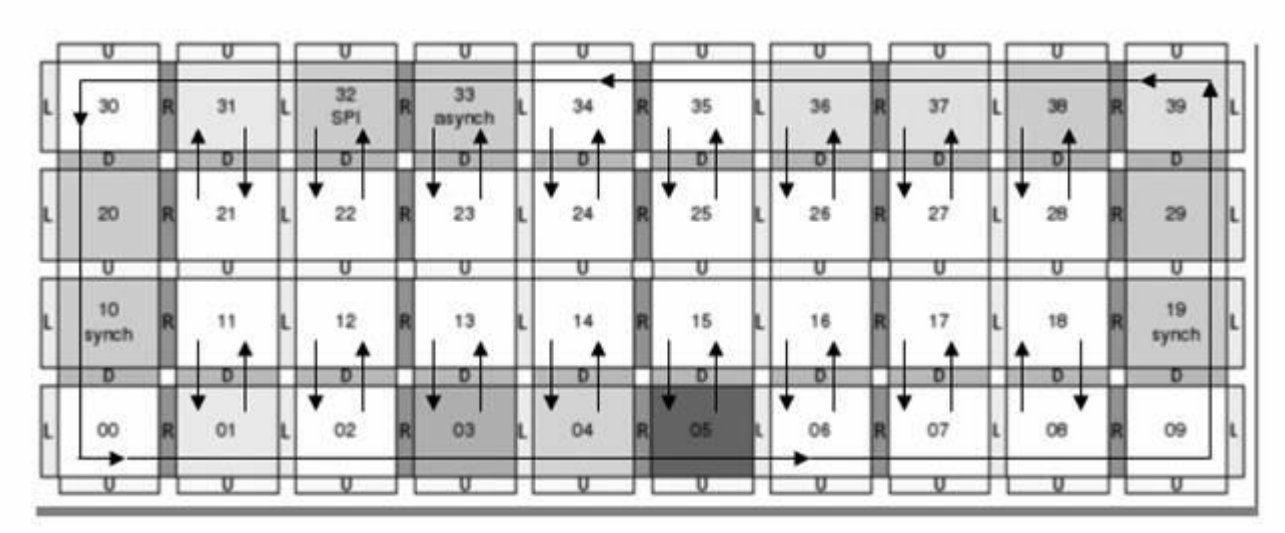
ko’paytiriladi. Olingan qiymatlarning yig’indisi uning faollashtiruvchi funksiyasiga beriladi va ma’lumot chiqishga o’zatiladi. Buning umumlashgan blok sxemasi qo’yidagicha tasvirlanadi:



10-rasm. Neyronning umumlashgan chiqish qiymatini hisoblashning blok sxemasi

Vektorlar qiymatini ikki hil yo’l yordamida kiritish mumkin: Barcha vektorning qiymati operativ yadroga kiritiladi yoki ketma ket ravishda birorta portdan kiritiladi.

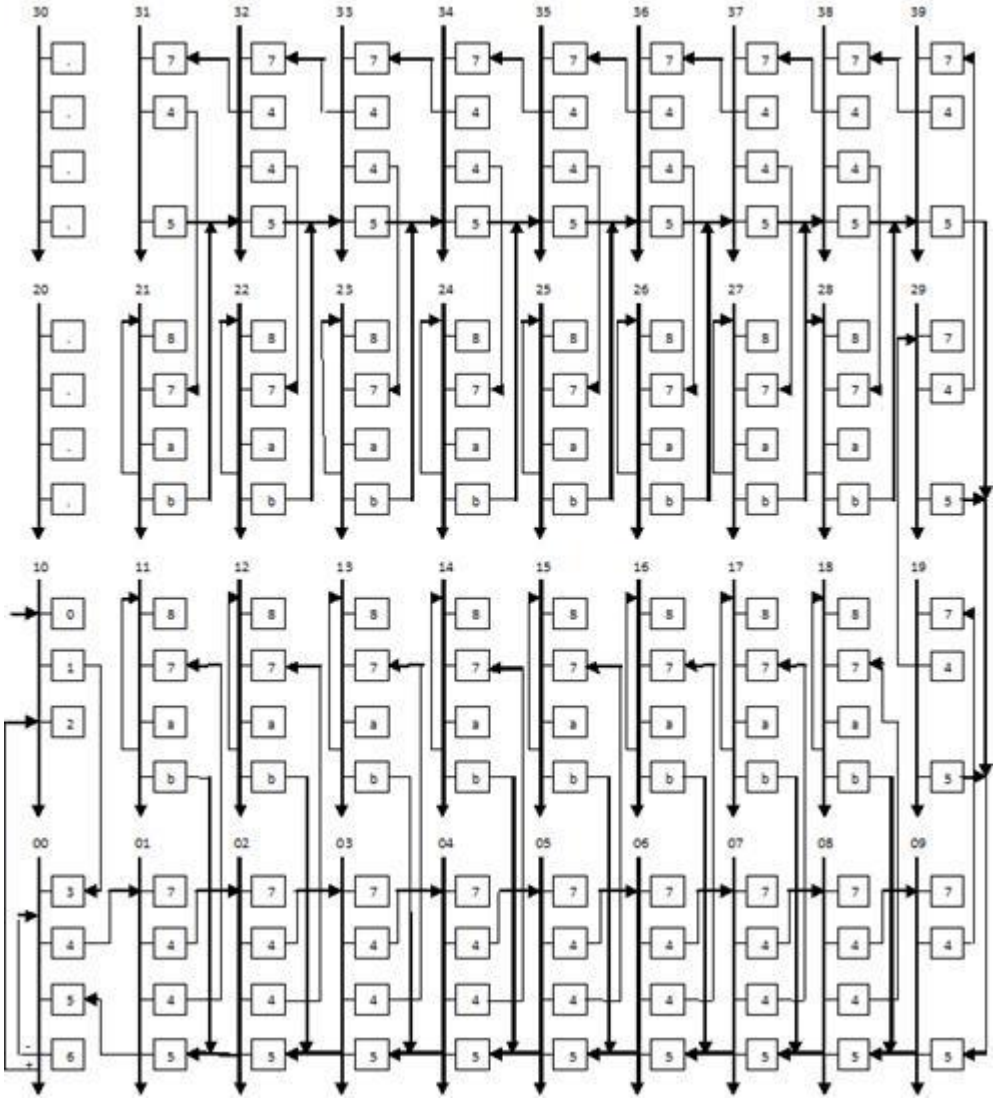
Qo’yidagi rasmda SNT larining SEAforth S40C18 prosessorida hisoblanish jarayoni keltirilgan.



11-rasm. SNTni prosessorda hisoblash jarayoni.

SNTni SEAforth S40C18 prosessorda hisoblash jarayoni umumiy algoritmi: [23,24,25]

1) n ulchamli $\vec{x}$ kiruvchi vektor №10 yadroga ketma-ket sinxron port orqali kiritiladi, sungra №00 yadroga kiritiladi;
2) bitlar bo’yicha $\vec{x}$ vektor №00 yadrodan “konveyr” ga tushadi, neyronning i-qatlami tushadi (№11-18, №21-28 yadrolar);
3) shundan sung chiquvchi vektor $\vec{y}$ olinadi va “konveyr”ga tushadi, ya’ni №00 yadroda shakllanadi;
4) tarmoq arxitekturasiga muvofiq SPI interfeysi orqali yadroga dastur kodi sifatida yuklanadi;
5) sungra oldingi iterasiyadan olingan vektor №00 yadrodan “konveyr”ga tushadi va keyingi neyronga o’tadi;
6) tarmoqning chiqish vektori olingandan sung, u №10 yadroga tushadi va sinxron port orqali tashqariga chiqadi.

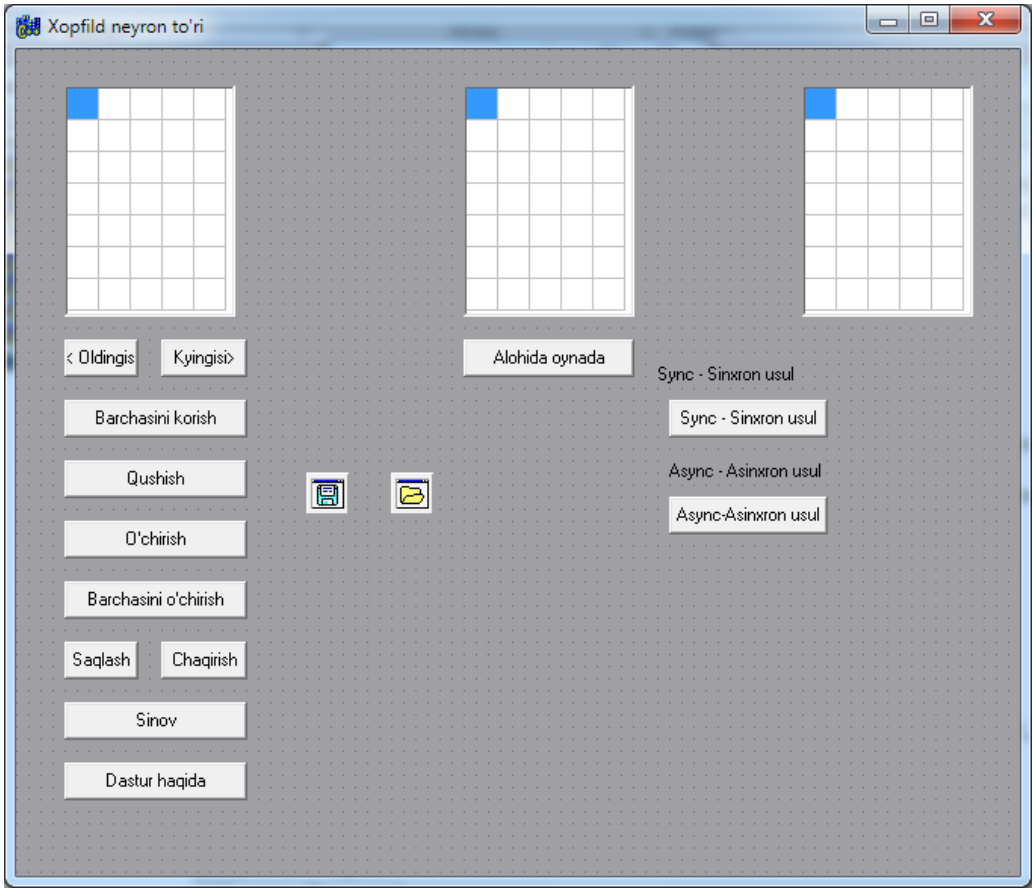


12-rasm. Prosessorda SNTni hisoblash jarayoni.

4-bob. Tasvirlarni tanishda neyron tarmoqlarining dasturiy vositasi

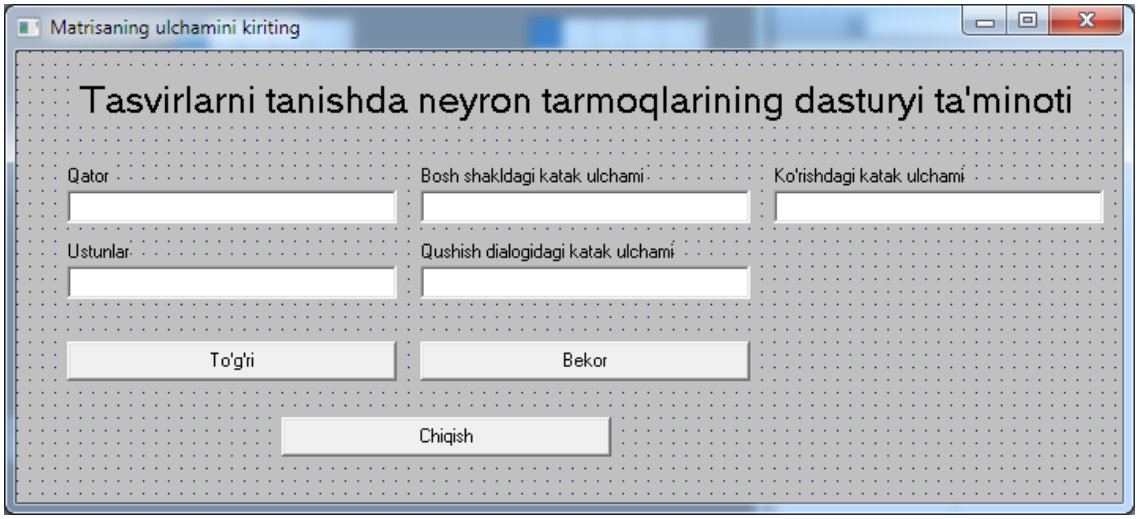
1.Dasturiy vositasini ishlab chiqish

Dasturiy vositani S++ dasturlash tilida yaratamiz, buning uchun Borland C++ Bulder dasturini ishga tushiramiz va shaklga zarur elementlarni o’rnatamiz (4.1-rasm) shakl nomini “Xopfild neyron to'ri” deb nomlaymiz va unga StringGrid1, StringGrid2, StringGrid3 larni o’rnatamiz. Shundan sung, zarur tugmachalarni shaklga qo’yib chiqamiz. Ularga mos datur kodlarini ilovada keltiramiz.



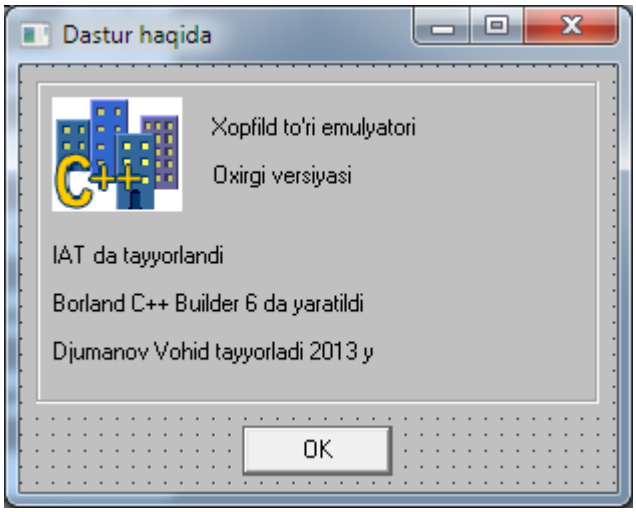
4.1-rasm. Xopfildning neyron to’ri shakli.

Eng oldin paydo bo’ladigan “Matrisaning o’lchamlarini kiritish” shaklni yaratamiz, bu shaklda dastur matrisadagi qatorlar soni, ustunlar soni, bosh shakldagi kataklar o’lchami, qo’shish dialog oynasidagi kataklar o’lchami, ko’rishdagi kataklar o’lchami va ularni ishga to’shiruvchi tugmalardan shaklni hosil qilish, bekor qilish, chiqish tugmachalarini o’rnatamiz va ularga mos dastur kodlarini kiritamiz. Dastur kodlarini ilovada keltiramiz.(4.2-rasm)



4.2-rasm. Matrisaning o’lchamlarini kiritish.

Dastur haqida ma’lumotlarni beruvchi shaklni yaratamiz, buning uchun unga oddiy Label elementlaridan foydalanib, dasstur nomi, dastur versiyasi, tayerlangan joyi, qaysi tilda yaratilganligi va kim tomonidan yaratilganligi haqida ma’lumotlarnikiritamiz va u elementlarni mos ravishda nomlaymiz.

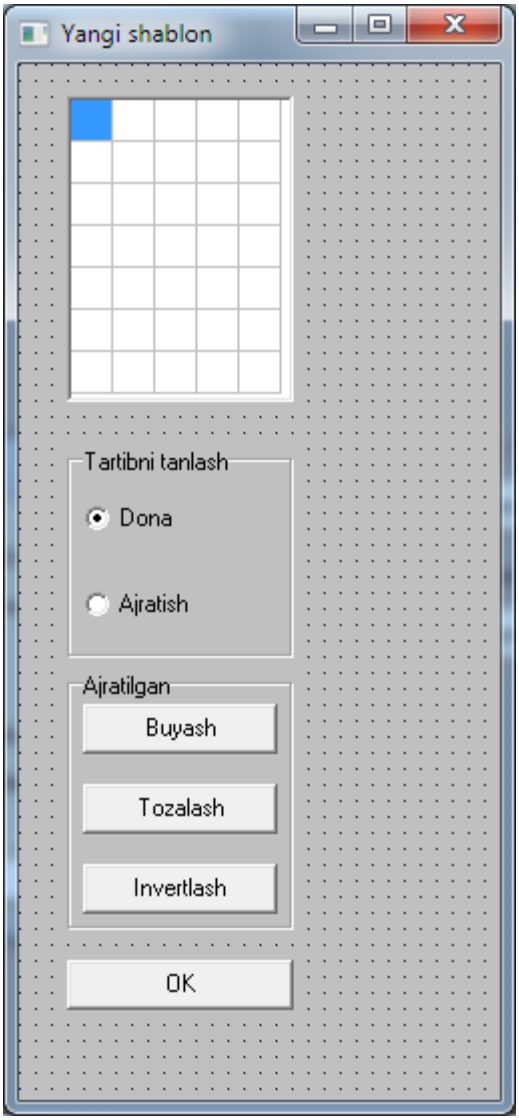


4.3-rasm. Dastur haqida.

Ushbu shakl dasturni yaratilganligi to’g’risidagi barcha ma’lumotlarni chiqarish uchun ishlatiladi.

Keyingi shakl, yangi shablonlarni yaratish uchun yordam beruvchi dasturiy kodlarni shakllantiradi. Bu shaklni yaratish uchun RadioGroup, StringGrid va Button elementlaridan foydalanamiz, bu elementlar mos ravishda shaklni yarattishda, unga ishlov berish usullari va unga yuklattilgan vazifalarni bajarish uchun foydalaniladi.

Tugmachalarga mos kodlarni yaratamiz, bu tugmachalar buyash, tozalash, invertlash va ishni bajarish uchun foydalaniladi.



4.4.-rasm. Yangi shablon yaratish shakli.

Ushbu yaratilgan shakl yangi shablon yaratishda foydalaniladi.(4.4-rasm.)

2. Hayot faoliyati havfsizligi

Zararli va havfli omillarning paydo bo'lishi. Insonning ish joyda ishlash EHM bilan doimiy ravishda bog'liq bo'lgani uchun, mehnat havfsizligi masalasi uning xavfsizligini ta'minlash nuqtaiy nazaridan qaraladi va ulardan foydalanishda inson sog'ligini saqlash shartlari ko'rib chiqiladi.

Laboratoriya xonalarida ishlashda quyidagi xavfli va noqulay omillarni ajratib ko’rsatish mumkin:

- Meteorologik muhit sharoiti (laboratoriya mikroklimati);
- Anomal yoritishlar;
- Shovqinning yuqori darajasi;
- Nurlanishning yuqori darajasi;
- Elektr toki bilan zararlanish havfi;
- Yuqori darajadagi psixofiziologik tulqinlanish;
- Yong’in havfi.

Elektron qurilmalarga nisbattan namlik darajasiga talablar, va mikromuhit, inson hayot faoliyati uchun har hil belgilanadi. EHMlardan foydalanishda GOST talablariga muvofiq EHMlar uchun xona namligi 40% - 60% dan kam bo’lmasligi va inson uchun xona namligi 70% dan kam bo’lmasligi zarur. Shuning uchun, xonaning optimal xavo namligi 50% qilib qabul qiligan. Xavo xarorati namligi insoning yuqori nafas yo’llaridagi qoplamaga ta’sir qilib, inson salomatligiga ta’sir qiladi va ish faoliyatini kamaytiradi.

Xonaning yuqori xarorati insonning organizimi tempraturasini qizdirishi va issiq urishiga olib keladi. Xonaning past xarorati esa inson tanasini sovishigi olib yelishi va shamaollashiga olib keladi. Shuning uchun xona xarorati GOST talablariga muvofiq – 22-24^0 S ni tashkil etishi zarur, nisbiy namligi – 40-60%ni tashkil etishi , xavoning xarakati 0.1 m/s bo’lishi zarur. Bulardan tashqari xonadagi yorug’lik talab darajasida bo’lishi kerak.

Xonadagi 1 dona EHMning shovqini [8]**,** 30-68 db diapazonida bo’lishi kerak. GOST talablariga muvofiq shovqin 40 db da bshlishi zarur. Biz bilamizki laboratoriyalarda bitta EHM emas balki bir nechta EHMlar bo’ladi. Xonaning uta shovqiln bo’lishi ham inson ish faoliyatiga tasir etadi. Shovqin inson asab tizimiga ta’sir etib e’tiborni susaytiradi va xotirani pasayishiga olib keladi. Ionizasiya [7] deb

har xil elektr zaryadlarining qushilishi bilan hosil bo'ladigan nurlanishga aytiladi. Ionlashtiruvchi nurlanishga: gamma-nurlanish, rentgen, korpuskulyar, infraqizil, mikrotulqinli va boshqa turdagilar kiradi. 10 sm. O'zoqlikda to'rgan monitordan chiqayotgan rentgen nurlanishlar 100 mkR/ch bo'ladi, ionizasiya darajasi esa ikkala zaryadlarda 1 sm^3 xavoda 1500-5000 bo'dadi.. Ishlab chiqarishdagi laboratoriyadagi nurlanish manbasi monitorlar hisoblanadi.. Yuqori darajadagi nurlanishda insonning boshi qattiq og'riydi, diqqati kamayadi va ishga susligini kuchaytiradi.

Laboratoriyadagi elektr qurilmalar havfli lillar manbasi hisoblanadi. EHMlar bevosita 220 V tok quchlanishi va 50 Gs chastota bilan ishlagani uchun ishchilarni elektr tokidan ehtiyot etilishi talab etildi. Elektr tokidan zararlanish tizimli blokning korpusidan, elektr simlaridan va boshqa shunga uxshashlardan kelib chiqishi mumkin. Elektr tokidan zararlanish inson terisini kuydirishi, uning tanasiga utgan tok ichki organlarini yuqori darajada kuydirishi mumkin. Inson tomonidan olingan zarar *elektrli* (organ suyuqligini zararlantirishi, jumladan qonni), *mexanik* (organizimning har xil darajada zararlanshi) va *biologik* (normal xarakatlanuvchi organizmda biologik o'zgarishlarga olib kelishi) xarakatlarga bo'linadi.

Psixofiziologik havfli faktorlar xarakteri jihatidan qo'yidagilarga bo'linadi: *fizik* (statistik va dinamik) va *asab-psixologik* [7] (uta aqliy kuchlanish, tahliliy tasirlanish, ishning monotonliligi va emosional ta'sirlanish). Bu holatlar bo'lmasligi uchun ishchi monitor oldida 4 sutkada va kompyuterga 30000 belgidan ko'p belgi kiritmaslik kerak.

Laboratoriyada yong'in xavfi EHMlarga simlarning notug'ri ulanishi, qisqa tutashuv va tok kuchlanishidan kelib chiqishi mumkin. Bulardan tashqari isitish tizimlaridan noto'g'ri foydalanish, ventelyasiya tizimini noto'g'ri qurilganligi va ishchilarning olovdan noto'g'ri foydalanishi oqibatida sodir etilishi mumkin.

Havfli va zararli omillardan himoyalanish tadbirlari. Laboratoriyada xovoni normallashtirish uchun tabiy va suniy ventilyasiyadan foydalanish zarur.

Laboratoriyada foydalaniladigan tabiiy ventilyasiyaga ishlab chiqarilgan tashkil etilmagan ventilyasiya urlari kiradi. Ulardan foydalanishning bir qator kamchiliklari mavjud: Binoga kirayotgan xavo isitilmaydi va namlanmaydi, shuning uchun xona uchun moslashtirilgan kamchiliklardan holi bo'lgan ventilyasiyalardan foydalanish kerak. Xonaning mos haroratini normal ushlab turish uchun qishda markazlashtirilgan issiqlik manbai, yozda esa mos ventilyasiyalardan foydalanish zarur.

1) Xonani normal yorug'lik bilan ta'minlash tabiy va suniy yoritish asboblaridan to'g'ri foydalanishni talab etadi. EHM monitorlarini yorug'lik tushishiga mos holda joylashtirish talab etildi.

2) Laboratoriyani shovqindan asrash uchun quyidagi usuldan foydalanish zarur:

- Uni hosil qiluvchi manbani yo'qorish;
- Uni tarqatish.

Maxsus shovqinni kamaytiruvchi amortizasiyalovchi tirkalmadan foydalanish zarur..

Tashqi shovqindan himoyalanish uchun laboratoriyani maxsus derazalar urnatilgan xonalarda joylashtirish.

4) nurlanishni kamaytirish uchun: uning manbalarini aniqlab elektron trubkali monitorlarni kamaytirish, ish vaqtini kamaytirish kabi ishlarni amalga oshirish zarur.

5) Elektr toki urishidan zararlanishni oldini olish uchun qurilmalarni to'g'ri joylashtirish elektr simlarini to'g'ri urnatish va uni ishonchli izolyasiyalash kerak.

6) psixofiziologiik ta'sirlanishni kamatirish uchun ishchi joylarini to'g'ri shakllantirish, ish vaqtini rasional taqsimlash, xonani har xil ranglar bilan jihozlash ekranlarning yorug'lik holatini normallashtirish kabi ishlarni amalga oshirish kerak (belgilarning kontrasti 0,8 belgidan kam bo'lmasligi; ekran yorug'ligi 10 kq/m^2; ekran imkoniyati; tasvirlarning regenerasiya chastotasi 72 MGs chastotadan kam bo'lmasligi).

7) yong’in xavfsizlik choralari (elektr kuchlanishini yerga ulanish simlaridan foydalangan bo’lishi, holatini nazorat qilish, chaqmoq qaytargich bo’lish, qisqa tutashuvdan ximoyalovichlar urnatilganligi), yong’inni dastlabki uchirgichlaridan foydalanish, elektr tokidan avariya holatidagi uchirish uskunalari bilan ta’minlangan bo’lishi zarur.

Xulosa

Dissertasiya ishining asosiy natijalari qo’yidagicha:

1. Maxsus adabiyotlar va Internet resurslarini analiz qilishi natijasida, neyrobiologiya, kibernetika, sunniy intellekt, suniy neyron to’rlari va neyroto’rli o’zi tashkil bo’luvchi, persiptronlar va tasvirlarni mashinada tanish, diskret matematika, ehtimollar nazariyasi va matematik statistika, obyektga yo’naltirilgan dasturlash tili S++ o’rganildi.
2. Muammoni hal etishda har bir belgini xarakteristikalari aniqlandi. Grafik tasvirlarni aniqlashda o’zi tashkil bo’luvchi suniy neyroan to’rlari tashkil etildi va parallel hisoblash tashkil etildi.
3. Ishda neyro-matrisali etalon bazasini yaratuvchi va testlanadigan tasvirlarni qayta ishlaydigan Xopfeld to’ri yaratildi. Yaratish belgilar bo’yicha ikki o’lchamli og’irlik koeffisiyentlarini massiv xarakteristikalarini o’zgartirishni amalga oshiriladi.
4. SNT larida ishlovchi algoritm o’rganildi va uning blok sxemasi, model loyihasi va berilgan usulning talablarini qanoatlantiruvchi dasturiy ilovalari yaratildi.
5. Olib borilgan ish natijasida tasvirlarni tanishdagi usullar o’rganildi, jumladan, raqamlar, lotin va rus grafikasidagi alfavitlar, bundan tashqari barcha xildagi belgilarga qo’llanilishi mumkin.
6. SNT lari modellashtirildi, testlandi va kompyuter dasturi yaratildi.

Foydalanilgan adabiyotlar ro’yxati

1. «Axborotlashtirish to’g’risida» O’zbekiston Respublikasining qonuni. Toshkent shahri, 1993 yil,7 may.
2. Karimov I.A. «O’zbekiston iqtisodiy islohatlarni chuqurlashtirish yo’lida». Toshkent. «O’zbekiston» – 1995y.
3. «Kompyuterlashtirishni yanada rivojlantirish va axborot – kommunikasiya texnologiyalarini joriy etish to’g’risida» O’zbekiston Respublikasi Prezidentining farmoni. 30 may 2002 yil.
4. «Kompyuterlashtirishni yanada rivojlantirish va axborot – kommunikasiya texnologiyalarini joriy etish chora – tadbirlari to’g’risida», O’zbekiston Respublikasi Vazirlar mahkamasining qarori. 6 iyun 2002 yil, 200-son.
5. Dj.Tu., Gonsales R. Prinsipы raspoznovaniya obrazov.Perevod s angliyskogo I.B.Gurevicha. M.:Mir.1978 g. 412 s.
6. Berkinblit M.B. Neyronnыye seti. Uchebnoye posobiye. M.: MIROS i VZMSh RAO, 1993. -96 s.
7. Neyronnыye seti: istoriya razvitiya teorii. Kn. 5: Ucheb. posobiye dlya vuzov. / Pod obщuyey red. A.I. Galushkina, Ya.Z. Sыpkina. - M.: IPRJR, 2001. - 840 s.
8. Kruglov V.V., Borisov V.V. Iskusstvennыye neyronnыye seti. Tioriya i praktika.- 2-ye izd.-M.:Goryachaya liniya-Telekom. 2002 g. 382 s.
9. Kallan R. Osnovnыye konsepsii neyronnыx setey.: Per. S angl.-M.: Izdatelskiy dom “Vilyams”, 2001 g. 287 s.
10. Gorban A.N., Rossiyev D. A. Neyronnыye seti na personalnom kompyutere.— Novosibirsk: Nauka, 1996. 275 s.
11. Tatuzov A.L. Neyronnыye seti v zadachax radiolokasii. Kn. 28. - M.: Radiotexnika, 2009. - 432 s.
12. Xaykin S. Neyronnыye seti: polnыy kurs, 2e izdaniye. : Per. s anrl. M. Izdatelskiy dom "Vilyams", 2006. 1104 s.

13. Komarsova L.G., Maksimov A.V. Neyrokompyuterы: Ucheb.posobiye dlya vuzov. –ye izd., pererab.i dop.-M: Izd-vo MGTU im. N.E.Baumana, 2004. -400 s.
14. Osovskiy S. Neyronnыye seti dlya obrabotki informasii. Perevod s polskogo. I.D.Rudinskogo. –M.: Finansы i statistika. 2002. 343 s.
15. Rutkovskaya D., Pilinskiy M., Rutkovskiy L. Neyronnыye seti, geneticheskiye i nechetkiye sistemы: Per.s polsk. I.D.Rudinskogo.- Goryachaya liniya – Telekom, 2006. -452 s.
16. Aksenov S.V., Novoselsev V.B. Organizasiya i ispolzovaniye neyronnых setey (metodы i texnologii) / Pod obщ. red. V.B. Novoselseva. – Tomsk: Izd-vo NTL, 2006. – 128 s.
17. Sivoxin A. V. Iskusstvennыye neyronnыye seti. Lab. praktikum / A. V. Sivoxin, A. A. Lushnikov, S. V. Shibanov. – Penza: Izd-vo Penz. gos. un-ta, 2004. – 136 s.
18. Vasilyev V.N., Pavlov A.V. Opticheskiye texnologii iskusstvennogo intellekta / Uchebnoye posobiye. Izd.2. V 2-x t. T.1. SPb: SPbGU ITMO, 2008. 81s.
19. Komashinskiy V.I., Smirnov D.A. Neyronnыye seti ix primeneniye v sistemax upravleniya i svyazi.-M.: Goryachaya liniya –Telekom, 2003.-94 s.
20. Igel S. Improving the Rprop Learning Algorithm / C. Igel, M. Husken // Proceedings of the Second International Symposium on Neural Computation. – 2000. – S. 115-121.
21. Otkrытaya spesifikasiya OpenMP API dlya parallelnogo programmirovaniya http://openmp.org/wp/.
22. V.V. Simonov. Osenka effektivnosti parallelnых algoritmov Dokladы TUSURa, № 1 (21), chast 2, iyun 2010
23. Svobodnaya Internet ensiklopediya «Vikipediya». Statya «SSE»: http://ru.wiki pedia.org/wiki/SSE.
24. Intel® 64 and IA-32 Architectures Optimization Reference Manual November 2009. http://www.intel.com/assets/pdf/manual/248966.pdf.

25. Otkrытыу standart parallelnogo programmirovaniya dlya geterogennых sred OpenCL. http://www.khronos.org/opencl/.

```
#include <vcl.h>
#pragma hdrstop
#include <cstdio>
using namespace std;
#include "Un_Main.h"
#include "Un_SetNumber.h"
#include "Un_About.h"
#include "Un_add_pattern.h"
#include "UnViewAll.h"
#include "UnExp.h"
#include "Un_functions.h"
#pragma package(smart_init)
#pragma link "trayicon"
#pragma resource "*.dfm"
TFrm_Main *Frm_Main;
int grid_cols=5, grid_rows=7;
int CellSize=20;
NeuralNetHopfield net;
int pattern_idx=-1;
const int
        len1=25,
        len2=20,
        len3=20,
        len4=80,
        len5=80,
        top1=25, left1=25;
__fastcall TFrm_Main::TFrm_Main(TComponent* Owner)
      : TForm(Owner)
{
}
void __fastcall TFrm_Main::gridClick(TObject *Sender)
{
      GridClear(grid3);
      ::gridClick(Sender);
}
void __fastcall TFrm_Main::gridDrawCell(TObject *Sender, int ACol,
    int ARow, TRect &Rect, TGridDrawState State)
{
      ::gridDrawCell(Sender, ACol, ARow, Rect, State);
}
void __fastcall TFrm_Main::Btn_go_syncClick(TObject *Sender)
{
      vector<bool> vec1, vec2;
      vec1.reserve(grid_cols*grid_rows);
      vec2.reserve(grid_cols*grid_rows);
      GridToVector(grid2, vec1);
```

```
	int n;
	if (n=net.restore_pattern_sync(vec1, vec2))
	{
		VectorToGrid(vec2, grid3);
		ShowMessage(IntToStr(n));
	}
	else
		ShowMessage("Neyron to'rida shablon yo'q");
}
void __fastcall TFrm_Main::Btn_add1Click(TObject *Sender)
{
	vector<bool> vec;
	if (GetVector(vec))
	{
		if (net.add_pattern(vec))
		{
			if (pattern_idx==-1)
				pattern_idx=0;
			LoadNet();
		}
		else
		{
			ShowMessage("To'rda bunday shablon bor");
		}
	}
}
void __fastcall TFrm_Main::Btn_removeClick(TObject *Sender)
{
	net.del_pattern(pattern_idx);
	if (pattern_idx == net.get_patterns_number())
	{
		pattern_idx--;
	}
	LoadNet();
}
void __fastcall TFrm_Main::Btn_prevClick(TObject *Sender)
{
	pattern_idx--;
	LoadNet();
}
void __fastcall TFrm_Main::Btn_nextClick(TObject *Sender)
{
	pattern_idx++;
	LoadNet();
}
void __fastcall TFrm_Main::LoadNet()
{
	if (pattern_idx==-1)
	{
```

```
            GridClear(grid1);
            Btn_remove->Enabled=false;
            Btn_prev->Enabled=false;
            Btn_next->Enabled=false;
        }
        else
        {
            VectorToGrid(net.get_pattern(pattern_idx), grid1);
            Btn_remove->Enabled=true;

            if (pattern_idx==0)
                Btn_prev->Enabled=false;
            else
                Btn_prev->Enabled=true;
            if (pattern_idx==net.get_patterns_number()-1)
                Btn_next->Enabled=false;
            else
                Btn_next->Enabled=true;
        }
}
void __fastcall TFrm_Main::Btn_go_asyncClick(TObject *Sender)
{
        vector<bool> vec1, vec2;
        vec1.reserve(grid_cols*grid_rows);
        vec2.reserve(grid_cols*grid_rows);
        GridToVector(grid2, vec1);
        int n;
        if (n=net.restore_pattern_async(vec1, vec2))
        {
            VectorToGrid(vec2, grid3);
            ShowMessage(IntToStr(n));
        }
        else
        {
            ShowMessage("Neyron to'rida shablon yo'q");
        }
}
void __fastcall TFrm_Main::Btn_aboutClick(TObject *Sender)
{
        AboutBox = new TAboutBox(Application);
        AboutBox->ShowModal();
        delete AboutBox;
}
void __fastcall TFrm_Main::FormCreate(TObject *Sender)
{
        Frm_SetNumber = new TFrm_SetNumber(Application);
        Frm_SetNumber->ShowModal();
        delete Frm_SetNumber;
        net.set_n(grid_rows*grid_cols);
```

```
for (int i=0; i<this->ComponentCount; i++)
{
	if (this->Components[i]->ClassNameIs("TStringGrid"))
	{
		TStringGrid *p=dynamic_cast<TStringGrid*>(this->Components[i]);
		p->RowCount = grid_rows;
		p->ColCount = grid_cols;
		p->DefaultRowHeight = CellSize;
		p->DefaultColWidth  = CellSize;
		p->Height = p->RowCount*CellSize + p->GridLineWidth*(p->RowCount+1)+2;
		p->Width  = p->ColCount*CellSize + p->GridLineWidth*(p->ColCount+1)+2;
		p->Top  = 50;
		GridClear(p);
	}
}
grid1->Top  = top1;
grid1->Left = left1;
grid2->Top  =grid1->Top;
grid2->Left = grid1->Left+grid1->Width+len4;
grid3->Top  = grid2->Top;
grid3->Left = grid2->Left+grid2->Width+len5;
Btn_prev->Top        = grid1->Top + grid1->Height+len1;
Btn_prev->Left       = grid1->Left;
Btn_prev->Width      = (grid1->Width-len2)/2;
Btn_next->Top        = Btn_prev->Top;
Btn_next->Left       = Btn_prev->Left+Btn_prev->Width+len2;
Btn_next->Width      = Btn_prev->Width;
Btn_view_all->Top    = Btn_prev->Top+Btn_prev->Height+len3;
Btn_view_all->Left   = grid1->Left;
Btn_view_all->Width  = grid1->Width;
Btn_add1->Top        = Btn_view_all->Top+Btn_view_all->Height+len3;
Btn_add1->Left       = grid1->Left;
Btn_add1->Width      = grid1->Width;
Btn_remove->Top      = Btn_add1->Top+Btn_add1->Height+len3;
Btn_remove->Left     = grid1->Left;
Btn_remove->Width    = grid1->Width;
Btn_remove_all->Top  = Btn_remove->Top + Btn_remove->Height+len3;
Btn_remove_all->Left  = grid1->Left;
Btn_remove_all->Width = grid1->Width;
Btn_save->Top        = Btn_remove_all->Top + Btn_remove_all->Height+len3;
Btn_save->Left       = grid1->Left;
Btn_save->Width      = (grid1->Width-len2)/2;
Btn_load->Top        = Btn_save->Top;
Btn_load->Left       = Btn_save->Left+Btn_save->Width+len2;
Btn_load->Width      = Btn_save->Width;
Btn_exp->Top         = Btn_save->Top+Btn_save->Height+len3;
Btn_exp->Left        = grid1->Left;
Btn_exp->Width       = grid1->Width;
Btn_about->Top       = Btn_exp->Top+Btn_exp->Height+len3;
```

```
	Btn_about->Left       = grid1->Left;
	Btn_about->Width      = grid1->Width;
	Btn_add2->Top         = grid2->Top+grid2->Height+len1;
	Btn_add2->Left        = grid2->Left;
	Btn_add2->Width       = grid2->Width;
	Btn_go_sync->Top      = Btn_add2->Top+Btn_add2->Height+len3;
	Btn_go_sync->Left     = Btn_add2->Left;
	Btn_go_sync->Width    = grid2->Width+grid3->Width+len5;
	Btn_go_async->Top     = Btn_go_sync->Top+Btn_go_sync->Height+len3;
	Btn_go_async->Left    = Btn_go_sync->Left;
	Btn_go_async->Width   = Btn_go_sync->Width;
	Lbl_Sync->Left        = grid2->Left;
	Lbl_Sync->Top         = Btn_go_async->Top+Btn_go_async->Height+len3;
	Lbl_Async->Left       = grid2->Left;
	Lbl_Async->Top        = Lbl_Sync->Top+len3;
	this->ClientWidth  = grid3->Left+grid3->Width+left1;
	this->ClientHeight = Btn_about->Top+Btn_about->Height+top1;
	LoadNet();
}
void __fastcall TFrm_Main::Btn_saveClick(TObject *Sender)
{
	if (SaveDlg->Execute())
	{
		FILE *fp;
		if ((fp=fopen(SaveDlg->FileName.c_str(), "w"))==NULL)
		{
			ShowMessage("Fayl ochilmayapti"+SaveDlg->FileName);
			return;
		}
		int n=grid_rows*grid_cols;
		fwrite(&n, sizeof(int), 1, fp);
		int n_pat = net.get_patterns_number();
		for (int i1=0; i1<n_pat; i1++)
		{
			for (int i2=0; i2<n; i2++)
			{
				bool tmp=net.get_pattern(i1)[i2];
				fwrite(&tmp, sizeof(tmp), 1, fp);
			}
		}
		fclose(fp);
	}
}
void __fastcall TFrm_Main::Btn_loadClick(TObject *Sender)
{
	if (OpenDlg->Execute())
	{
		FILE *fp;
		if ((fp=fopen(OpenDlg->FileName.c_str(), "r"))==NULL)
```

```
            {
                ShowMessage("Fayl ochilmayapti "+OpenDlg->FileName);
                return;
            }
            int n;
            if (fread(&n, sizeof(n), 1, fp)!=1)
            {
                ShowMessage("Fayl formati noto'g'ri"+OpenDlg->FileName);
                return;
            }

            if (n!=grid_rows*grid_cols)
            {
                ShowMessage("Faylda "+OpenDlg->FileName+" shablon ulchami boshqa ("+n+")");
                return;
            }
            net.clear_patterns();
            bool *bool_arr = new bool[n];
            while (fread(bool_arr, sizeof(bool), n, fp)==n)
            {
                vector<bool> bool_vec;
                for (int i1=0; i1<n; i1++)
                {
                    bool_vec.push_back(bool_arr[i1]);
                }
                net.add_pattern(bool_vec);
            }
            delete [] bool_arr;

            if (net.get_patterns_number()==0)
                pattern_idx=-1;
            else
                pattern_idx=0;

            LoadNet();

            fclose(fp);
        }
}
void __fastcall TFrm_Main::FormKeyPress(TObject *Sender, char &Key)
{
    switch (Key)
    {
    case VK_ESCAPE:
        this->Close();
        break;
    }
}
void __fastcall TFrm_Main::Btn_remove_allClick(TObject *Sender)
```

```
{
	net.clear_patterns();
	pattern_idx=-1;
	LoadNet();
}
void __fastcall TFrm_Main::Btn_view_allClick(TObject *Sender)
{
	ShowMessage("Qullanilmagan");
/*
	ViewAllFrm = new TViewAllFrm(Application);
	ViewAllFrm->ShowModal();
	delete ViewAllFrm;
*/
}
void __fastcall TFrm_Main::Btn_expClick(TObject *Sender)
{
	ExpFrm = new TExpFrm(Application);
	ExpFrm->ShowModal();
	delete ExpFrm;
}
void __fastcall TFrm_Main::Btn_add2Click(TObject *Sender)
{
	vector<bool> vec, start_vec;
	GridToVector(grid2, start_vec);
	if (GetVector(vec, start_vec))
	{
		VectorToGrid(vec, grid2);
	}
}
```

2-fayl

```
#include <vcl.h>
#pragma hdrstop
#include "UnExp.h"
#include "Un_Main.h"
#include "UnViewAll.h"
#include "Un_functions.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TExpFrm *ExpFrm;
__fastcall TExpFrm::TExpFrm(TComponent* Owner)
	: TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TExpFrm::Btn_GoClick(TObject *Sender)
{
	int
```

```
        n_prob,
        n_perc;
    try
    {
        n_prob = StrToInt(LblEd_Num->Text);
        n_perc = StrToInt(LblEd_Perc->Text);
    }
    catch (EConvertError &err)
    {
        ShowMessage("Noto'g'ri kiritish");
        return;
    }

    if (n_perc<1 || n_perc>100)
    {
        ShowMessage("Foyiz noto'g'ri kiritilgan");
        return;
    }

    int n_iter_tot=0;
    int n=net.get_patterns_number();
    for (int i1=0; i1<n; i1++)
    {
        for (int i2=0; i2<n_prob; i2++)
        {
            vector<bool> vec=net.get_pattern(i1);
            CorruptVector(vec, n_perc);
//*
            TForm *FormTest = new TForm(Application);
            TStringGrid *pattern, *corrupt_pattern;
            pattern = new TStringGrid(FormTest);
            pattern->Parent=FormTest;
            corrupt_pattern = new TStringGrid(FormTest);
            corrupt_pattern->Parent=FormTest;
            for (int i1=0; i1<FormTest->ComponentCount; i1++)
            {
                if (FormTest->Components[i1]->ClassNameIs("TStringGrid"))
                {
                    TStringGrid *p=dynamic_cast<TStringGrid*>(FormTest-
>Components[i1]);
                    p->RowCount = grid_rows;
                    p->ColCount = grid_cols;
                    p->DefaultRowHeight = CellSizeView;
                    p->DefaultColWidth  = CellSizeView;
                    p->Height = p->RowCount*CellSizeView + p->GridLineWidth*(p-
>RowCount+1)+2;
                    p->Width  = p->ColCount*CellSizeView + p->GridLineWidth*(p-
>ColCount+1)+2;
                    p->FixedRows = 0;
```

```
                        p->FixedCols = 0;
                        p->Top    = 20;
                    }
                }
                pattern->Left   = 20;
                corrupt_pattern->Left = pattern->Left+pattern->Width+20;
                VectorToGrid(net.get_pattern(i1), pattern);
                VectorToGrid(vec, corrupt_pattern);
                FormTest->ShowModal();
                delete FormTest;
//*/
                vector<bool> vec_restore;
                int n_iter;
                n_iter=net.restore_pattern_sync(vec, vec_restore);
                ShowMessage(IntToStr(n_iter));
                n_iter_tot += n_iter;
            }
        }
        int n_iter_avg=n_iter_tot/(n*n_prob);
        ShowMessage("Shaklni tiklash uchun qadamlarning o'tacha qiymati" + IntToStr(n_iter_avg));
}
//---------------------------------------------------------------------------
void CorruptVector(vector<bool> &vec, int perc)
{
        if (vec.empty() || perc<1 || perc>100)
                return;
        int n_to_invert = vec.size()/100*perc;
        vector<bool> mask(vec.size(), false);
        while (n_to_invert!=0)
        {
                int idx;
                while (mask[idx=rand()%vec.size()]);
                mask[idx]=true;
                vec[idx]=!vec[idx];
                n_to_invert--;
        }
}
```

3-fayl

```
#include <vcl.h>
#pragma hdrstop
using namespace std;
#include "Un_functions.h"
#pragma package(smart_init)
void __fastcall gridClick(TObject *Sender)
{
        TStringGrid *p=(TStringGrid *)Sender;
        if (p->Cells[p->Col][p->Row] == '1')
```

```
            p->Cells[p->Col][p->Row] = ' ';
        else
            p->Cells[p->Col][p->Row] = '1';
}
void __fastcall gridDrawCell(TObject *Sender, int ACol,
     int ARow, TRect &Rect, TGridDrawState State)
{
        TStringGrid *p=(TStringGrid *)Sender;
        p->Canvas->Brush->Color = clBlack;
        if (p->Cells[ACol][ARow] != ' ')
            p->Canvas->FillRect(Rect);
}
void GridClear(TStringGrid* grid)
{
        for (int i=0; i<grid->ColCount; i++)
            for (int j=0; j<grid->RowCount; j++)
                grid->Cells[i][j] = ' ';
}
void GridToVector(TStringGrid *grid, vector<bool> &vec)
{
        vec.clear();
        vec.reserve(grid->ColCount*grid->RowCount);

        for (int i=0; i<grid->ColCount; i++)
            for (int j=0; j<grid->RowCount; j++)
            {
                if (grid->Cells[i][j]=='1')
                    vec.push_back(true);
                else
                    vec.push_back(false);
            }
}
void VectorToGrid(vector<bool> &vec, TStringGrid *grid)
{
        int idx=0;
        for (int i=0; i<grid->ColCount; i++)
            for (int j=0; j<grid->RowCount; j++)
                if (vec[idx++])
                    grid->Cells[i][j]='1';
                else
                    grid->Cells[i][j]=' ';
}
```

4-fayl

```
#include <vcl.h>
#pragma hdrstop
#include "UnViewAll.h"
#include "Un_Main.h"
#include "Un_functions.h"
```

```
#pragma package(smart_init)
#pragma resource "*.dfm"
TViewAllFrm *ViewAllFrm;
int CellSizeView = 20;
const int
        len1=20,
        len2=20;
__fastcall TViewAllFrm::TViewAllFrm(TComponent* Owner)
      : TForm(Owner)
{
}
void __fastcall TViewAllFrm::FormKeyPress(TObject *Sender, char &Key)
{
      switch (Key)
      {
      case VK_ESCAPE:
            this->Close();
            break;
      }
}
void __fastcall TViewAllFrm::FormCreate(TObject *Sender)
{
//*
      int n=net.get_patterns_number();
      int top=len1, left=len1;
      for (int i1=0; i1<n; i1++)
      {
            TStringGrid *grid = new TStringGrid(this);

            grid->Parent = this;
            grid->RowCount = grid_rows;
            grid->ColCount = grid_cols;
            grid->FixedRows = 0;
            grid->FixedCols =0;
            grid->DefaultRowHeight = CellSizeView;
            grid->DefaultColWidth  = CellSizeView;
            grid->Height = grid->RowCount*CellSizeView + grid->GridLineWidth*(grid-
>RowCount+1)+2;
            grid->Width  = grid->ColCount*CellSizeView + grid->GridLineWidth*(grid-
>ColCount+1)+2;
            //grid->Top  = 0;
            //grid->Left = 0;
            VectorToGrid(net.get_pattern(i1), grid);
      }
//*/
}
```

5-fayl

```
#include <vcl.h>
#include <cstdlib>
#pragma hdrstop
USEFORM("Un_Main.cpp", Frm_Main);
USEFORM("Un_add_pattern.cpp", Frm_Add);
USEFORM("Un_SetNumber.cpp", Frm_SetNumber);
USEFORM("Un_About.cpp", AboutBox);
USEFORM("UnViewAll.cpp", ViewAllFrm);
USEFORM("UnExp.cpp", ExpFrm);
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
	randomize();

	try
	{
		Application->Initialize();
		Application->Title = "Xopfild";
		Application->CreateForm(__classid(TFrm_Main), &Frm_Main);
		Application->Run();
	}
	catch (Exception &exception)
	{
		Application->ShowException(&exception);
	}
	catch (...)
	{
		try
		{
			throw Exception("");
		}
		catch (Exception &exception)
		{
			Application->ShowException(&exception);
		}
	}
	return 0;
}
```

6-fayl

```
#pragma hdrstop
#include "neural_net.h"
NeuralNetHopfield::NeuralNetHopfield(unsigned int n, bool autoinit):
n(n), autoinit(autoinit), is_init(false), weights(n), out1(-1), out2(1), porog(0)
{
	for (vector< vector<int> >::iterator i=weights.begin(); i!=weights.end(); i++)
	{
		(*i).resize(n);
	}
```

```
}
NeuralNetHopfield::NeuralNetHopfield():
n(0), autoinit(false), is_init(false), out1(-1), out2(1), porog(0)
{
}
unsigned int NeuralNetHopfield::get_n()
{
	return n;
}
void NeuralNetHopfield::set_n(unsigned int n)
{
	patterns.clear();
	is_init=false;
	this->n=n;
	weights.resize(n);
	for (vector< vector<int> >::iterator i=weights.begin(); i!=weights.end(); i++)
	{
		(*i).resize(n);
	}
}
NeuralNetHopfield::~NeuralNetHopfield()
{
}
bool NeuralNetHopfield::add_pattern(vector<bool> &pattern)
{
	if (find(patterns.begin(), patterns.end(), pattern)==patterns.end())
	{
		patterns.push_back(pattern);
		is_init=false;
		if (autoinit)
			init_weights();
		return true;
	}
	else
	{
		return false;
	}
}
bool NeuralNetHopfield::del_pattern(unsigned int pattern_num)
{
	if (pattern_num<patterns.size())
	{
		patterns.erase(patterns.begin()+pattern_num);
		is_init=false;
		if (autoinit)
			init_weights();
		return true;
	}
	else
```

```
        {
            return false;
        }
}
unsigned int NeuralNetHopfield::restore_pattern_sync(vector<bool> &vec1, vector<bool> &vec2,
unsigned int max_iter)
{
        if (patterns.empty() || n==0 || vec1.size()!=n)
                return 0;

        if (!is_init)
                init_weights();
        vector<int> tmp_vec1, tmp_vec2;
        tmp_vec1.reserve(n);
        tmp_vec2.reserve(n);
        unsigned int n_iter=0;
        for (vector<bool>::iterator i=vec1.begin(); i!=vec1.end(); i++)
                tmp_vec2.push_back(bool_to_int(*i));
        do
        {
                tmp_vec1 = tmp_vec2;

                for (unsigned int i1=0; i1</*tmp_vec2.size()*/n; i1++)
                {
                        tmp_vec2[i1] = calc_neuron(i1, tmp_vec1);
                }
                n_iter++;
        } while(tmp_vec2!=tmp_vec1);
        vec2.clear();
        vec2.reserve(n);
        for (vector<int>::iterator i=tmp_vec2.begin(); i!=tmp_vec2.end(); i++)
                vec2.push_back(int_to_bool(*i));
        return n_iter;
}
unsigned int NeuralNetHopfield::restore_pattern_async(vector<bool> &vec1, vector<bool> &vec2,
unsigned int max_iter)
{
        if (patterns.empty() || n==0 || vec1.size()!=n)
                return 0;
        if (!is_init)
                init_weights();
        vector<int> tmp_vec;
        tmp_vec.reserve(n);
        for (unsigned int i=0; i<n; i++)
                tmp_vec.push_back(bool_to_int(vec1[i]));
        bool is_updated;
        vector<bool> mask;
        unsigned int n_neurons_done;
        unsigned int neuron_num;
```

```
	unsigned int n_iter=0;
	do {
		is_updated=false;
		mask.assign(n, false);
		n_neurons_done=0;
		while (n_neurons_done<n)
		{
			while (mask[neuron_num=rand()%n]);
			n_neurons_done++;
			mask[neuron_num]=true;
			int tmp_out=calc_neuron(neuron_num, tmp_vec);
			if (tmp_out!=tmp_vec[neuron_num])
				is_updated=true;
			tmp_vec[neuron_num]=tmp_out;
		}
		n_iter++;
	} while (is_updated);
	vec2.clear();
	vec2.reserve(n);
	for (unsigned int i=0; i<n; i++)
		vec2.push_back(int_to_bool(tmp_vec[i]));
	return n_iter;
}
bool NeuralNetHopfield::init_weights()
{
	if (n==0)
	{
		return false;
	}
	unsigned int i1, i2, i3;
	for (i1=0; i1!=/*weights.size()*/n; i1++)
		for (i2=0; i2!=/*weights[i1].size()*/n; i2++)
		{
			weights[i1][i2]=0;
			if (i1==i2)
				continue;
			for (i3=0; i3!=patterns.size(); i3++)
				weights[i1][i2] += bool_to_int(patterns[i3][i1])*bool_to_int(patterns[i3][i2]);
		}
	is_init=true;
	return true;
}
int NeuralNetHopfield::calc_neuron(unsigned int neuron_num, vector<int> &layer)
{
	int sum=0;
	for (unsigned int i1=0; i1</*weights.size()*/n; i1++)
		sum += weights[i1][neuron_num]*layer[i1];
	if (sum>porog)
		return out2;
```

```
        else if (sum<porog)
                return out1;
        else
                return layer[neuron_num];
}
int NeuralNetHopfield::bool_to_int(bool var)
{
        return var?out2:out1;
}
bool NeuralNetHopfield::int_to_bool(int var)
{
        if (var==out1)
                return false;
        else if (var==out2)
                return true;
        else
                return false;
}
unsigned int NeuralNetHopfield::get_patterns_number() const
{
        return patterns.size();
}
vector<bool> & NeuralNetHopfield::get_pattern(int num)
{
        return patterns[num];
}
bool NeuralNetHopfield::clear_patterns()
{
        if (n==0)
        {
                return false;
        }
        patterns.clear();
        is_init=false;
        return true;
}
#pragma package(smart_init)
```